Dynamically provisioned virtual infrastructures : specification, allocation and execution

Guilherme Piêgas Koslovski

Dynamically provisioned virtual infrastructures: specification, allocation and execution

Guilherme Piêgas Koslovski
advised by
Dr. Paulo Gonçalves and *Dr. Pascale Vicat-Blanc*

June 24, 2011

Acknowledgement

Initially, I would like to thank my friend and advisor Pascale Vicat-Blanc for giving me opportunity to do a PhD and for trusting in my potential during all these years. She offered me the chance of learning from her knowledge on research and life experiences. Also, I would like to express gratitude to Paulo Gonçalves for accepting to advise me during the final steps of this thesis. He was always available for answering a lot of questions and solving many administrative problems.

I am also immensely grateful to the reviewers, Dr. Yves Denneulin and Dr. Olivier Festor, and the other members of the jury, Dr. Djamal Zeghlache and Dr. Frédéric Desprez.

During all these years in Lyon, I had the opportunity to meet special people as my dear friends Fabienne Anhalt, Sebastien Soudan, Marcos Dias de Assunção, Dinil mon Divakaran, Olivier Mornard, and Doreid Ammar. Thanks for the time we have spent together discussing about research and life. Surely, these discussions have helped this thesis take shape.

I also express my gratitude towards past and current members of the RESO team, Augustin Ragon, Philippe Martinez, Romaric Guillier, Suleyman Baikut, Hugo Caro, Ghislain Landry, Ludovic Hablot, Anne-Ccile Orgerie, Sylvie Boyer, and Thomas Begin, to name just a few. In addition, I thank the members of Lyatiss and some external collaborators, Tram Truong Huu, Johan Montagnat, Cédric Westphal, and Wai-Leong Yeow.

Marcelo Pasin deserves a special thanks. This friend has guided me to my PhD, in addition to helping me getting settled in Lyon.

Considering my support in Brazil, my heartfelt thanks to my wife Lutielen Paniz for her support, patience, and for providing me with encouragement during rough times. Without her kind words during the difficult moments and her happiness during the good times, this thesis could not have been possible. I do not have enough words to express how much I thank you.

Finally, I thank my family, Tanea, Alcides, Raquel, and Marina (in memoriam) for giving me the base principles of my life. Their support was essential during all these years.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Virtual Infrastructures (VIs) have recently emerged as result of the combined on-demand provisioning of Information Technology (IT) resources (consolidated by Cloud Computing) and dynamic virtual networks (introduced by Cloud Networking). By combining IT and network virtualization to realize dynamic service provisioning, the VI concept is turning the Internet into a world-wide reservoir of interconnected resources, where computational, storage, and communication services are available on-demand for different users and applications.

A VI is fully virtualized, provisioned and managed by an Infrastructure Provider (InP). During the VI's lifetime, the requesting user is given full control over the aggregation of IT and network resources. Several projects are studying mechanisms to create, allocate, deploy, and manage VIs, exploring technologies such as virtualization of machines, links, switches, and routers for composing and providing isolated and time-limited virtual infrastructures.

The innovation introduced by VIs brought along a set of challenges requiring the development of new models, technologies, and procedures to assist the migration of existing applications and services from traditional infrastructures to VIs. The complete abstraction of physical resources and network topologies, coupled with the indeterminism of required computing and communication resources to execute applications, turned the specification and composition of a VI into a challenging task. In addition, efficiently mapping a set of VIs onto a distributed and large-scale physical substrate is an *NP-hard* problem. Besides considering common objectives of InPs (e.g., efficient usage of the physical substrate, cost minimization, increasing revenue), an effective allocation algorithm should consider the users' expectations (e.g., allocation quality, data location and mobility).

This thesis contributes to these challenges and related ongoing research initiatives by proposing the following:

A language for describing virtual infrastructures: We propose the *Virtual Infrastructure Description Language (VXDL)* as a descriptive and declarative language that allows users and systems to completely describe the relevant components of a virtual infrastructure, including the set of virtual resources, the network topology, and the organization of the internal timeline [6] [1]. The main advantages of such language are to abstract the request formulation from the effective provisioning and thus increase the application portability. VXDL has been proven to be an efficient solution for describing VIs, being adopted by several applications and solutions for provisioning dynamic VIs. In addition, an open forum for discussions and improvements is being launched (`http://www.vxdlforum.org/`). This forum will allow developers and InPs to exchange experiences, suggestions, and extensions, guiding the standardization of VXDL.

Specifying and executing distributed applications: The execution of a distributed application is facilitated by a mechanism for translating workflows into VI specifications. The mechanism extracts information from workflows and combines it with the users' expertise on the application's behavior. Both node and network requirements are addressed.

For a proof of concept, a large-scale distributed application has been executed atop VIs provisioned following different specifications. The results highlight the importance and impact of data location and proximity of virtual nodes [5]. In addition, the capability of specifying elastic VIs results in an efficient tradeoff between cost and performance [2].

Allocating resources to virtual infrastructures: We propose an approach to reduce the search space in an automatic and efficient way, accelerating the process of VI allocation and finding an effective mapping solution. In addition, our allocation algorithm exploits attributes defined in VXDL, and is guided by a problem formulation that reconciles the perspectives of both users and InPs. Experimental results have shown that by using our optimized algorithm, the objectives of InPs and users are satisfied [4] [VXAlloc] [VXCap].

Providing reliable virtual infrastructures: Finally, we propose and develop a mechanism where a user can delegate the reliability expectations of his application to the InP, contracting a transparent service. By using VXDL, each VI component can request a specific level of reliability. The internal InP mechanism translates these requirements into the effective number of replica nodes and links that must be provisioned together with the original VI. The results demonstrate that when a failure happens the service automatically activates the replica nodes, keeping the entire reliability management completely transparent to the running application [7].

Résumé

Le concept d'*Infrastructures Virtuelles* (*VI,* ou *Virtual Infrastructures*), approvisionnées dynamiquement, a émergé récemment suite la combinaison de l'approvisionnement de ressources informatiques (consolidés par le calcul dans le nuage) d'une part, et des réseaux virtuels dynamiques (introduits par les réseaux de nuage) d'autre part. Grâce la virtualisation combinée des ressource de calcul et de réseau pour l'approvisionnement dynamique de services, le concept de VI a transform l'Internet en un réservoir mondial de ressources interconnectées, où des services de calcul, de stockage et de communication sont disponibles á la demande, pour des utilisateurs et applications avec des besoins différents.

Une VI est un infrastructure de calcul et de réseau complètement virtualisée, qui est approvisionnée et gérée par un *Infrastructure Provider (InP)* ou fournisseur d'infrastructure en français. Pendant sa durée de vie, le contrôle sur la VI est transféré á l'utilisateur émetteur de la requête, qui a désormais le contrôle total sur cette agrégation des ressources de calcul et de réseau. Plusieurs projets ont proposé des mécanismes pour créer, allouer, déployer et gérer des machines, liens, commutateurs et routeurs, afin de pouvoir composer et fournir des infrastructures virtuelles isolées ayant une durée de vie limitée.

Avec l'innovation des VIs viennent aussi un ensemble de nouveaux défis nécessitant le développement de nouveaux modèles, technologies et procédures, pour assister la migration d'applications et de services existants d'infrastructures traditionnelles vers des VIs. L'abstraction complète des ressources physiques et de la topologie réseau, et l'indéterminisme dans les besoins des applications en termes de ressources de calcul et de communication ont fait de la spécification et de la composition de VI un problème difficile. En outre, l'allocation efficace d'un ensemble de plusieurs VIs sur un substrat physique distribué à grande échelle est un problème *NP-hard*. En plus de considérer les objectifs traditionnels des VIs (par exemple un usage efficace du substrat physique, un cout minimal, un revenu croissant), un algorithme d'allocation efficace doit également satisfaire les attentes des utilisateurs (par exemple la qualité de l'allocation, la localisation des données et la mobilité).

Ce manuscrit contribue aux initiatives de recherche en cours et est organisées autour des propositions suivantes :

Un langage pour décrire des infrastructures virtuelles : nous proposons VXDL (pour *Virtual Infrastructure Description Language* - langage de description d'infrastructures virtuelles), qui permet aux utilisateurs et aux systèmes de décrire entièrement les composants pertinents d'une infrastructure virtuelle, comprenant un ensemble de ressources virtuelles, leur topologie réseau et leur organisation temporelle [6] [1]. VXDL est reconnu comme étant une solution efficace pour décrire des infrastructures virtuelles. Il est adopté par plusieurs applications pour approvisionner des VIs dynamiques. De plus, un forum ouvert de discussions pour la progression du langage est en cours de développement (`http://www.vxdlforum.org/`). Ce forum permettra aux développeurs et aux InPs d'échanger leurs expériences, suggestions et propositions d'extensions, pour conduire VXDL à la standardisation.

Spécification et exécution d'applications distribuées : l'exécution d'applications distribuées est facilitée par un mécanisme qui traduit un flux de travail en une spécification de VI. Ce mécanisme extrait l'information nécessaire des flux de travail pour la combiner avec l'expertise qu'a l'utilisateur sur le comportement de l'application. Cela répond aux besoins la fois des nœuds de calcul et du réseau. Pour valider ce concept, une application distribuée à large échelle est exécutée plusieurs fois dans des VIs, approvisionnées selon différentes spécifications à chaque fois. Les résultats mettent en évidence l'importance et l'impact de la localisation des données et de la proximité entre nœuds virtuels [5]. De plus, la possibilité de spécifier des VIs élastiques permet d'aboutir à un meilleur compromis entre coût et performance [2].

Allocation d'infrastructures virtuelles : une solution d'allocation est développée pour réduire l'espace de recherche d'une façon automatique et efficace. Elle accélère le processus d'allocation et trouve une solution d'allocation efficace. Cette solution a été brevetée. De plus, notre algorithme d'allocation exploite les attributs proposés dans VXDL. Il est guidé par la formulation du problème qui prend en compte à la fois le point de vue des utilisateurs et celui des InPs. Des résultats expérimentaux montrent qu'en utilisant notre algorithme optimisé, les objectifs des InPs ainsi que ceux des utilisateurs sont satisfaits [4] [VXAlloc] [VXCap].

Fournir des infrastructures fiables : la fiabilité devient un service offert par des InPs. Nous proposons et développons un mécanisme, avec lequel un utilisateur peut déléguer les besoins en fiabilité de son application à l'InP, en souscrivant à un service de fiabilité qui est transparent pour lui-même. En utilisant VXDL, chaque composant d'une VI peut demander un niveau de fiabilité spécifique. Le mécanisme, interne à l'InP, traduit ces niveau de fiabilité en le nombre effectif de répliques de nœuds et de liens qui doivent être approvisionnés en même temps que la VI de base. Des résultats d'expérience montrent que lorsqu'une défaillance se produit, le service active automatiquement les répliques des nœuds, tout en gardant la gestion de la fiabilité complètement transparente pour les applications en cours d'exécution [7].

One
Introduction

1.1 Motivation

The Internet is constantly evolving, metamorphosing from a communication network into a computation-and-communication system. Nowadays, users can reserve and utilize resources and services on a *pay-as-you-go* basis according to their applications' requirements [Buyya, 2009]. The maturity of resource virtualization and Grid technologies has contributed to the emergence of virtualized infrastructures and business models such as Cloud Computing, which allow users to tap into a virtually unlimited reservoir of resources [Rochwerger et al., 2009].

Cloud Computing comes at various shapes and flavors, where typically different layers of an Information Technology (IT) infrastructure are virtualized and delivered to end users as services. Examples include Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). By renting only the resources that they need, when they need, small organizations can reduce their costs with IT infrastructure as they no longer need to spend on buying and managing their own servers. In fact, more and more companies have opted to outsource their IT infrastructure to Cloud providers [Rosenberg and Mateos, 2010].

Until recently, several infrastructures virtualized only server and storage resources, and paid little attention to network. Cloud Networking has changed this scenario by enabling dynamic provisioning of virtual links to interconnect virtual server components. A combination of Cloud Computing and Cloud Networking allows for the creation of entirely virtualized infrastructures (e.g., IT and network) atop a distributed physical substrate. A *Virtual Infrastructure (VI)* is a fully virtualized IT infrastructure that appears to be physical, but in reality, shares the underlying physical substrate with other VIs during a given time frame.

A VI can be formulated as a graph of computing and communication resources that carries substantial information. The virtual IT or network components are the vertices (e.g., nodes, routers, storage), interconnected by a set of virtual links, the edges. The dynamic provisioning of virtualized resources enables each component of a VI to vary its capacity during its lifetime. Moreover, a component can be present only in specific stages of a VI's lifetime. *The resulting VI graph is a malleable and elastic entity*, composed and modeled according to users' and applications' requirements.

VIs have a well-defined lifecycle composed of specification, allocation, provisioning,

and release. For illustrating the use of VIs, the bottom part of Figure 1.1 presents a physical substrate as a combination of several virtualized resources. In this example, three Cloud providers are exposing their resources to the management framework. At the top, a user (e.g., person, institution, or enterprise) wants to run his legacy application (e.g., scientific application, meeting platform, game, corporate control software) in the Cloud. The application, which used to run on a physical substrate (e.g., clusters, Grids, private data centers), now has to migrate to execute on a virtualized infrastructure that is dynamically provisioned, and where each execution is charged according to the resource usage. The VI specification is submitted to a management framework responsible for provisioning the virtual resources according to the user's specifications. The framework knows about the capacities of available resources that compose the distributed physical substrate. With this information, the framework can allocate, provision, and control the VI during its lifetime.

Provisioning a VI poses a series of challenges due to: i) the indeterminism of the required VI composition for running a legacy application efficiently; ii) the abstraction of existing computational power, architecture, and geographical location; and iii) the lack of specific tools to describe and manage VIs, that consider their dynamic aspect.

1.2 Problem and objectives

The composition and usage of VIs, and the management of virtualized physical substrates are challenging tasks because although *a VI is a simple entity, its virtual resources can be defined in a complex manner, carrying substantial information.* By being elastic, a VI can vary its configuration, capacities, and usage on the fly. In addition, virtual resources are mobile and can move around the physical substrate.

Moreover, users who used to know their applications' behavior on physical substrates and are now migrating to the Cloud, have difficulty in defining an efficient VI composition to execute their applications. Different from a private infrastructure, in the Cloud, users have to find the tradeoff between cost and application performance before submitting the application.

From the Infrastructure Providers' (InPs) perspective, allocating and provisioning VIs are not trivial operations. Allocating resources is a complex step of the VI's lifecycle because finding an optimal mapping solution between virtual and physical resources (represented as graphs) is an *NP-hard* problem [Chowdhury and Boutaba, 2009]. Besides to find a satisfactory solution (considering the InP objectives) in an acceptable response time, an efficient heuristic must consider the user's expectations.

New models and tools are required to ease the composition of VI's specification, translating the abstract application requirements in well-defined attributes for composing the virtual resources. Several approaches from different projects have investigated the management of VIs [GEN] [HIP] [SAI] [GEY], but some key-points still require research and engineering efforts. We identified four key open questions characterizing the main objectives of this thesis:

1. **A declarative language capable of easing the specification of distributed applications is required.** A representative model should consider the different

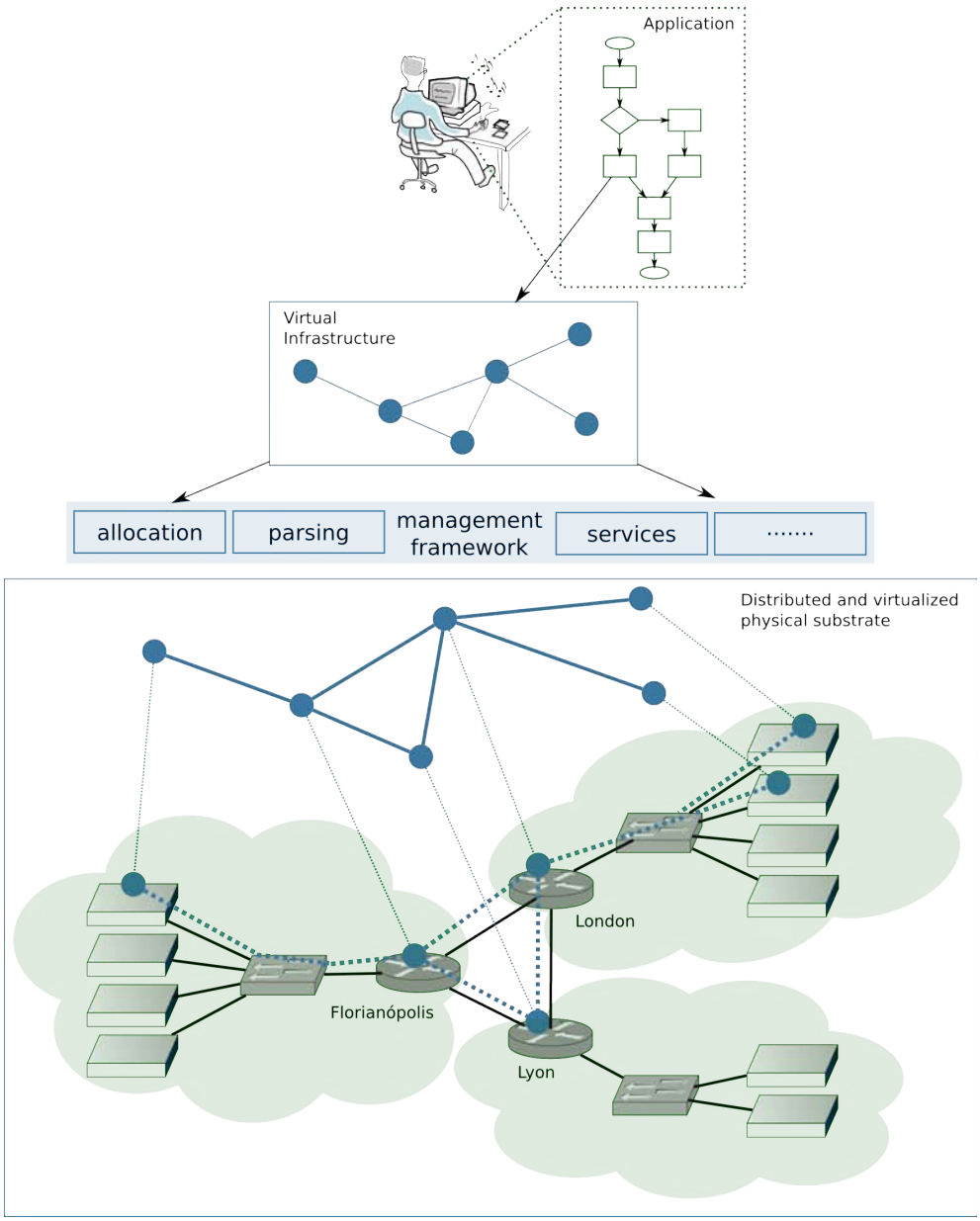


Figure 1.1: A user requesting a set of compute-and-communication resources for executing his application.

aspects of IT and network virtualization, describing their functionalities and composition. The dynamic and elastic aspects of VIs must be considered in the descriptive process. Today, Cloud providers are representing, describing, and requesting virtual resources thanks to different application programming interfaces.

2. **Models, mechanisms, and tools for simplifying the VI manipulation.** As users are migrating their applications from traditional infrastructures (e.g., clusters, Grids, private data centers) to a virtualized and often external substrate based on

dynamic provisioning, new models and tools are required to ease the composition of VIs, translating the abstract application requirements into defined attributes to compose virtual resources.

3. **A solution for allocating VIs considering both users' and InPs' expectations.** Finding an efficient mapping between the users' requirements and the substrate objectives is not trivial. In addition, the VI concept brings new objectives and expectations for the formulation of the allocation problem: i) the elasticity aspect of virtual resources; ii) the user's expectations in terms of application efficiency and quality of experience; iii) and the on-demand provisioning including *pay-as-you-go* model.

4. **Reliability as a Service.** The on-demand service provisioning model explored by InPs leaves opportunities for the creation of new services. The provisioning of new services generally aggregates revenue for InPs, while facilitates the development and execution of complex applications. For example, a physical resource (IT or networking) is subject to the occurrence of random failures. Consequently, a VI allocated on top of a virtualized physical substrate can be also affected by physical failures. Some applications are critical and cannot tolerate failures as the absence of a single physical resource can compromise the whole execution. In addition, in a *pay-as-you-go* execution model, users have no control of the physical substrate, and usually have no information to treat and prevent failures.

These four open questions are routing the contributions of this thesis.

1.3 Contributions

Considering the challenges of the previously defined research issues, and defining VIs as the key entity manipulated in this work, we summarize our contributions in four main areas: i) the creation of a language for describing VIs; ii) the application and validation of the language with a distributed application; iii) efficient algorithms for allocating virtual infrastructures onto virtualized and distributed physical substrate; and iv) the proposition of VI's reliability as a service. Figure 1.2 presents the four contributions as they are distributed over the chapters, describing their interaction and workflow. User input is dealt with at the highest level. A user should be located on top of the highest level, contributing with his application and expertise, and below at the bottom level, a distributed and virtualized physical substrate is present to receive the user VI. This thesis contributions are the intermediate layers, as describe below:

Virtual Infrastructure Description Language (VXDL): The Virtual Infrastructure Description Language (VXDL) enables the description of virtual infrastructures [6]. By using VXDL, a VI can be described and modeled defining the exact requirements in terms of computing resource and network configuration [1] [8]. In addition, VXDL includes elastic and dynamic aspects of VIs and is constructed for enabling an allocation and provisioning of the VI guided by users. By defining some cross-layer attributes, users can interact with the service provisioning, indicating rules and requirements which are indispensable for the efficient execution of their applications.

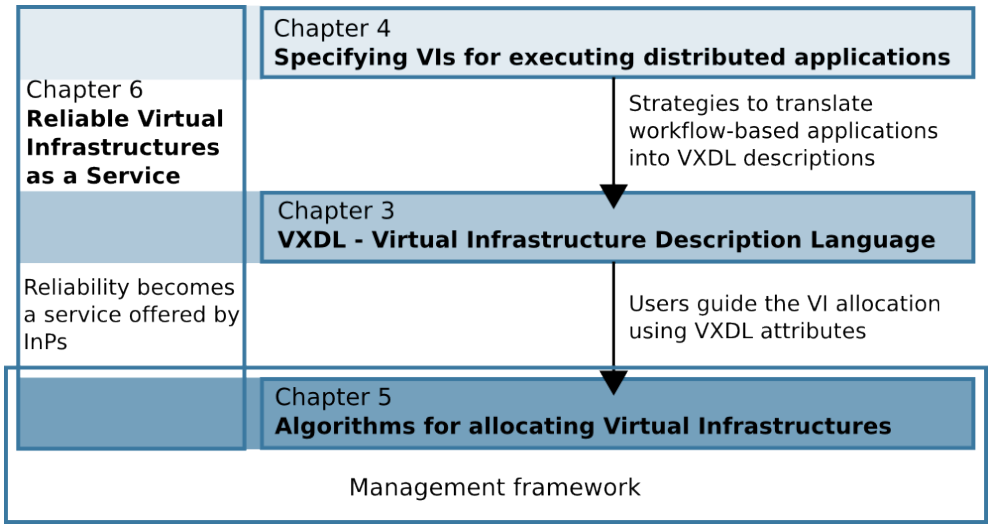


Figure 1.2: The organization of chapters and contributions.

The descriptive capacity of VXDL has been demonstrated through use cases that represent real scenarios. Furthermore, a set of tools have explored VXDL as the main language for describing VIs [LYaTiss, 2011] [MOT].

Specifying VIs for executing distributed applications: As users migrate their applications to the Cloud, a facilitator mechanism is highly required to translate the application's requirements into VXDL documents. For translating workflow-based applications into VI specifications, we propose a mechanism whose translation strategies extract information (e.g., data dependency, execution sequence) from a workflow combining it with the application behavior (given by results of micro-benchmarks and user's expertise) composing VI specifications [5].

A real large-scale distributed application is translated into different VXDL documents, which are them submitted to a management framework. An analysis is performed by comparing the results of the application when executed on top of different VIs. The analysis highlight that an optimal VI composition is essential for finding a good tradeoff between provisioning cost and application performance [2], and the results show the importance of data and nodes location when executing the application inside a VI [5].

Allocating resources to Virtual Infrastructures: As VXDL introduced attributes with which the user can define the behavioral rules (e.g., capacity and composition can vary), and enabled the definition of specific allocation constraints (e.g., a specific location, disabling the resource sharing, and composing shared risk groups), the VI allocation problem becomes also controllable by the user, and not only by the InPs. As a consequence, both users' and InPs' perspectives must be combined into the problem formulation. We formulate the VI allocation problem considering both perspectives and develop an allocation algorithm guided by these constraints and objectives [4]. The results show that our allocation algorithm reduces the costs and substrate fragmentation while guaranteeing the allocation quality.

Reliable Virtual Infrastructures: Following the current reservation and execution model, the provisioning of reliable virtual resources becomes a service provided by the InP. We propose and develop a mechanism capable of interpreting the reliability requirements (specified within VXDL) and compose a reliable virtual infrastructure. VXDL acts as a participant in the Service Level Agreement (SLA) composition. The reliability level requested for each component is translated into the effective number of replicas that must be dynamically provided to guarantee the user's requirements. When a failure occurs, a replica automatically assumes the execution of the resource that failed. The recovery process is abstracted from the user.

The service is evaluated by executing a distributed application with different requirements of reliability level. The results identify the cost for provisioning, and perform a comparison between the reliable service and the application level reliability support [7].

1.4 Thesis organization

This manuscript is organized as follow. Chapter 2 reviews the literature on the techniques and concepts related to virtual infrastructures. Chapter 3 presents our language for describing VIs. Techniques to translate workflow-based applications into VI requirements are described in Chapter 4. The execution of a large-scale distributed application is presenting as a validation of the proposition. Chapter 5 presents our algorithms for allocating virtual infrastructures. In Chapter 6, a mechanism for providing reliable virtual infrastructures as a service is introduced. Finally, Chapter 7 concludes and presents perspectives for future work.

Two
State of the Art

Part of the contents of this chapter was published at the International Journal of Network Management - special issue on Network Virtualization and its Management (IJNM 2010) [1], and at the 20th ITC Specialist Seminar 2009 [8].

2.1 Introduction

The concept of Virtual Infrastructure (VI) is emerging from a combination of resource virtualization techniques (computing and network), Grid Computing, and the innovative service model explored by Cloud Computing. This chapter reviews concepts, technologies and models related to dynamically provisioned virtual infrastructures. Section 2.2 reviews the techniques and concepts proposed to virtualize IT and network resources. Section 2.3 discusses Cloud Computing technology, its key concepts, and future directions. In Section 2.4, the concept and composition of virtual infrastructures are presented, whereas their provisioning is described in Section 2.5. Finally a summary of the literature survey is presented in Section 2.6, positioning the contributions of this thesis.

2.2 Virtualization

The virtualization of resources consists in dematerializing their physical capacity and functionalities, and representing them as virtual entities and services. This technology has been studied for decades in different contexts, such as for virtualizing resources like desktops, applications, systems, storage, and networks [Jeong and Colle, 2010].

Currently, IT infrastructures are using virtualization to *consolidate* computational resources and to *decrease administrative costs* [VMWare, 2009]. Different services with specific and distinct requirements can be allocated on a single computational element or platform, sharing the capacity of physical resources. This consolidation reduces costs with energy consumption, cooling requirements, administration and management, while it increases the platform exploitation [Microsoft, 2009]. In addition, virtualization brings *isolation and protection*: the hardware abstraction that it provides guarantees that services hosted by virtualized elements are kept completely isolated from one another.

The available technologies exploit virtualization following four paradigms for resources sharing: abstraction, partitioning, aggregation, and transformation [Garcia-Espin et al., 2010].

- *Abstraction (1:1)* represents the class of physical resources exposed as a single entity that can be reserved and configured by a user (e.g., the reservation mode explored by Grid'5000 [Cappello et al., 2005]).
- *Partitioning (1:N)* a physical resource into N virtual entities is the main paradigm explored by IT resource virtualization technologies. The N virtual resources share the capacities and functionalities of the physical resource;
- *Aggregation (N:1)* consists in grouping a set of physical resources (IT or network) and exposing the resulting combined capacity as a single entity; and
- *Transformation (N:M)* is a combination of both partitioning and aggregation. The virtual entities exposed by partitioned physical resources can be combined independently of the physical source.

These paradigms are explored at different levels of an IT infrastructure, including hardware, operating systems, programming languages, network equipments and links. The next sections focus on virtualization of computing and network resources.

2.2.1 Computing resources virtualization

In the 1960s, Goldberg explored virtualization techniques to share the computing power of large mainframes [Popek and Goldberg, 1973]. Until the 2000s, most of these techniques remained confined to these platforms, having not been much explored by academia and industry. Due to the decreasing price of microcomputers and the development of modern multitask operating systems, resource virtualization techniques were not mandatory. In recent years, motivated by the increasing computing power of modern platforms coupled with a need for higher flexibility, both academia and industry have directed efforts toward resource virtualization.

Virtualization became an alternative to overcome some difficulties and limitations of existing hardware and operating systems, such as platform dependency, and inefficient

usage of parallel machines. Nowadays, the computing virtualization technology consists in *partitioning* a physical resource and representing it as abstract entities called Virtual Machines (VM) [Rosenblum and Garfinkel, 2005]. Each virtual machine offers features, services, and interfaces that are similar to those of the underlying virtualized hardware (e.g., CPU, memory, and Input/Ouput - I/O - devices). An abstraction layer, the Virtual Machine Monitor (VMM), controls the access to the physical hardware [VMWare, 2010] [Barham et al., 2003]. We review below the techniques used by some of the current VMMs to virtualize CPU, memory, and I/O devices.

2.2.1.1 CPU virtualization

The x86 platform was not initially designed to be virtualized and to share a single physical resource among multiple virtual machines. Although this architecture has four levels of execution privileges, operating systems assume a single and isolated execution, using by default the most privileged level, the *ring 0*.

Virtualizing this platform naturally requires the introduction of a virtualization layer under the operating system. Some explicit instructions cannot be virtualized due to their ring dependency (they can be only executed in the most privileged level). When executed in a higher (unprivileged) level, they require a semantic rewrite, performed by the underlying VMM.

Over the past few years, research on CPU virtualization attempted to overcome these x86 limitations, introducing new concepts and consequently new CPU technologies. The current scenario of CPU virtualization can be decomposed in three main categories [VMWare, 2010]: full virtualization; virtualization at the operating system level and paravirtualization; and hardware-assisted virtualization. The categories are reviewed below and the main technologies, advantages, and drawbacks are summarized in Table 2.1.

Full virtualization: This technology, based on the simulation of physical resources, does not require any update of operating system and legacy software running inside a virtual machine. Non-virtualizable instructions and operations requested by a hosted system are translated into real platform instructions before the execution. More specifically, full virtualization is the only technique that requires no hardware assistance or operating system updates.

Examples of common software that provides full virtualization include VirtualBox [Vir], VMWare [VMW], QEMU [QEM], and Microsoft Hyper-V Server [HYP]. VMWare combines full virtualization with real time binary translation [Sugerman et al., 2001]. This approach translates kernel code replacing privileged instructions with new sequences that have the intended effect on the virtualized hardware. QEMU shares the same approach, also relying on dynamic binary translation to emulate a processor, but provides an intermediate mode for supporting native execution combined with binary translation. One particularity of QEMU is its capability to run on any computer without administrative rights. The technology explored by Microsoft Hyper-V Server is based on virtual partitions. Each partition can be considered as a virtual machine, in which a hosted operating system can be executed. Similarly to the other systems, operations that require privileged access to physical hardware are dynamically intercepted and translated into operations of the native physical machine.

Operating system-level virtualization and paravirtualization: Virtualization at the operating system level explores multiple isolated user-spaces offered by certain kernels implementations. Instead of using the hardware instructions directly, hosted operating systems running on user-space virtual machines use the normal system calls of the host operating system. Consequently, the interface exposed to hosted operating systems can be more limited than the physical interface. Implementations like OpenVZ [OPEb], UML [UML], Linux-VServer [VSE], and Jails [Kamp and Watson, 2000] explore this virtualization technology, also known as *containers*.

Differentiating from this approach, the paravirtualization technology [Whitaker et al., 2002] [Barham et al., 2003] is applied on x86 architectures to overcome their existing limitations. The hosted operating systems have knowledge about the virtualized environment (the existence of a virtual machine). This approach requires modifications on virtualized operating systems introducing specific calls (the *hypercalls* that give the control to VMM) aiming to reduce the execution overhead with architecture dependent instructions, which in a full virtualized scenario must be trapped by the VMM before execution.

Hardware-assisted virtualization: This virtualization technique reflects the efforts of the industry to contribute with the virtualization wave and to explore it commercially. Current physical architectures, such as Intel VT-x [INT] and AMD-V [AMD] introduce a new CPU execution mode that allows the VMM to run on a privileged mode below ring 0. In this scenario, hosted privileged instructions are automatically trapped and directed to the VMM. Dynamic binary translation or paravirtualization are no longer required. Consequently, the changes needed in hosted operating systems are eliminated. Virtual machine monitors such as Xen [Barham et al., 2003], VMWare [VMW], KVM [KVM], and Microsoft Hyper-V [HYP] offer implementations exploring the hardware-assisted virtualization.

Categories	Main technology	Advantages	Drawbacks
Full virtualization	translation, emulation	updates of operating system and legacy software are not required; isolation	execution overhead
Operating system-level virtualization and paravirtualization	operating system hypervisor	better performance than full virtualization	update required in hosted operating system; less isolation
Hardware assisted virtualization	virtualization support directly in hardware	no need for updates of hosted operating systems	execution overhead in some I/O operations

Table 2.1: Summary of virtualization categories including their main technologies, advantages and drawbacks.

2.2.1.2 Memory virtualization

The virtualization of random access memory (RAM) is considered as a critical part of a VMM implementation [Barham et al., 2003]. The traditional approach is to give to VMM the control of a shadow image of the virtual machine's memory-management data structure. With this information, the VMM can precisely control the access to physical memory [Waldspurger, 2002], all active memory pages, and consequently, the paging mechanism. This technique can induce overheads on hosted operating systems as it frequently

accesses and changes page tables. The future direction of memory virtualization is the usage of hardware-managed shadow page tables for accelerating x86 virtualization [Rosenblum and Garfinkel, 2005].

2.2.1.3 Virtualization of I/O devices

The sharing of physical devices among hosted VMs requires a routing access control to efficiently distribute their capacity among the guest operating systems, while guaranteeing their isolation and protection. Some implementations rely on emulation of virtual devices proposing standard and consistent device drivers to improve the portability across platforms [Sugerman et al., 2001] [VMWare, 2010].

Other proposals expose a set of device abstractions [Barham et al., 2003], where access is controlled by the VMM via shared-memory and asynchronous buffer-descriptor rings, similarly to hardware interrupts implemented on operating systems. For example, the virtualization of network interface cards (NICs) enables the creation of virtual networks, including virtual switches and bridges, on a single physical host [Sugerman et al., 2001] [Barham et al., 2003] [8] .

Future research in this area includes the introduction of hardware support on device virtualization [Rosenblum and Garfinkel, 2005], which can efficiently eliminate the I/O overhead imposed by the emulation and abstract device techniques.

2.2.2 Network virtualization

This kind of virtualization has been explored to co-allocate multiple virtualized networks on top of physical networks [4WA]. Similarly to server virtualization, this technology enables the sharing of physical resources, in this case links and network equipments, among hosted virtual resources. This concept has been applied in distinct scenarios and layers of the Open System Interconnection (OSI) model [ISO, 1994], exploring different technologies [Chowdhury and Boutaba, 2010], which have paved the road toward full virtual networks.

We review these technologies, organizing the concepts into four main categories: virtual LANs, virtual private networks, overlay networks, and programable networks.

2.2.2.1 Virtual LAN

Virtual LANs (VLANs), standardized as 802.1Q by IEEE [IEE], are logically isolated networks that can co-habit on a single central network switch or switching hub [Bell et al., 1999] [802.1Q, 2005] [Sincoskie and Cotton, 1988]. Typically, this virtualization technique operates at OSI's layers 2 and 3. VLANs were initially explored to reduce the collision domain of Ethernet segments interconnected by hubs in Local Area Networks (LANs). With the introduction of switches, the collision problem was controlled and VLANs started to be used for reducing broadcast domains.

This virtualization technology does not cope with traffic engineering, except when coupled with 802.1p [Nichols et al., 1998] it enables to prioritize part of the traffic. Current technologies for switches and routers have proposed a set of configurable options for guaranteeing quality of service provisioning, being applied to logically integrate network members into a single view, increasing the organization, segmentation and planing of local networks [Bin Tariq et al., 2009].

2.2.2.2 Virtual Private Network

A Virtual Private Network (VPN) is a communication network used to interconnect one or more IT endpoints geographically distributed over a Wide Area Network [Baugher et al., 1999] [Rosen and Rekhter, 2006], by establishing tunnels over the public communication infrastructure. It was introduced for security reasons (at layer 3) maintaining privacy on data exchange through cryptography, and for accelerating routing operations (at layer 2).

VPNs have been used to interconnect distributed branches of companies, exploring secure and trusted communications [VPN]. In secured VPNs the tunnel is encrypted when transmiting sensitive data, guaranteeing that if someone intercepts the traffic, the information cannot be directly accessed. In trusted VPNs, the Service Provider (SP) responsible for the VPN and the client establish a Service Level Agreement (SLA) that aims to guarantee a secure virtual communication path over the specific trusted physical path.

VPNs comprise three classes of components [Andersson and Madsen, 2005]: Customer Edges (CE), Provider Edge routers (PE), and P routers, as presented in Figure 2.1. Costumer Edges are the source and target border nodes. Provider Edges and P routers are part of the SPs network. PEs are directly connected to the CEs, and P routers are used to interconnect the PEs.

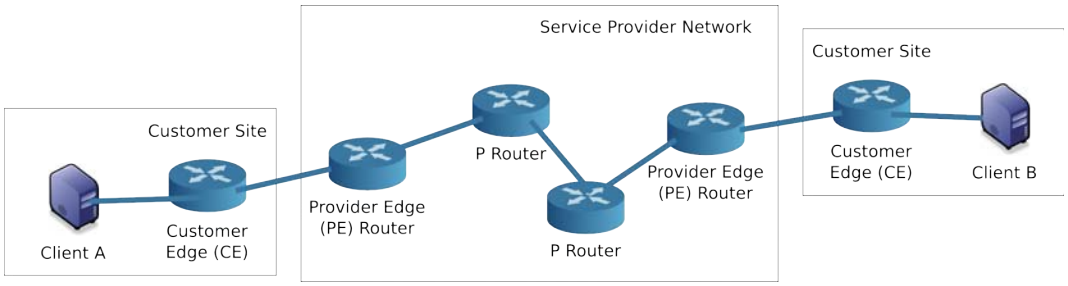


Figure 2.1: Representation of basic VPN components. Customer Edges (CE) are located at Customer Site. Provider Edge routers (PE) and P routers are part of Service Provider (SP) network.

VPNs can be established at different levels of the OSI model, exploring multiple protocols and services. We detail these levels below.

Layer 1: The virtualization of Layer 1 network elements, such as Optical Cross-connect (OXC), or Time-Division Multiplexing (TDM) switches, enable the composition and realization of L1VPNs [Takeda, 2007]. By definition, a Layer 1 VPN uses GMPLS RSVP-TE for signaling both within the service provider network (between PEs), and between the costumer and the SP (between CE and PE) [Fedyk et al., 2008].

Generalized Multiprotocol Label Switching (GMPLS) [Farrel and Bryskin, 2005] is an extension of the MPLS architecture [Rosen et al., 2001]. MPLS has been defined to support label-based forwarding of data. In the original architecture, the Label Switching Routers (LSRs) were assumed to have a forwarding plane capable of recognizing packet or cell boundaries. The GMPLS extension includes Label Switching Routers (LSRs) that do not forward data based on packet or cell boundaries. These LSRs include devices where the

forwarding decision is based on time slots, wavelengths, or physical ports [Berger, 2003]. This extension, combined with the concept of nested Label Switched Paths (LSPs), allows the system to scale by building a forwarding hierarchy. According to their topologies, Layer 1 VPNs can be classified in Virtual Private Wire Services (VPWS) which are point-to-point, and Virtual Private Line Services (VPLS) which are point-to-multipoint [Benhaddou and Alanqar, 2007].

Layer 2: Layer-2 VPNs are usually provided over Ethernet, ATM, and Frame-Relay technologies. There are two fundamental types of Layer 2 VPN services that a SP could offer to a customer [Andersson and Rosen, 2006]: Virtual Private Wire Service (VPWS) and Virtual Private LAN Service (VPLS). A VPWS is a VPN service that supplies an L2 point-to-point service, and a VPLS is an L2 service that emulates LAN service across a Wide Area Network (WAN) [Augustyn and Serbest, 2006].

The Layer 2 VPN protocols encapsulate the Layer 3 packets into point-to-point frames. The advantage of Layer 2 VPNs is that the higher level protocols remain with their normal operation, more specifically, the virtualization is transparent to Layer 3 protocols. Point-to-point Tunneling Protocol (PPTP) [Hamzeh et al., 1999], Layer 2 Tunneling Protocol (L2TP) [Townsley et al., 1999], and Layer 2 Forwarding (L2F) [Valencia et al., 1998] are examples of L2VPN protocols.

Layer 3: A Layer-3 VPN is characterized by the use of IP and/or MPLS to virtualize a shared network infrastructure (VPN backbone) [Carugi and McDysan, 2005]. The SP edge devices determine how to route VPN traffic by looking at the IP and/or MPLS headers of the packets they receive from the customer's edge devices [Callon and Suzuki, 2005].

Layer-3 VPNs are implemented as two basic types [Chowdhury and Boutaba, 2009]: CE- and PE-based. In a CE-based VPN, the CE devices create and manage the tunnels, without interference of network provider. The CE devices encapsulate the packets and route them through carrier networks. In this scenario, three protocols are identified to create the tunnel: a carrier protocol responsible for carrying the VPN packets (such as IP); an encapsulating protocol, such as PPTP [Hamzeh et al., 1999], L2TP [Townsley et al., 1999], and IPSec [Kent and Atkinson, 1998] [Kent and Seo, 2005]; and the original CE passenger protocol. Already in a PE-based approach, the network provider is responsible for the establishment and management of the existing VPNs.

2.2.2.3 Overlay networks

Overlay networks are topologies constructed over existing networks. In this virtualization layer, virtual links are logically established over physical links or paths to interconnect the nodes that are part of the overlay.

Nowadays, this technology is used as an alternative to overcome diverse issues in different contexts, such as Internet routing resilience [Andersen et al., 2001], enable multicasting [Eriksson, 1994] [Jannotti et al., 2000], provide QoS guarantees on communications [Subramanian et al., 2004], and content distribution [Krishnamurthy et al., 2001]. Peer-to-Peer networks, often built at level 4 of the OSI model, are examples of overlay networks [Androutsellis-Theotokis and Spinellis, 2004], where the content is typically exchanged atop of an IP network, as explored by Akamai solutions [AKA].

2.2.2.4 Programmable networks

This type of network can be programmed via a minimal set of Application Programming Interfaces (APIs) with which one can ideally compose an infinite spectrum of higher-level services [Campbell et al., 1999].

This technology supports the construction of networks enabling dynamic deployment of new services and protocols. As identified by [Chowdhury and Boutaba, 2009], two schools of thought emerged for the implementation method:

- Open Signaling: proposes a clear distinction between transport, control, and management planes. It defines an abstraction layer for physical network devices with a well-defined interface. For example, OpenFlow switches explore a bottom-up approach to dynamically configure networks [McKeown et al., 2008];

- Active Network: explores the dynamic configuration based on routers and switches information collected by analyzing packet exchange, which allows a real-time adaptation of the network according to the traffic [Tennenhouse and Wetherall, 2002].

2.2.3 Discussion

Virtualization is a driving force for flexibility in the next-generation Internet, guaranteeing its growth and success, and preventing its ossification [Handley, 2006]. It can enable ubiquitous deployment of applications and foster innovations by adding a layer of abstraction between the actual hardware and running and exposed resources [Anderson et al., 2005] [Niebert et al., 2008] [Keller and Rexford, 2010].

Although the isolation mechanisms of VMMs have an impact on the performance of virtualized systems [Wang and Ng, 2010] [Schlosser et al., 2011] and some issues remain opened [Gupta et al., 2006], virtualization enables an efficient usage and sharing of physical resources. By being *de-materialized*, virtual resources can be deployed on demand, configured, started, paused, saved, or deleted, like a set of programmable objects. This gives to IT and network providers the possibility to set up and manage virtual resources, allowing the dynamic provisioning of new services [Carapinha and Jiménez, 2009].

Virtualization has over the years been applied to specific scenarios, such as server consolidation and VLAN composition. The timeline depicted in Figure 2.2 shows the progress of the main technologies, projects and software for virtualization of network and IT resources. Following the timeline, one can observe that virtualization technologies have increasingly shifted from solutions for specific endpoints of the Internet (e.g., servers, local networks) to more distributed resources (e.g., VPNs). Currently, different virtualization techniques can be combined for providing more elaborated services. For example, two providers with heterogeneous solutions for network virtualization (i.e., one exposing optical Layer 1 virtualization, and another a Layer 2 VPN service) can cooperate and interconnect to serve the needs of a customer who is unaware of the difference of underlaying technology. In addition, an external network operator can orchestrate resources by allocating them and composing virtual networks using services exposed by different providers [GEYSERS, 2011].

One of the main contributions introduced by virtualization techniques is that resources can be dynamically reconfigured. IT resources can have their memory, storage, and CPU

configurations redefined during the execution process, whereas a virtual link can be controlled in terms of bandwidth capacity. These dynamic configuration and provisioning features have been explored by recent technologies, as detailed in the following sections.

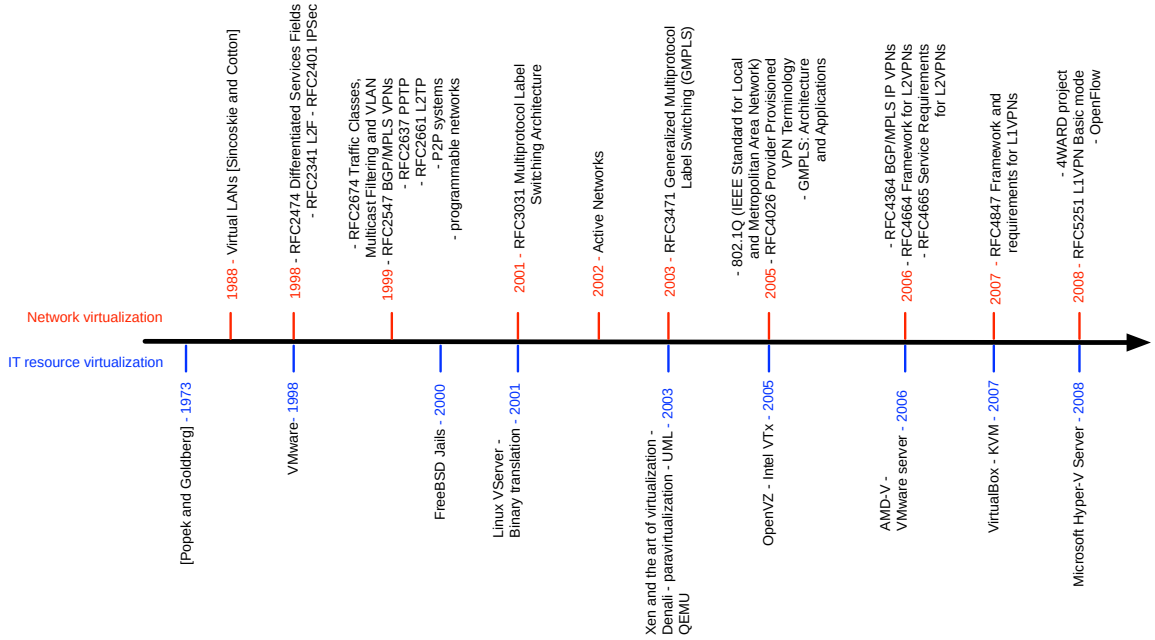


Figure 2.2: A network and IT resources timeline comprising the main technologies, projects, and software explored for virtualization.

2.3 Cloud Computing

The maturity of resource virtualization and Grid Computing technologies [Kesselman and Foster, 1998] have led to the emergence of Cloud Computing. Concepts such as virtual organizations and services exposed via transparent interfaces have initially been explored in Grids [Foster and Kesselman, 1997] [Foster, 2001] [Foster, 2005]. Cloud Computing combines these concepts with new virtualization techniques and business models to enable on-demand access to a shared pool of configurable resources (e.g., servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort [Mell and Grance, 2009] [Buyya, 2009] [Rimal et al., 2011].

In this way, Cloud Computing provides a new means for delivering IT services, based on cost-efficient, scalable and on-demand provisioning guided by the user's requirements. Exploiting the flexibility offered by virtualization, IT elements are exposed as a set of services (or even resources) that can be reserved by users for a specific timeframe. Currently, single users, companies, and institutions deploy dynamic computing resources on Clouds to execute their applications, exploring the advantages offered by the different levels of virtualization, including cost reduction [2] and reliability support [7].

The commercial model explored in Clouds differs from traditional IT commercialization by demanding smaller up-front investment. Traditionally, a user buys a software or

hardware, and then uses it respecting the conditions imposed by contractual agreements (e.g., number of licenses, or number of allowed executions). Cloud Computing proposes the on-demand availability of different virtualized services that can be rented, accessed, customized, and exploited by users in a pay-as-you-go manner.

With the Cloud approach, both users and providers can explore advantages such as:

- users are not locked into a single IT solution or provider;
- software and hardware licenses' costs are diluted and distributed between providers and users;
- providers can operate close to the maximum computing capacity and, at same time, maximize revenues;
- IT administration costs are also proportionally shared by users and providers.

2.3.1 Types of Clouds

Clouds are classified according to the servers' location and the users who access them. The location distinguishes private resources, belonging to a single organization, and public resources, located over a distributed infrastructure, while the access identifies the users and how they access the resources (e.g., reserving, renting, combining). A commonly used classification identifies mainly three types of Clouds: Private Clouds, Public Clouds, and Hybrid Clouds [Zhang et al., 2010]. Figure 2.3 shows an example of a scenario composed of a Private Cloud, two Public Clouds (A and B), and two hybrid Clouds (A and B). The following sections detail these types of Clouds.

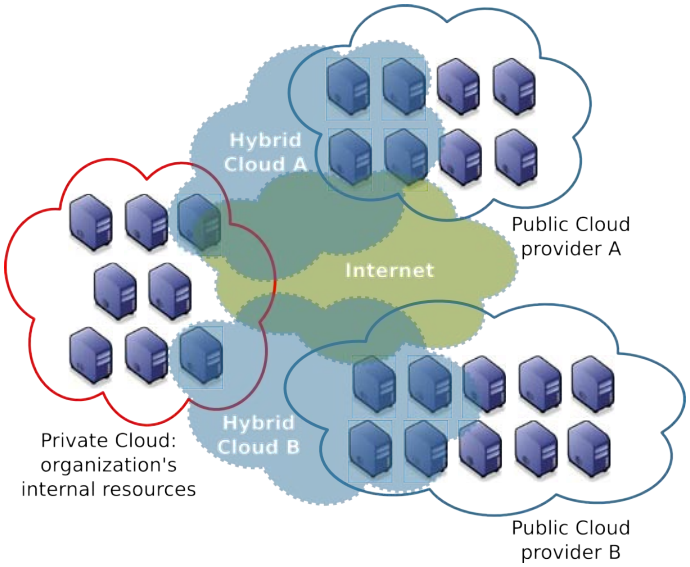


Figure 2.3: A common classification defining three types of Clouds: Private Clouds, Public Clouds, and Hybrid Clouds.

2.3.1.1 Private Clouds

Private Clouds (also called Internal Clouds) are built using the internal resources of organizations. The exclusive use of computing power is usually restricted to members of the organization to guarantee the confidentially of internal and critical data. Administrators of private computing infrastructures have the highest level of control over performance, reliability, and security of resources. Both commercial and open-source frameworks and tools, such as VMware vCloud Director [vCl], OpenNebula [Opea], Xen Cloud Platform (XCP) [XCP], Eucalyptus Systems [Euc], and LYaTiss Weaver [LYaTiss, 2011], can be used to compose and manage Private Clouds.

2.3.1.2 Public Clouds

Public Clouds (or External Clouds) make their resources available to users as services. The set of virtualized computing resources is often geographically distributed across several data centers. In comparison to Private Clouds, Public Clouds offer less control and security guarantees over critical data, and require the establishment of a communication channel to access virtual machines.

Examples of Public Cloud providers comprise Amazon EC2 [AMAa], Microsoft Windows Azure platform [AZUa], Salesforce.com [SAL], 3Tera [TER], Google App Engine [GOOa], NetSuite [Net], among others. As of writing, Cloud providers do not offer the provisioning of virtual links to interconnect virtual machines. Instead, the communication is performed following a best-effort approach.

2.3.1.3 Hybrid Clouds

At times, an organization can feel compelled to reserve public on-demand resources and interconnect them with its Private Cloud. It may be the case under peak-load conditions where the organization's private resources may not suffice the application workload. A set of public services are then provisioned on-demand, hence combining private and public resources for a given timeframe.

Figure 2.3 details the composition of Hybrid Clouds, where a subset of resources from Public Cloud A is combined with internal resources of the Private Cloud, creating the Hybrid Cloud A. Resources are interconnected by a communication channel established over the Internet. The Hybrid Cloud B is similarly composed using the on-demand resources of Public Cloud B.

Combining public and private resources enables a finer control on sensitive data location and security than using only Public Clouds. An organization can keep the sensitive data on its Private Cloud, exploring the public resources to execute less critical tasks.

2.3.2 Service models

Cloud Computing can dynamically provision virtualized services at different levels. A set of virtualized servers and IT infrastructure can provide *Software, Platform,* and *Infrastructure as a Service* [Mell and Grance, 2009] as shown by the layers in Figure 2.4. A set of distributed and virtualized servers composing the physical substrate is shown at the bottom, while the other layers represent different models for exposing services, which are detailed below.

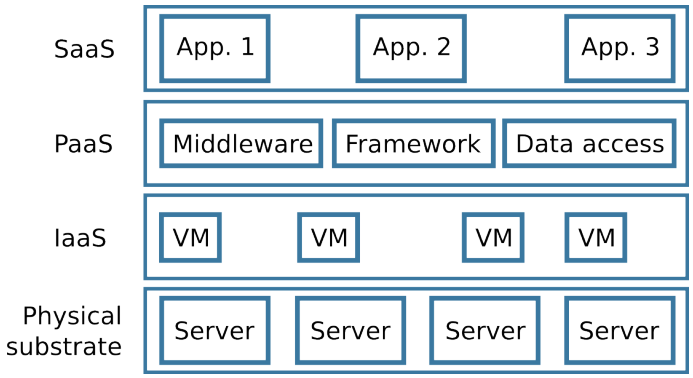


Figure 2.4: The service models offered by Cloud Computing providers in 2011.

2.3.2.1 Software as a Service (SaaS):

SaaS, the higher level of abstraction, offers on-demand execution of applications allocated on remote servers. In contrast to the one-time licensing model commonly used in traditional software, *SaaS* is frequently sold using a subscription model, where customers pay an ongoing fee to use the application [Jacobs, 2005]. Examples of *SaaS* providers include Microsoft Online [BPO] Salesforce.com [SAL], Rackspace [RACa], SAP Business ByDesign [SAP], Google Apps [GOOb] and NetSuite [Net].

2.3.2.2 Platform as a Service (PaaS):

The *PaaS* model provides features to users for deploying a complete application lifecycle (for example, tools to design, develop, test, host, and integrate software) without the need for special software installation and configuration. *PaaS*-layer resources, such as software development frameworks, are exposed as on-demand services.

In addition, tools to manage and control scalability, security and persistency are provided as optional service components. Google App Engine [GOOa], Microsoft Windows Azure [AZUa], and Force.com [FOR] are examples of *PaaS* providers.

2.3.2.3 Infrastructure as a Service (IaaS):

This model offers Virtual Machines (VMs) as services. Instead of buying new servers to run his application, the client can lease a set of virtual machines. Virtual machines are dynamically provisioned and interconnected through a best-effort approach, generally without any quality of service guarantees to the network communications. Examples of *IaaS* providers include Amazon EC2 [AMAa], GoGrid [GOG], Rackspace [RACa], and Flexiscale [FLE].

2.3.3 Cloud Networking

Although Cloud Computing has consolidated the on-demand provisioning of virtual machines, an efficient network interconnection of these elements has not been addressed. Cloud providers instantiate a set of virtual machines and interconnect them using best-effort approaches in which the network is generally not controlled and the usage of communication resources is charged according to the total volume of transferred data.

The best-effort approach is not enough for network-sensitive applications. A bottleneck or increased latency between distributed modules can compromise the entire application

execution [5]. To overcome this limitation, Cloud Networking enables networking resources to be controlled, deployed and provisioned in a much more dynamic, flexible and scalable way [SAI]. This technology proposes the dynamic provisioning of connectivity services following the same delivery model explored by Cloud Computing: on-demand and billed according to usage.

Figure 2.5 exemplifies the dynamic provisioning of a connectivity service between a Private Cloud and a Public Cloud provider. A virtual link is reserved and dynamically provisioned to interconnect both Clouds. The provisioned capacity can vary during the reservation time: a peak on data to be computed or to be transferred requires more capacity then the nominal rate originally provided. During the peak, more resources are dynamically provisioned by the Cloud Networking provider.

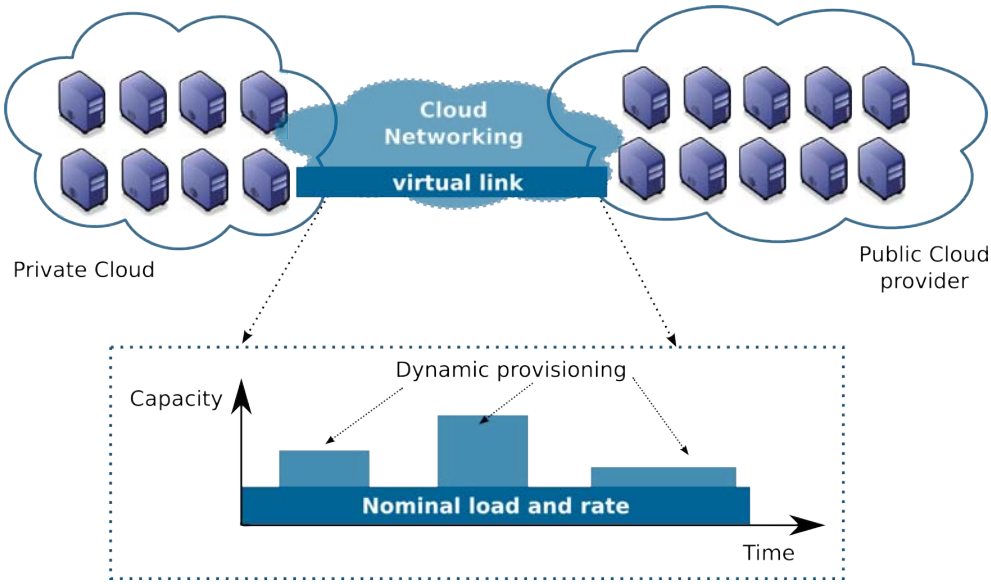


Figure 2.5: Example of Cloud Networking provisioning between a Private Cloud and a Public Cloud (example discussed in the context of SAIL project [SAI]).

2.3.4 Discussion

Cloud Computing consolidates the on-demand provisioning of configurable virtual machines to perform computations [AMAa] [ENO] and provide large data storage [TER]. Cloud users have been outsourcing their computing and storage infrastructure to Public Cloud providers who provision services in a *pay-as-you-go* service provisioning. In addition, the concept of Cloud Networking aims at dynamically provision virtual interconnections to enable efficient and secure communication between virtual entities.

Many research and engineering issues are still open in Clouds. For example, Public Cloud providers differ in the ways that their infrastructure is composed and the form their features and services are exposed (e.g., virtualization techniques, internal network topology). A user can thus expect his application to have a different behavior when executing it over different Cloud providers [Li et al., 2010].

Usually, a provider has an API or a set of web services that enable the interaction with its reservation and provisioning systems, such as Amazon EC2 API [AMAb], Win-

dows Azure API [AZUb], and Rackspace API [RACb]. These APIs can be exposed using SOAP [Gudgin et al., 2007], HTTP [Fielding et al., 1999], or RESTful [Gregorio and de Hora, 2007] interfaces. They do not follow a well-defined standard, which hence induces issues in communication and information exchange among these systems, management frameworks, and users.

Moreover, the security of Clouds is a major concern, especially when sensitive information must be outsourced to distributed public resources [Barrios Hernandez et al., 2010]. In addition to traditional security aspects that should be provided (e.g., authentication, authorization, integrity, and confidentially), Cloud Computing introduces the possibility of the misuse of resources. A large amount of computing power can be allocated and used for illegal services (e.g., DoS attack, spamming) [Schoo et al., 2010].

In spite of the issues above, it is incontestable that Cloud Computing has changed the usage of the Internet. This technology has been adopted by many enterprises, each following a specific business model. Predictions for the next years indicate that more than 80% of the IT will be outsourced by the end of 2020, and Clouds will be cheaper, more reliable, more secure, and easier to use. In addition, the Cloud providers' cost will be less than 25% of that of corporate data centers [Rosenberg and Mateos, 2010].

2.4 Virtual Infrastructures

Virtual Infrastructures (VIs) extend the original IaaS paradigm to provide dynamic virtual networks of computing (also virtualized) resources [SAI] [GEN] [GEYSERS, 2011]. With the VI extension, the Internet's computational, storage and network resources can be abstracted from physical infrastructure and exposed as services that are dynamically reserved and provisioned [Laganier and Vicat-Blanc Primet, 2005] [Rochwerger et al., 2009].

Figure 2.6 illustrates this new scenario: two virtual infrastructures (A and B) are allocated on a distributed and virtualized physical substrate, sharing its IT and networking capacities. Both VIs are completely isolated and have no knowledge of sharing physical resources.

2.4.1 Concept

We defined a *Virtual Infrastructure (VI)* as a time-limited group of virtual computing resources interconnected by a virtual network [1]. By combining resource and network virtualization, the user of a VI has the illusion that he is using a private distributed system, while in reality he is using multiple systems that are part of a virtualized physical substrate. VI instances are kept isolated from one another.

VIs can span multiple networks belonging to different administrative domains. Independently of the underlying physical topology and the applied virtualization techniques, a VI user has a consistent view of a regular network stack. He can join the VI from any location, deploying and using the same application used on the Internet or his intranet.

A VI can be formally represented as a graph whose vertices are in charge of active data-processing functions and edges of moving data between vertices. Figure 2.7 illustrates this concept representing a virtual infrastructure composed of virtual machines interconnected through virtual links. It shows two virtual routers (vertices r_vA and r_vB) which are used to interconnect and perform the bandwidth control among the other virtual resources (vertices r_v1 to r_v8). The virtual routers can independently forward the traffic of the different

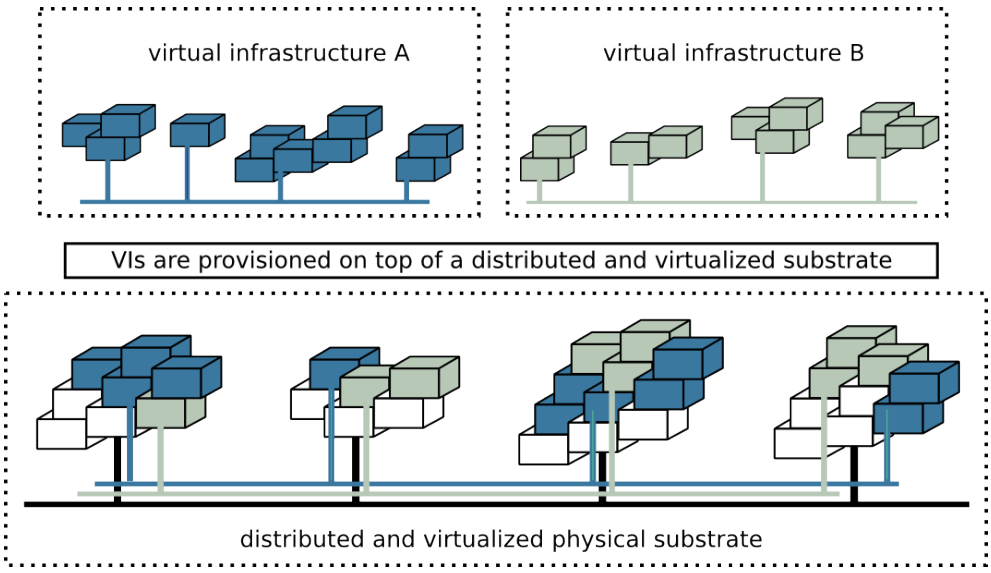


Figure 2.6: Two VIs are sharing a distributed and virtualized physical substrate.

virtual infrastructures which share the same physical network. Each edge represents a virtual link (as l_v1 and l_v2) with different configurations used to interconnect a pair of virtual resources.

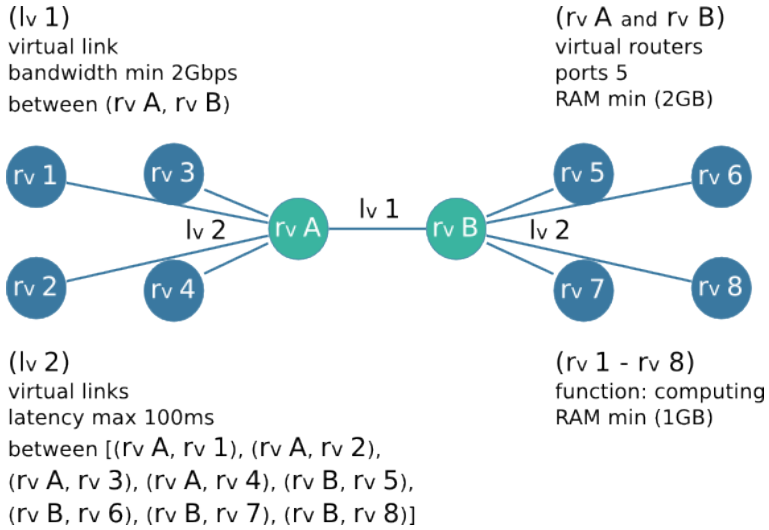


Figure 2.7: Example of a VI composition using a graph notation. The virtual IT resources are the vertices and the virtual links the edges.

Moreover, a VI can be defined and modeled according to user requirements [6]. Different information can be defined for the graph components, individually or aggregated:

- Temporal attributes for each resource (time window for provisioning).
- Parameters to represent the necessary configuration of IT resources (e.g., RAM,

CPU speed, and storage capability) and to define the link requirements (bandwidth and latency). This information can vary during the VI lifetime.

- The required software (e.g., operating systems) and tools (e.g., communication and programming tools) that must be provisioned with the VI.
- The resource's functionality (e.g., a set of *computing* nodes or *storage* nodes).
- Security attributes for each component (e.g., access control, confidentiality level).
- Reliability level required for each VI component.
- Commercial attributes (maximum cost).

2.4.2 Building blocks

As defined by LYaTiss Weaver [LYaTiss, 2011], a VI is composed of the following components or building blocks (edges and vertices) [1] [16]: vNodes, vStorage, vRouters, vAccessPoints, and vLinks. They can be organized individually or in groups. For example, a vertex of a VI graph can be composed of a group of vNodes, vRouters, and vLinks.

vNode: is a fully customizable virtual machine that is provisioned (e.g., assigned CPU, RAM, and storage resources) and configured (e.g., set the operating system and communication tools) atop virtualized physical hosts according to the user's requirements.

vRouter: the virtual routers are special components that virtualize the data, control, and management network planes. Users can deploy customized routing protocols and configure the packet-queuing disciplines, packet filtering and monitoring mechanisms they want. In addition, as they concentrate aggregated VI traffic, vRouters represent strategic points of the network for transfer rate control. By limiting the rate and policing the traffic of different vRouters, the traffic of VIs can be controlled so that the user is provided with a fully isolated VI. The advantage of having controlled environments is twofold: users have strong guarantees, and the network provider can better exploit the network by sharing it efficiently among users.

vStorage: the location, mobility, and security aspects of data are critical factors for many applications and users. vStorages are special virtual machines that must be allocated and provisioned on top of specialized physical hosts, which implement Storage Area Network (SAN), Direct-Attached Storage (DAS), or Network-Attached Storage (NAS) technologies. In this case, physical hosts must provide enough capacity to fulfill the application requirements in terms of data volume, encryption and security.

vAccessPoint: virtual access points are the entry points of VIs. They allow the interconnection among external components and VI resources, and between different virtual infrastructures. In addition, vAccessPoints can offer functionalities such as Network Address Translation (NAT), masquerade network, load balancing, firewall and data encrypting.

vLink: the components of VIs (which can include vNodes, vRouters, vAccessPoints, vStorage and groups of these) are interconnected using vLinks (virtual links). Each vLink is a temporary path allocated over multiple physical channels for which metrics such as bandwidth (forward and reverse) and latency can be defined. In addition, by asking for vLinks it is possible to compose and model the VI, identifying for example the critical communication paths.

2.4.3 Discussion

Virtual Infrastructures have emerged as the blend of different concepts and techniques, such as resource and network virtualization, Grid Computing, and resources as services. Within this new concept, users can reserve a set of interconnected IT resources and configure them according to their application requirements. VIs are provisioned on top of a distributed and virtualized physical substrate, sharing the resources in a transparent and isolated way. During a given time slot, all VI resources are made available to, configured and managed by the user.

2.5 Provisioning dynamic Virtual Infrastructures

The provisioning of a virtual infrastructure is a complex task as each of its components can vary its configuration in three dimensions: capacity, time, and cost. Following the pay-as-you-go model, cost represents the InP perspective, who is charging for the resource provisioning according to time and capacity requirements.

Figure 2.8 shows the three dimensions that should be considered during a virtual resource provisioning. The virtual resource r_v1 can vary its capacity per time resulting in a specific usage cost. Four volumes of capacity per time have their own provisioning costs. Moreover, each virtual resource composing a VI can similarly vary its configuration.

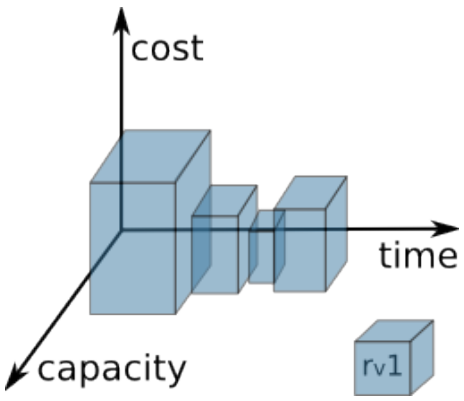


Figure 2.8: Dimensions that should be considered during a VI provisioning.

The following sections review the requirements for efficient VI provisioning, and present existing solutions for this purpose, detailing specifically a framework used as base for developing the contributions of this thesis.

2.5.1 Requirements for efficient VI provisioning

VIs have a lifecycle composed of specification, allocation, deployment, and release [16]. The three dimensions (time, cost, and capacity) are present and should be treated in all steps. For provisioning efficient VIs, a management framework should respect and integrate the following requirements:

- Enabling the specification of dynamic resources. The requirements of virtual IT and network resources can vary their capacity during the reservation time. In addition, a virtual resource can be present only partially in the whole VI reservation time;
- The internal mechanism for embedding VIs should also respect this elastic aspect, as well as the user's and InP's expectations. The allocation should be performed efficiently for both, optimizing the trade-off between their goals; and
- Deploying the VIs respecting the user's requirements. Tools for guaranteeing the Quality-of-Service (QoS) should control the provisioned capacity, assuring isolation and performance for each individual component.

2.5.2 Existing solutions for provisioning VIs

Several projects, prototypes, and commercial solutions have been proposed to provide virtual resources, virtual networks, and VIs within different contexts, such as in Grids and Clouds. We discuss these solutions here by organizing them in IT resource provisioning, and network provisioning. They are analyzed respecting the requirements identified in Section 2.5.1.

2.5.2.1 IT virtualization

Grids have explored decoupling services from physical infrastructure with virtualization techniques. In [Rezmerita et al., 2006], the authors proposed PVC, combining various Grid, P2P and VPN approaches to create instant Grids. This distributed system is composed of a daemon process (peer) and a brokering service, which helps establish the connections. Distributed peers are aggregated composing a virtual Grid.

Existing commercial products, such as Amazon's Elastic Compute Cloud (EC2) [AMAa], Enomaly's Elastic Computing Platform (ECP) [ENO] and GOGRID [GOG], allow users to reserve a set of resources, choose an operating system, and customize them. Other products are less configurable, like 3Tera's AppLogic [TER] where the operating system cannot be chosen. Some frameworks for Cloud orchestration, such as OpenNebula [Opea] and Eucalyptus [Euc], provide tools for IaaS provisioning. All these Cloud solutions are suitable for performing computation and storage and are not aim to host virtual routers. Moreover, these Cloud Computing solutions do not intend to control the network or its communication performance.

2.5.2.2 Network virtualization

Some proposals have explored virtualization for enabling end-to-end control while using physical equipments from different physical infrastructure providers [Feamster et al., 2007]. In [Bavier et al., 2006], the authors propose VINI, a virtual network infrastructure that allows several virtual networks to share a single physical infrastructure. Researchers can run experiments in virtual-network slices running inside User Mode Linux (UML) instances.

They can configure the slices and choose a protocol, among those available, to communicate internally. This also provides full-isolation between virtual nodes and controlled resource sharing.

Trellis [Bhatia et al., 2008], a network-hosting platform deployed on the VINI facility, is a virtual-network substrate that can run on commodity hardware. It is implemented using VServer's [VSE] container-based virtualization because of its network performance. VINI and Trellis were evaluated on PlanetLab [Bavier et al., 2004] which provides users with slices of virtual resources distributed across its overlay infrastructure. The virtual resources are VServers generally providing end-host functionalities. In the GENI design [GEN], the users are provided with slices, like in VINI, but composed of virtual machines or partitions of physical resources. GENI's goal is to provide users with multiple shareable types of resources.

ShadowNet [Xu Chen, 2009] proposes an architecture for an operational trial/test network consisting of ShadowNet nodes interconnected by a single backbone. A ShadowNet node is composed of a collection of carrier-grade equipments, namely routers, switches and servers. Each node is connected to the Internet as well as to other ShadowNet nodes via a (virtual) backbone. VIs are developed on top of ShadowNet nodes mainly for testing purposes.

In addition to these initiatives, some proposals have specifically treated the bandwidth sharing in virtual networks. A design and mathematical model of dynamic adaptation of virtual networks to their performance objectives has been proposed by DaVinci [He et al., 2008]. In addition, isolation between virtual networks is provided by slicing mechanisms, controlling the attribution of bandwidth to virtual links [Kim et al., 2010] [Sherwood et al., 2010].

Recent projects have addressed the VI provisioning [SAI] [GEY]. Geysers project [GEYSERS, 2011] is investigating solutions for implementing virtual networks atop distributed optical providers, while SAIL project [SAI] is investigating the provisioning of virtual networks to interconnect distributed Cloud data centers, proposing the dynamic creation of Flash Network Slice.

2.5.3 HIPerNet framework

We present in this section the HIPerNet, the first framework to provide and manage dynamic virtual infrastructures proposed in the context of HIPCAL [HIP] and CARRIOCAS projects [Audouin et al., 2009] [Vicat-Blanc Primet et al., 2009]. This framework was chosen to serve as the base for the development and implementation of the contributions of this thesis. The concepts and technologies addressed by HIPerNet, which justifies this choice, are discussed here.

2.5.3.1 Concepts and design principles

HIPerNet is a framework for dynamic provisioning of networking and computing infrastructures. Conceptually, it innovates by combining system and network virtualization technologies with bandwidth sharing and advance reservation mechanisms [Laganier and Vicat-Blanc Primet, 2005] [1]. By definition, HIPerNet is transparent to all types of upper layers, such as: protocols (e.g., TCP, UDP), APIs (e.g., sockets), middleware (e.g., Globus [Foster and Kesselman, 1997]), applications, services and users. It helps end users with intensive-application deployment, execution and debugging, by providing an envi-

ronment to run them on scalable, controlled, distributed and high-capacity platforms. In addition, it is mostly agnostic of lower layers and can be deployed over IP as well as lambda networks.

The HIPerNet framework is responsible for the creation, management and control of VIs supervising and monitoring their status and behavior during their whole lifetime. The key principle is to have operations realized automatically. For this, HIPerNet defines and activates mechanisms to automatically control the VIs as well as the status of the underlying exposed and virtualized physical substrate.

Figure 2.9 shows the main components of HIPerNet. At the lower level, HIPerNet accesses and controls a part of the physical infrastructure that is virtualized and exposed. The physical resources are registered on the HIPerNet database and can be allocated to VIs. Once the resources have been exposed, HIPerNet gets full control over them. This set of exposed virtualized resources composes the substrate hosting the VIs. At the up layer, HIPerNet receives VI requests using a special language: VXDL (a contribution of this thesis whose is discussed in Chapter 3).

The internal mechanism of HIPerNet allocates, deploys, and configures VIs following the user's requirements. All modules of HIPerNet can be customized. For example, the allocation or deployment mechanisms can be replaced as long as they respect the defined API. This has been an important feature for devising the allocation mechanism detailed in Chapter 5, and the reliability service is discussed in Chapter 6. At run-time, the HIPerNet manager communicates with the virtual and physical resources to monitor their status and configure control tools to supervise the resource usage [HIPerNet].

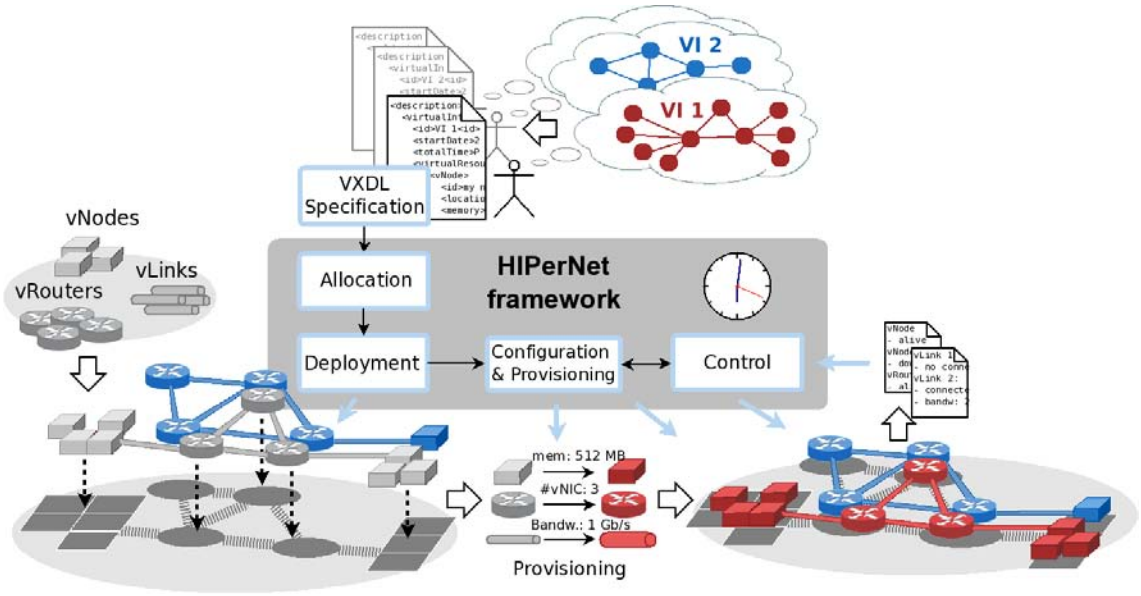


Figure 2.9: Dynamic provisioning of VIs with the HIPerNet framework.

Provisioning of IT resources: HIPerNet provides virtual IT resources based on Xen VMM [Barham et al., 2003]. The virtualized physical substrate is composed of a set of physical machines running a special administrative image, which is basically a Xen *dom0*

with a set of private management tools. To spontaneously join the virtualized physical substrate, each machine must deploy the HIPerNet image and declare its existence, configuration, and available capacity to HIPerNet, becoming a potential host for virtual IT resources. This information is automatically declared during the bootstrap.

Virtual machines are composed according to the user's specification and instantiated on physical hosts. Considering the filesystem deployment, HIPerNet follows two approaches: an image can be copied directly to physical hosts; or placed on a network file system. The operation mode is selected by the user during the instantiation process. Some services, such as live migration [Clark et al., 2005] and reliability support [7], require the provisioning using the network file system to guarantee the mobility of virtual machines.

Dynamic network provisioning: HIPerNet has explored tools and technologies for virtualizing networks at layers 1 to 3, enabling multiple virtual overlay networks to cohabit on a shared communication infrastructure. All network overlays and virtual resources are kept isolated at the network level to ensure security, performance control and adaptability.

Within HIPerNet, vLinks implemented at layer 3 are based on IPsec [Kent and Seo, 2005] and the HIP protocol [Moskowitz and Nikander, 2006]. In addition, at this layer, the network provisioning and traffic control are based on vRouters. The initial prototyping of a vRouter consists in a software router implemented inside the Xen virtual machines [8]. vRouters perform an efficient channel provisioning by combining routing and traffic engineering in order to satisfy the quality of service requirements of each VI [Divakaran and Vicat-Blanc Primet, 2007].

The implementation and provisioning of virtual links at lower layers (L1 and L2) was addressed within the CARRIOCAS projects [Audouin et al., 2009] [Vicat-Blanc Primet et al., 2009]. A prototype named Scheduling, Reconfiguration, and Virtualization (SRV) was developed for interconnecting IT resources with ultra-high speed optical networks [Verchere et al., 2008].

Security model: The security model is based on a combination of Simple Public Key Infrastructure (SPKI) and HIP protocol [9]. The cryptographic identification scheme is the core of the SPKI [Ellison et al., 1999], which defines a flexible private authorization management layer for the VI resources. A certificate is signed by the sender using its private key. When receiving the certificate, the receiver verifies the signature using the sender's public key available in the certificate, in order to check the message integrity and authenticate the sender. Unlike the traditional PKI model, there is no need for a high level naming service to guarantee the certificate authenticity. The SPKI certificate itself carries a signed authorization associated to the resource identified by its public key.

In addition, the HIP protocol automatically associates public keys to entities [Moskowitz and Nikander, 2006]. Traditionally, the IP address plays two independent roles: a *locator* and an *identifier*. Network layer protocols (e.g., IPv4, IPv6) use the locator role to route packets, while the upper layer protocols (e.g., TCP, UDP) use the identifier role. As upper layer protocols depend on location, they fail when mobility causes a modification of the IP address. HIP decouples these two roles while maintaining a binding between an identifier and a set of associated locators. The identifier namespace defined by HIP comprises the public key of the host and is called a Host Identity (HI). HIP also defines the Host Identity Tag (HIT), a 128 bit long truncated hash of the HI, used by applications for communicate.

2.5.4 Discussion

The new challenges posed by VIs, particularly its multidimensional aspect, requires the development of specific tools for efficient provisioning and management. As VIs are comprehensive entities gathering substantial information, solutions for provisioning IT and network resources should be combined for efficiently manage all their dynamic aspects.

One of the first framework that proposed the combined provisioning of IT and network resources was the HIPerNet. It has demonstrated to be a flexible solution for the dynamic provisioning of virtual infrastructures. *The modules composing the HIPerNet can be replaced, enabling its adaptability to different developers.*

HIPerNet was used as the base framework to deploy and validate the contributions that are presented in the following chapters. Several patents have been produced [VXCap] [VXAlloc], and currently an industrialized version of the HIPerNet framework has been developed and maintained by the LYaTiss company (for more information please refer to `http://www.lyatiss.com`).

2.6 Summary

Virtualization is the main technology that drives the research to Future Internet. Distributed data centers have adopted virtualization as an efficient technology to consolidate IT platforms, improve resource usage, and decrease costs (e.g., with IT administration, power consumption, and cooling) [Festor et al., 2010].

A new business model has been explored for outsourcing IT infrastructures [Rosenberg and Mateos, 2010]. Instead of buying new resources, several enterprises are renting on-demand services to complete their IT infrastructures, following a *pay-as-you-go* model. Clouds have consolidated the on-demand provisioning of virtualized services (e.g., IaaS, PaaS, SaaS). In addition, Cloud Networking has proposed a new wave of service provisioning dedicated to dynamic interconnection of virtual entities. A dynamic virtual link can be established with a specific configuration (e.g., security, bandwidth) to interconnect distributed IT resources (e.g., data centers, Private and Public Clouds).

Recently, the combined provisioning of IT and networking resources has been proposed, composing a malleable *computing-and-communication* entity identified as *Virtual Infrastructure*. VIs can be provisioned on top of multiple distributed physical substrate, sharing their resource capacities. The HIPerNet framework was developed as a proof of concept, to provide dynamic VIs.

Although the VI concept is a motivating scenario, reviewing the existing solutions for dynamic resource provisioning, we observe that research and engineering work is still required to enable its real exploitation. As VIs are a relatively new concept, the development of tools to specify and manipulate these entities (e.g., allocate, provision, and execute) are required to facilitate its adoption by users.

Four of these tools are described in the following chapters: a language for specifying VIs; a mechanism to translate a workflow-based application into a VI specification; an allocation algorithm for mapping VIs on top of distributed and virtualized substrates; and a service for providing transparent reliability for virtual resources. Each chapter presents a literature review specifically related to the subject and problem that it addresses.

Three

A language for describing VIs

The work presented in this chapter was published at the Second International Conference on Networks for Grid Applications (GridNets 2008) [6]. VXDL has been adopted by recent projects (such as HIPCAL [HIP], CARRIOCAS [Audouin et al., 2009], GEYSERS [GEY], and SAIL [SAI]) as one of the languages used to specify virtual infrastructures, and is part of the solutions patented by the LYaTiss company (`http://www.lyatiss.com`) and the Institut National de Recherche en Informatique et en Automatique (INRIA, `http://www.inria.fr/`) [VXCap] [VXAlloc]. In addition, an open forum for discussions and improvements is being launched (`http://www.vxdlforum.org/`).

3.1 Introduction

We have seen that Virtual Infrastructure (VI) is a new concept based on combined and interconnected dynamic provisioning of IT and network resources. VIs are malleable entities requested and customized by users according to their specific requirements.

Applications differ in the hardware and software configurations required to run successfully. Standalone applications usually require VIs with high computing capacity, whereas distributed applications, in addition to computational resources, often demand VIs with large communication capabilities. Distributed applications may transfer large databases across networks for processing while exchanging very small and sensitive messages. A deadline can be specified to enforce efficient use of reserved resources as well as to shorten the task completion time [Agapi et al., 2009]. Low latency communications (for example, Message Passing Interface – MPI [MPI, 1995]) require strong quality-of-service and can benefit from end-to-end bandwidth reservation and isolation services.

The VI concept introduces a complete abstraction of physical resources and their geographic locations, which poses new challenges in specifying and executing high-end applications. Many factors can directly impact application performance, such as the amount of data to be processed, the geographical location, distribution and confidentiality of the data, and the indeterminism of the required computational power. These factors must be translated into virtual infrastructure requirements in terms of computation, storage, and communication capacities.

In general, descriptive languages are a way to specify and detail the resources users need. Due to a VI's complexity, a language for describing its composition must be abstract enough and more adaptive than conventional resource-description languages and models. In addition, it needs to combine the spatial and temporal aspects of virtual infrastructures.

This chapter proposes the Virtual Infrastructure Description Language (VXDL), an XML-based language for specifying and modeling VIs. VXDL allows the identification and parameterization of virtual resources and groups of resources (according to their functionalities), as well as the network topology (based on the link-organization concept). It also introduces the internal virtual infrastructure timeline, which explores the elasticity of VIs, enabling application providers to specify the exact intervals when virtual resources are provisioned.

The rest of this chapter is organized as follows. Section 3.2 discusses the main projects, languages, and standards that have been proposed to describe computing resources and network topologies. It also reviews the motivations and needs for a new language for specifying virtual infrastructures. VXDL is presented in Section 3.3, and Section 3.4 concludes this chapter.

3.2 State of the art

The description of IT resources and network topologies has been the focus of several studies over the past years. Essentially, the proposed languages differ in the context and expressiveness of their grammars. In this section, we review resource and network description languages, discuss languages for describing virtual infrastructures, and review standardization efforts.

3.2.1 IT resource description languages

The list of languages presented here contains those used to describe computing resources in Grids and others for general purpose description. The languages are reviewed in the order they were proposed and published since a chronological analysis helps observe constant improvements in the descriptive process. In early languages, users specified only the number of computers required to execute their applications. As Grid Computing evolved, the level of detail increased.

ClassAd: this language was developed in the context of the Condor project [CON] to allow for job submission and resource reservation in Grids [Raman et al., 1998]. ClassAd is a semi-structured language that does not specify a rigid grammar. By comparing expressions, it aims to perform a mapping between described attributes (substrate) and specified values (request). Attributes can be default values (e.g., integers, reals) or com-

plete requests (queries) with parameters for resource reservation (start and end dates). ClassAd does not provide attributes to describe network topology.

Redline: proposed to improve the resource description performed by ClassAd. It presents three main innovations [Liu and Foster, 2003]: the description and parameterization of groups of resources; the introduction of pre-defined functions, such as *sum(), count()* and *max()* to personalize the definition of resources; and the introduction of predicates including *forall x in set*, which allowed the application of a single rule for a set of resources.

SWORD: is a resource discovery system for shared wide-area platforms [Oppenheimer et al., 2005]. This system supports queries described in a specific XML format (SWORD XML) or using the ClassAd language. The proposed description format, XML SWORD, enables the specification of individual resources, clusters and the union of clusters. Queries specified with SWORD can also inform simple parameters for network configuration, such as the latency and bandwidth allowed on interconnections.

JSDL: the Job Submission Description Language (JSDL) [Anjomshoaa et al., 2005] allows the description of simple computational tasks of non-interactive applications for traditional Grid environments. A job defined in JSDL specifies parameters such as required resources, execution limits (in terms of CPU time or consumed memory), and command line to execute the application.

GLUE: is an information model composed of objects developed to unify the modeling and representation of Grid resources and their relation [Andreozzi et al., 2007]. The concept discussed in GLUE is the usage of *Computing Elements (CE)* that include resource features, policy tasks, and the interaction among CE's. GLUE allows the representation of homogeneous clusters and sets of heterogeneous resources. Each CE can be individually specified and parameterized; its required software and functionality can be explicitly defined.

BPDL: Broker Property Description Language follows the XML standard and proposes a grammar that enables resource description in terms of quality-of-service attributes, required software and middleware, and security policies [Kertesz et al., 2007]. This language was developed to represent a whole IT substrate, and later improved to include attributes for specifying monitoring metrics, fault tolerance requirements and scheduling policies.

3.2.2 Network description languages

The languages to describe network resources and topologies discussed here were developed with distinct contexts and goals, such as VPN representation, generic and abstract network description, network simulation, and telecommunications management network [Pras et al., 1999] [ITU-T, 1995]. In general, these languages enable a potential representation of networks, but without attention to IT resources.

CIM: the Common Information Model (CIM) specification [CIM, 1999] describes a common set of objects and their relationships. The CIM architecture is based on UML and

provides the CIM Query Language (CQL) to select sets of properties from CIM-object instances. CQL, a subset of SQL-92 [SQL, 1992] with extensions specific to CIM, allows queries for a set of resources with certain configurations, but does not have parameters to interact with the allocation system (e.g., informing the basic location of a component).

NDL: the Network Description Language (NDL) [NDL] [der Ham et al., 2007] is a collection of schemas (topology, layer, capability, domain and physical) used to represent a network infrastructure at different levels [Dijkstra et al., 2008]. This language is guided by the general-purpose Resource Description Framework (RDF) [RDF], used for representing information on the Web. RDF (and NDL) explores a graph data model composed of a set of RDF triples (a *subject*, an *object* and a *predicate*). A subject (resource) has a set of describing objects that are interconnected by predicates.

3.2.3 Virtual infrastructure description languages

Other resource description models are found when analyzing projects that are also investigating network virtualization and provisioning of virtual resources.

vgDL: *Virtual Grid Description Language* [Chien et al., 2004] enables users to provide an initial description of the desirable virtual resources (similar to templates). vgDL proposes three types of virtual aggregations: *LooseBag, TightBag* and *Cluster*. The meaning of each type is abstracting the kind of network interconnection that is provisioned.

libvirt: this API [LIB] is a toolkit to interact with different virtualization platforms. It contains an XML format that enables low-level description and configuration of virtual machines and virtual network entities (including bridges, Network Address Translation - NAT, and routed networks).

Rspec model: the *Rspec model* [RSp], developed in the context of the GENI project [GEN], aims to provide interoperability and data exchange between components of a virtualized execution scenario (e.g., users, resource allocator). The proposed language does not define boundaries in the description, which means that each component can be described with a different level of detail. This approach can help the interaction among computational entities, but it introduces complexity in defining the level of description that is enough for each situation.

3.2.4 Standardization efforts

Currently, several working groups are investigating and proposing standard models to enable the interaction among different contexts and technologies. We review the standardization projects related to network and resource description, and Cloud Computing models.

OGF NML-WG: the Open Grid Forum (OGF) Network Markup Language Working Group (NML-WG) [NML] is addressing the problem of inter-domain light-path provisioning by defining a standard schema for network topology description and information exchange at the network-domain level. The schema describes network objects and their

relationships. The basic NML objects and extended objects can be represented in multiple syntaxes, including XML and RDF.

OGF NSI-WG: the OGF Network Service Interface Working Group (NSI-WG) [NSI] is investigating recommendations to define a generic network service interface. More specifically, this working group focuses on defining an interface that is exposed by service providers (including functionalities), and called by different entities, such as users, other network service providers, and middleware.

OGF OCCI-WG: OGF Open Cloud Computing Interface Working Group (OCCI-WG) [OCC] is investigating a solution to interface with Cloud Infrastructures exposed as a service. OCCI defines a modular API and Cloud infrastructure resources (compute, network and storage) are described using a simple key-value based descriptor format.

DMTF OVF: the specification level addressed by the Distributed Management Task Force (DMTF) Open Virtualization Format (DMTF OVF) [DMTF, 2009] is the packaging and distribution of software to be run on virtual machines. OVF enables the description and parameterization of virtual machines in terms of hardware configuration (devices such as hard-disks, network interfaces, and memory) and the information of software (e.g., operating system and specific legacy services) that runs during the execution time. In addition, OVF permits the configuration of the operating system (e.g., host names, IP address, subnets) and application (e.g., DNS name, databases and external services). An extension of OVF, proposed to address the elastic aspect of virtual infrastructures, includes the definition of key performance indicators that are monitored during the reservation time [Galán et al., 2009].

W3C USDL: the Unified Service Description Language Incubator (USDL) Group Charter [USD] is defining a language for describing general parts of technical and business services and allow them to become tradable and consumable. Technical services are those based on WSDL, REST or other specifications, whereas business services are defined as activities offered by a service provider to a consumer to create value for the latter. The business services are more general and comprise manual and technical services. This work group started in late 2010 and as of writing of this chapter the documentation and proposition were still under development.

3.2.5 Discussion and analysis

Users need to define the virtual infrastructures they request according to their applications' requirements. A language dedicated to virtual infrastructure description must be abstract enough and more adaptive than conventional resource description. During the specification process, new challenges coming from virtualization techniques have to complement the management, configuration and execution features of classical infrastructure description languages. For example, the description languages explored in Grids only enable the specification of computing resources and jobs that the users submit. They do not offer attributes to deal with the elastic aspect of VIs' components.

A VI is modeled as a graph composed of vertices (IT resources) and edges (network links). Due to its complexity and large number os components, attributes, and particularities, this *compute and communication* resource graph must be described as a single entity. Standardization groups are identifying the requirements and proposing models to specify resources and applications in the context of Cloud Computing (IT resource only), but the combined specification of IT resources and network topologies are not addressed by these models.

Usually, languages for describing networks do not offer enough attributes for specifying computing resources, which are interconnected by these networks. Although a network description language offers enough attributes to specify the network topology, it becomes necessary to complement it with an comprehensive model for defining IT resources. Moreover, a language dedicated to VI description needs to combine the space and temporal aspects of these entities.

Table 3.1 summarizes the languages reviewed in this section. The column values were selected considering the definition of a virtual infrastructure, its components and properties. The components, identified in Section 2.4 in Chapter 2, consist of: vNodes, vAccessPoints, vRouters, vStorage, and vLinks. The selected properties represent the key attributes for specifying a VI: location, monitoring, software configuration, reliability, and elasticity. *Location* is related to the interaction between user and management framework. A user can define the exact geographical location where the virtual component must be provisioned due to specific reasons (e.g., data dependency, security, governmental law). *Monitoring, reliability*, and *elasticity* indicate the presence of attributes for configuring the services exposed by an InP. For instance, defining the reliability and monitoring levels, or defining the elasticity rules (capacity variation per time). *Software configuration* indicates the availability of attributes to configure and interact with the legacy software running on the virtual resources (e.g., configuring rules for load balancing, number of active threads of a pager server) The value $\surd$ indicates the availability of an attribute to specify the required VI component. As we can observe, the language that offers more attributes covers only 55% of VI components and particularities. Most languages do not reach 37%, and none allows the complete description of VI components. *This analysis hence shows the need for a specific language to describe virtual infrastructures, by enabling the parameterization of all their components, attributes, and particularities.*

3.3 Virtual Infrastructure Description Language - VXDL

VXDL allows the description of a VI or a *compute-and-communication* resource graph [6]. As presented in Figure 3.1, VXDL enables the specification of a VI according to different levels of application requirements (e.g., SaaS and PaaS). It fulfills all the requirements in Table 3.1 and brings four original aspects:

- the joint specification of network and computing elements;
- a language that permits a simple and abstract description of complex virtual infrastructures;

	VI components						VI properties				
Language	vNodes	vAccessPoints	vRouters	vStorage	vLinks	Groups	Location	Monitoring	Software configuration	Reliability	Elasticity
ClassAd	√	-	-	-	-	-	-	-	-	-	-
Redline	√	-	-	-	-	√	-	-	-	-	-
SWORD	√	-	-	-	-	√	√	-	-	-	-
JSDL	√	-	-	-	-	√	-	-	√	-	-
GLUE	√	-	-	-	√	√	-	-	√	-	-
BPDL	√	-	-	-	-	√	-	√	√	-	-
CIM	√	-	√	-	√	√	-	√	-	√	-
NDL	√	-	-	-	√	√	√	-	-	-	-
vgDL	√	-	-	-	-	√	-	-	-	-	-
libvirt	√	-	-	-	-	-	-	-	√	-	-
Rspec	√	-	-	√	√	√	-	-	-	-	-
NML	√	√	√	-	√	√	-	-	-	-	-
NSI	√	-	-	√	√	-	-	√	-	-	-
OCCI	√	-	√	√	-	-	√	-	√	-	-
OVF	√	-	√	-	-	-	-	√	√	-	√
USDL	√	-	-	-	-	-	-	-	-	-	√
VXDL	√	√	√	√	√	√	√	√	√	√	√

Table 3.1: A comparison among some resource description languages focused on virtual infrastructure specification. √ identifies the presence of an attribute. VXDL, described in Section 3.3, fulfills all the VI requirements.

- the VI timeline specification used to describe the temporal and elastic aspects of virtual infrastructures, defining intervals where each resource is really required; and
- the specification of attributes representing the expected Quality-of-Experience (QoE).

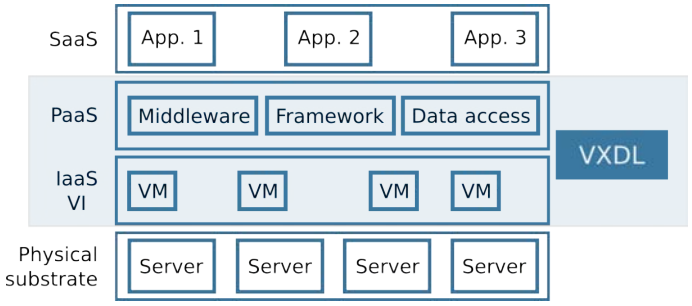


Figure 3.1: The positioning of VXDL considering the service models offered by Cloud Computing providers.

As discussed earlier, users can model VIs based on the requirements of their applications. In this context, VXDL enables the development of requests such as *"I am a user from London, and I would like the provisioning of a VI composed of 2 vNodes, with reservation slot starting on 10/11/2011 at 14:00:00, lasting 1 hour. One node must be geographically located in Lyon and the other in Paris. The vNodes must have 2 GB of RAM, and 1 CPU core of 2.0 GHz. My vNodes are critical and must be provisioned with a reliability of 99.9%. My application requires a bi-directional virtual link of 10 Mbps to communicate efficiently."*.

In addition, Infrastructure Providers (InPs) can share requests, propose templates, and expose a subset of virtualized resources. This information is essential in distributed discovery and allocation systems. For example, an InP A could share an information and request a VI composition using resources of InP B: *"I would like to extend the VI of my client over your substrate for a reservation period starting on 10/08/2011 at 08:00:00, during 3 months. To be executed, the financial-information system operated by this company requires 8 storage servers and 20 computing nodes. Each data server must have a storage capacity of 1 TB, while the computing nodes must have 4 GB of memory. These resources will be used by a temporary branch of my client's company, and must be interconnected with the headquartered access point located in California, through a dedicated communication channel with 100 Mbps of bandwidth capacity. Both IT and network resources must be provisioned with a high-level security to guarantee the data confidentiality."*.

These two sentences indicate the scenarios where VXDL acts. In Section 3.3.3 we describe examples to show how the language helps the description of VIs in both contexts.

3.3.1 Language grammar

VXDL proposes a simple and high level grammar to specify virtual infrastructures hiding hardware details. The syntax and lexical rules of VXDL are presented in Figure 3.2 and Figure 3.3 using the Extended Backus-Naur Form (EBNF) [EBNF, 1996]. EBNF adds the regular expression syntax of regular languages to the BNF notation [Crocker and Overell, 1997], in order to allow very compact specifications. The definitions of *string, integer, float,*

double, boolean, date and *time* are not presented to simplify the description, concentrating only in defining the elements of VXDL. These elements follow the international format in accordance with W3C recommendations [Biron and Malhotra, 2001]. As observed in these figures, VXDL follows the XML standard. An XML Schema Definition (XSD) and the documentation of VXDL are maintained on `http://www.ens-lyon.fr/LIP/RESO/Software/vxdl/`.

Following a tree organization, a VXDL description begins by the definition of *virtualInfrastructure*, as shown by the first line of Figure 3.2. Thus, the virtual infrastructure is specified and composed respecting the defined rules. We can note that a set of elements are optional permitting the adaption of VXDL to different contexts with specific requirements. In the next section, we discuss the semantic of VXDL and demonstrate the usage of its grammar.

3.3.2 VXDL file structure

As presented in Figure 3.4, a VXDL file structure is conceptually composed of four parts that represent all aspects of a VI:

- the general description of the VI, and the definition of default values;
- the description of virtual resources;
- the description of the virtual network topology; and
- the description of the internal VI timeline.

A VXDL document is organized as follows: the root element *virtualInfrastructure* contains recursive entries for vNode, vRouter, vAccessPoint, vStorage, vLink, and vGroup elements. Each virtual resource is identified and referred by a unique identifier (*id* attribute). In addition, the document contains a single entry for *virtualTimeline* description. The description of VXDL elements is partially optional. It is possible to specify a simple communication infrastructure (for example, a virtual private network) or a simple aggregate of end resources without any network topology description.

The UML specification of VXDL is presented in Figure 3.5, showing the interaction and dependency of its components. To simplify the language definition, some of the attributes are factorized and grouped in special classes such as *common* and *capacity*. Moreover, a special component, vResource, is used to abstract the interaction among other virtual resources.

In the following sections, we detail the conceptual organization of VXDL documents.

3.3.2.1 VI general description

The first part of a VXDL document concerns the general and default attributes of a virtual infrastructure. Performing an analogy with a graph specification, this section of VXDL defines the *default* attributes for all VI components, and additional attributes not placed in a graph. Looking at Figure 3.5, the object *virtualInfrastructure* represents the general description, being composed of the following attributes (following the inheritance of *common*):

- the identification of the infrastructure;


```
VXDL = '<virtualInfrastructure>', owner, {user}, [extension], common,
    {vNode | vRouter | vAccessPoint | vStorage | vLink | vGroup},
    [virtualTimeline], sharedRiskGroup, '</virtualInfrastructure>';

common = id, [startDate], [totalTime], [reliability], [location], [security], [monitoring],
    [monitoringURL];

id = '<id>', string, '</id>';
startDate = '<startDate>', date, time, '</startDate>';
totalTime = '<totalTime>', time, '</totalTime>';
owner = '<owner>', string, '</owner>';
user = '<user>', string, '</user>';
extension = '<extension>', boolean, '</extension>';
reliability = '<reliability>', string, '</reliability>';
security = '<security>', string, '</security>';
monitoring = '<monitoring>', string, '</monitoring>';
monitoringURL = '<monitoringURL>', string, '</monitoringURL>';
location = '<location>', string, '</location>';

sharedRiskGroup = '<sharedRiskGroup>', id, resources, '</sharedRiskGroup>';
resources = '<resources>', {id}, '</resources>';

vResource = common, [model], [exclusivity], [tags];
model = '<model>', string, '</model>';
exclusivity = '<exclusivity>', boolean, '</exclusivity>';
tags = '<tags>', {key, value}, '</tags>';
key = '<key>', string, '</key>';
value = '<value>', string, '</value>';

vNode = '<vNode>', vResource, [devices], [ip],
    [imageURL], [storage], [cpu], [memory], '</vNode>';

imageURL = '<imageURL>', string, '</imageURL>';
devices = '<devices>', {string}, '</devices>';
ip = '<ip>', string, '</ip>';

memory = '<memory>', (longInterval | longSet | longSimple), memoryUnit, '</memory>';
cpu = '<cpu>', frequency, [cores], [architecture], '</cpu>';
frequency = '<frequency>', (doubleInterval | doubleSet | doubleSimple), frequencyUnit,
    '</frequency>';
cores = '<cores>', integer, '</cores>';
architecture = '<architecture>', string, '</architecture>';
storage = '<storage>', (longInterval | longSet | longSimple), memoryUnit, '</storage>';

vStorage = '<vStorage>', vResource, type, [volume], [imageURL], [ip], '</vStorage>';
type = '<type>', 'NAS' | 'DAS' | 'SAN', '</type>';
volume = '<volume>', string, '</volume>';

vAccessPoint = '<vAccessPoint>', vResource, [ip], {loadBalancer | nat | masquerade},
    '</vAccessPoint>';
loadBalancer = '<loadBalancer>', {inEndpoint}, {publicIP}, '</loadBalancer>';

inEndpoint = '<inEndpoint>', string, '</inEndpoint>';
publicIP = '<publicIP>', ip, '</publicIP>';
nat = '<nat>', [inEndpoint], [inPort], protocol, [publicIP], [outPort], '</nat>';
inEndpoint = '<inEndpoint>', string, '</inEndpoint>';
inPort = '<inPort>', long, '</inPort>';
outPort = '<outPort>', long, '</outPort>';
protocol = '<protocol>', string, '</protocol>';
masquerade = '<masquerade>', {inEndpoint}, publicIP, '</masquerade>';
```

Figure 3.2: Specification of the VXDL grammar (version 2.7) using EBNF (part 1 of 2).


```
vLink = '<vLink>', vResource, source, destination, [latency], [bandwidth], '</vLink>';
source = '<source>', string, '</source>';
destination = '<destination>', string, '</destination>';
bandwidth = '<bandwidth>', forward, [reverse], '</bandwidth>';
forward = '<forward>', (doubleInterval | doubleSet | doubleSimple), bandwidthUnit,
    '</forward>';
reverse = '<reverse>', (doubleInterval | doubleSet | doubleSimple), bandwidthUnit,
    '</reverse>';
latency = '<latency>', (doubleInterval | doubleSet | doubleSimple), latencyUnit,
    '</latency>';

vRouter = '<vRouter>', vResource, [memory], dataPlane, controlPlane, '</vRouter>';
dataPlane = '<dataPlane>', scheduling, '</dataPlane>';
scheduling = '<scheduling>', ('LQF' | 'RR' | 'FCF'), '</scheduling>';
controlPlane = '<controlPlane>', layer, type, routingProtocol, [routingTable], '</controlPlane>';
type = '<type>', ('static' | 'dynamic'), '</type>';
routingProtocol = '<routingProtocol>', ('OSPF' | 'BGP' | 'RIP'), '</routingProtocol>';
layer = '<layer>', string, '</layer>';
routingTable = '<routingTable>', rtSize, {route}, '</routingTable>';
rtSize = '<rtSize>', integer, '</rtSize>';
route = '<route>', source, destination, gateway, '</route>';
gateway = '<gateway>', string, '</gateway>';

vGroup = '<vGroup>', id, multiplicity, [location], {vNode | vRouter | vAccessPoint |
    vStorage | vLink}, '</vGroup>';
multiplicity = '<multiplicity>', integer, '</multiplicity>';

virtualTimeline = '<virtualTimeline>', id, {timeline}, '</virtualTimeline>';
timeline = '<timeline>', id, [after], {activate}, totalTime, '</timeline>';
after = '<after>', string, '</after>';
activate = '<activate>', string, '</activate>';

longInterval = '<interval>', longValues, '</interval>';
longValues = '<min>', long, '</min>', '<max>', long, '</max>';
longSet = '<set>', longValues, '<step>', long, '</step>',  '</set>';
doubleInterval = '<interval>', doubleValues, '</interval>';
doubleValues = '<min>', double, '</min>', '<max>', double, '</max>';
doubleSet = '<set>', doubleValues, '<step>', double, '</step>',  '</set>';
memoryUnit = 'MB' | 'GB' | 'TB';
frequencyUnit = 'GHz' | 'MHz';
bandwidthUnit = 'Gbps' | 'Mbps';
latencyUnit = 'us' | 'ms' | 's';
```

Figure 3.3: Specification of the VXDL grammar (version 2.7) using EBNF (part 2 of 2).

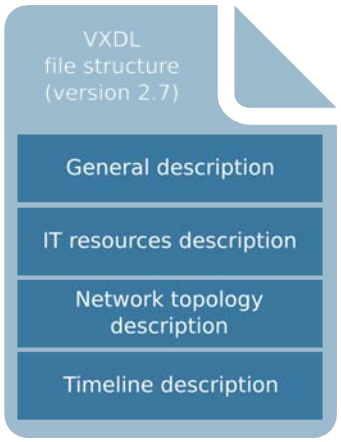


Figure 3.4: The conceptual VXDL file structure (version 2.7).

- an attribute informing if the VI being described is an extension of a VI already provisioned;
- the reservation period (start date and duration);
- the default security and reliability levels;
- the required monitoring level;
- the global geographical location;
- the owner and the list of users allowed to access and manipulate the VI;
- and the composition of Shared Risk Groups (SRGs).

The values of some attributes (e.g., the levels of monitoring, security and reliability are defined by the substrate configuration and availability) are defined and exposed by the InP during the SLA negotiation. The default definitions informed in this section can be refined for each element. For example, a VI defined with default reliability requirement of 99.9% and default security level *bronze*, can have internal critical components with 99.99% and *gold* requirements, for reliability and security, respectively.

By defining SRGs, users can identify groups of critical resources that should not be allocated on the same physical resources. This definition is an extension of the Shared Risk Link Groups (SRLGs) used to avoid the allocation of back-up MPLS tunnels in the same SRLG of the interfaces they are protecting [Kompella and Rekhter, 2005]. Besides virtual links, VXDL allows SRGs composed of IT resources. SRGs can be used for defining the reliability requirements at the user level, instead of requesting an InP service (as discussed in Chapter 6).

In addition, the *virtualInfrastructure* element also has entries for the components composing the other three main conceptual parts of VXDL, as described in the sequence.

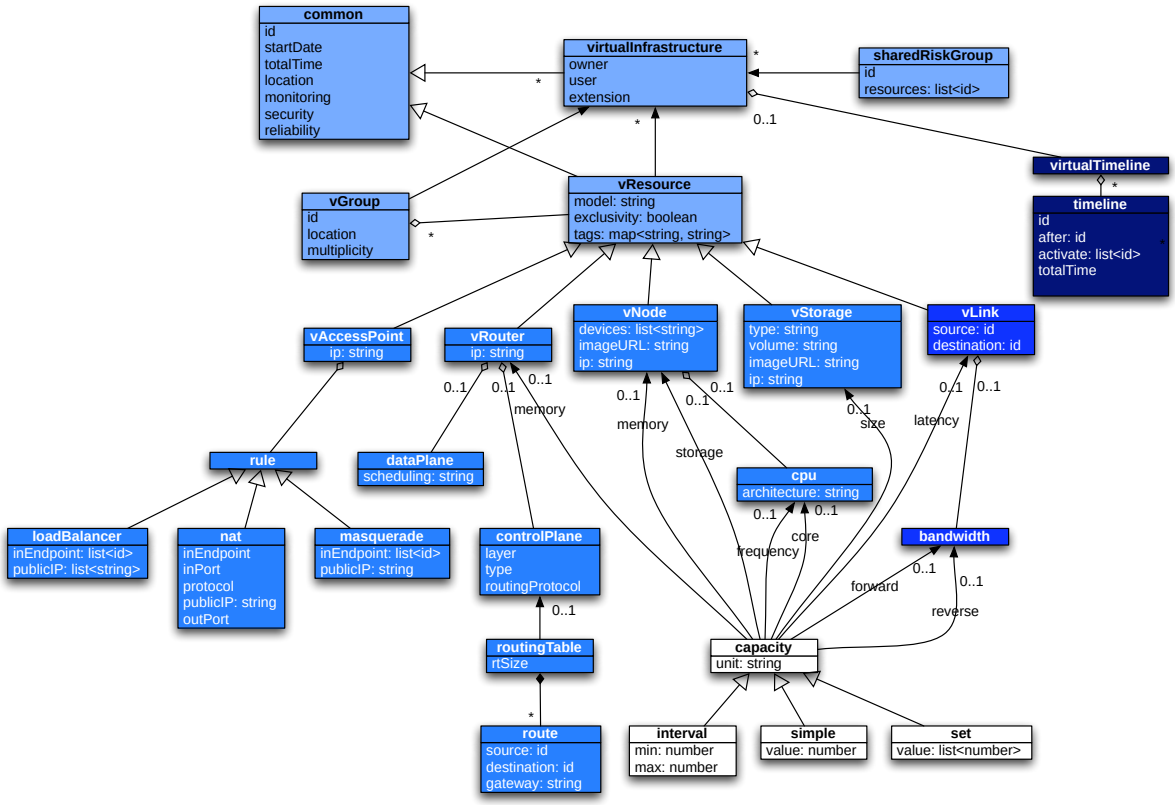


Figure 3.5: UML specification of VXDL (version 2.7).

3.3.2.2 Virtual resources description

The second part of VXDL describes, in a simple and objective way, all the IT resources and groups of them. Keeping the analogy with a graph description, these components are responsible for describing the vertices. The objects with the blue color and white font shown in Figure 3.5 are mainly explored to describe virtual resources. One can observe the presence of attributes that specify the basic building blocks of virtual infrastructures: vNodes, vAccessPoints, vStorage, and vRouters, as discussed in Section 2.4.2 in Chapter 2.

VXDL allows the basic resource parameterization (values for memory size, CPU speed, among others), using *simple* values, *intervals*, and *sets* of acceptable values. A specific attribute, the *tags*, enables the specification of a set of key-value pairs, allowing the interaction of VXDL with other description languages and models. Tags can also be used to transmit software configuration to the management framework, aiming for automatic configuration.

An important feature of VXDL is that it proposes cross-layer parameters (i.e. application level and physical level attributes) for all components. For example, with the specification of *location* and *exclusivity*, and the composition of shared risk groups, users can directly transmit application-specific information and constraints to the management framework. The *location* attribute corresponds to a physical embedding constraint, which places a virtual resource (group, or an entire VI) in a given physical location (e.g. a city, a country, a site, or a machine), for an application- or user-specific reason.

On the other hand, on a virtualized physical substrate, multiple virtual machines can be executed in the same physical host sharing the available resources. VXDL enables the specification of *exclusivity*, meaning that only one virtual machine must be allocated in a physical host, disabling any sharing to offer explicit security and performance guarantees. In addition, software to be deployed on resources can be specified by this part of the VI description.

Analyzing the virtual resources individually, we observe that the attributes proposed to describe vRouters enable the specification of data-planes and control-planes individually, being more adaptable to different technologies and usage scenarios [GEY] [SAI]. Also, the attributes to specify vAccessPoints enable the definition of VI boundaries, and the rules that guide the interaction of the VI with the external world, not part of the VI (e.g., a public resource, the Internet). A NAT, a set of masquerade rules, and a Load Balancer are proposed to interconnect and organize the incoming and outcoming traffic.

Finally, besides the storage capacity, attributes to identify the *type* of storage and the *volume* configuration are proposed. The type of a vStorage can be chosen among DAS (Direct-attached storage), NAS (Network-attached storage), or SAN (Storage Area Network) technologies.

3.3.2.3 Virtual network topology description

Continuing the line of a graph description, these elements define attributes to specify the edges of VI graph. These virtual links define interconnections between IT resources, between individual resources and groups, inside groups, and among groups.

The dark-blue color and white font components presented in Figure 3.5 show the main attributes available to describe virtual links. The required capacity and quality-of-service values are defined by the *bandwidth (forward and reverse)* and *latency* attributes. Maximum and minimum values can be defined for latency and bandwidth (forward and

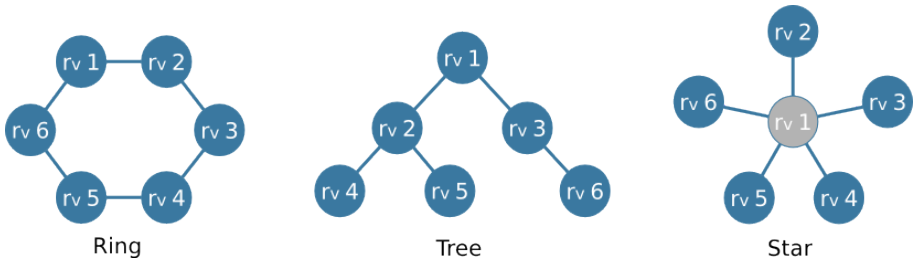


Figure 3.6: Examples of network topologies used to interconnect virtual resources.

reverse).

vLinks are specified as source-destination pairs. The extremities of a virtual link can be a single IT resource (e.g., vNode, vRouter, vAccessPoint, vStorage) or a group of resources. Figure 3.6 presents some topologies that can be described with VXDL: a ring, a tree, and a star. While many topologies can be described using VXDL, we select these three to exemplify the source-destination pairs and the positioning of a vRouter. In this example, the *Ring* topology is described requiring a uni directional path with bandwidth of *25 Mbps* to interconnect six vNodes. The fragment of VXDL presented in Figure 3.7 shows the description of this topology, where only one vLink was specified and a set of pairs defined the resources that are interconnected by this link. The *Star* topology exemplifies the presence of a vRouter (r_v1) to control the traffic between five vNodes.

3.3.2.4 Virtual timeline description

Finalizing the analogy with a graph description, this section of VXDL describes the elastic aspects of VIs or, in other words, the subgraphs grouped by time slots. Virtual infrastructures are allocated and provisioned for a defined time slot. Time slot duration is specific to the substrate-management framework and consequently this parameter is configured by the manager of the virtualized environment.

Often, the virtual infrastructure components are not used simultaneously or all along the virtual infrastructure lifecycle. Thus, the specification of an internal timeline for each virtual infrastructure can help optimizing the allocation, scheduling, and provisioning of virtual resources.

Figure 3.8 explains the timeline description of a VI composed of five IT resources: three computing clusters (*cluster A, B, and C*), a database node, and a visualization node. This VI is divided in three stages according to the resource usage. In *stage I*, a data transfer is performed between the database server and the computing clusters. At this moment, a network interconnection with enough capacity is required to perform the transfer before a defined deadline. After the data transfer, *stage II* begins, during which a computation is performed and clusters must be interconnected with a different network configuration. Finally, in *stage III* a data transfer must be carried out for the final visualization of results. Again, virtual links with enough capacity must be provisioned to interconnect the computing clusters with the visualization resources.

By analyzing this example it is possible to observe that the specification of internal timelines can help management frameworks in resources allocation and provisioning: the timeline indicates the exact moment where resource are required helping InPs in not wasting over-provisioned capacities. If a VI lifetime is long enough and can be divided in


```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="my VPXI" owner="koslovski">
    <user>guilhermekoslovski</user>
    <startDate>2011-09-21T08:00:00</startDate>
    <totalTime>PT3H</totalTime>
    <vLink id="virtual link 1">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 1</source>
      <destination>rv 2</description>
    </vLink>
    <vLink id="virtual link 2">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 2</source>
      <destination>rv 3</description>
    </vLink>
    <vLink id="virtual link 3">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 3</source>
      <destination>rv 4</description>
    </vLink>
    <vLink id="virtual link 4">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 4</source>
      <destination>rv 5</description>
    </vLink>
    <vLink id="virtual link 5">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 5</source>
      <destination>rv 6</description>
    </vLink>
    <vLink id="virtual link 6">
      <bandwidth>
        <forward>
          <simple>25.0</simple>
          <unit>Mbps</unit>
        </forward>
      </bandwidth>
      <source>rv 6</source>
      <destination>rv 1</description>
    </vLink>
  </virtualInfrastructure>
</description>
```

Figure 3.7: Description of a ring topology using VXDL.

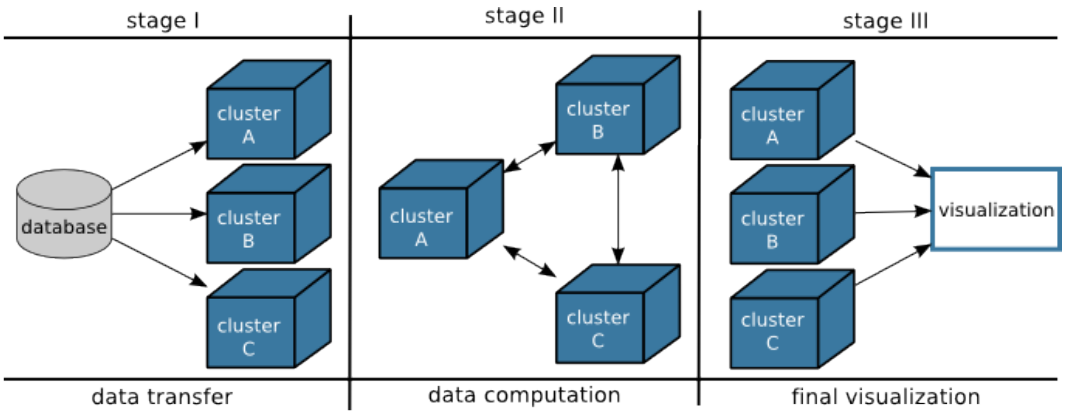


Figure 3.8: Internal timeline of a Virtual Infrastructure. Three distinct stages are identified: i) data transfer, ii) data computation, and iii) visualization of resulting data.

well-defined stages of resource usage, the infrastructure provider can activate the virtual resource only when they are really required by the application. In Figure 3.5 the dark-blue objects represent the attributes offered to specify virtual timelines.

3.3.3 Describing VI scenarios with VXDL

This section presents examples of scenarios where VXDL can be used. It aims to demonstrate the usage and the expressiveness of VXDL.

3.3.3.1 Requesting and provisioning vNodes and vLinks

The first example, explored in the context of the HIPCAL project [HIP], demonstrates the usage of a simple set of vNodes during the reservation and activation. The goal of this example is to show how the attributes are used in the description and provisioning processes.

Figure 3.9 presents the description of two vNodes interconnected by a single vLink. The vNodes require a 2 GHz CPU (1 core) and a memory capacity between 2 GB and 4 GB. Any value within this range is acceptable. Observe that the VI requires a monitoring service described as *gold*. The meaning of this keyword is defined by the infrastructure provider during the service definition and exposition.

Figure 3.9 also describes a vLink to interconnect *node 1* and *node 2*, requiring a bandwidth capacity between 10 Mbps and 20 Mbps. For the sake of simplicity, this example does not describe the VI timeline.

A management framework that receives this VXDL description must allocate and provision the virtual resources. During the allocation, the framework decides on a single value to be provisioned within the acceptable ranges informed by the user. Usually this decision if performed by the infrastructure provider considering the load distribution, economic aspects, physical substrate fragmentation, among others factors (we discuss the allocation problem in detail in Chapter 5).

After the decision process, a new VXDL description is composed and returned to inform relevant information about the provisioned VI, such as: i) which values were provisioned between the ranges specified by the user; ii) the IP address through which the virtual resources can be accessed; and iii) the monitoring address of every resource from which a

user can access on-line information and metrics (in accordance with the service exposed by the infrastructure provider).

Figure 3.10 presents a VXDL description that states the values selected to be provisioned by the management framework. In this example, *node 1* was provisioned with 3 GB of memory, and *node 2* with 2 GB. Also, the virtual link was configured with a bandwidth of 10 Mbps for both forward and reverse. Finally, the attributes highlighted in green show the address used to access the virtual resources, and where monitoring information is available.

3.3.3.2 Multiuser game service

This scenario presents a virtual infrastructure requested to host a multiuser game service where client requests can arrive at any time during the reservation period. This example was developed in the context of HIPCAL [HIP] and CARRIOCAS [CAR] projects, and the main objective is to demonstrate how VXDL can be used to specify dynamic VIs where the number of resources and their configuration change during the execution time.

As shown in Figure 3.11, a virtual server must be provisioned to host the game service during a defined period. Each client who wishes to play must connect to the central server. Virtual links between clients access-point and game server are dynamically provisioned to ensure the Quality-of-Service (QoS) of players. Clients can be located at different cities. The dashed boxes and links represent resources that must be dynamically provisioned, whereas the solid boxes and links represent the connected clients.

The VI that will host the multiuser game service requires a reservation slot with a duration of 10 days, starting at *2011-04-21 08:00:00* and is composed of:

- Virtual Infrastructure:
  - a centralized game server (vNode) able to connect to a maximum of 100 clients (vAccessPoints) using TCP protocol for communication;
  - The security level required is *gold* among the options: *basic, silver, gold,* and *platinum.*
- Game server requirements:
  - RAM capacity: 8 GB;
  - Storage capacity: 100 GB;
  - The game server is a critical component of this VI. Thus, a high reliability level is required (*platinum* in this case, among the options: *basic, silver, gold,* and *platinum*).
- Virtual links:
  - 100 vLinks must be dynamically provisioned, respecting the arrival of remote clients;
  - Each virtual link must have at least 10 Mbps of bandwidth reserved to be provisioned when the client requests the connection.


```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="example 1" owner="koslovski">
    <user>guilhermekoslovski</user>
    <startDate>2011-04-21T08:00:00</startDate>
    <totalTime>PT4H</totalTime>
    <monitoring>gold</monitoring>
    <vNode id="node 1">
      <cpu>
        <cores>1</cores>
        <frequency>
          <simple>2</simple>
          <unit>GHz</unit>
        </frequency>
      </cpu>
      <memory>
        <interval>
          <min>2.0</min>
          <max>4.0</max>
        </interval>
        <unit>GB</unit>
      </memory>
    </vNode>
    <vNode id="node 2">
      <cpu>
        <cores>1</cores>
        <frequency>
          <simple>2</simple>
          <unit>GHz</unit>
        </frequency>
      </cpu>
      <memory>
        <interval>
          <min>2.0</min>
          <max>4.0</max>
        </interval>
        <unit>GB</unit>
      </memory>
    </vNode>
    <vLink id="link 1">
      <bandwidth>
        <forward>
          <interval>
            <min>10.0</min>
            <max>20.0</max>
          </interval>
          <unit>Mbps</unit>
        </forward>
        <reverse>
          <interval>
            <min>10.0</min>
            <max>20.0</max>
          </interval>
          <unit>Mbps</unit>
        </reverse>
      </bandwidth>
      <source>node 1</source>
      <destination>node 2</description>
    </vLink>
  </virtualInfrastructure>
</description>
```

Figure 3.9: Description of 2 vNodes interconnected by a vLink using the VXDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="example 1" owner="koslovski">
    <user>guilhermekoslovski</user>
    <startDate>2011-04-21T08:00:00</startDate>
    <totalTime>PT4H</totalTime>
    <vNode id="node 1">
      <ip>200.142.35.10</ip>
      <monitoringURL>http://200.142.35.50/node1</monitoringURL>
      <cpu>
        <cores>1</cores>
        <frequency>
          <simple>2</simple>
          <unit>GHz</unit>
        </frequency>
      </cpu>
      <memory>
        <simple>3</simple>
        <unit>GB</unit>
      </memory>
    </vNode>
    <vNode id="node 2">
      <ip>200.142.35.11</ip>
      <monitoringURL>http://200.142.35.50/node2</monitoringURL>
      <cpu>
        <cores>1</cores>
        <frequency>
          <simple>2</simple>
          <unit>GHz</unit>
        </frequency>
      </cpu>
      <memory>
        <simple>2</simple>
        <unit>GB</unit>
      </memory>
    </vNode>
    <vLink id="link 1">
     <monitoringURL>http://200.142.35.50/link1</monitoringURL>
      <bandwidth>
        <forward>
          <simple>10</simple>
          <unit>Mbps</unit>
        </forward>
        <reverse>
          <simple>10</simple>
          <unit>Mbps</unit>
        </reverse>
      </bandwidth>
      <source>node 1</source>
      <destination>node 2</destination>
    </vLink>
  </virtualInfrastructure>
</description>
```

Figure 3.10: Example of a VXDL description used to inform the configuration provisioned by the management framework according to the request in Figure 3.9.

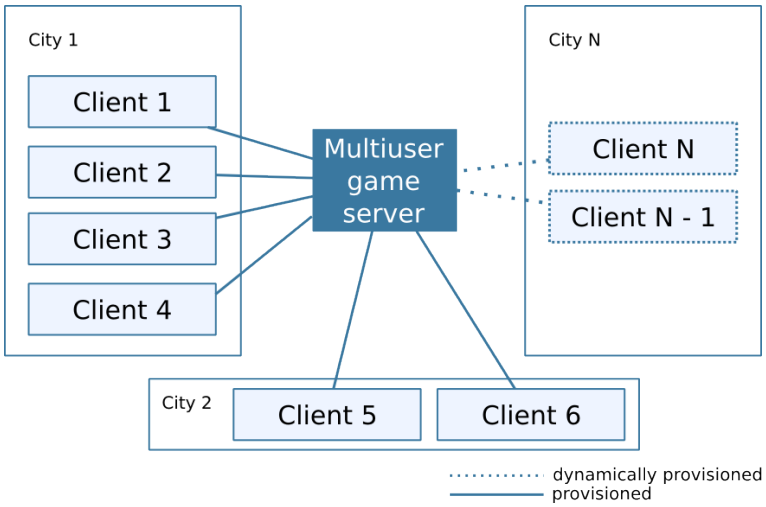


Figure 3.11: Example of a multiuser game server connected to a set of remote clients.

Within VXDL, this scenario can be presented using two types of description: one to specify the game server, and another to specify the clients. An initial VXDL file describes the game server configuration and prepares the remote vAccessPoints. The submission of a new VXDL informing the location of the new client access point is necessary for each client that arrives to play the game. At this moment, the virtual link is provisioned to interconnect client and game server. The VXDL file presented in Figure 3.12 shows the initial configuration.

The requests of client connections can arrive at any time during the reservation period. The VXDL description shown in Figure 3.13 represents the demand for provisioning a vLink to interconnect the new client to game server. In other words, this VXDL extension file informs that one of the vAccessPoints prepared during the initial configuration became active. In this case, a client located at *City N* informs its IP address and requests the connection to the original VI for a period of 2 hours.

We extend the discussion of this example to identify how vLinks can be provisioned to interconnect the new client with the game server. Figure 3.14 exemplifies two possible scenarios for this dynamic provisioning.

- Case A: the client is outside the administrative domain. By administrative domain we mean the domain of virtualized and exposed resources that is administered by the management framework. Thus, the vLink can be provisioned with all required guarantees (forward and reverse bandwidth) until the end of the administrative domain where the vAccessPoint is located.

- Case B: the client is within the administrative domain. In this scenario, a vLink can be provisioned until the client's vNode instead of the vAccessPoint. Consequently, the full path that provisions the vLink has the same guarantees.

3.3.3.3 VISUPIPE

The High Performance Collaborative Remote Visualization (VISUPIPE) software generates 30 images of 20 megapixels x 32 bits per second x 2 for stereo, which results in a

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="multiuser game service" owner="koslovski">
    <startDate>2011-04-21T08:00:00</startDate>
    <totalTime>P10D</totalTime>
    <security>gold</security>
    <vNode id="game server">
      <reliability>platinum</reliability>
      <memory>
        <simple>8</simple>
        <unit>GB</unit>
      </memory>
      <storage>
        <simple>100</simple>
        <unit>GB</unit>
      </storage>
    </vNode>
    <vGroup id="access points" multiplicity="100">
      <vAccessPoint id="client">
        <nat>
          <protocol>TCP</protocol>
        </nat>
      </vAccessPoint>
    </vGroup>
    <vLink id="dynamic link">
      <bandwidth>
        <forward>
          <simple>10</simple>
          <unit>Mbps</unit>
        </forward>
        <reverse>
          <simple>10</simple>
          <unit>Mbps</unit>
        </reverse>
      </bandwidth>
      <source>game server</source>
      <destination>access points</destination>
    </vLink>
  </virtualInfrastructure>
</description>
```

Figure 3.12: VXDL description for the initial VI configuration of the multiuser game service example.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="multiuser game service" owner="koslovski">
    <extension>true</extension>
    <user>new client</user>
    <startDate>2011-04-21T10:00:00</startDate>
    <totalTime>PT2H</totalTime>
    <vAccessPoint id="new client">
      <location>CityN</location>
      <nat>
        <protocol>TCP</protocol>
        <publicIP>200.142.35.110</publicIP>
      <nat>
    </vAccessPoint>
    <vLink id="dynamic link">
      <bandwidth>
        <forward>
          <simple>10.0</simple>
          <unit>Mbps</unit>
        </forward>
        <reverse>
          <simple>10.0</simple>
          <unit>Mbps</unit>
        </reverse>
      </bandwidth>
      <source>new client</source>
      <destination>game server</destination>
    </vLink>
  </virtualInfrastructure>
</description>
```

Figure 3.13: VXDL description of a client request for the multiuser game service example.

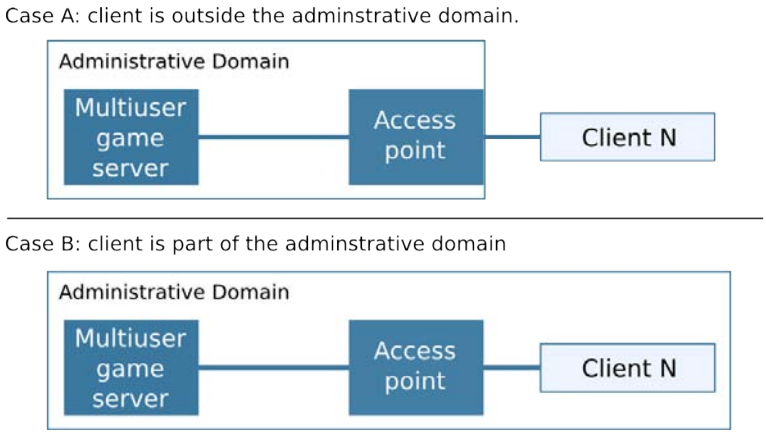


Figure 3.14: Example of clients provisioning for the multiuser game service use case.

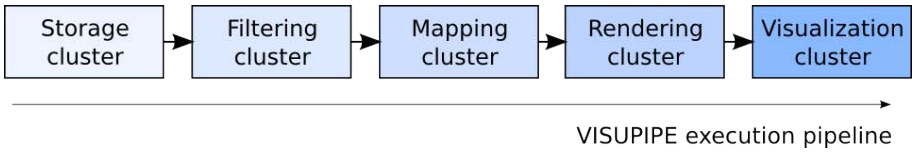


Figure 3.15: Execution pipeline of the High Performance Collaborative Remove Visualization (VISUPIPE) application.

bandwidth requirement of 38.4 Gbps when a researcher wants to explore the data and images in real time. The VISUPIPE execution pipeline is presented in Figure 3.15, where five clusters, each with different requirements (computing and communication), are interconnected by unidirectional links. The data information stored on *Storage cluster* is transferred to computing clusters (identified as *Filtering, Mapping,* and *Rendering*). Each cluster manipulates the data and transfers the resulting information to the next one, until it reaches the data visualization cluster (*Visualization cluster*). The goal of this example is to show how a virtual timeline is specified, informing the internal behavior of a VI.

The execution of the VISUPIPE application requires a virtual infrastructure composed of five clusters interconnected by virtual links with different requirements. The virtual infrastructure must be reserved for 11 hours, starting at *2011-04-21 08:00:00*. More specifically, the virtual infrastructure is composed of:

- Storage cluster:
  - Composed of 20 nodes, each requiring a minimum storage capacity of 200 GB;
  - Nodes must be interconnected by links with maximum latency of 0.200 ms.
- Filtering cluster:
  - Composed of 40 nodes, each requiring a minimum memory capacity of 4 GB and a minimum CPU speed of 2 GHz;
  - Nodes must be interconnected by links with maximum latency of 0.400 ms and minimum bandwidth of 100 Mbps.

- Mapping cluster:
  - Composed of 30 nodes;
  - Each node requires a minimum memory capacity of 4 GB, a minimum CPU speed of 2 GHz, and a minimum storage capacity of 80 GB.
- Rendering cluster:
  - Composed of 10 nodes, each requiring a minimum memory capacity of 8 GB, and a minimum CPU speed of 2 GHz;
  - The nodes interconnection tolerates a maximum latency of 0.150 ms.
- Visualization cluster:
  - Composed of 30 components. Each node requires a minimum memory capacity of 8 GB, and a minimum CPU speed of 2 GHz;
  - The nodes interconnection required is a maximum latency of 0.200 ms;
  - All nodes must be interconnected with a specific device (*visualization wall*) located in Lyon, France.
- Network topology configuration connecting the clusters:
  - Unidirectional links with minimum bandwidth capacity of 38.4 Gbps between: *Storage cluster* to *Filtering cluster*, *Filtering cluster* to *Mapping cluster*, *Mapping cluster* to *Rendering cluster*, and *Rendering cluster* to *Visualization cluster*.

Figures A.1, A.2, A.3, and A.4 of Appendix A present a VXDL specification for the VISUPIPE example. This VXDL file describes the five clusters and all network links required to execute VISUPIPE. Particularly, Figure A.3 shows the timeline composition of this scenario. The computing and data transfer stages are defined in advance (before the start of the application's execution). Each timeline (or stage) has a specific duration and begins after the end of the previous timeline.

3.4 Conclusions

VIs are malleable entities that can be composed according to users' and applications' requirements, which can differ in terms of computing, storage and communication [Vicat-Blanc et al., 2011]. The absence of one may affect the application's performance, but the over-provisioning can increase the cost for users.

An efficient and expressive description language is crucial for VI instantiation as it is the way (tool) users define the requirements of their applications. In this chapter, we presented the *Virtual Infrastructure Description Language (VXDL)*, a powerful language that enables the complete description and parameterization of VI components. VXDL has a simple and objective grammar that offers attributes to specify virtual resources, virtual network topology, and the internal timeline of virtual infrastructures.

Examples applied to different projects were presented. In each example, we described the scenario, objectives, and how VXDL can be used. In some cases, the discussion was

extended to show how a VXDL description (or a VI specification) is instantiated by infrastructure providers, and how VXDL is exploited for carrying the provisioned configuration.

Some attributes proposed by this language open opportunities for further studies. For example, a user can specify the required reliability level for a service, which is then implemented by the InP in a transparent way and charged from the final user (we discuss a mechanism to provide reliable virtual infrastructures in Chapter 6). On the other hand, an advanced user can define the shared risk groups. The benefits and drawbacks of each scenario should be identified, considering allocation costs and application performance.

Finally, it is important to mention that VXDL has been adopted by several software solutions as an intermediate communication tool between application and the execution environment, including: the HIPerNet framework [HIPerNet], the MOTEUR workflow manager [MOT], and the LYaTiss solutions for VI orchestration in the Clouds (VXDL parser [VXDLParser], LYaTiss Weaver [LYaTiss, 2011], LYaTiss Designer, VXDL translator, among others).

Four

Specifying VIs for executing distributed applications

The work presented in this chapter was published at the Journal of Grid Computing (JoGC 2010) [2], the First International Conference on Cloud Computing (CLOUDCOMP 2009) [5], and as a research report [16]. The work has been done in collaboration with the I3S team (`http://www.i3s.unice.fr`) in the framework of the HIPCAL project [HIP].

4.1 Introduction

Cloud Computing is increasingly being exploited for providing resources on-demand to users who reserve and then adapt the allocated virtual infrastructures to their application requirements. A challenging problem, both for InPs and users, is to estimate the number of resources to allocate from the Cloud for a specific period. Commercial Clouds implement several schemes to bill for resource usage, most based on coarse-grained metering of the amount of CPU and disk space consumed [AMAa], although network bandwidth is also a critical resource in many applications. Estimating the proper number of resources to allocate is the user's responsibility, an estimation that is far from trivial, especially when considering distributed applications. Assistance in estimating resource consumption and cost management is therefore highly desirable.

The task of determining the size of a VI for supporting the execution of a given application is often difficult. Although a quasi-unlimited number of computing resources can be allocated, a balance has to be found between the VI cost and the expected application

performance. Without assistance, the user has to specify a VI solely based on his previous experience in executing the application on the InP substrate.

In theory, a system can estimate the cost of a VI deployment and usage scenario if enough information is known on the application and the VI. However, it is hardly feasible to anticipate the precise needs or behavior of a parallel application on a given a VI.

The general problem of estimating resource requirements for a VI, while minimizing cost and maximizing performance, and translating them into a VI specification is intractable given that it depends on knowing the exact execution behavior of a distributed application. To make the problem tractable, we focus on workflow-based applications by proposing a solution that models the application behavior using the formalism of workflow descriptions. Such a solution is not too restrictive as many coarse-grained distributed applications can be modeled as workflows whose tasks are module invocations [Glatard et al., 2008]. In this solution, modules to be executed are the nodes of a workflow's directed graph, whereas module dependencies are the edges. Each application module might be invoked a number of times depending on the database size and, as long as no dependency exists between invocations, modules can be performed concurrently. With the workflow formalism, the application logic can be interpreted and exploited to produce an execution schedule estimate.

This chapter thus investigates a mechanism whereby users can foresee the optimal VI components and network topology needed for their application. The mechanism allows users to automatically deploy a VI for executing their applications. Figure 4.1 positions the contribution of this chapter considering the modular models offered by Cloud Computing providers. The proposed strategies translate an application description (e.g., computing nodes, middleware, framework) into VXDL files.

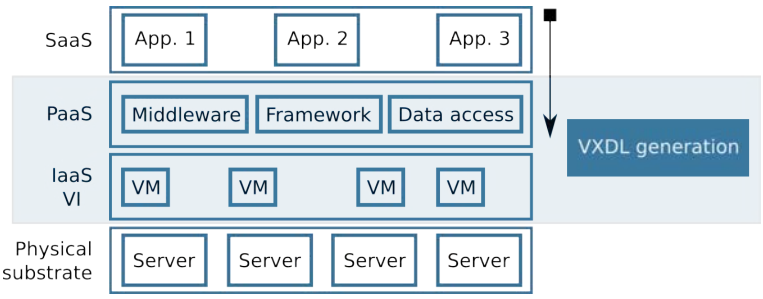


Figure 4.1: Positioning of the specification strategies and VXDL considering the module models offered by Cloud Computing providers.

The rest of this chapter is structured as follows. Section 4.2 reviews the state of the art related to translating workflow-based applications. Section 4.3 describes strategies to translate workflow-based applications into VXDL descriptions. An experimental analysis is presented in Section 4.4, and Section 4.5 compares the proposed strategies to generate VXDL descriptions. Conclusions are presented in Section 4.6.

4.2 State of the art

The VI design optimization problem is to determine both the number of resources and the network topology needed to run a given application efficiently. Minimizing the number of resources needed is not necessarily the best objective function as it may be better to find a trade-off between VI cost and application performance. Such an objective can be formulated as a cost function whose parameters depend on the size of the allocated infrastructure and its configuration. Both computing resources and network bandwidth have to be considered in this cost function.

Several existing resource allocation and task scheduling strategies for Grid applications focus on matchmaking algorithms whose goal is not to find an optimal allocation, but to identify suitable resources [Braun et al., 2001]. Best-effort algorithms such as *Min-Min*, *Max-Min* [Maheswaran et al., 1999] or *HEFT* [Topcuoglu et al., 2002] focus only on minimizing the application makespan whereas other QoS constrained algorithms consider a multi-objective scheduling problem [Yu and Buyya, 2005] [Sakellariou et al., 2005]. However, none takes into account the bandwidth of a link for data exchange between workflow modules. Silva et al. [Silva et al., 2008] presented a heuristic for resource allocation on utility computing infrastructures that optimizes the number of machines allocated to process tasks and speed up the execution within a budget. However, this heuristic is only suitable for bag-of-tasks problems where tasks are independent and do not communicate.

Some workflow-based allocation algorithms [Blythe et al., 2005] [Guo et al., 2006] [Mandal et al., 2005] [Xiao et al., 2007] can deliver better performance than matchmaking [Bittencourt and Madeira, 2010]. However, the objective of these algorithms is to minimize the application makespan without taking into account the execution cost on a pay-per-use platform such as Clouds. Ramakrishnan et al. [Ramakrishnan et al., 2009] presented a fault tolerant scheduling algorithm that orchestrates multiple workflows on Grid and Cloud infrastructures by duplicating the execution of some workflows thereby increasing the probability of success of individual tasks. This kind of approach, although potentially efficient in reducing execution time, does not consider the VI allocation cost. Senkul et al. [Senkul and Toroslu, 2005] presented an architecture for workflow scheduling that considers allocation cost and co-allocation of tasks on the same resource, but it does not consider resource heterogeneity.

In addition, resource allocation algorithms for mapping grid-based workflows onto Grid resources have been proposed in the context of service level agreements [Dang and Hsu, 2008] [Dang and Altmann, 2009]. These algorithms try to assign the workflow tasks to Grid resources so as to meet the user's deadline and minimize cost. The algorithms do not take into account network bandwidth.

4.3 Strategies to translate workflows into VXDL

We propose a process to specify distributed applications represented using workflows into VXDL descriptions [5]. This *application-mapping process* comprises three main steps:

1. *Workflow generation:* the workflow is generated using information extracted from the application, such as benchmark results, data input description, data transfer in each module, and the number of nodes required to perform a satisfactory execution.

A workflow description represents the input/output data, the data-processing modules, and the relationship of the application's modules. This representation has been well-studied into the literature (as reviewed in Section 4.2).

2. *Workflow translation into VXDL:* taking into account the application requirements (such as RAM, CPU speed, and storage size), users can develop a VXDL description asking for the desirable configuration of the VI. At this point, users can also declare that some components must be allocated in a specific location as well as define the virtual network topology, stating the proximity of the components (latency configuration) and the required bandwidth. In addition, users can assign critical components for identifying the virtual resources that require reliability support (a mechanism for providing reliability support is described in Chapter 6).

3. *VI provisioning:* a management framework allocates the VI components respecting the configuration expressed by the user (such as parameterizations and timeline composition). In addition, the software configuration (operating system, programming tools, and communication libraries), extracted directly from the application and described using VXDL, are deployed within the virtual resources that compose the VI. This step is detailed in Chapter 5.

4.3.1 Workflow-based applications

Complex and large scale applications are generally described as workflows and interpreted by engines that convert the work description into execution scripts. An example of a workflow composition is given in Figure 4.2, where *Floating* and *Reference* represent data units to be processed, and *CrestLines*, *CrestMatch*, *PFMatchICP*, *PFRegister*, *Yasmina* and *Baladin* are processing modules. Modules are invoked as many times as needed to process all data units received. The user describing the application focuses on the data processing logic rather than on the execution schedule. The structural application workflow is transformed into an execution schedule dynamically while the workflow engine is being executed.

Several languages have been proposed in the literature to describe workflows. On Grid-based infrastructures, Directed Acyclic Graph (DAG)-based languages, such as MA-DAG used in the DIET middleware [Caron and Desprez, 2006], have often been employed. To ease definition of large-scale distributed applications with a complex logic, more abstract languages have been introduced. For instance, Scufl was proposed within the myGrid project [myG] to present data flows enacted through the Taverna workflow engine [Oinn et al., 2007]. It has been one of the first languages to focus on the application data flow rather than on the generated task graph.

The language we chose for representing workflow, *GWENDIA* [GWE], is a data flow oriented language that aims at easing the description of complex application data flows from a user's perspective. GWENDIA is represented in XML using the tags and syntax defined below:

- Types: values flowing through the workflow are typed. Basic types are *integer*, *double*, *string*, and *file*;

- Processors: a *processor* is a data production module. A regular processor invokes a service through a known interface. Special processors include workflow *input* (a

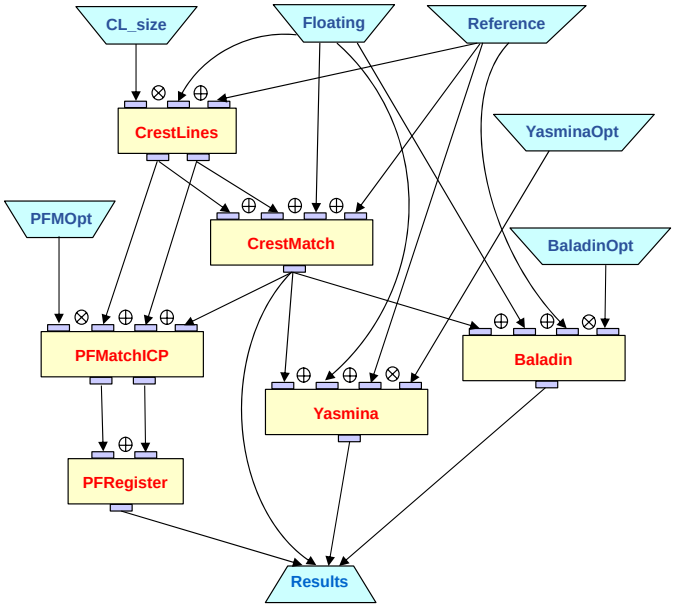


Figure 4.2: Bronze Standard's workflow.

processor with no inbound connectivity, delivering a list of externally defined data values), *sink* (a processor with no outbound connectivity, receiving some workflow output) and *constant* (a processor delivering a single, constant value);

- Processor ports: processor input and output ports are named and declared. A port may be an input (<*in*> tag), an output (<*out*> tag) or both an input/output value (<*inout*> tag). The input ports also define iteration strategies that control the number of invocation of the processor as a function of its inputs. A simple example is presented in Figure 4.3.

- Data link: a data link is a simple connection between a processor output port and a processor input port as exemplified in Figure 4.4.

Workflow-based applications are represented using descriptive languages and submitted to workflow managers, which are in charge of submitting the tasks and optimizing the execution process. For example, the data-intensive Grid-interfaced workflow manager (MOTEUR) [MOT] enacts workflows represented with both Scufl and GWENDIA languages. MOTEUR interprets the workflow description and submits the workflow tasks to a distributed infrastructure, optimizing the execution via three levels of parallelism: workflow parallelism, data parallelism and pipelining [Glatard et al., 2008].

4.3.2 Permanent and variable parts of workflows

Considering the workflow translation into VXDL, usually, applications are composed of two different sets of modules (or workflow entities): *permanents* and *variables*. This separation is performed according to the modules' lifetime, where it is considered permanent if it is present during the whole life of a workflow, and variable otherwise. For example, in

```
<workflow>
  <interface>
    <constant name="parameter" type="integer">
      <value>50</value>
    </constant>
    <source name="reals" type="double"/>
    <sink name="results" type="file"/>
  </interface>
  <processors>
    <processor name="docking" type="webservice">
      <wsdl url="http://localhost/docking.wsdl" operation="dock"/>
      <in name="param" type="integer"/>
      <in name="input" type="file"/>
      <out name="result" type="double"/>
      <iterationstrategy>
        <cross>
          <port name="param"/>
          <port name="input"/>
          </cross>
      </iterationstrategy>
    </processor>
    <processor name="statisticaltest" type="diet">
      <service path="weightedaverage"/>
      <in name="weights" type="double"/>
      <in name="values" type="list(integer)"/>
      <in name="coefficient" type="double"/>
      <out name="result" type="file"/>
      <iterationstrategy>
        <cross>
          <port name="coefficient"/>
          <match tag="patient">
            <port name="values"/>
            <port name="weights"/>
          </match>
        </cross>
      </iterationstrategy>
    </processor>
  </processors>
</workflow>
```

Figure 4.3: Description of workflow tasks using the GWENDIA language. This figure presents a few *processors* (or data production unit) and their interaction using input/output *ports*.

```
<links>
  <link from="reals" to="statisticaltest:coefficient"/>
  <link from="docking:result" to="statisticaltest:weights"/>
  <link from="statisticaltest:result" to="results"/>
</links>
```

Figure 4.4: Example of a workflow's task interconnections (*data links*) using the GWENDIA language.


```
...
<vNode id="database">
  <memory>
    <interval>
      <min>1</min>
    </interval>
    <unit>GB</unit>
  </memory>
  <storage>
    <simple>100</simple>
    <unit>GB</unit>
  </storage>
</vNode>
...
```

Figure 4.5: Description of a *permanent* database node using the VXDL language.

Notation	Meaning
$m_{\max}$	maximum number of computing nodes available on the VI
n	number of input data items
s	number of execution stages of the application
$m = (m_1, m_2, ..., m_s)$	number of nodes used at each execution stage with $\forall i, m_i \leq m_{\max}$
c_r	per-second cost of a computing resource
c_b	per-Mbps cost of bandwidth
Td_i	deployment time of stage i (in seconds)
$T_i(m_i, n, b)$	execution time of stage i (in seconds)
$b = (b_1, b_2, ..., b_{ki}), i \in [1..s]$	links bandwidth used at stage i (in Mbps)

Table 4.1: Notations used in the cost function model.

Figure 4.2 the data modules (represented in blue) are used by different computing modules and must be available during the entire workflow execution. The computing modules, on the other hand, are only used at determined moments.

The permanent part also comprises the dedicated nodes hosting middleware, framework, and database required for running the application. Figure. 4.5 presents a partial VXDL description of *permanent* resources, where a simple database node is described to store the data presented in Figure 4.2.

The *variable* part of a VI description depends directly on the information extracted from the workflow, such as input data, the number of modules, and the links between the modules. In addition, the virtual network topology is directly related to the *variable* part composition. The more dependence exists between workflow modules, the more complicated is the network topology.

4.3.3 Translation strategies

This section details the proposed strategies implemented in the MOTEUR workflow manager for designing VIs to execute workflow-based applications and for producing the VXDL files [2]. Table 4.1 summarizes all the parameters used for formulating the strategies.

4.3.3.1 Naive strategy

This strategy is naive in the sense that it only considers a single execution stage and the resources are statically allocated to each module, even though a module may not be used during the whole execution of the workflow.

Let p be the number of modules in a workflow, and t_i the benchmarked execution time of each module $i \in 1..p$. A set of $m_{\max}$ virtual computing nodes is allocated and split proportionally to each module execution time: $m_{\max} t_i / \sum_{j \in s} t_j$. The network bandwidth

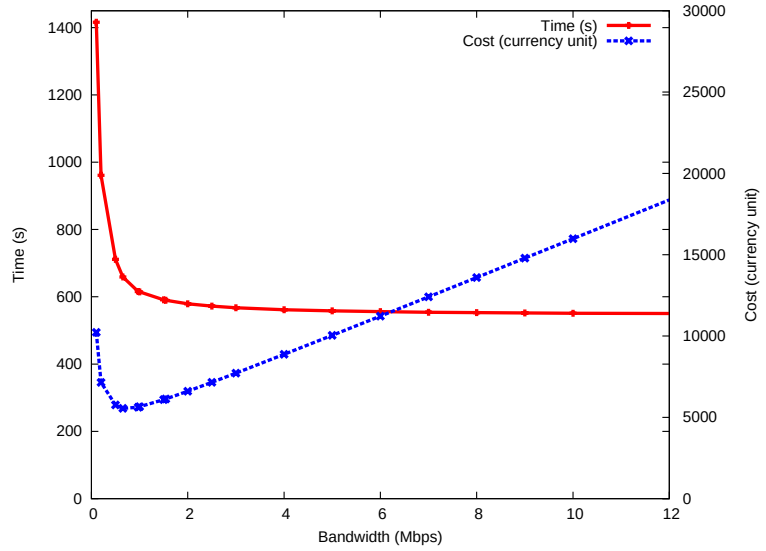


Figure 4.6: Estimation of the execution time and total cost with regard to the bandwidth of the *FIFO* strategy.

is similarly allocated proportionally to the amount of data to transferred between each pair of modules, or the same bandwidth is reserved for all links in the infrastructure. This strategy serves as a performance baseline for comparisons.

4.3.3.2 FIFO strategy

In this approach, we assume that the resources are indistinguishable, and the workflow scheduler can request any module to be executed on any available virtual resource. A *FIFO* scheduling strategy is optimal in the sense that VI redeployment is unnecessary ($T = T_1$). In addition, the same bandwidth is reserved for all links in the infrastructure ($b_1 = b_2 = ... = b_k$).

Figure 4.6 displays the estimated execution time and the total cost of the workflow from Figure 4.2 with regard to the bandwidth (for $n = 32$ input data items and unit costs $c_r = c_b = 0.2$). When the bandwidth is small, the total cost is high due to the increased data transfer time, but as the bandwidth increases, both the execution time and cost decrease. However, after a 2.0 Mbps threshold, the execution time slightly reduces and the bandwidth allocation cost increases. The optimization method used to approximate numerically the optimal bandwidth leads to 0.6517 Mbps.

4.3.3.3 Optimized strategy

The *FIFO* strategy can only be applied with homogeneous resources without optimizing the bandwidth between each resource pair. The optimized strategy described here considers dividing the workflow execution into multiple stages and allocating resources and bandwidth independently for each stage. A cost minimization algorithm is executed for identifying the optimal number of virtual resources to be allocated to the modules involved in the stage.

Moreover, the algorithm should decide on the number of stages and when a VI reconfiguration should happen. Initially, it transforms the workflow into a Directed Acyclic

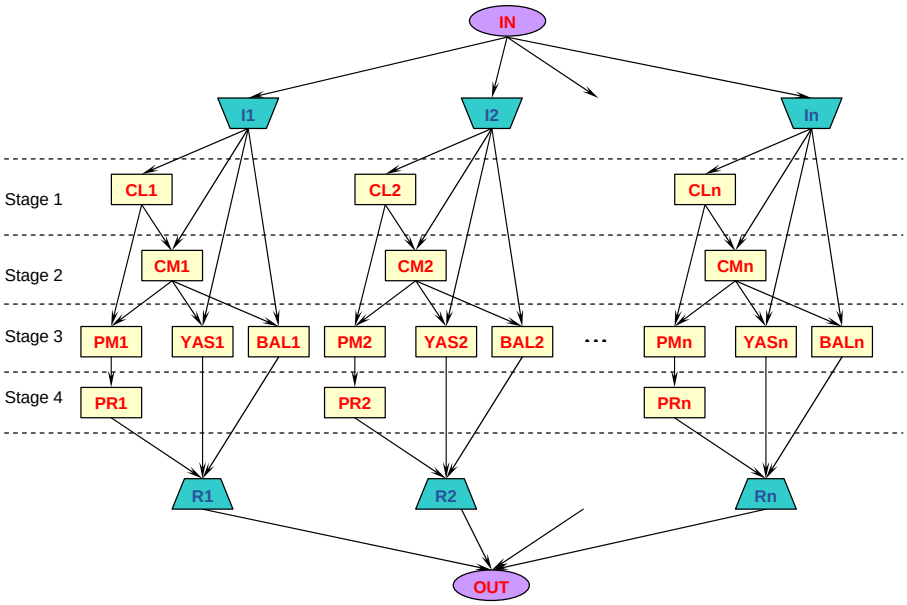


Figure 4.7: DAG jobs of Bronze Standard application for n inputs.

Graph (DAG) using the approach presented in [Zhao and Sakellariou, 2006]. Secondly, the DAG is divided in execution stages, as shown in the example in Figure 4.7 for the workflow of Figure 4.2. *IN* and *OUT* are special entries and exit nodes that are not considered in estimating the execution and data transfer times. An execution stage is defined as the set of invocations that have the same depth in the DAG graph.

Note that the DAG generation is only possible for workflows without unbounded loops (i.e., the exact number of invocations of each module must be known in advance), so that the workflow planer can determine a complete execution schedule. Workflows including *while* loops, or *foreach* constructs iterating over data structures of unknown size are unresolvable prior to execution. Although DAG representation limits the class of applications that can be planed, it represents a broad category of workflow applications in e-Science [Deelman et al., 2003]. A solution for dealing with workflows with unresolvable constructs is to divide them into smaller resolvable sub-workflows. This strategy was implemented, for example, in the workflow manager of the DIET middleware (MA DAG) to deal with workflows that could not be represented by DAGs [MA].

At each execution stage identified, the infrastructure is reconfigured for only deploying the specific modules involved in that stage. The resources are allocated proportionally to the number of invocations needed for each module. In a typical data intensive application execution, there are more data items to process (n) than resources available ($m_{\max}$). For instance, in the case of a stage i with only one module j (e.g., stage 1, 2 or 4 in Figure 4.7), $m_{\max}$ data items are processed concurrently by j and the process is repeated $n/m_{\max}$ times, leading to the execution time:

$$T_i = \left\lceil \frac{n}{m_{\max}} \right\rceil \times T_j \tag{4.1}$$

where T_i is the execution time for j.


```
...
<virtualTimeline id="Application timeline">
  <timeline id="T1">
    <activate>Service_i</activate>
    <totalTime>PTtiS</totalTime>
  </timeline>
  <timeline id="T2">
    <after>T1</after>
    <activate>Service_j1</activate>
    <activate>Service_j2</activate>
    <activate>Service_j3</activate>
    <totalTime>PTtjS</totalTime>
  </timeline>
</virtualTimeline>
...
```

Figure 4.8: Timeline description for the *optimized* strategy.

The next step is to introduce the bandwidth requirements as well as the IT resource configurations. Let $inv_j, j = 1..s$ be the number of invocations of module j at stage i where s is the number of modules being executed at this stage. Let vector $m = (m_1, m_2, ..., m_s)$ be a combination of numbers of resources allocated to the module j, satisfying the condition $\sum_{j=1}^{s} m_j \leq m_{\max}$. The resulting optimal execution time to complete inv_j invocations of module j is:

$$T_j = \left\lceil \frac{inv_j}{m_j} \right\rceil \times T_{uj} \tag{4.2}$$

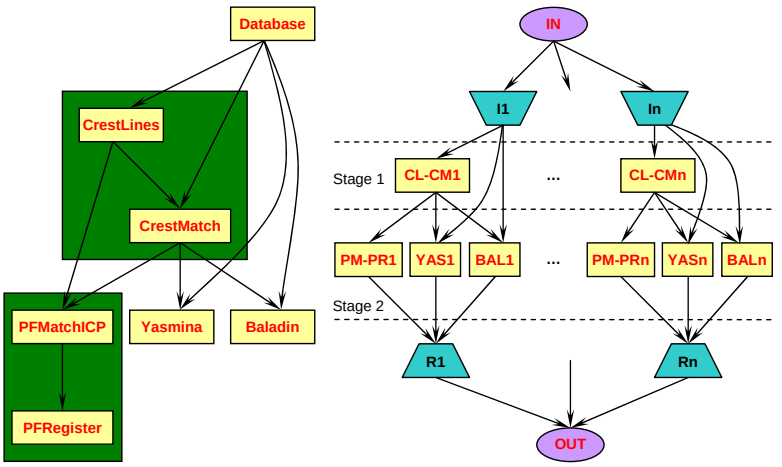
where T_{uj} is the unit execution time of module j.

The *optimized* strategy composes an internal *virtual timeline* of a VI. Figure 4.8 presents an application example with two stages described in VXDL. The first stage has a module that executes in t_i seconds, and the second stage has three modules starting together after the first stage completes.

4.3.3.4 Modules grouping optimization

The total execution cost also depends on the time at which each stage is deployed. An optimization of the total resource reservation and redeployment time was designed, extending the job grouping strategy without loss of parallelism using a technique introduced in [Glatard et al., 2008]. This strategy minimizes the application makespan by grouping modules that would have been executed sequentially, thus reducing data transfers and the number of job invocations needed. Applying this strategy to the workflow of Figure 4.2, two module groups are identified, which do not cause loss of parallelism as shown in Figure 4.9a. The number of execution stages can also be reduced as shown in Figure 4.9b.

Starting from the execution DAG split into stages, module invocation groups are evaluated for each consecutive pair of stages. For each module a of the workflow involved in the stage i, let $a_0, a_1, ..., a_j$ be all children from a in stage $i+1$. All possible combinations of grouping a with one or more of the a_k modules is tested and the resulting execution cost is evaluated by optimizing the number of resources and the bandwidth allocated. Figure 4.10 shows the best solution for the workflow example presented in Figure 4.2.



(a) Grouping without parallelism loss (b) Resulting execution DAG considering n input data items

Figure 4.9: Modules grouping without parallelism loss.

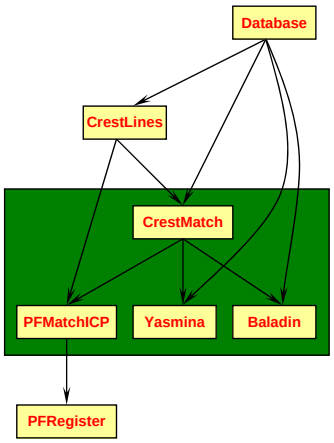


Figure 4.10: Grouping CrestMatch, PFMatchICP, Yasmina and Baladin modules.

4.4 Executing workflow-based applications

The experiments presented here assess the performance of a workflow-based application atop several VIs allocated over a distributed and virtualized physical substrate. The main goal is to show the impact and importance of well placing the *permanent* and *variable* parts of a workflow, in terms of resources proximity. A set of VIs were composed based on the information available on the workflows and benchmarks, which represents the users' knowledge about the application. The VIs were allocated and provisioned considering different approaches for placing the virtual resources.

4.4.1 Medical application example

We selected a complex, real-scale medical-image analysis application, known as *Bronze Standard*, to illustrate the translation of a workflow into a VXDL description. The Bronze Standard technique addresses the difficult problem of validating procedures for medical-image analysis [Glatard et al., 2006], where there is usually no reference, or *gold standard*, to assess objectively the quality of computational results. The statistical analysis of images enables the quantitative measurement of computational errors, thereby the bronze standard technique quantifies the maximal error resulting from widely used *image registration algorithms*. The larger the sample image database and the number of registration algorithms to compare with, the most accurate the method. This very-scalable procedure requires the composition of a complex application workflow including different registration-computation modules with data transfer inter dependencies.

Modules	Time (average ± standard deviation)	Input data	Produced data
CrestLines	31.06s ± 0.57	15 MB	10 MB
CrestMatch	3.22s ± 0.51	25 MB	4 MB
PFMatchICP	10.14s ± 2.41	10.2 MB	240 kB
PFRegister	0.64s ± 0.22	240 kB	160 kB
Yasmina	52.94s ± 12.96	15.2 MB	4 MB
Baladin	226.18s ± 19.36	15.2 MB	4 MB

Table 4.2: Benchmarks of the Bronze Standard's modules: execution time, data input and produced data volumes.

The estimated performance of Bronze Standard depends on the size of the image database. In the experiments we use a clinical database of 32 pairs of patient images to be registered by the different algorithms in the workflow. For each run, the processing of the complete image database results in the generation of approximately 200 computing tasks. As illustrated in Figure 4.2, the workflow of this application has a completely deterministic pattern, all modules of the application have the same number of invocations. The execution time and the data volume transferred by each module have been measured in initial micro-benchmarks reported in Table 4.2.

Bronze Standard's workflow is enacted with the MOTEUR workflow manager [Glatard et al., 2008]. MOTEUR submits the workflow tasks to the VI infrastructure through the DIET middleware [Caron and Desprez, 2006], a scalable Grid scheduler based on a hierarchy of agents communicating through CORBA [COR].


```
...
<vNode id="Moteur">
  <memory>
    <interval>
      <min>4</min>
    </interval>
    <unit>GB</unit>
  </memory>
</vNode>
<vGroup id="Cluster_Baladin" multiplicity="22">
  <vNode id="Node_Cluster_Baladin">
    <memory>
      <interval>
        <min>512</min>
      </interval>
      <unit>MB</unit>
    </memory>
  </vNode>
</vGroup>
...
```

Figure 4.11: Description of a vNode and a computing cluster required to execute the Bronze Standard application.

To exemplify the translation of Bronze Standard's workflow into a VXDL file, let us consider a request for a VI of 35 nodes. Three nodes will be dedicated to the *permanent* part: 1 node for MOTEUR, 1 node for the middleware server and 1 node for the database server. The 32 nodes left are distributed and allocated proportionally to the execution time of the workflow modules using, for simplicity, the *Naive strategy* (discussed in Section 4.3.3.1); 3 nodes are used for CrestLines, 1 node for CrestMatch, 1 node for PFMatchIP, 1 node for PFRegister, 22 nodes for Baladin, and 4 nodes for Yasmina. Figure 4.11 presents the specification of a virtual node (MOTEUR) and a computing cluster (Baladin). Several variants of VI descriptions with different network topologies can be expressed for the same computing-resources set. We give examples developing two different VI compositions.

4.4.2 VI COMPOSITION

Figure 4.12 illustrates the graph of a VI composition to execute the Bronze Standard's workflow, where the vertices represent the virtual resources and virtual clusters required to execute the application, and the edges are the network links required to enable the communication between the workflow's modules. The *permanent* part is differentiated from other components (i.e., MOTEUR, middleware and database nodes) including those that are critical in terms of computing and data transfer (i.e., CrestMatch, Yasmina, Baladin and PFMatchICP).

Based on this composition, we developed two descriptions for this scenario varying the network topology and requirements:

- **VI 1**. The network is composed of two types of links: one with low intra-cluster latency, and another with a maximum latency of 10 ms for interconnecting the clusters;
- **VI 2**. The network comprises three virtual links: i) one with low intra-cluster latency (maximum latency of 0.200 ms); ii) one with latency of 10 ms interconnecting the components, except the links with iii) a maximum latency of 0.200 ms between *CrestMatch* (dark blue) and clusters *PFMatchICP*, *Yasmina* and *Baladin*

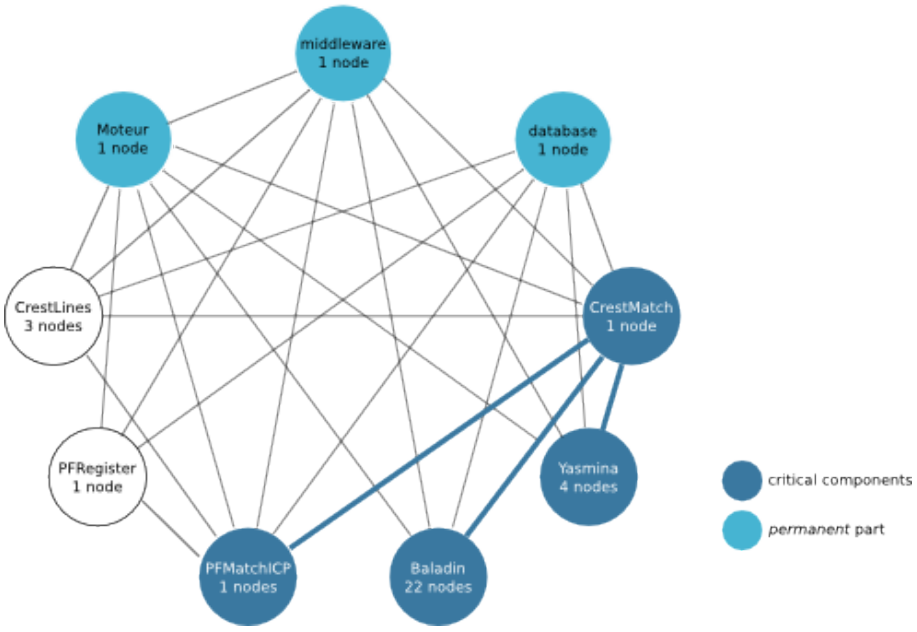


Figure 4.12: A VI composition for executing the Bronze Standard's workflow.

(presented in dark blue). The communication-intensive links of **VI 2** are presented in Figure 4.13.

To illustrate how each description can be allocated on a physical substrate, we propose two solutions for each VI, which corresponds to a total of four different allocations. For the sake of illustration, these allocations were arbitrarily performed, not based on any particular algorithm (a detailed discussion is available in Chapter 5). In this example, *Location 1* and *Location 2* represent two geographically distributed resource sets. We use the notations r_v and r_p to represent virtual and physical resources, respectively.

Figure 4.14 summarizes the allocation described below:

- **VI 1 - Allocation I**: the intra-cluster link specification motivates the allocation of loosely connected resources with one virtual machine per physical node.
- **VI 1 - Allocation II**: the virtual resources of clusters *CrestMatch*, *PFRegister*, *Yasmina*, and *Baladin* share physical nodes. Each physical node allocates 2 virtual machines.
- **VI 2 - Allocation III**: due to the required interconnection capacity (maximum latency) all virtual resources must be allocated at the same location (for example, in the same site of a Grid, or a specific geographical location that respects the maximum latency defined). This allocation explores the allocation of 1 virtual machine per physical node.
- **VI 2 - Allocation IV** explores the same physical components of *Allocation III* but with 2 virtual machines per physical node for the clusters *CrestMatch*, *PFRegister*, *Yasmina*, and *Baladin*.


```
...
<vLink id="Communication Intensive 1">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Cluster_CrestMatch</source>
  <destination>Cluster_Baladin</destination>
</vLink>
<vLink id="Communication Intensive 2">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Cluster_Yasmina</source>
  <destination>Cluster_PFMatchICP</destination>
</vLink>
<vLink id="Communication Intensive 3">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Cluster_CrestMatch</source>
  <destination>Cluster_PFMatchICP</destination>
</vLink>
<vLink id="Communication Intensive 4">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>database</source>
  <destination>Cluster_PFMatch</destination>
</vLink>
...
```

Figure 4.13: VXDL description of the communication-intensive virtual links of Figure 4.12.

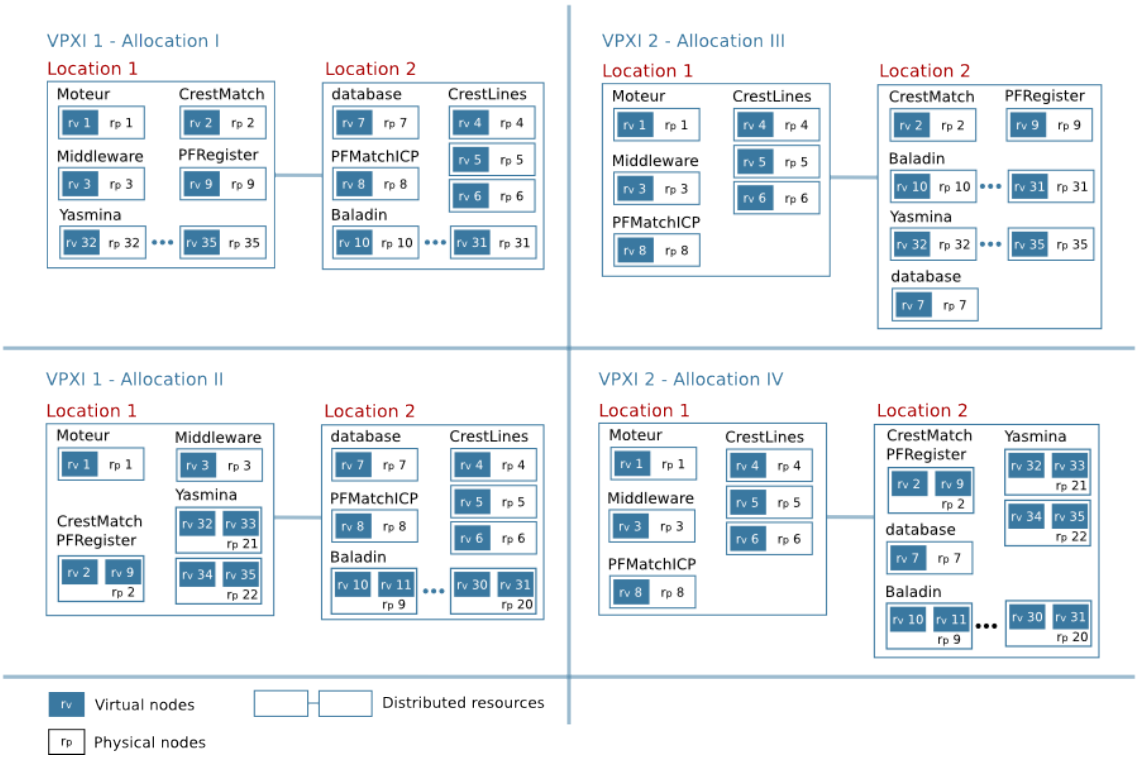


Figure 4.14: Allocations of descriptions *VI-1* and *VI-2*.

4.4.3 Application execution

The experimental testbed for executing the Bronze Standard application used physical resources from the Grid'5000 testbed [Cappello et al., 2005]. Grid'5000 is an experimental testbed distributed across 9 sites in France, for research in large-scale parallel and distributed systems [GRI]. The sites are interconnected by a backbone infrastructure comprising 10 Gbps dedicated lambda paths provided by the French National Telecommunication Network for Technology (RENATER) [REN] as depicted in Figure 4.15. The reserved physical infrastructure was composed of the clusters *capricorne* (located in *Lyon*), *bordemer* (located in *Bordeaux*), and *azur* (located in *Sophia*), whose nodes contain CPUs of 2.0 GHz dual-core Opterons.

The resources were virtualized by the HIPerNet framework [1], responsible for the provisioning and management of all virtual infrastructures composed to execute the application. The VIs used a system image containing the operating system based on a standard Linux distribution Debian *Etch* with a kernel version *2.6.18-8* for *AMD64*. Each VI is composed of 35 nodes divided in *permanent* and *variable* parts: 3 nodes are dedicated to the *permanent* part (MOTEUR, DIET, and storage server) using 1 CPU per node, and the remaining 32 nodes to the *variable* part. The allocation of *variable* resources was performed in accordance with the following VI descriptions: *VI 1 - Allocation I* and *VI 2 - Allocation III* used 1 CPU per node while *VI 1 - Allocation II* and *VI 2 - Allocation IV* used 1 CPU core per node. For each each experiment, we executed the application 10 times and obtained the averages and standard deviations of the application makespan,

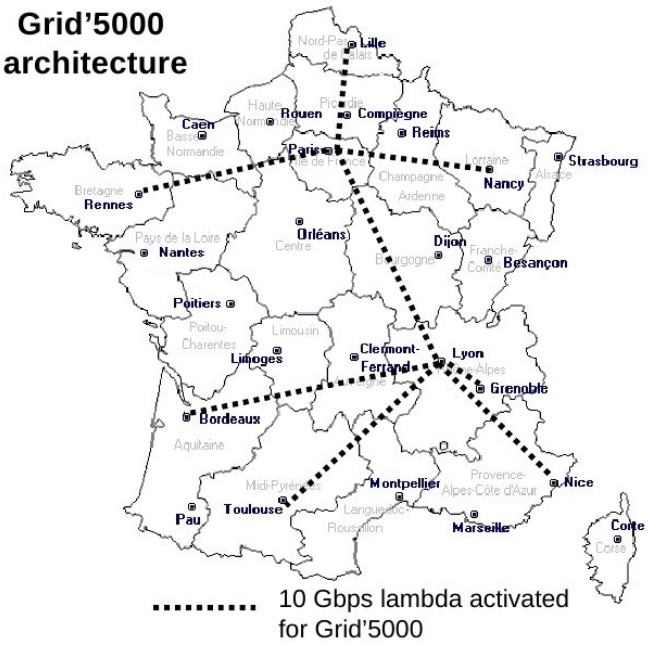


Figure 4.15: Architecture of Grid'5000: 9 sites interconnected by 10 Gbps dedicated lambda paths (figure courtesy of the Grid'5000 community [GRI]).

the data transfer and the task execution time.

4.4.3.1 Placing resources on one location

This experiment analyzes the application execution on VI's components in the same geographical location. The VI locations follow their specification: *VI 2 - Allocation III* and *VI 2 - Allocation IV*.

The application's makespans on *VI 2 - Allocation III* and *VI 2 - Allocation IV* are 11min 44s (±49s) and 12min 3s (±50s), respectively. This corresponds to a +3.8% makespan increase due to the execution overhead when two virtual machines are collocated on the same physical resource. In addition, we present in the Table 4.3 the average execution time (in seconds) of the application modules on *VI 2 - Allocations III and IV*. The average execution overhead is 5.17% (10.53% in the worst case and 1.28% in the best case).

4.4.3.2 Resources distributed across 2 locations

In this experiment the VI's components were provisioned across 2 locations. We compare the application execution of a VI locally provisioned (*VI 2 - Allocation IV*) with that of a VI whose resources are distributed (*VI 1 - Allocation II*). The resources' locations follow the map solution proposed by *VI 1 - Allocation II* and *VI 2 - Allocation IV*.

When porting the VIs from a local provisioning to a large scale distributed provisioning, data transfers increase. Table 4.4 presents the data transfer time (in seconds) of the application modules on *VI 2 - Allocation IV* (local) and *VI 1 - Allocation II* (distributed across 2 locations). The overhead is 150% in the worst case. Conversely, some local

Modules	VI 2 - Allocation III	VI 2 - Allocation IV	variation
CrestLines	34.12 ± 0.34	36.84 ± 5.78	+7.97%
CrestMatch	3.61 ± 0.48	3.99 ± 0.63	+10.53%
PFMatchICP	11.93 ± 2.76	12.75 ± 5.35	+6.87%
PFRegister	0.78 ± 0.18	0.79 ± 0.18	+1.28%
Yasmina	59.72 ± 14.08	61.53 ± 13.98	+3.03%
Baladin	244.68 ± 16.68	247.99 ± 19.51	+1.35%

Table 4.3: Average execution time (in seconds) on *VI 2 - Allocation III* and *VI 2 - Allocation IV*.

transfers may be slightly reduced. However, in this case this overhead has little impact on the application makespan since it is compensated for by the parallel data transfer and computations introduced by MOTEUR [Glatard et al., 2008].

Indeed, the makespans of VIs distributed across 2 locations are 12min (±12s) and 12min 11s (±20s) on *VI 1 - Allocation I* and *VI 1 - Allocation II*, respectively, very similar to the performance of the local *VI 2 - Allocation IV*.

Modules	VI 2 - Allocation IV	VI 1 - Allocation II	variation
CrestLines	2 ± 0.45	3.01 ± 1.6	+50.5%
CrestMatch	1.99 ± 0.34	1.83 ± 0.36	-8.04%
PFMatchICP	1.3 ± 0.4	3.25 ± 0.13	+150%
PFRegister	0.51 ± 0.23	0.43 ± 0.09	-15.69%
Yasmina	1.19 ± 0.27	1.16 ± 0.21	-2.52%
Baladin	1.17 ± 0.38	1.81 ± 1.03	+54.7%

Table 4.4: Data transfer time (in seconds) on the local *VI 2 - Allocation IV* and large scale distributed *VI 1 - Allocation II*.

4.4.3.3 Resources distributed across 3 locations

In the final experiment of this section, the VIs were provisioned over 3 locations. The *permanent* part of allocation solution *VI 1 - Allocation II* was placed in a single location (*Lyon*, in this case) whereas the *variable* part was randomly distributed among *Lyon*, *Bordeaux*, and *Sophia* locations.

As expected, further distributing computational resources causes an additional increase in the data-transfer overheads among the application's modules, with an execution makespan of 12min 13s (± 30s) and a data-transfer overhead of 176% in the worst case.

4.5 Analysis of strategies for composing VIs

The second set of experiments, also using Bronze Standard, investigates the different strategies for composing VIs. Different VIs were elaborated following the proposed strategies and the information extracted from Bronze Standard. In the following subsections we describe the testbed and the main obtained results.

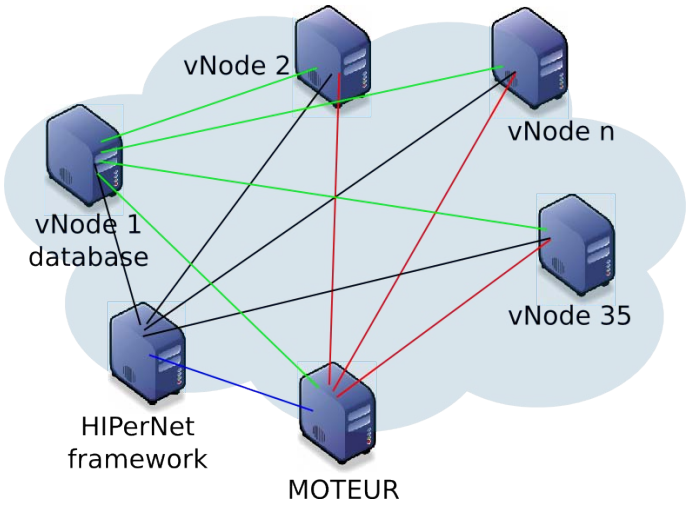


Figure 4.16: Experimental infrastructure used to compare the different strategies to compose VIs.

4.5.1 Testbed composition

The testbed also uses physical resources from Grid'5000, but this time only resources located in *Lyon* site were reserved. The physical resources are *Sun Fire V20z* machines with 2.4 GHz CPUs, 2 cores, 2 GB RAM, interconnected through 1 Gbps Ethernet. Once again, the VIs were allocated and managed by the HIPerNet framework.

The experimental infrastructure is depicted in Figure 4.16. For the experiments, 35 vNodes were deployed, but this time, the MOTEUR workflow engine acted as a client of the HIPerNet framework, being hosted on a dedicated physical host, outside the VI. The HIPerNet engine deploys and manages virtual machines on-demand on the computers (dark connections), either with an operating-system image of the input database server or with the application modules. Each physical computer hosts a single virtual machine. MOTEUR produces VXDL descriptions that are submitted to the HIPerNet engine (blue connection). After receiving the information about all components allocated to the VI, MOTEUR connects to the computing nodes to invoke the application modules (red connections). The computing nodes connect to the database host to copy the input data and send the computational results. The final results are sent to MOTEUR (green connections).

For the needs of the MOTEUR planner which implements the strategies described in this Chapter, all modules involved in the Bronze Standard's workflow have been benchmarked for execution time and amount of data transferred, as reported in Table 4.2. For each experiment, the application was executed 5 times and the makespan was averaged to minimize the execution time variations encountered in distributed computing. The standard deviation is also reported.

For each strategy, the planner optimizer of MOTEUR was executed to determine the configuration with the minimal execution cost. The number of virtual machines allocated to the application and the bandwidth among the database node and computing nodes is specified by corresponding VXDL documents.

4.5.2 Cost model

The execution of a workflow-based application can occur in several *stages*. At each stage, the VI can be reallocated respecting a specific configuration, to perform the execution of part of the workflow. The VI reconfiguration between different stages, which may involve redeploying resources, is time-consuming. An extreme condition is to create a static VI for the whole workflow execution, thus sparing the redeployment cost. Another extreme is to allocate new resources, one by one, on demand.

The cost model presented in this section makes a fine-grained estimate of the resources consumed by each application run [2] [Truong Huu and Montagnat, 2010]. Note that the model is applicable to estimate the cost of a single run of an application on the infrastructure. It does not take into account other costs, such as the long term storage of data onto Cloud resources. Should users need data storage before and/or after execution, they would be charged additionally and independently of the cost calculated below.

The cost was formulated following the notation presented in Table 4.1. Let $m_{\max}$ be the maximum number of computing nodes available on the infrastructure and s be the number of execution stages of the application. The vector $m = (m_1, m_2, ..., m_s)$ is the number of nodes used at each execution stage with $\forall i, m_i \leq m_{\max}$. Let c_r be the per-second cost of a computing resource. The total computing cost of the infrastructure allocated for the application is:

$$C_r = c_r \times \sum_{i=1}^{s} m_i \times (Td_i + T_i(m_i, n, b)) \tag{4.3}$$

where Td_i is the deployment time (including resource reservation and initialization time) and $T_i(m_i, n, b)$ is the execution time at stage i. T_i depends both on computing time and on data transfer time involved within stage i. It is parameterized by the number of resources reserved (m_i), the number of input data items to process (n) and the bandwidth ($b = (b_1, b_2, ..., b_{ki}), i \in [1..s]$) of the network links used for data transfer. The computation of T_i is possible using the application logic described through the workflow. The workflow engine used in our experiment, MOTEUR [Glatard et al., 2008], was seminally designed to produce an execution schedule and control the distribution of an application at runtime. It was enriched with a resource allocation and scheduling planner used to estimate T_i, given that information on the workflow modules execution time and transferred data is available.

The total infrastructure cost is also impacted by the data transfer time. If the per-Mbps cost of the reserved bandwidth is c_b, then the total data transfer cost is:

$$C_b = c_b \times \sum_{i=1}^{s} (Td_i + T_i(m_i, n, b)) \sum_{j=1}^{k_i} b_j \tag{4.4}$$

This cost applies to a VI where the amount of allocated bandwidth is controlled. It sums all data transfer costs involved in the workflow execution, including workflow input data transferred from outside the Cloud (at stage 1), the temporary data generated during workflow execution (at all stages) and the output data transferred to external resources (at stage s). From equations 4.3 and 4.4, the total infrastructure cost to execute the applications can be computed as:

$$C = C_r + C_b \tag{4.5}$$

This cost has to be optimized considering a maximum admissible cost and the application performance scalability. A trade-off has to be found between the number of computing and network resources allocated (which impacts T_i) and the resulting cost.

4.5.3 Comparing strategies

The Bronze Standard application was executed over VIs composed with the different discussed strategies. We compare the *single-stage strategy* and *multi-stage strategies* to identify the application behavior when executing in an elastic VI.

4.5.3.1 Single-stage strategies

The *naive* and *FIFO* strategies are single-stage. They use the maximum available computing resources with an optimal bandwidth yielding a minimal execution cost. The virtual infrastructures of the *naive* and *FIFO* strategies are represented in Figure 4.17 and 4.18, respectively. The *naive* allocation strategy allocated the 34 computing nodes to application modules as follows: 3 nodes for *CrestLines*, 1 node for *CrestMatch*, 1 node for *PFMatchICP*, 1 node for *PFRegister*, 5 nodes for *Yasmina*, and 23 nodes for *Baladin*. The same bandwidth, 2.69Mbps, is required to interconnect all computing nodes. The application makespan is 67.08min ± 0.10min.

This experiment shows that the virtual resources are not well exploited during the execution. Figure 4.19 shows a schedule of this strategy. Each colored line represents one task duration: it starts once the corresponding task has been submitted and stops at the end of its execution. The first, brighter part of a line represents the task's waiting time spent from submission until a resource becomes available for execution. Colors are arbitrary and just help distinguish the different tasks. As can be seen, at the beginning of the execution, only three nodes are used to execute the *CrestLines* module. Other resources are wasted. Similarly, the result of *CrestMatch* is needed by three modules: *PFMatchICP*, *Yasmina* and *Baladin*, but there is only one resource allocated to this module according to this strategy, thereby becoming a bottleneck.

The makespan of the *FIFO* strategy is lower: 46.88min ± 0.78min with the optimal bandwidth (1.16Mbps). The standard deviation of this strategy is higher due to the variable arriving order of the tasks. Some long tasks are executed on the same computing resource, leading to an increase in the application makespan. Figure 4.20 shows a typical task schedule for this strategy.

4.5.3.2 Multi-stage strategies

The *optimized* strategies are multi-staged, optimized bandwidth required, and may allocate less resources than the maximum available when doing so leads to no gain. The planer determines the number of virtual resources and the bandwidth yielding a minimal execution cost.

Within the multi-stage optimization, virtual resources from stage n can be reused in stage $n + 1$. If stage $n + 1$ uses more virtual machines than stage n, additional virtual machines are deployed during the execution of stage n.

Without module grouping there are 4 execution stages, which are represented in Figure 4.21. According to the optimization results, only 30 nodes were allocated to the first,

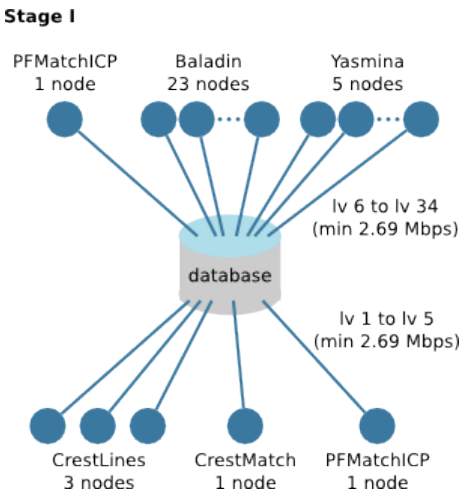


Figure 4.17: Virtual Infrastructure composition considering the *naive* strategy.

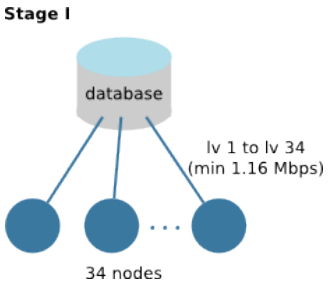


Figure 4.18: Virtual Infrastructure composition considering the *FIFO* strategy.

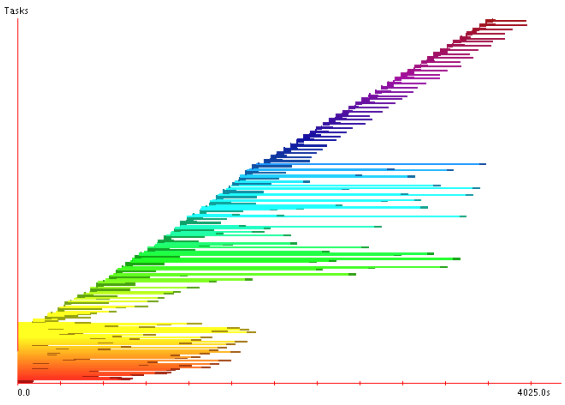


Figure 4.19: Task schedule with the *naive* strategy.

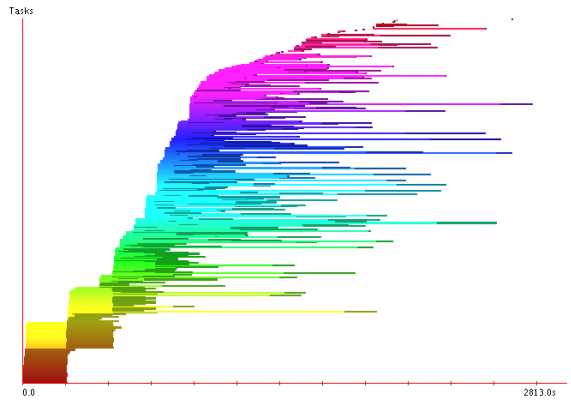


Figure 4.20: Task schedule with the *FIFO* strategy.

second and fourth stages (additional resources would be wasted). The required bandwidths are 4.62 Mbps, 14.74 Mbps and 3.87 Mbps, respectively. For the third stage, 4 nodes were allocated to *PFMatchICP*, 6 nodes for *Yasmina* and 20 nodes for *Baladin*. The bandwidth for each module in this stage is 0.87 Mbps, 1.36 Mbps and 1.29 Mbps, respectively. The corresponding application makespan is 37.05min ± 0.25min.

Further grouping the application modules as shown in Figure 4.10, the application is divided into three stages only, using 30 nodes each. As presented in Figure 4.22, the bandwidth allocated to each stage is 4.90 Mbps, 1.95 Mbps and 3.87 Mbps, respectively. The application makespan is then 22.93min ± 0.35min. Besides the execution time improvement, the number of resources consumed is lowered. As we can observe in Figure 4.23, not all tasks of the same stage finish exactly at the same time though, due to variations in execution time of the image analysis tools, which depends on the exact processed image content. This has an impact as the tasks of stage n have to wait for the longest task of stage $n-1$ before the system can be reconfigured.

4.5.3.3 Analysis

For complementing the analysis, we also measured the deployment time of the virtual infrastructure before running the application and the reconfiguration time between stages of the *optimized* strategies. The reconfiguration time takes into account bandwidth reconfiguration between the database host and computing nodes allocated to application modules in each stage.

Table 4.5 compares the performance of the strategies presented above and the associated platform cost computed using Equation 4.5. The worst case is the *naive* strategy which uses the maximum number of resources for a very large makespan and a long deployment. The *FIFO* strategy spends the same time to deploy the infrastructure, but it has a better makespan than the *naive* strategy. The *naive* and *FIFO* strategies reconfiguration time is null since they are single-stage. The *optimized* strategy without module grouping has better results than the *naive* and *FIFO* strategies, both in terms of application makespan and number of resources consumed, although it has to spend time to reconfigure the infrastructure after each stage. The best case is obtained for the *optimized* strategy with module grouping. It uses less resources, spends less time to reconfigure the infrastructure and returns the results faster. In terms of deployment time, the *naive* and

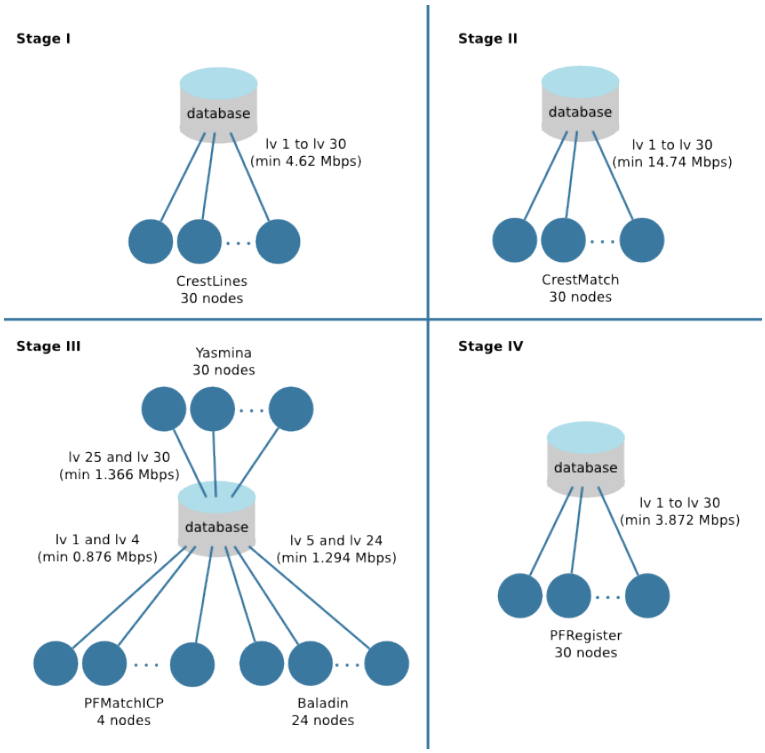


Figure 4.21: Virtual Infrastructure composition considering the *optimized* strategy without grouping modules.

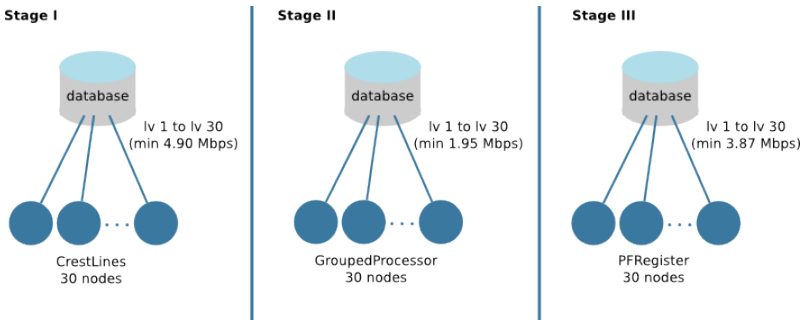


Figure 4.22: Virtual Infrastructure composition considering the *optimized* strategy with grouping modules.

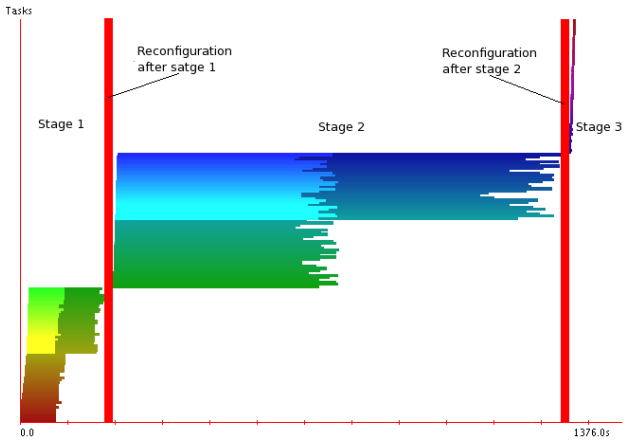


Figure 4.23: Task schedule with *optimized* modules grouping.

FIFO strategies take 29.83 min to deploy a VI composed of 35 virtual resources. This duration corresponds to the time needed to copy the operating-system images (319 MB) from the image repository to the virtual machines and start them sequentially. The *optimized* strategies use only 31 machines, reducing the deployment time to 25.68 min. It is expected that a parallel deployment decreases this redeployment overhead. As expected, the cost estimated is lowered for higher performing strategies toward the reduction of the application makespan and of the network bandwidth consumed.

Strategy	Makespan	#VM	Deployment time	Reconfiguration time	Execution cost ($\times 10^5$)
Naive	67.08min ± 0.10	35	29.83min	0	$1.40 \times c_r + 3.68 \times c_b$
FIFO	46.88min ± 0.78	35	29.83min	0	$0.98 \times c_r + 1.10 \times c_b$
Optimized (without grouping)	37.05min ± 0.25	31	25.68min	79.29s	$0.69 \times c_r + 0.98 \times c_b$
Optimized (with grouping)	22.93min ± 0.58	31	25.68min	52.86s	$0.42 \times c_r + 0.48 \times c_b$

Table 4.5: Performance comparison among the four strategies.

4.5.4 Impact of bandwidth control on application cost and performance

Finally, we complemented the set of experiments by evaluating the application behavior when executing following the bandwidth reservation and control mechanisms offered by the HIPerNet framework. For that, the application was executed using the optimized strategy with module grouping under two additional network bandwidth configurations: lower and higher bandwidth values than the optimal found were tested (i.e., 1 Mbps and 10 Mbps respectively).

Table 4.6 displays, for each configuration, the data transfer time in each stage (in seconds), the application makespan (in minutes), and the corresponding cost. Comparing the results with the optimized bandwidth allocation, it appears that the makespan increases as expected when using a low bandwidth. However, the cost increases as well because the cost gain on network bandwidth is compensated by the loss in the computing nodes'

reservation time. With the high bandwidth, the application makespan can be reduced (-22.72% in this case) at a higher cost (+102% computed with $c_r = c_b = 0.10$).

Bandwidth	Stage 1 (seconds)	Stage 2 (seconds)	Stage 3 (seconds)	Makespan (minutes)	Execution cost ($\times 10^5$)
Low (1 Mbps)	222.59 ± 2.51	316.57 ± 40.37	2.91 ± 0.50	34.78 ± 0.67	$0.65 \times c_r + 0.31 \times c_b$
Optimized	53.8 ± 4.56	171.72 ± 24.66	1.53 ± 0.23	22.93 ± 0.58	$0.42 \times c_r + 0.48 \times c_b$
High (10 Mbps)	30.79 ± 3.85	42.68 ± 9.55	1.09 ± 0.18	17.72 ± 0.23	$0.33 \times c_r + 1.61 \times c_b$

Table 4.6: Evaluation of the bandwidth control mechanism.

4.6 Conclusions

This chapter addressed the difficult problem of composing an optimal VI to execute a distributed application. Applications have different requirements in terms of computing and communication that can vary during the execution time. In a *pay-as-you-go* scenario, such as Cloud Computing, an optimal VI must consider the trade-off between allocation cost and application performance. We proposed and evaluated strategies to determine optimal VIs specification for executing workflow-based distributed applications on VIs, provisioned following the pay-as-you-go model.

We identified that applications can explore the dynamism provided by the elasticity of VIs. By decomposing a VI in well-defined stages, *application performance can be improved while costs can be reduced,* as shown by results that assess the performance of the *optimized* strategy with a job grouping optimization. In addition, the advanced network bandwidth reservation and control capabilities offered by management frameworks (such as HIPerNet) can be exploited to improve the applications' performance.

VXDL has acted as a key *facilitator* for describing VIs, since the attributes offered by this language enabled the specification of the required VI configuration in terms of computation, communication, and usage intervals. However, to guarantee the user's expectations, an efficient mechanism is highly required for translating and allocating a set of VIs onto a distributed physical substrate. The allocation mechanism should interprets and exploits all attributes of VXDL, and in addition, considers the user's expectations during the formulation of the allocation problem.

An open research line that we will investigate is the elaboration of translation mechanism for legacy applications, represented and characterized with other information (e.g., MapReduce [Dean and Ghemawat, 2008], web servers).

Five

Allocating resources to Virtual Infrastructures

The work presented in this chapter has been accepted for publication at the 12th IEEE/IFIP International Symposium on Integrated Network Management - Special Track on Management of Cloud Services and Infrastructures (IM 2011 - STMCSI) [4]. In addition, it is part of the solutions patented by the LYaTiss company (http://www.lyatiss.com) and the Institut National de Recherche en Informatique et en Automatique (INRIA, http://www.inria.fr/) [VXAlloc] [VXCap].

5.1 Introduction

As illustrated in Figure 5.1, once the needs of an application are identified and a VI for it is specified, the resulting VXDL file carrying the specification and substantial information on the user's requirements must be interpreted, allocated and provisioned over the distributed substrate. As a key step in the dynamic provisioning process, the allocation of a VI atop a physical substrate must reconcile the users' expectations with the Infrastructure Providers (InP) objectives. While users can compose VXDL descriptions, considering their requirements in terms of application performance and reservation cost, InPs, on the other hand, usually aim to minimize resource usage and substrate cost, or to maximize the acceptance ratio of new requests.

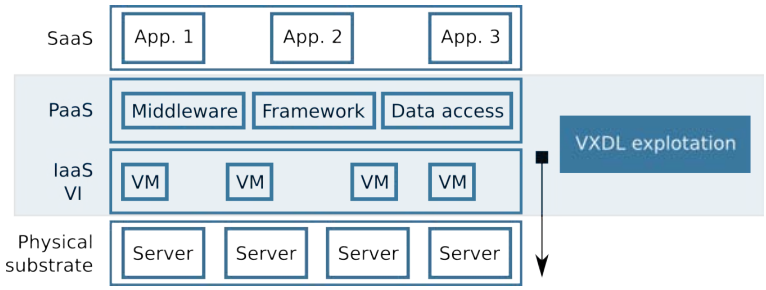


Figure 5.1: Positioning of a mechanism for allocating VIs specified with VXDL, considering the service models offered by Cloud Computing providers.

Due to combinatorial explosion, the computational time to find a feasible solution depends exponentially on the size of the search space considered by the allocation algorithm. The optimal allocation of a virtual infrastructure with constraints on physical nodes and links is an *NP-hard* problem (it cannot be solved in polynomial time) [Chowdhury and Boutaba, 2009].

Moreover, the complete abstraction of physical network and IT resources, combined with the geographical independence of VIs, brings together new challenges to the allocation process, since a VI can be allocated anywhere on top of a distributed substrate. The allocation of geographically fragmented virtual infrastructures raises the following concerns for users and InPs: i) the network-communication latency between distributed virtual resources increases, which can augment the application's runtime [5]; and ii) the physical substrate's fragmentation decreases the potential for accepting new requests due to increased congestion on communication channels and IT resources [Zhu and Ammar, 2006]. The more geographically distributed are the physical resources that host a VI, the more bandwidth capacity is required (reserved) to interconnect the virtual IT resources. In the long term, the physical substrate's fragmentation can increase costs such as energy consumption, cooling and administration, due to the simultaneous activation of several distributed physical racks and network equipments.

In the face of these challenges, the main contributions of this chapter are:

- an allocation-problem formulation that considers the user and InP metrics; and
- an allocation heuristic guided by the geographical location of virtual and physical components, which introduces a procedure to reduce the search space in an automatic and intelligent way, accelerating the allocation algorithm without compromising the cost.

The results highlight an improvement in allocation quality (the user perspective) of about 39% for different sizes of virtual infrastructures allocated on a medium-size physical substrate. In addition, the fragmentation can decrease by almost 28% on a medium-size physical substrate, while the cost can decrease by approximately 21% (the InP perspective).

This chapter is organized as follows: Section 5.2 illustrates the allocation of a virtual infrastructure and reviews the state of the art. Section 5.3 formulates the allocation problem using a graph notation, and describes the formal representation of constraints

and objective functions. In Section 5.4 we propose the allocation method, describing its composition and implementation. The experiments performed with our proposition are presented in Section 5.5. Section 5.6 concludes the chapter.

5.2 State of the art

In the lifecycle of virtual infrastructures, the allocation step is responsible for identifying the mapping between virtual and physical resources. Within this step, a VI request must be interpreted and available distributed resources must be reserved. This mapping is used by the management framework to identify the exact location where the virtual resources must be provisioned. The following sections describe this mapping process and review the approaches that have been elaborated for allocating IT resources, virtual networks, and VIs.

5.2.1 Example scenario

VIs and physical substrates can be modeled using a graph notation, where the vertices represent the IT resources (virtual and physical), and the edges represent the communication channels (virtual links and physical paths). Consequently, a simple way to exemplify the allocation process is through a mapping of graph components: the virtual edges and vertices are mapped into physical edges and vertices, respectively, respecting the constraints and capacities of each component (virtual or physical).

To illustrate the allocation of VIs, we create two specifications (presented in Figure 5.2) and allocate them on a distributed physical substrate (as presented in Figure 5.3). Both VIs presented in Figure 5.2 are described using graphs. The VIs (*VI A* and *VI B*) and the physical substrate have Q values attributed to links and resources. For VIs, these values represent the requirements in terms of network communication and computing configuration, and for a physical substrate, they represent the available capacity. *VI A* is composed of four virtual resources (r_v1, r_v2, r_v3, and r_v4), which are interconnected by three virtual links (l_v1, l_v2, and l_v3). *VI B* comprises three virtual resources (r_v5, r_v6, and r_v7) interconnected by two virtual links (l_v4, and l_v5).

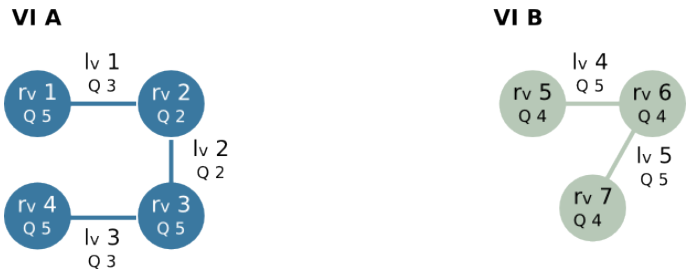


Figure 5.2: Example of two virtual infrastructures (*VI A* and *VI B*) using graphs.

The physical infrastructure (described on the left side of Figure 5.3) is composed of twelve resources (r_p1 to r_p12) interconnected by fifteen links (l_p1 to l_p15). The resources are geographically distributed over two locations (*Location 1* and *Location 2*) interconnected by a dedicated backbone.

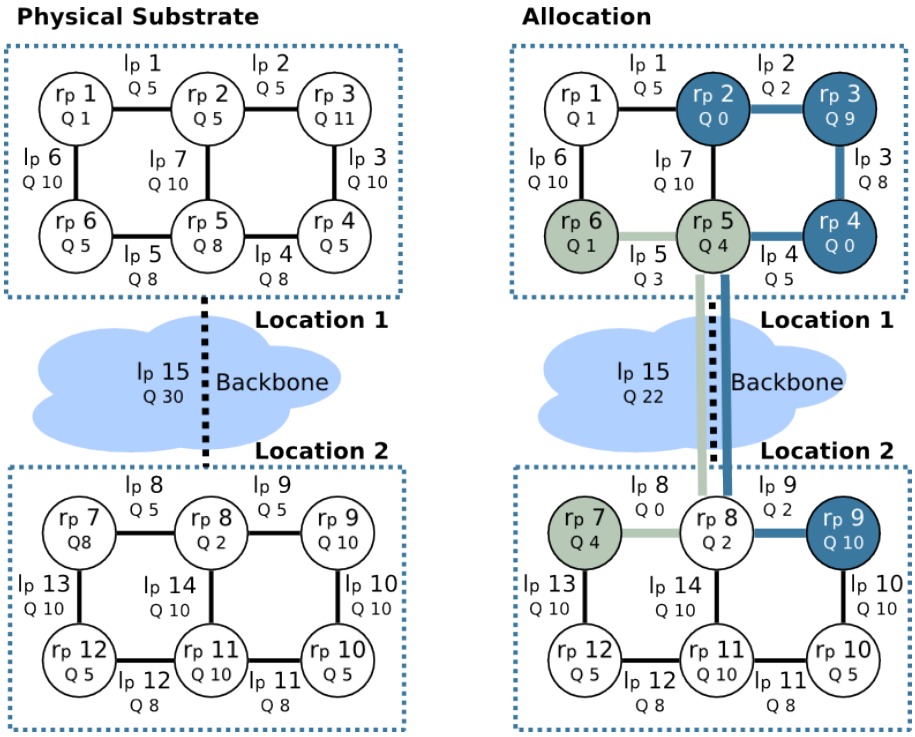


Figure 5.3: The representation of a physical substrate, and an example of a map solution between the VIs presented in Figure 5.2 and the physical substrate.

The mapping solution (present on the right side of Figure 5.3) is an allocation alternative for the *VI A* and *VI B* over the physical substrate. In this example, the IT and network resources of both VIs are distributed over the two locations. The Q values presented in this mapping solution show the available capacity of physical resources after the allocation process. Table 5.1 summarizes this allocation mapping using tuples in the format $<$ virtual resource, physical resource, capacity, $\Delta t >$, where Δt means the time period during which the capacity was reserved. To simplify the description, we consider a unique time $t0$ in this example. Following the VI's life cycle, this mapping information is given as input to management frameworks that provision the virtual resources.

5.2.2 Existing solutions for allocating VIs

The allocation of virtual resources (IT and network) has been investigated over the past few years in different contexts. Initially, this problem was addressed to allocate communication channels between IT endpoints in different networking technologies, such as IP, MPLS, ATM and frame-relay, composing VPNs. In VPNs, the allocation process consists in finding paths between source-destination pairs respecting the bandwidth requirements [Duffield et al., 1999] [Gupta et al., 2001] [Lu and Turner, 2006]. The allocation of virtual computing resources was not addressed by these approaches. In addition, the location of source and destination endpoints were known in advance by the allocation algorithms.

[Dias de Assunção and Buyya, 2009] proposed a system termed as the InterGrid for allocating virtual machines at multiple computing sites. The concept was further extended

(r_v or l_v)	(r_p or l_p)	Q	Δt
r_v1	r_p2	5	$t0$
r_v2	r_p3	2	$t0$
r_v3	r_p4	5	$t0$
r_v4	r_p9	5	$t0$
l_v1	l_p2	3	$t0$
l_v2	l_p3	2	$t0$
l_v3	l_p4	3	$t0$
l_v3	l_p15	3	$t0$
r_v5	r_p6	4	$t0$
r_v6	r_p5	4	$t0$
r_v7	r_p7	4	$t0$
l_v4	l_p5	5	$t0$
l_v5	l_p15	5	$t0$
l_v5	l_p8	5	$t0$

Table 5.1: Map of the allocation solution presented in Figure 6.4 using tuples in the format $<$ virtual resource, physical resource, capacity, $\Delta t >$.

to allow an organization to extend the capacity of its local cluster by borrowing resources from a Public Cloud provider. This work targeted only bag-of-tasks applications and the allocation of virtual links was not taken into consideration.

Recent work has focused on problem formulations considering node requirements together with network configuration. Due to its NP-hard complexity [Chowdhury and Boutaba, 2009], there are different proposals on solving this graph-embedding problem, most of which aim to find an optimal solution in an acceptable response time. These proposals include isomorphism-based detection, path-splitting methods, multi-commodity flow modeling, and heuristics based on substrate characteristics. In addition to exploring different approaches, these proposals have their own objectives and metrics, for example, maximizing resource usage, minimizing maximum link load, proposing fault-tolerant algorithms, and allocating virtual resources across multiple domains. We detail here selected proposals, their context, main contributions and particularities.

[Ricci et al., 2003] proposed a search-space restriction to accelerate the response time of the allocation process. The proposed program, called *assign*, explores the resource homogeneity of the Emulab testbed [White et al., 2002] by aggregating resources into equivalence classes (*vclasses and pclasses*). A group of aggregated resources is interconnected by network links with capacities described in terms of bandwidth. Particularly, this program does not allow computing resources to be shared among multiple virtual machines. *Assign* implements an algorithm based on simulated annealing [Kirkpatrick et al., 1983] for finding a mapping solution.

[Lu and Turner, 2006] investigated the mapping of virtual networks on shared substrates. The proposed algorithm attempts to find the best topology in a family of backbone-star topologies. More specifically, a set of intermediate nodes is assigned as the backbone, and each remaining node is connected to the nearest backbone node aiming to minimize the network total cost. By positioning intermediate backbone nodes the search space is

limited. Only a subset of nodes directly connected by the backbone-star topology are considered to determine the shortest path in terms of resource distance.

[Zhu and Ammar, 2006] proposed a set of heuristics to allocate virtual networks on a physical substrate. This work aimed to increase the allocation efficiency by minimizing the stress on nodes and links, where the stress of a resource represents the load already supported and reserved. Based on this metric, new resources are provisioned on top of physical resources with a relatively low stress. The authors also proposed an algorithm to reconfigure the substrate by reallocating a subset of virtual resources, keeping the entire substrate with a low load. In addition, the reallocation can be guided by reconfiguration policies [Fan and Ammar, 2006].

[Houidi et al., 2008] proposed a distributed algorithm for load balancing and to allocate virtual nodes and links to physical substrates. They have designed a protocol for asynchronous communication between agents located on distributed resources to guide the allocation process. The algorithm performs a star based decomposition of the virtual network, using a hub-and-spoke approach, which maps a central node as hub and connects the other nodes as spokes (similar to the *backbone-star* approach, proposed in [Lu and Turner, 2006]). The implementation was based on a multi-agent system [Kephart and Chess, 2003], where a set of autonomous and independent agents communicate and collaborate to identify a map solution. A recent approach extended this algorithm by proposing a distributed fault-tolerant allocation algorithm to handle failures and reselect new resources to replace those no longer available [Houidi et al., 2010] [Houidi et al., 2011]. This framework (mainly composed of resource description [Renault et al., 2010], discovery and allocation mechanisms) monitors the physical substrate to detect when a resource fails, acting to select and allocate a new resource to replace the one that failed.

[Cadere et al., 2008] modeled an embedding graph problem considering three parameters to evaluate the quality of an allocation map: the dilatation that is the maximum length of a path and represents the communication delay; the *congestion* that is the maximum number of virtual links using a physical path and represents the bandwidth control; and the load that is the number of virtual resources allocated on a physical resource. The proposed multi-objective allocation heuristic aims at minimizing these metrics and consequently the number of resources used to accommodate a virtual infrastructure. The evaluation results indicate that the heuristic may have a long computation time under certain distributed physical substrates.

[Yu et al., 2008] discussed the design of the substrate network considering virtual path migration and splitting, wherein a virtual link can be split among a set of virtual paths enabling a more efficient usage of the substrate network. In addition, a path migration is periodically performed to optimize the substrate network utilization. The problem is modeled as a multi-commodity flow problem without node remapping. According to their evaluation, the splitting of virtual links helps maximize revenue and minimize substrate resource usage.

[Lischka and Karl, 2009] proposed a backtracking algorithm to identify a subgraph isomorphism between the virtual and the physical graph. The proposed algorithm is an extension of the *vflib* graph matching algorithm [Cordella et al., 2004] and is solvable in polynomial time. By limiting the length of a physical path in which a virtual link can be extended, the authors reduced the search space and consequently limited the response time. Within this approach, nodes and links are considered together during the allocation

process.

[Chowdhury et al., 2009] developed two algorithms (one deterministic and one randomized) to allocate virtual infrastructures with coordinated node and link mapping. With link and node constraints, the problem is formulated as a Mixed Integer Program (MIP) [Schrijver, 1998], and the integer constraints were relaxed to obtain a linear programing. In [Chowdhury et al., 2010] the allocated problem was formulated considering the allocation of VIs across multiple administrative domains that compose the physical substrate. They proposed a decentralized framework based on a hierarchical addressing scheme and on a location awareness protocol. Each local domain (or intra-domain) receives a partition of a VI request to allocate on their private resources by executing the centralized heuristic described in [Chowdhury et al., 2009]. The protocol carries information to request the VI, to negotiate the allocation, and to inform its status (success and failure). Recently, [Yeow et al., 2010] extended this formulation and investigated the virtual-infrastructure allocation considering mechanisms to pool backup nodes in order to achieve the desired level of reliability together with resource allocation.

The work developed in [Butt et al., 2010] investigates a topology-aware mechanism to re-optimize and allocate initially rejected requests. This approach identifies the bottleneck of rejected requests and re-optimizes the physical substrate to accommodate their requirements. The metric used to control the physical substrate load and to determine which resources must be re-optimized are based on a scaling factor that identifies the probability of a resource becoming a bottleneck.

We summarize these approaches in Table 5.2 identifying the following characteristics:

- *Centralized or distributed implementation:* the majority of the proposed algorithms have been implemented with a centralized approach. The decentralization of the algorithm is justified by an improvement in scalability, robustness, and local management of partial failures.

- *Reconfigurability:* refers to the algorithm's skill in reconfiguring the virtual infrastructures already allocated in order to balance the load of the physical substrate. The existence of this option may be related to the context in which the approach is inserted. Usually, this reconfiguration task can be performed by management frameworks outside of the allocation mechanism.

- *Type of incoming requests:* the *online* problem formulation refers to the scenario where the virtual infrastructure requests are not known in advance, arriving dynamically with a non-arbitrary duration (reservation time). Over the past years, this model of incoming requests has been adopted mainly to represent a real scenario, where the services (VIs) could be requested at any time.

- *Problem formulation and heuristic:* different solutions can be applied for solving the allocation problem. In general, software implementations have used substrate based heuristic, shortest path [Dijkstra, 1959], and subgraph-isomorphism [West, 2000] detection techniques, while simulations have explored Mixed Integer Problem formulations [Schrijver, 1998].

Proposal	Implementation	Reconfigurability	Requests	Problem formulation / heuristic
[Ricci et al., 2003]	centralized	no	offline	Simulated Annealing
[Lu and Turner, 2006]	centralized	no	offline	Shortest path, substrate based heuristic
[Zhu and Ammar, 2006]	centralized	yes	offline	Shortest path, Greedy
[Cadere et al., 2008]	centralized	no	online	Shortest path, substrate based heuristic
[Yu et al., 2008]	centralized	yes	online	Multi-commodity flow problem, Greedy
[Chowdhury et al., 2009]	centralized	yes	online	Multi-commodity flow problem, Rounding Mixed Integer Program
[Lischka and Karl, 2009]	centralized	no	online	Subgraph isomorphism detection
[Houidi et al., 2010]	distributed	yes	online	substrate based heuristic
[Chowdhury et al., 2010]	distributed	yes	online	Mixed Integer Program
[Yeow et al., 2010]	centralized	no	online	Mixed Integer Program
[Butt et al., 2010]	centralized	yes	online	Multi-commodity flow problem, Rounding Mixed Integer Problem

Table 5.2: A list of selected proposals developed to allocate virtual resources and their particularities: implementation scenario, reconfigurability, type of requests, and problem formulation.

5.2.3 Requirements for efficient VI allocation

As highlighted in the previous section, the allocation of virtual infrastructures has been well studied, forming an extensive background of good practices and future action points. For example, often virtual resources are not mapped adjacently, but distributed over physical resources. Consequently, the vLinks required to interconnect these resources must be extended over multiple physical communication paths, with a vLink being split over the physical substrate to provide communication between virtual source and target. The *splitting* of a vLink means its decomposition in many virtual links allocated over different physical components [Yu et al., 2008], while the *extension* is related to providing a QoS control mechanism along the physical path where the vLink is provisioned.

Conceptually, the abstract decomposition and extension are completely transparent to final users [Yu et al., 2008], who are only aware of the dynamic virtual resources provisioned to compose the VI (called by *User layer*). An *abstract layer* is introduced between the *physical layer* and the *user layer* to implement the required mechanisms to perform traffic control and QoS provisioning [Anhalt et al., 2010]. We explain this decomposition in Figure 5.4, where a vLink (part of the *user layer*) is divided and extended on multiple virtual paths (the *abstract layer*) allocated on top of a virtualized and distributed physical substrate (the *physical layer*). vLink l_v1 is decomposed into four new vLinks (l_v2, l_v3, l_v4, and l_v5), and two virtual routers (r_vA and r_vB), forming the virtual paths. The communication between virtual resources r_v1 and r_v2 is effectively divided over two physical paths resulting from vLink decomposition.

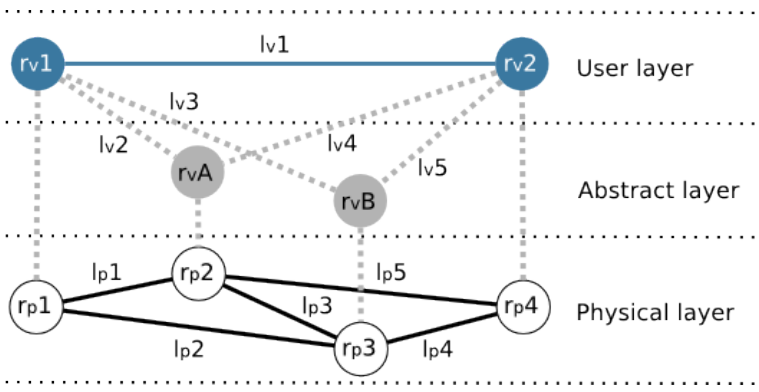


Figure 5.4: Illustration of a vLink decomposition using extension, splitting, and vRouters. *User layer* represents the components exposed to users; *Abstract layer* remains transparent, being controlled by the InP to guarantee the users' requirements; and *Physical layer* represents the materialized resources that compose the distributed physical substrate.

[Yu et al., 2008] argued that splitting techniques can decrease the usage of physical substrate resources and increase the acceptance ratio of new requests. However, the allocation of virtual paths increases the complexity since a virtual path is directly related to the location of virtual source and target. These locations also are discovered during the mapping process, and consequently, also the virtual paths composition.

In addition, [Ricci et al., 2006] drew lessons from the implementation of an allocation algorithm in the context of PlanetLab [Bavier et al., 2004] and Emulab [White et al., 2002] testbeds. They concluded that the resource allocator should have a well-defined

and asynchronous interface to enable the *negotiation* with users and other allocators. Although the scenario explored in their work is simpler than the allocation of virtual infrastructures, the requirements are similar and must be considered. For example, as detailed in Chapter 3, a description language plays an important role in the negotiation step.

Recently, [Haider et al., 2009] highlighted new challenges in the allocation of virtual networks, most of which related to the need for an adaptive approach capable of dynamically adapting the resources allocated to VIs requirements. In this context, the SAIL project has determined the elastic aspect of virtual resources as an important requirement for efficient usage and implementation of VIs [SAI].

We considered these requirements along with the expectations and needs of real users to produce a complementary list of key aspects that should be addressed in allocating virtual infrastructures. The users' expectations were defined based on [5] and [2]. As discussed in Chapter 4, users can generate efficient VI descriptions that represent the application requirements, and decompose them in different stages, identifying the exact capacity required for each one. The key requirements are divided in Infrastructure Providers' (InP) and users' allocation requirements:

InPs' requirements:

- The main objective of an InP is to allocate the maximum number of VI in an efficient manner, minimizing the number of physical resources activated. To achieve this purpose, an allocation algorithm should **consider nodes and links with the same weight, allocating both at the same time**, as highlighted in [Chowdhury et al., 2009]. Several algorithms perform the allocation in a two-step process, first allocating the nodes and then allocating the links, solving a shortest-path problem or a multi-commodity flow problem. This approach may lead to inefficient solutions in terms of resources usage.

- Virtual infrastructure can be positioned anywhere on top of a virtualized substrate. The allocation mechanism should **minimize the spreading and fragmentation of physical resources**. Decreasing the number of distributed physical resources activated reduces the administrative costs (e.g., energy and cooling).

- The virtual resources composing a VI can vary their capacity requirements during the reservation time. Instead of comparing single values representing the capacity requirements, allocators should **manipulate capacity profiles**, which represent the capacity variation during the reservation time.

- In some cases, an IT virtual resource cannot be allocated on a single resource because its requirements exceed the physical capacity. An allocator should consider the **aggregation of similar physical resources composing a single virtual entity able to provide the required capacity**. A similar approach has been implemented in vLinks extension and splitting [Yu et al., 2008].

Users' requirements:

- As discussed in Chapters 3 and 4, the new languages and techniques developed to compose and describe virtual infrastructures enable the direct participation of users in the allocation process. **Users and management framework must interact to efficiently allocate a virtual infrastructure**. The constraints and requirements are explicitly detailed guiding the allocator. Moreover, a VI description can be improved after a monitored execution, and the acquired knowledge should be used to improve the next VI allocation and execution, applying for example, the adaptability concept [Andrzejak et al., 2006].

- Following the pay-as-you-go model applied to Cloud Computing and Cloud Networking technologies, **new services can be requested by users, such as security, reliability and adaptability**. To provide these new services, the allocation process must be aware of the requirements and particularities of each service. In Chapter 6, we demonstrate how a service that offers reliable virtual infrastructures can be implemented, and the important role performed by the allocation algorithm.

5.3 Problem formulation

In this section we formulate the allocation problem using a graph representation. This formulation is guided by the requirements identified in Section 5.2.3 and aims to cover the attributes proposed by the descriptive language presented in Chapter 3.

5.3.1 Graph embedding problem

The problem of mapping VIs to a physical substrate corresponds to a classical graph embedding problem. The graph describing the VI, $G^v(R^v, L^v)$, must be mapped on the physical substrate graph, $G^p(R^p, L^p)$, where R^v and R^p are the set of virtual and physical nodes, respectively, and L^v and L^p are the set of virtual and physical links, respectively. Table 5.3 summarizes the notation used to formulate the allocation problem.

Let's denote by $Q_R(r,t)$ the vector of capacities (e.g., memory and CPU) of node r ($\in R^v$ or $\in R^p$) at time $t \in [0,T]$. In addition, let P^p be the set of all the simple physical paths between any two physical nodes, $Q_P(p,t)$ be the vector of capacities of physical path $p \in P^p$, and $Q_L(l,t)$ the vector of capacities of link $l \in L^v$, both at time t. For a physical path $p = (l_1, l_2, \dots)$, the bandwidth capacity is the minimum of all bandwidth values of l_i in p. By adopting capacity vectors indexed by time, the capacities of IT resources and links can vary during the reservation's time.

Let $A_R(r)$ be the set of geographical locations of resource r ($\in R^v$ or $\in R^p$). For virtual resources, the value of $A_R(r)$ can be specified by users, and for physical resources it represents the exact geographical location. Finally, let $E_R(r)$ be a binary function which identifies if a virtual resource requires exclusivity on a physical host, not sharing its capacity.

A map of a VI on a physical substrate represents the reservation of all the capacity requirements specified by the user, noted as:

$$\text{Resources mapping} : \mathcal{M}_R : R^v \rightarrow R^p \tag{5.1}$$

Notation	Meaning
$G^v(R^v, L^v)$	a graph of virtual resources R^v (vertices) and the virtual links L^v (edges)
S^v	a set of virtual graphs $G_i^v \in S^v$
$r_i \in R^v$	a virtual IT resource
$r_j \in R^p$	a physical IT resource
$l_i \in L^v$	a virtual link
$l_j \in L^p$	a physical link
$p = (l_1, l_2, \dots)$	a physical path p composed of a set of links
P^p	set of all physical paths between any two nodes
$t \in [0, T]$	time instant of reservation period $[0, T]$
$Q_R(r, t)$	the vector of capacities of a IT resource r indexed by time t
$Q_L(l, t)$	the vector of capacities of a link l indexed by time t
$Q_P(p, t)$	the vector of capacities of a path p
$A_R(r)$	geographical locations of resource r
$E_R(r)$	binary function identifying exclusivity of virtual resources

Table 5.3: Notations used to formulate the allocation problem.

$$\text{Links mapping} : \mathcal{M}_L : L^v \rightarrow P^p \tag{5.2}$$

Given a set of VI requests S^v, the embedding problem is to obtain a mapping of virtual nodes R^v to physical nodes R^p, denoted by $\mathcal{M}_R$, and virtual links L^v to physical paths P^p, denoted by $\mathcal{M}_L$, such that the following conditions are satisfied:

$$Q_R(\mathcal{M}_R(r_i), t) \geq Q_R(r_i, t), \forall r_i \in R_j^v, \forall G_j^v \in S^v, \forall t \in [0, T]; \tag{5.3}$$

$$Q_P(\mathcal{M}_L(l_i), t) \geq Q_L(l_i, t), \forall l_i \in L_j^v, \forall G_j^v \in S^v, \forall t \in [0, T]; \tag{5.4}$$

$$A_R(r_i) \subset A_R(\mathcal{M}_R(r_i)), \forall r_i \in R_j^v, \forall G_j^v \in S^v; \tag{5.5}$$

$$E_R(r_i) = 1 \implies \mathcal{M}_R(r_i) \neq \mathcal{M}_R(r_j), \forall i \in R_k^v, \forall G_k^v \in S^v, \forall j \in R_m^v, \forall G_m^v \in S^v. \tag{5.6}$$

5.3.2 Objective functions

As identified in Section 5.2.3, different objectives and functions can be exploited for solving the allocation problem. We formulate the objective functions considering both InP and users perspectives. The former was formulated in terms of allocation cost and physical substrate fragmentation, while the latter in terms of allocation quality.

To facilitate the introduction of these functions, explaining their motivations in a real scenario, we present in Figure 5.5 a VI request and in Figure 5.6 a physical substrate that must host the virtual infrastructure. The VI is composed of three virtual resources (r_vA, r_vB, and r_vC) which are interconnected by a set of virtual links. A particular user has access to all VI components during the reservation time.

This VI must be allocated and provisioned on top of a distributed and virtualized physical substrate. Usually, this physical substrate is hierarchically organized and in-

terconnected. Figure 5.6 presents an example of this hierarchical organization using a tree structure where the leaves are the IT resources, and the parent nodes represent the geographical organization and interconnection. Starting from the bottom, the physical resources (for example, r_p1, r_p2, and r_p3) are grouped into racks. These racks are positioned at different locations, such as *lyon.fr.eu* and *paris.fr.eu*. More specifically, in this example the exact location of the physical node r_p1 is noted as *rp1.rack1.lyon.fr.eu*. The distance between physical resources in this hierarchical organization is given by the number of intermediate hops. For example, the distance between r_p2 and r_p3 is one hop.

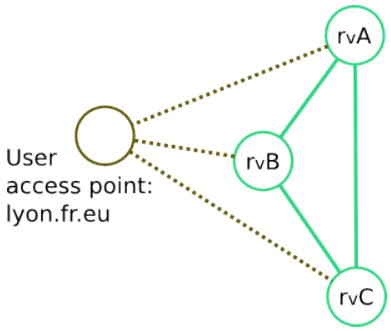


Figure 5.5: A VI composed of resources r_vA, r_vB and r_vC is requested by a user with access point located at *lyon.fr.eu*.

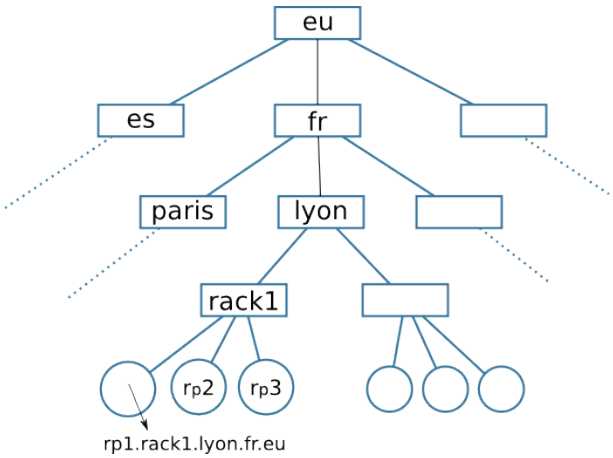


Figure 5.6: The physical substrate that must allocate the VI described in Figure 5.5 comprises components hierarchically distributed and interconnected.

5.3.2.1 Physical substrate cost

For an InP, the cost to allocate a virtual infrastructure is proportional to the number of resources required to accommodate the user's requirements. Let's define the functions $C_r(r,t)$ and $C_l(l,t)$, which set the physical substrate cost for an amount of resource $r \in R^v$ and the cost for an amount of link $l \in L^v$, respectively, both at time t. The total cost of resources R^v, over a reservation time $[0,T]$ is given by:

$$C_R(R^v, T) = \int_0^T \Big(\sum_{r_i \in R^v} C_r(r_i, t) \Big)\ dt \tag{5.7}$$

Similarly, the total cost of links L^v, over a reservation time $[0, T]$ is defined as:

$$C_L(L^v, T) = \int_0^T \Big(\sum_{l_i \in L^v} C_l(l_i, t) \times len(\mathcal{M}_L(l_i)) \Big)\ dt \tag{5.8}$$

where $len(\mathcal{M}_L(l_i))$ gives the length of the path provisioned to allocate virtual link l_i. Consequently, the total cost of a VI G^v is noted as:

$$C_{VI}(G^v, T) = \alpha C_R(R^v, T) + \beta C_L(L^v, T) \tag{5.9}$$

Constants α and β are tunable weights that allow the balance and the normalization between resources' and links' costs. Given a set of VI requests S^v, an immediate metric is the minimization of the total cost for the infrastructure provider, defined by:

$$\text{minimize: } \sum_{G_i^v \in S^v} C_{VI}(G_i^v, T) \tag{5.10}$$

subject to the constraints defined by conditions (5.3)-(5.6).

5.3.2.2 Physical substrate fragmentation

The cost of a virtual infrastructure allocation is usually negotiated and charged to the final user. However, InPs have administrative costs that vary depending on the number of resources activated. These costs are difficult to be charged during the SLA negotiation, but can have a negative impact on the InP objectives. For example, the allocation of spread VI components induces long-term issues: i) it increases costs related to energy consumption, cooling, and provisioning of high-speed networks to interconnect data-centers due to the simultaneous activation of several racks and network equipments; and ii) it decreases the acceptance ratio of new requests due to increased congestion on communication and computational resources [Zhu and Ammar, 2006]. These issues are directly related to the physical substrate fragmentation.

We define the *physical substrate's fragmentation* as a metric to qualify the number of physical resources (IT and network) reserved and activated on a distributed and virtualized substrate. This metric is given by the ratio of the number of activated resources to the total number of physical resources available.

To exemplify the fragmentation, lets consider the virtual infrastructure request presented in Figure 5.5 and the physical substrate described in Figure 5.6. Considering the physical substrate fragmentation, an efficient allocation mapping allocates all the virtual nodes and virtual links using the minimum number of physical components. For example, allocating the virtual nodes into physical machines r_p1, r_p2, and r_p3, respectively, only requires activating one rack and using a single physical communication path to provide network access to those components. Furthermore, an optimal allocation places all virtual nodes (r_vA, r_vB, and r_vC) into physical machine r_p1 only requiring the activation of a single computing resource. In the long term, it is expected that minimizing the physical substrate's fragmentation increases the acceptance ratio of an InP.

InPs aim to minimize the physical resources' fragmentation. Thus, let's denote by F_R the subset of physical resources $r \in R^p$ that supports at least one virtual resource running, and similarly, by F_L the subset of physical links $l \in L^p$ that hosts at least one virtual link activated. The objective is to minimize the number of physical resources (IT and networking) to allocate S^v:

$$\text{minimize: } \frac{\#F_R + \#F_L}{\#R^p + \#L^p} \tag{5.11}$$

subject to the constraints defined by conditions (5.3)-(5.6).

5.3.2.3 Allocation quality

From the user's perspective, the allocation and provisioning of spread VI components increases latency of network communications [5] as follows:

- node-to-node: usually, the more distant apart the physical hosts are, the higher the latency in communication. This issue is more perceptible in communication intensive applications, but can also affect regular applications;
- user interaction: interactive applications (such as remote terminals or visualization tools) are explored by users to control the virtual infrastructure and their applications. The response time of these applications increases proportionally to the physical distance between the user and the virtual infrastructure components.

Chapter 4 showed that *advanced users* can specify the exact configuration required to execute their applications efficiently including the exact location where the virtual resources should be provisioned [6]. Reasons why a certain application should run in a certain location, include data-location dependency, security, and even limitations on data mobility because of governmental law. Although this location-based provisioning can be explicitly required, *regular users* are not aware of the efficient configuration in terms of networking and computational power. Moreover, some users are not familiar with the meaning of latency in communications, and only wish a set of VMs to execute their applications, as the approach explored in Clouds.

In this context, defining the allocation quality from the user's perspective is a difficult task as it is optimal when the virtual request is precisely defined and all resources correctly provisioned. When the request is non-absolutely defined, quality is subjective: the user does not care about the optimal configuration to execute his application, but he wants the application to run well, usually with efficient interaction among its distributed components. Figure 5.5 describes this scenario: a VI is requested by a user with access point located at *lyon.fr.eu*. The user does not specify any requirement in terms of virtual-resources' location or network configuration. Thus, the InP can allocate this request anywhere on top of the distributed substrate represented by Figure 5.6.

The resources' proximity can be optimized for both types of users (e.g., advanced or regular) independently of the VIs description level. We call *allocation quality* of a VI the average of all distances (calculated in hops) between the virtual resources and one specific geographical landmark (the reference point). More specifically, the reference point can be the location of the user; or the location of a certain virtual resource, as specified by the user.

To illustrate the definition of allocation quality, let's consider the user's location as the geographical landmark for hops calculation (*lyon.fr.eu*). Allocating resources distributed across other locations (such as *es.eu*) results in a greater number of hops from the user's location than allocating resources near *lyon*, or even *fr*. More specifically, allocating all the components of this VI on physical node *rp1.rack1.lyon.fr.eu* results in a minimum distance (in this case, only one hop, from *lyon*'s access point), and consequently, an optimal allocation quality.

The quality of an allocation is directly related to the location of the virtual resources. Let's define the function $D_R(a_i, a_j)$ that gives the distance between locations a_i and a_j in number of hops, where $a \in A_R$, and each hop is equivalent as one unit. Further, define a^u as the location specified by the user (the reference point). The optimal allocation quality is given by the minimization of the average resources' distance:

$$\text{minimize: } \sum_{r_i \in R^v} \frac{D_R(A_R(\mathcal{M}_R(r_i)), a^u)}{\#R^v} \tag{5.12}$$

subject to the constraints defined by the conditions (5.3)-(5.6).

Conceptually, the quality improvement is related to the minimization of physical substrate fragmentation. This relationship is observed in results presented in Section 5.5.

5.4 A heuristic for allocating VIs

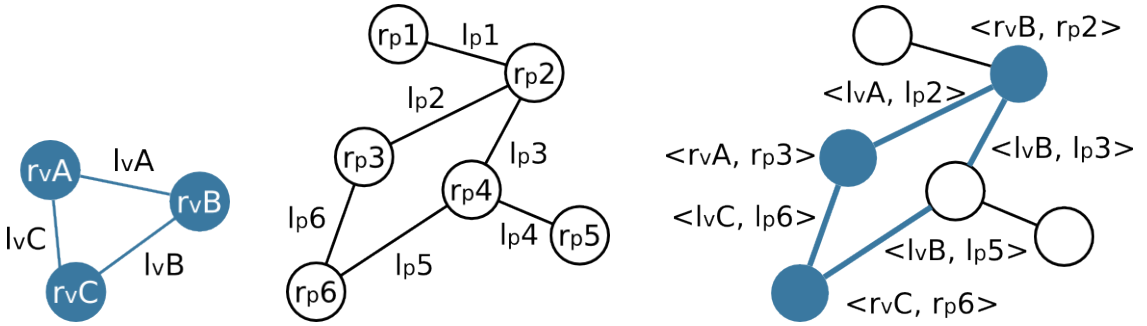
As discussed beforehand, there are different approaches to solve the graph-embedding problem, such as heuristics based on substrate characteristics, path-splitting methods, multi-commodity flow modeling, and isomorphism-based detection. Among these options, we choose a subgraph-isomorphism detection [West, 2000], solvable in polynomial time, to incorporate the allocation constraints and to examine the metrics proposed in this chapter. As result of this work, we have proposed a heuristic to allocate virtual infrastructures [VXAlloc] that drastically decreases the decision time by proposing a method to restrict the search-space. This method has been patented and is exploited by the LYaTiss startup, and for the sake of confidentiality its critical part cannot be detailed here. Therefore this section overviews the method and details experimental results obtained with our implementation of the heuristic.

5.4.1 Subgraph-isomorphism detection

The technique for identifying isomorphic subgraphs acts as an alternative for embedding a virtual graph in a physical one [West, 2000] [Cordella et al., 2004] [Lischka and Karl, 2009]. An isomorphism, with edges extension, from G^v to G^p is a function f that maps R^v to R^p and L^v to P^p such that each edge $l_i \in L^v$ with endpoints $r_m \in R^v$ and $r_n \in R^v$ is mapped to a path of edges $l_j \in P^p$ with endpoints $f(r_m) \in R^p$ and $f(r_n) \in R^p$, subject to the constraints defined by the conditions (5.3)-(5.6).

Figure 5.7 exemplifies a subgraph isomorphism map between graphs G^v and G^p. Applying function f for all $r_i \in R^v$ and for all $l_i \in L^v$ results in a set of maps noted $< x, f(x) >$, where x represents both vertices or edges, contextually.

A subgraph-isomorphism mapping of non-simple graphs (e.g., VIs) requires the extension of edges to interconnect non-adjacent vertices. Figure 5.7c exemplifies this require-



(a) graph $G^v(R^v, L^v)$.

(b) graph $G^p(R^p, L^p)$.

(c) Subgraph isomorphism between graphs G^v and G^p.

Figure 5.7: A subgraph-isomorphism mapping between G^v (5.7a) and G^p (5.7b) given by the application of a function f, where $f(r_vA) = r_p3, f(r_vB) = r_p2, f(r_vC) = r_p6, f(l_vA) = l_p2, f(l_vB) = l_p3 \cup l_p5, f(l_vC) = l_p6$, as exemplified by 5.7c.

ment: r_vB and r_vC were mapped on r_p2 and r_p6, respectively. Consequently, l_vB must be extended over l_p3 and l_p5.

5.4.2 Algorithms formalization

This section formalizes the main heuristics composing the allocation algorithm. The notation presented in Table 5.3 is used to describe the pseudo-algorithms.

In Algorithm 5.1 we present a general view of the location-aware algorithm. Initially, the location-aware algorithm identifies the set of physical landmarks specified by the user, as well as the set of virtual resources without location constraints. An iteration is performed on these sets. Each time, a physical location specified by the user is defined as the required location constraint for components that do not have this information. At this moment, a subgraph-isomorphism detection is performed to find a map solution. If no allocation solution is found for this configuration, the location constraint is relaxed for these resources (the location's precision is decreased).

Let's use Figure 5.6 as example. In the first iteration, the required location of virtual resources r_v1, r_v2, and r_v3 are defined as *lyon.fr.eu* (the user's location). Considering that no solution is found with this configuration, the geographical location of these virtual resources is relaxed from *lyon.fr.eu* to *fr.eu*. Observe that, in this way, the algorithm always tries to allocate virtual components as close as possible to the geographical landmark. The algorithm focuses on solving the embedding problem, optimizing the allocation quality and decreasing the substrate fragmentation, as discussed in Section 5.3. A'_R represents the combination of all locations defined in the VI request, including the original user's location (the first one to be used as reference). To induce the minimization of virtual resource distances, the algorithm temporarily sets a location constraint for these resources that have not specified this requirement. The resulting set of virtual resources ($R^{v'}$) is used as input for the allocation procedure described in Algorithm 5.2. If no solution is found, the location constraints of those resources are relaxed.

Algorithm 5.2 solves a subgraph-isomorphism detection respecting the allocation constraints defined in Section 5.3. This algorithm is based on the formulations proposed in [Cordella et al., 2004] and [Lischka and Karl, 2009]. We mainly differ from the origi-

Algorithm 5.1: A heuristic for allocating VIs.

```
for a ∈ A'_R do
    while a ≠ ∅ do
        R^v' ← ∅ : temporary set of virtual resources;
        for r_i ∈ R^v do
            if A_R(r_i) = ∅ then
                set a for r_i;
            add r_i in R^v';
        M = alloc(R^v', L_v, ∅, ∅);
        if M ≠ ∅ then
            return M;
        relax a;
return ∅;
```

Algorithm 5.2: *alloc*: subgraph-isomorphism detection.

```
R^v: requested virtual resources;
L^v: requested virtual links;
R^v': temporary set of allocated virtual resources;
L^v': temporary set of allocated virtual links;
C ← candidates(R^v', L^v', R^v, L^v);
if C = ∅ then
    return ∅;
for (r_i, r_j) ∈ C do
    L^t ← all links between (r_i, R^v');
    if allocation constrains (5.3)-(5.6) from Section 5.3 are satisfied then
        add r_i in R^v';
        extend vLinks from L^t as described in Section 5.2.3;
        add L^t in L^v';
        if (R^v' = R^v) ∧ (L^v' = L^v) then
            return (R^v', L^v');
        n ← alloc(R^v, L^v, R^v', L^v');
        if n ≠ ∅ then
            return (R^v', L^v');
return ∅;
```


nal proposition by adding the optimized constraints on generating candidates and by not using a backtracking approach.

This recursive algorithm propagates the virtual request (R^v and L^v) and a set of temporary allocated resources ($R^{v'}$ and $L^{v'}$). With this information, pairs of virtual-physical candidates are generated (line 5) and covered in the following lines aiming the total allocation of a virtual infrastructure. Each allocated resource is added to a temporary set and the recursion is continued.

5.5 Experiments

This section describes a set of experiments performed with VXAlloc and the respective analysis considering the metrics proposed in Section 5.3. A complementary comparison between the wasted capacity and the acceptance-ratio of VXAlloc and non-virtualized infrastructures are discussed.

5.5.1 Scenario composition

We ran the experiments on machines belonging to Grid'5000 [Cappello et al., 2005] with the following configuration: 2 CPUs Intel Xeon L5420, 4 cores, 2.5 GHz, 6 MB cache, and 32 GB RAM. We used Java Runtime Environment version 1.6.0_20 to run the allocator and the other HIPerNet tools.

The physical-substrate graphs and the virtual-request graphs were generated by the topology generation tool GT-ITM [Calvert et al., 1997]. Physical substrates use the *transit-stub model*, which results in graphs with domains interconnected by a backbone. Virtual requests were generated considering the *normal model* without backbone routers, following setups similar to previous work [Yu et al., 2008] [Chowdhury et al., 2009].

Two physical substrates were simulated:

- a small-size substrate composed of 100 resources (domains and backbone routers) and approximately 200 physical links, organized in 4 geographical domains, and interconnected by a backbone composed of 8 resources;
- a medium-size substrate composed of 500 resources, approximately 4000 links, divided in 8 geographical domains, and interconnected by a backbone composed of 20 resources.

The values of CPU cores (2, 4 or 8) and memory capacity (2 GB, 4 GB, 8 GB or 16 GB) follow a uniform distribution. The network's bandwidth capacity was defined as 1 Gbps within domains and 10 Gbps between domains (in the backbone). To represent a realistic scenario, the allocation of computing nodes on backbone resources was disabled.

Virtual requests can require 2, 4, 6, 8, or 10 nodes. The probability of two virtual nodes being connected is 0.5, as defined in GT-ITM configuration. All VI requests require a reservation period of one hour. The values of CPU cores (1, 2 or 4) and memory capacity (256 MB, 512 MB, 1 GB, or 2 GB) also follow a uniform distribution. The same approach was used to generate network-bandwidth requirements (10 Mbps, 20 Mbps, 40 Mbps, 100 Mbps, 200 Mbps, 400 Mbps, or 600 Mbps).

The values of α and β used to calculate the VI cost (C_{VI}) were defined as $\alpha = \beta = 1$, indicating that nodes and links have the same weight in the InP, as explored on previous

scenarios [Yu et al., 2008]. The cost functions ($C_r(r,t)$ and $C_l(l,t)$) require an equivalent metric to calculate the costs. As an example of this equivalence, in terms of cost calculation, we arbitrarily set that 1 GB, 1 core, and 100 Mbps are the basic units for memory, CPU and bandwidth, respectively. The definition of these values requires a specific study based on InP policies and current substrate load, as proposed by [Zhu and Ammar, 2006]. We leave this implementation and analysis for future work.

The number of VI requests submitted were: 100 to the small-size substrate, and 300 to the medium-size substrate. The number of requests' sources varies (1 or 3) to represent different request-submission scenarios. Results identified by *allocation with basic algorithm* were obtained by executing the regular subgraph-isomorphism detection without the geographical-location optimization. The optimized execution is identified as *allocation with our optimized algorithm*.

All averages presented here were calculated considering 10 executions and have a confidence interval of 95%.

5.5.2 Physical substrate fragmentation and cost

The second experiment investigates the variation of InP allocation costs and physical-substrate fragmentation. For both metrics, the results were obtained considering two configurations: *small-size* and *medium-size* physical substrate.

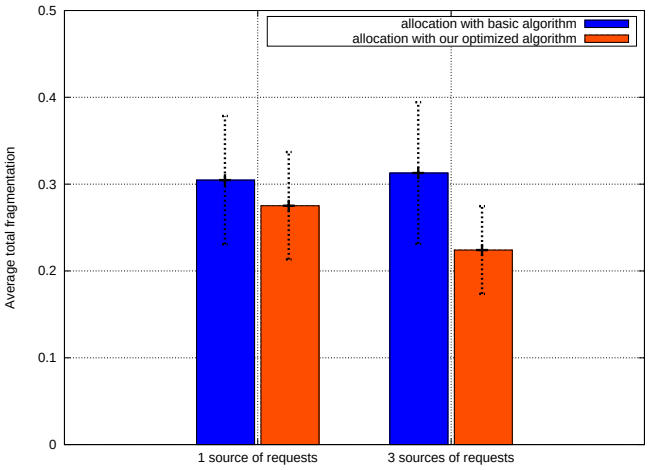


Figure 5.8: Fragmentation of a small-size physical substrate. The number of sources requesting VIs is 1 and 3.

Figure 5.8 shows the average total fragmentation of a small-size physical substrate. The comparison performed in Figure 5.8 highlights that our algorithm improves the physical substrate usage by decreasing the fragmentation by approximately 10% in the case where all requests come from the same location. Its performance increases with the number of requests source: 28% when 3 sources are submitting requests. Similar results are obtained with requests submitted to the medium-size substrate (see Figure 5.9).

The average of the total cost required to allocate these requests were analyzed and presented in Figure 5.10 and Figure 5.11. For these scenarios, our optimized algorithm also decreases the average cost. Considering requests submitted by only 1 source, the cost

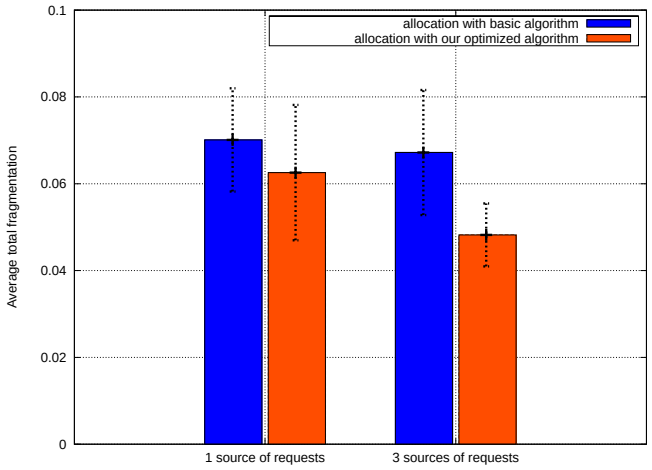


Figure 5.9: Fragmentation of a medium-size physical substrate. The number of sources requesting VIs is 1 and 3.

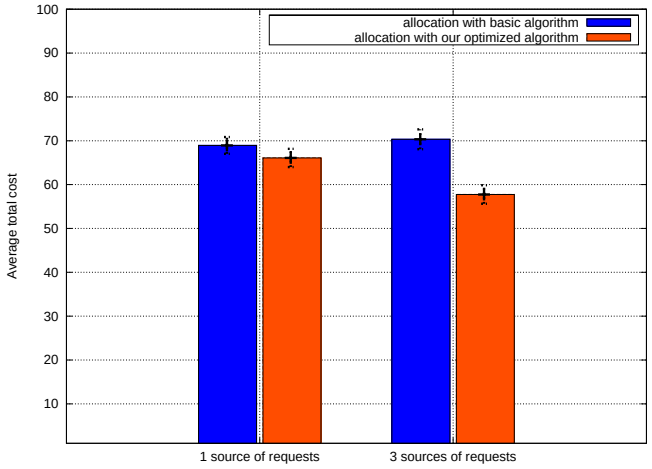


Figure 5.10: InP cost of allocations on a small-size physical substrate. The number of sources requesting VIs is 1 and 3.

decreases in both scenarios: approximately 4% on a small-size physical substrate, and 8% on a medium-size substrate. When analyzing requests submitted by 3 sources, the cost also decreases nearly by 17% and 21% for the small-size and the medium-size substrates, respectively.

5.5.3 Allocation quality

We also measured the the allocation quality. The user location (source of requests) was defined as the reference point for calculating the distances.

Figure 5.12 and Figure 5.13 present the average for the small-size and medium-size substrate, respectively. The results show that the average total distance is smaller with our algorithm in both scenarios, for requests submitted by 1 and 3 source(s). When only 1 source was submitting requests, the average total distance decreased by almost 13%

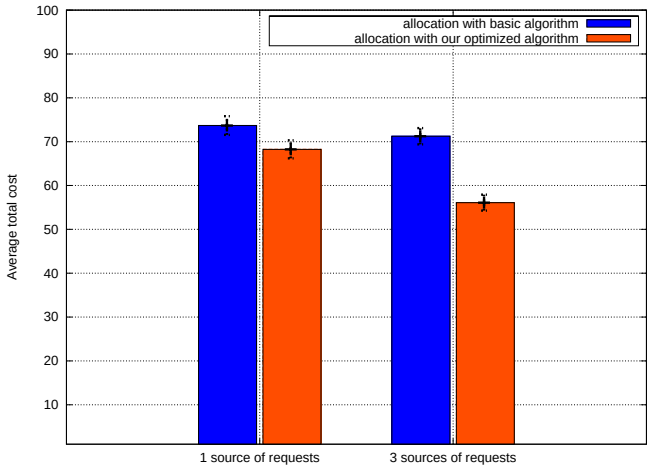


Figure 5.11: InP cost of allocations on a medium-size physical substrate. The number of sources requesting VIs is 1 and 3.

for small-size and medium-size substrates. When 3 sources were submitting requests the average total distance is highly decreased: by approximately 36% and 39% for small-size and medium-size substrates, respectively. Consequently, it is expected that the users of VIs allocated by our optimized algorithm have a lower impact of the physical-resource distribution in terms of network communication.

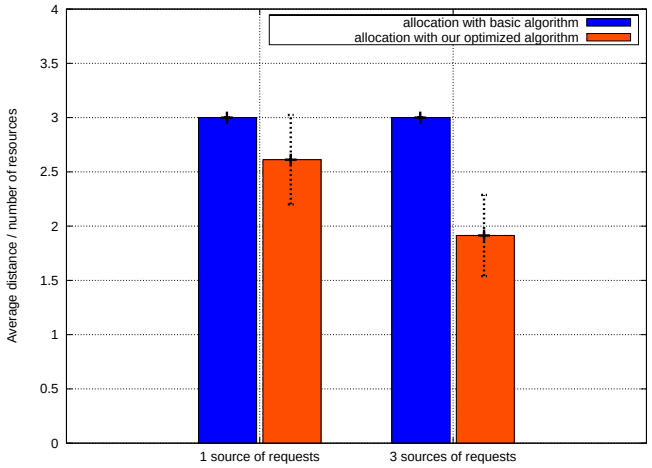


Figure 5.12: Aggregated distance of VI components allocated on a small-size physical substrate. The number of sources requesting VIs is 1 and 3.

An analysis of the results of the optimized algorithm highlights the relationship between performance and the number of geographical landmarks used as reference points (in this case, the user's location). For all metrics, the results obtained with three sources of requests showed better performance than those obtained with one source of requests, independently of the physical substrate size. Consequently, increasing performance of these metrics (physical fragmentation, allocation cost, and allocation quality) is expected when

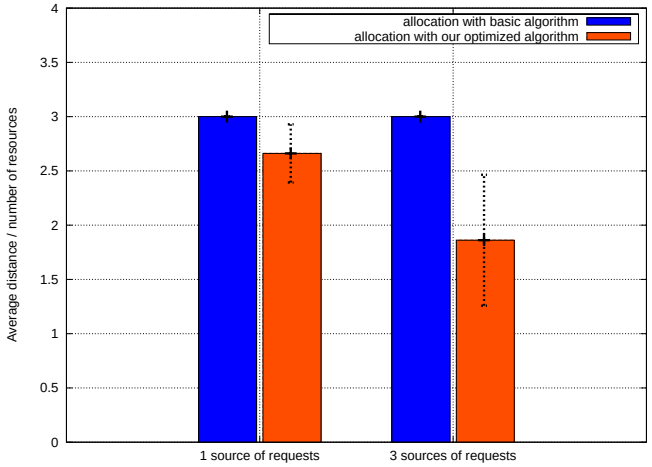


Figure 5.13: Aggregated distance of VI components allocated on a medium-size physical substrate. The number of sources requesting VIs is 1 and 3.

the number of sources submitting VI requests is augmented.

5.5.4 Comparing VXAlloc with non-sharing allocation

A complementary study was performed to compare the percentage of wasted capacity and the acceptance ratio of VXAlloc (a virtualized allocation) and non-sharing allocation. For that, we simulated a physical substrate network comprising 5 racks, each one composed of 24 nodes, interconnected by a hypercube network. Every physical node disposes of a 8 core 2.4 GHz CPU, and a memory capacity of 32 GB. Physical nodes of a rack are interconnected with a bandwidth capacity of 1 Gbps, while the capacity between racks is 10 Gbps.

The virtual infrastructures were also generated by the GT-ITM tool and follow the *normal model* [Calvert et al., 1997]. A set of virtual requests composed of 200 VI descriptions vary the number of virtual nodes (among 2, 4, and 8) following a uniform distribution. All virtual nodes require the provisioning of 1 core, varying the memory configuration among 4 GB, 8 GB, 16 GB, and 24 GB. We used the *load* notation to identify the percentage of total memory required by virtual resources when compared with the available 32 GB.

Results identified by *non-sharing allocation* are the result of requests that require *exclusivity* for all nodes, allocated by VXAlloc. Those with the label: *allocation with our optimized algorithm* are the result of the normal execution of VXAlloc. Note that by requiring *exclusity* for all virtual resources allocation, the physical substrate sharing is automatically disabled.

5.5.4.1 Evaluation

The first results identify the wasted capacity (in percentage) of both allocation solutions. Figure 5.14 presents this comparison. In the first graph (left side) the loads (percentage of virtual resources over the physical capacity) are analyzed individually. The virtual requests vary following the configuration described above. As expected, we observe better performance of VXAlloc in terms of physical substrate usage, highlighting one of the main advantages of virtualization: the possibility of sharing a physical resource among multiple

virtual resources, respecting the user's expectations. The wasted capacity decreases by 65.6%, 72.9%, and 87.26% for loads 0.125, 0.25, and 0.5, respectively. With load 0.75 both algorithms present the same wasted capacity due to the physical limitation.

The second graph of Figure 5.14 (right side) gives the comparison when load varies (respecting the values defined in the configuration). Following the same line, VXAlloc also presents better performance in terms of resource usage, decreasing the wasted capacity by 34.2%. In addition, we performed a comparison of the acceptance ratio of both scenarios. The first graph of Figure 5.15 (left side) shows that VXAlloc has higher acceptance ratio than the non-sharing allocation under all load configurations. When the load varies following a uniform distribution (right-side graph), VXAlloc also exceeds the non-sharing values by 178% percent.

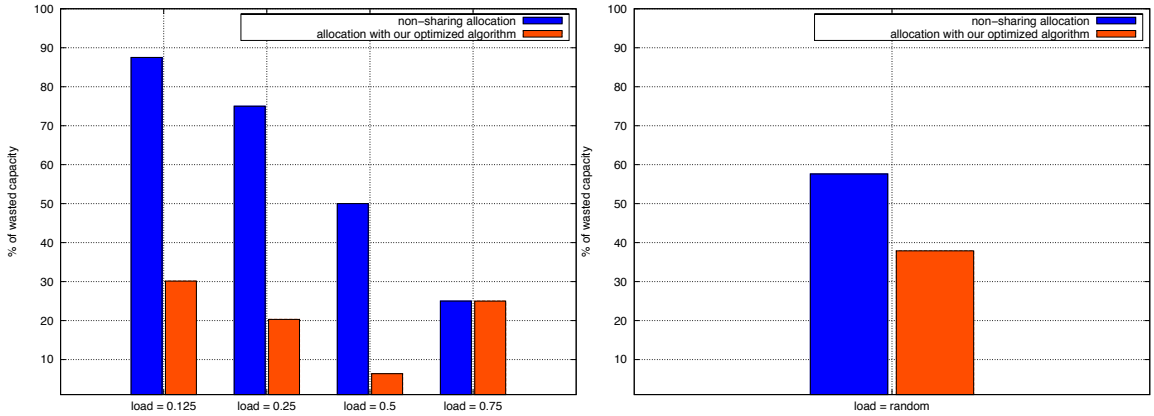


Figure 5.14: A comparison of wasted capacity between a non-sharing scenario and one virtualized, allocated using VXAlloc.

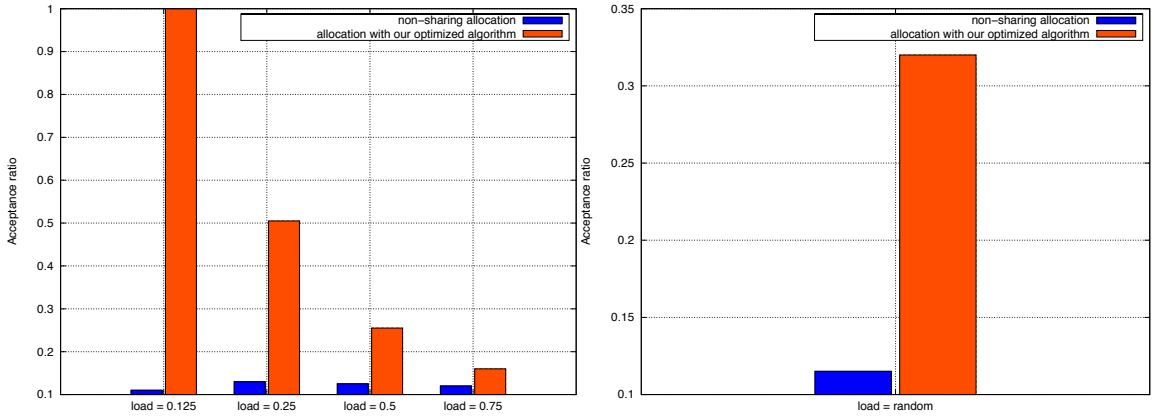


Figure 5.15: A comparison of the acceptance ratio between a non-sharing scenario and one virtualized, allocated using VXAlloc.

5.6 Conclusions

This chapter concentrates on the allocation of virtual infrastructures guided by the geographical location of virtual resources. Thanks to VXDL (as discussed in Chapter 3), users can interact with the allocation mechanism, informing specific attributes and setting constraints that represent their expectations and QoE requirements.

We formulated the allocation-problem and proposed metrics considering the users' (allocation quality) and the InPs' (cost and fragmentation) perspectives. An allocation heuristic that optimizes both perspectives has been implemented as a patented module of the LYaTiss Weaver. Our experiments show that it is possible to improve the virtual-infrastructure allocation's quality (approximately 39% for VIs allocated on a medium-size substrate), and simultaneously decrease the physical substrate's fragmentation and the substrate's cost (almost 28% and 21% on a medium-size substrate, respectively).

A future research line that we will explore is the interconnection of allocation strategies with information exposed by the InP management framework. For instance, the buffer capacity of virtual switches and routers are relevant information for the efficient provisioning of split and extended paths [Anhalt et al., 2010]. In addition, a different approach for modeling the allocation problem is planned as future work (discussed in Chapter 7).

Six

Reliable Virtual Infrastructures

The work presented in this chapter was published at the Second IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010) [7]. It has been done in collaboration with DoCoMo Labs USA (http://www.docomolabs-usa.com/).

6.1 Introduction

One of the main benefits of infrastructure virtualization is the capability of providing new services to accomplish the users' expectations. Among these new services we highlight the provisioning of transparent reliability. In other words, *reliability becomes a service provided by the InPs.*

Independent of the application context, some tasks are critical and their failure would cause the system to collapse. In addition, even the failure of non-critical tasks could significantly impact (for instance, delay) the completion of an application or service. In case of failure, a component of a virtual infrastructure can be migrated to a different location in the physical substrate. One key aspect in this scenario is for the infrastructure user to be able to specify the reliability associated with a task during the VI bootstrap, and for the InP to transparently provide the reliability to the user and effectively provision it through the allocation of virtual backup nodes ready to take over in case of node failure.

We discuss the motivations and gains of introducing customizable reliability of virtual infrastructures when executing large-scale distributed applications, and present a framework to specify, allocate and deploy virtualized infrastructure with reliability support. This chapter proposes an approach to efficiently specify and control the reliability at runtime. We illustrate the ideas described above by analyzing the introduction of reliability at the

VI level on a real application. Experimental results, obtained with the Bronze Standard application (described in Section 4.4.1 of Chapter 4) running in virtual infrastructures provisioned on the experimental large-scale Grid'5000 platform, show the benefits of the reliability service.

The rest of this chapter is organized as follows. Section 6.2 reviews the state of the art and technologies related to reliability on distributed systems. Section 6.3 motivates the introduction of reliability in virtual infrastructures, and identifies system goals which guide the design of our proposition. In Section 6.4, we discuss the issues associated with the translation of the reliability requirements, specified with VXDL, into a reliable VI. Section 6.5 describes the allocation and provisioning of a virtual reliable graph onto the physical substrate. In Section 6.6, we show experimental results with the Bronze Standard application as well as a cost analysis. We conclude this chapter and present perspectives for future work in Section 6.7.

6.2 State of the art

Providing reliability and availability on virtualized environments is an issue that has been studied in the recent years. Within virtual nodes, hypervisors such as Xen [Barham et al., 2003] provide the capability to store live snapshots of the virtual machines to reliable storage, which can be resumed on other physical nodes if failures occur. Remus [Cully et al., 2008] and Kemari [Tamura et al., 2008] improved static snapshots by periodically updating live snapshots to replica nodes that are on standby. From another perspective, proactive migration of virtual machines to other healthy nodes is considered upon early warnings of impending failures [Nagarajan et al., 2008] [Clark et al., 2005].

Also, recovering from failures has received some attention in the literature. On a large-scale execution environment, the re-submission mechanism is one of the solutions used to make the application continue running when a failure is detected [Lingrand et al., 2009a] [Lingrand et al., 2009b]. The application's makespan is longer in this case when the submitted task's execution time is long. Another possible solution is to periodically save static snapshots of the entire VI [Kangarlou et al., 2009] to disk, while execution is in progress. The live snapshots are reloaded as a new submission if failures are encountered in the current execution. The application's makespan then depends on the re-submission interval and the snapshot interval, which may be long due to disk-access times. Moreover, synchronizing clocks of reloaded virtual machines is critical as it can compromise the execution of time sensitive applications [Broomhead et al., 2010].

Fault tolerance is provided in some contexts, such as data centers [Mysore et al., 2009] [Guo et al., 2009]. However, it is achieved through specific engineering of the network nodes and links over-provisioned for redundancy. [Yeow et al., 2010] provides mechanisms to pool backup nodes to achieve a desired level of reliability. However, it is mostly a theoretical work and does not provide any vertical integration from the user specification to the physical substrate allocation. In [Menth et al., 2009], the authors focus on providing link reliability in wide-area network, by considering the most likely link failure combinations, and providing backup links for these failures. The reliability of virtual nodes is not addressed in this work.

These mechanisms, unfortunately, do not provide sufficient transparency against failures. Re-initiating or resuming applications at a later time to recover from failures will

impact any time-sensitive applications. Therefore, a live protection mechanism such as Remus [Cully et al., 2008] or Kemari [Tamura et al., 2008] is needed. In both Remus and Kemari, the memory state of a protected (critical) node is continuously *synchronized* with a replica (backup node), as with checkpointing. When a failure in the protected node occurs, the backup node can resume execution immediately, and the failover process can be made transparent to other nodes in the VI. This live protection mechanism has another advantage over prior snapshot mechanism: instead of the entire VI, only the critical nodes need to be checkpointed.

The key difference between Remus and Kemari is that Kemari initiates a checkpoint only when external events occur, such as disk writing and network-packet sending, whereas Remus checkpoints at a regular interval. One important feature of Remus is that, at every checkpoint, the external output is buffered locally in the critical node until it is assured that the backup node completes that checkpoint update. This ensures that any failover operation will be transparent to other unaffected nodes. Moreover, the protected node continues execution in parallel until the next checkpoint, thereby increasing system performance over classical lock-step checkpointing. Kemari, on the other hand, does not perform any buffering and relies on pausing the protected node to achieve the required transparency. We chose to use Remus over Kemari in our proof of concept as it provides a finer and customizable granularity between checkpoints, which can be as frequent as tens of milliseconds. As of Xen 4.0.0 [Barham et al., 2003], Remus is included in the official Xen releases.

6.3 Motivations and goals

Networking and computing infrastructures are subject to random failures of nodes and links. These failures are not rare in the case where the number of physical entities is large, especially in distributed systems. Currently, Cloud Computing providers have defined Service Level Agreements (SLAs) comprising the percentage of reliability, availability, or uptime of their systems. For instance, Amazon EC2 [AMAa] and Windows Azure [AZUa] inform an availability of 99.95%, while SalesForce [SAL] informs 99.9% of availability. However, the levels are not guaranteed: when an SLA is not respected (e.g., a failure occurs compromising the execution) a compensation in terms of credits for a new execution is offered to the user. For some users this compensation is sufficient, but for critical applications, a failure can compromise the entire execution.

The impact of a node failure to a distributed application can be very different; a failed worker node amongst hundreds of others is less significant than the failure of a database server. The reliability of a system may be evaluated quantitatively and qualitatively. The Mean Time Between Failures (MTBF) is a statistical metric to determine the failure rate of the underlying infrastructure, which can be evaluated by the InP's management system.

One approach is for the application designer to ensure reliability himself, by providing redundancy in the components and modules composing the application. However, this requires different sets of expertise: one is the expertise to design the system in order to deliver the intended application; another is to ensure that the components are integrated so as to support the desired reliability.

Furthermore, the actual reliability will depend on the physical resource upon which the system is deployed. If the application developer provides his own physical substrate

(servers and switches), then he can specify the reliability of each individual element. If on the other hand the system is deployed using a virtualized infrastructure, the reliability characteristics of the physical resources might be unknown.

Since it is common for the application developer to delegate the elastic provisioning of resources to the infrastructure provider, in order to only use the proper amount of resource in the face of varying demands, the corresponding provisioning of the reliability must by the same token be delegated as well. From the application provider's perspective, it is easier and more flexible to specify a level of reliability and have the reliable virtual infrastructure provided transparently as part of a SLA. On the other hand, from the InP's perspective, reliability becomes a service that can be added and that can generate new revenue streams. Further, the infrastructure provider is free to manage reliability in light of his own constraints and optimization opportunities: a backup might be associated to different resources from different independent applications. This multiplexing of the backup resources provides economy of scale to the infrastructure provider [Yeow et al., 2010].

These observations highlight a few requirements for a reliable virtualized infrastructure:

- the virtual infrastructure user should be able to specify reliability in a flexible and expressive manner (we exploit VXDL for specifying the reliability requirements, as discussed in Chapter 3);
- the InP should be able to implement and allocate reliable and backup resources efficiently (we extended the formulation presented in Chapter 5 as presented in Sections 6.4 and 6.5);
- both the virtual-infrastructure user and the physical-network provider need to see their business objectives satisfied (we discuss the results in Section 6.6).

Figure 6.1 summarizes the stages and requirements to provide reliable virtual infrastructures considering the application-provider specification. Our goal is to describe tools which allow to satisfy these requirements, and to deliver the efficient provisioning of reliability in a virtual infrastructure, as specified by the virtual infrastructure user.

6.4 From reliability requirements to reliable VIs

The VXDL language allows for describing the required reliability level for a VI. The application provider can set the reliability requirement individually for each virtual resource (nodes and links), or for a group of resources. This approach enables the composition of a VI with different requirements in terms of reliability level. For instance, an application with a master-worker architecture, e.g., MapReduce [Dean and Ghemawat, 2008], can require more reliability support for masters, and set the same reliability level for a group of workers. The example presented in Figure 6.2 illustrates the flexibility of the specification language. Part of a VXDL file, the example describes a group of 30 virtual nodes with a reliability specification of 99.9% (among other parameters).

The VXDL file containing the reliability requirements is interpreted by the VXDL parser [VXDLParser], a versatile tool that translates a VI specification into a resource request to the InP management framework. Specifically, it analyzes the VI specification,

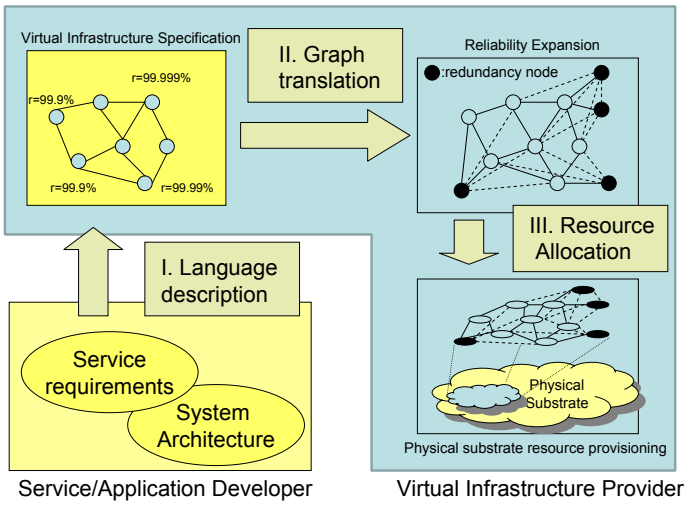


Figure 6.1: The integration of reliability service from the user specification to the VI allocation.

```
...
<vGroup id="workers" multiplicity="30">
 <vNode id="worker">
  <reliability>99.9%</reliability>
  <memory>
    <simple>512</simple>
    <unit>MB</unit>
  </memory>
  <cpu>
   <cores>1</cores>
   <frequency>
     <simple>1.0</simple>
     <unit>GHz</unit>
   </frequency>
  </cpu>
 </vNode>
</vGroup>
...
```

Figure 6.2: Part of a VXDL file exemplifying the the description of reliable virtual resources: a group of critical resources requires a reliability level of 99.9%.

automatically fills in any missing components (e.g., default elements and values by some predefined templates), and translates the VI into a graph representation for resource allocation (see next section). Furthermore, automated inclusion of backup components into the graph for targeted reliabilities is added onto the VXDL parser. The procedure is described below.

6.4.1 Automatic generation of backup nodes

A targeted reliability, in general, can be achieved with sufficient backup nodes. A critical node with a low MTBF will require more backup nodes on standby (synchronized with the active node) than another node with a higher MTBF for the same reliability level, if physical failures are independent. As noted in [Yeow et al., 2010], backup nodes can be shared among different groups of critical nodes to minimize the total number of backup nodes (and hence, minimal idle nodes).

For example, a VI has two groups of critical nodes with n_1 and n_2 critical nodes

respectively, requires at least r_1 and r_2 reliability respectively, and k_1 and k_2 backup nodes respectively. It is possible to share the backup nodes for $n_1 + n_2$ nodes such that the total number of backup nodes is lower than $k_1 + k_2$ provided that every backup node is a standby for all other critical nodes. In [Yeow et al., 2010], the Opportunistic Redundancy Pooling (ORP) mechanism imposes a sharing policy between groups of critical nodes such that it is possible to have $\min(k_1, k_2)$ backup nodes so long as the reliabilities of every group is satisfied.

The VXDL parser implements ORP to evaluate the number of backup nodes required. Since ORP assumes independent physical failures, it also generates additional physical-embedding constraints such that the physical locations of all shared backup nodes and critical nodes validate that assumption. For example, virtual nodes may not be embedded onto the same physical host, or rack that is connected to the same switch, or power supply.

6.4.2 Backup links: consistent network topology

Failovers from the critical nodes to backup nodes are expected to be transparent to the unaffected nodes of the VI. While the virtual machine protection tool (e.g., Remus) guarantees the failover time in tens of milliseconds and consistency across the VI through output buffering, consistency in the network topology has to be guaranteed through additional links to the backup nodes. That is, failed critical nodes which are resumed at the backup nodes must be connected to the rest of the VI as described in the original specification.

To ensure failover transparency, the additional backup links are pre-allocated (together with the VI) rather than on demand after failures occurred. In the latter case, resources for backup nodes cannot be guaranteed and, even if sufficient resources are available, undesired delays may be incurred during failover. Furthermore, active synchronization from the critical nodes to backup nodes consume bandwidth, which has to be allocated as well. Harary and Hayes [Harary and Hayes, 1996] have devised methods to minimize the number of additional links required. Specifically, a new graph G' is constructed with $n + k$ nodes such that the original VI is always a subgraph of G' when *any* k nodes are removed. Unfortunately, this class of solutions is infeasible in our system:

1. Guaranteeing that the VI is a subgraph of G' only ensures that the graph after k node failures is isomorphic. Hence, recovering from failures may result in unaffected nodes being moved around in order to recover the VI.

2. Exact solutions are found only for regular graphs such as rings, lines, square-grids and trees. For general graphs, heuristics are used [Dutt and Mahapatra, 1997] [Ajtai et al., 1992].

3. The solution assumes unweighted links; adding weights on top of the solution introduces an additional layer of complexity.

As such, similar to the approach in [Yeow et al., 2010], we add i) links from nodes of the VI to backup nodes such that every backup node is linked to neighbors of critical nodes, and ii) links interconnecting the backup nodes since two critical nodes which are neighbors of each other may fail simultaneously. We call the former set *first-order* backup links and the latter set *second-order* backup links. The second-order links are only required if two critical nodes are linked in the original VI.

In addition to the results of [Yeow et al., 2010], we reuse the first-order backup links for synchronization between critical nodes and backup nodes whenever possible. Algorithm 6.1 shows the procedure for the generation and reuse of first-order backup links and their attributes: bandwidth (bw) and latency (lat). We omit details on the generation of second-order links and the remaining synchronization links that were not reused from the first-order links, since the procedure is similar to that of the first order.

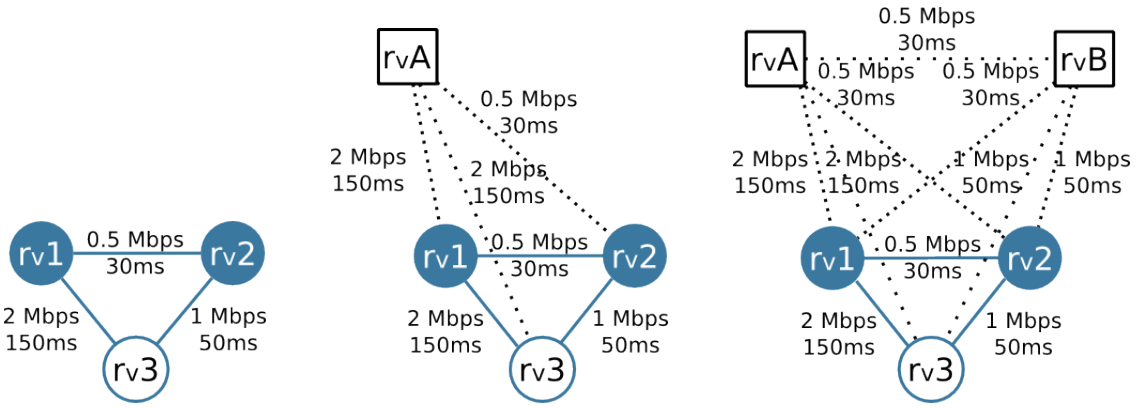
Algorithm 6.1: Generating First-Order Backup links

```
1  R^b: set of backup nodes;
2  C(b): set of critical nodes which uses b as a backup node;
3  L^v: set of links in VI ;
4  for b ∈ R^b do
5      for (i, j) ∈ L^v do
6          if i ∈ C(b) then
7              bw(b, j) ← bw(b, j) + bw(i, j);
8              lat(b, j) ← min {lat(b, j), lat(i, j)};
9              if j ∈ C(b) then
10                 Label (b, j) as synchronization link;
11                 Ensure bw and lat suffice for Remus;
```

Figure 6.3 shows an example of how backup links are generated. Nodes r_v1 and r_v2 request a backup node each one, identified as r_vA and r_vB, respectively. A new link between a backup node and some other node is created if it is a neighbor of a critical node. Hence, in Figure 6.3b, node r_vA connects to all three nodes. Furthermore, the attributes of link (r_vA, r_v3) can function as links (r_v1, r_v3) or (r_v2, r_v3). Links (r_vA, r_v1) and (r_vA, r_v2) are reused for synchronization. With one more backup node (as in Figure 6.3c), the first-order backup links of node r_vB are the same as those of node r_vA, and with a second-order backup link between node r_vA and r_vB to function as the link (r_v1, r_v2) when both critical nodes fail.

6.5 Reliable VI provisioning

Translating a VI to a graph representation results in a unified input to a resource allocation manager, regardless whether a VI requires reliability support. This immediately translates the resource allocation problem into a graph embedding problem. However, reliability support demands tighter allocation constraints, which are not present in VI's that does not require any reliability guarantee. That is, virtual nodes should be mapped in a way that virtual node failures resulting from the physical substrate should be independent. Then, virtual nodes of the same VI should not be allocated onto the same physical node. Moreover, the framework management should be able to provision and control the execution of reliable virtual infrastructures, identifying the occurrence of physical resources failures, and monitoring the efficiency of the service execution. In this section we describe the graph constraints related to the reliable VI provisioning, and describe the implementation of this service using the HIPerNet framework.



(a) Original VI (b) Add one backup (c) Add two backups

Figure 6.3: The figures show the steps (from left to right) as each backup node is added to the original VI for reliability. Nodes r_v1 and r_v2 are critical nodes, and nodes r_vA and r_vB are backup nodes. Backup links are used for synchronization (in dotted lines), and the respective attributes are determined by the existing links in the original VI.

6.5.1 Graph embedding and mapping constraints

In Chapter 5 we presented an algorithm to allocate virtual infrastructures based on subgraph-isomorphism detection. In this section, we extend that formulation by adding the constraints related with reliability support. These constraints complement the generation of backup links described by Algorithm 6.1 and guarantee the allocation of reliable virtual nodes. While the classical graph embedding problem enforces virtual nodes to be placed only onto unique physical nodes, it is insufficient to assure independent failures.

In the example presented in Figure 6.4, the physical substrate of Figure 6.4a must host the virtual infrastructure of Figure 6.3a. The nodes to be protected r_v1 and r_v2, and backup node r_vB would need to be placed on different physical resources, hence creating additional mapping constraints to the graph embedding problem.

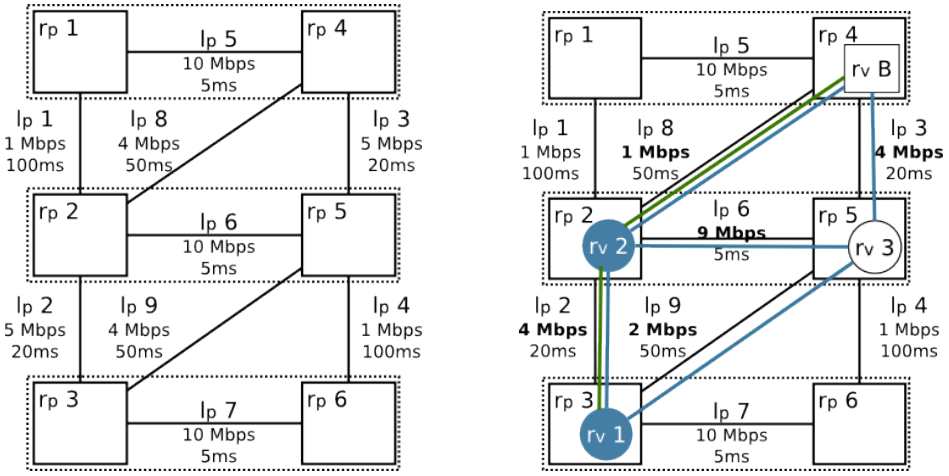
We formulate these conditions according to the notation used in Chapter 5 (refer to Table 5.3). Initially, let $B(r_i)$ be the set of backup nodes of virtual resource $r_i \in R^v$. An efficient mapping solution will not allow the sharing of physical resources among r_i and the set of backup nodes $R(r_i)$. Also, the physical resources mapping the backup nodes must have enough capacity to allocate the original virtual component in the event of a failure. More specifically, the conditions are:

$$\mathcal{M}_R(r_i) \neq \mathcal{M}_R(r_j), \forall r_i \in R^v, \forall r_j \in B(r_i); \tag{6.1}$$

$$\mathcal{M}_R(r_i) \neq \mathcal{M}_R(r_j), \forall r_k \in R^v, \forall r_i \in B(r_k), \forall r_j \in B(r_k); \tag{6.2}$$

$$Q_R(\mathcal{M}_R(r_k), t) \geq Q_R(r_i, t), \forall r_i \in R^v, \forall r_k \in B(r_i), \forall t \in [0, T]. \tag{6.3}$$

The allocation mapping presented in Figure 6.4b follows the reliable allocation conditions discussed above. Virtual resources and their relative backups are not sharing the physical resources, and the virtual links were allocated to keep the network topology consistent in the case of failures.



(a) Physical substrate. (b) Embedded Graph.

Figure 6.4: Example of embedding a graph in Figure 6.3b onto a physical substrate represented by (a), respecting the reliability constraints: original and backup nodes are not placed in the same physical resource.

6.5.2 Mapping solution

The graph embedding problem is well-known to be NP-hard [Chowdhury and Boutaba, 2010]. The additional allocation constraints between virtual nodes, however, does not make the problem less complex since solution space remains the same even though the search space may be reduced. To this end, we implemented the allocation constraints to the VXAlloc software. The subgraph isomorphism detection implemented by VXAlloc is relatively straightforward to incorporate the additional allocation constraints.

We briefly review the graph embedding method as follows. It is essentially a depth-first search that looks at all possible node mappings and eliminate the choices based on feasibility of virtual links emanating from the node in consideration. Initially, all possible node mappings are generated and sorted in some order that optimize some objective such as minimize cost and fragmentation, while maximizing the allocation quality. The sorting and ordering of physical candidates is limited to the number of candidates configured by the InP to avoid looking too deep into a search tree. We refer the reader to Chapter 5 for further details. For our purpose of incorporating the additional placement constraints, a filtering step is added to the list of generated possible mappings.

6.5.3 From mapping to provisioning

The mapping provided by the allocation step is interpreted and instantiated using the HIPerNet framework [1]. The HIPerNet framework combines system and networking virtualization technologies with bandwidth sharing and advance reservation mechanisms to offer dynamic networking and computing infrastructures as services [1] [16] [9].

The reliability mechanisms of HIPerNet presented in this work, are based on a modified version of Remus. To enable Remus protection, all VMs file-system were deployed on a Network File System (NFS) server. The first benchmarks performed with Remus

[1] We refer the reader to Section 2.5.3 in Chapter 2 for further details of the HIPerNet framework.

demonstrated that communication between the NFS server (source) and virtual machines (destination) should be limited to a maximum transfer rate of 100 Mbps. Otherwise, Remus cannot keep a stable copy of the critical VM for the default checkpoint intervals of 200ms. In this work, the reliability mechanisms are applied to virtual nodes. The protection of network communication and data persistent on NFS are out of scope and they are not discussed.

6.6 Evaluation through a use case application

We now evaluated the reliability service by executing an existing large-scale distributed application, named *Bronze Standard*, for proof of concept purpose. The bronze standard [Glatard et al., 2006] technique, described in Section 4.4.1 of Chapter 4, tackles the difficult problem of validating procedures for medical-image analysis.

The bronze standard application can be represented as a workflow of computational processes with I/O data dependencies, as illustrated in Figure 4.2. In the experiments reported below, this workflow is enacted with the data-intensive Grid-interfaced MOTEUR workflow manager [Glatard et al., 2008]. A clinical database of 59 pairs of patient images to be registered by the different algorithms involved in the workflow is used. Each service depicted in Figure 4.2 is instantiated as an independent computing task that is delegated to one of the computing nodes. For each run, the processing of the complete image database results in the generation of 354 computing tasks (with a computation time of 30 seconds to 5 minutes each on a state-of-the-art PC). The data volume transferred for each task is in the order of 30 MB. The makespan of the application's execution is in the order of 20 minutes in the absence of failures.

The experiments are carried out using VIs managed by HIPerNet within the Grid'5000 testbed [Cappello et al., 2005]. In our experiments, we use 100 physical nodes to compose a pool of virtualized physical resources. The number of physical nodes exceeds that of virtual resources specified (see next section) because virtual machines of *the same application* cannot be co-located in the same physical node to prevent correlated failures and additional virtual backup nodes are needed to protect the critical nodes.

Faults are simulated by shutting down physical machines respecting the MTBF parameter. The MTBF of each node in each experiment is 60000s, 30000s and 15000s. Assuming the largest makespan to be 30 minutes, the failure probability of each node is then 0.03, 0.06 and 0.12, respectively. The initial MTBF value (60000s) is based on failure rate of servers (with a probability between 0.02 and 0.04) identified by [Atwood and Miner, 2008]. The other lower MTBF values represent worse failure rates which could be attributed to a variety of reasons. Some examples are improper cooling in racks, irregular maintenance and inadequate protection from power interruptions.

6.6.1 VI composition

The VI specification to *bronze standard* is based on the results presented in Chapter 4, where 31 virtual resources are configured with 512 MB of RAM, and 1 GHz of CPU, and 10 Mbps of bandwidth requirement for each virtual link between the database and the workers. Virtual nodes require exclusivity on physical nodes. As shown in Figure 4.16, the HIPerNet engine deploys and manages virtual machines on these computers on demand (dark arrows).

The MOTEUR workflow engine, as a client of the HIPerNet framework, was hosted on one physical host outside the VI. MOTEUR produces VXDL descriptions including the reliability requirements that are requested to the HIPerNet engine (blue connection). After receiving all virtual machines allocated to the VI, MOTEUR connects to the computing nodes (worker nodes) to invoke the application services (red connections). The computing nodes connect to the database host to copy the input data and send the computational results, and the final results are sent to MOTEUR (green connections).

6.6.2 Cost model

From an infrastructure provider point of view, the major challenge is to account (financially or not) for resource usage according to specific criteria (e.g., fair share among users, digressive price, reliability level etc). Although a quasi-unlimited amount of computing resources may be allocated, a balance has to be found among i) the allocated infrastructure cost, ii) the expected performance, and iii) the optimal performance achievable, which depends on the level of parallelization of the application.

Considering this scenario, we introduce a simple cost model for the pricing of a VI with reliability support. The InP estimates the provisioning cost of an extended VI (already with backup resources identified). We consider different prices for active and backup resources.

Following the notation presented in Table 5.3 (Chapter 5), we define a price function $\Psi_R(Q_R(r,t),t)$ which sets the price for an amount of resource r at time t. Similarly, $\Psi_L(Q_L(l,t),t)$ would set a price for the bandwidth. The total price for the use of the resource is thus, over the lifetime $[0,T]$ of the virtual infrastructure G^v:

$$P_G(T) = \int_0^T \left(\sum_{i \in R^v} \Psi_R(Q_R(r_i,t),t) + \sum_{j \in L^v} \Psi_L(Q_L(l_j,t),t)\right) dt \tag{6.4}$$

Introducing link and node redundancy to increase the reliability corresponds to an additive cost to the user which has to be evaluated. The cost function is extended to calculate the total price ($P_{G'}(T)$) for the extended graph ($G^{v'}(R^{v'}, L^{v'})$), including reliable resources (R^b, L^b). The price of reliability (P_B) of a virtual infrastructure is given by

$$P_B(T) = \int_0^T \left(\sum_{i \in R^b} \Psi_R(Q_R(r_i,t),t) + \sum_{j \in L^b} \Psi_L(Q_L(l_j,t),t)\right) dt \tag{6.5}$$

and the total price of a reliable virtual infrastructure is of course $P_{G'}(T) = P_G(T) + P_B(T)$.

For a first order assessment of the performance of our model, we consider a pricing model based on the published prices of Amazon EC2 [AMAa] for Europe. A detailed economic analysis is outside the scope of this document and we set $\Psi_R(Q_R(r,t),t)$ to correspond to a fixed price per hour use for one of two types of nodes, and one of two types of contract: basic node (with 1.7 GB RAM) and high performance node (with 7.5 GB RAM); short term lease, and long term lease, respectively. Those prices are given in Table 6.1. EC2 does not charge any link cost in between nodes of its data center, and since the data transfers in our application can be fulfilled by typical ethernet links, we do not

include any specific link pricing in our basic cost analysis. For EC2-like infrastructures, there is a cost and delay associated with uploading the medical images to process up to the data center over the Internet, however this is independent of the reliability and outside of the scope discussed here.

VM Specifications	1.7 GB RAM	7.5 GB RAM
Short term lease	$0.095	$0.38
Long term lease	$0.031	$0.031

Table 6.1: Amazon EC2 Europe prices for VMs (per hour or part thereof).

We consider prices for the user, but an analysis of the costs to the provider would yield similar results. Our intent is to provide rough estimates to illustrate the trade-off between resource and reliability.

6.6.3 Experimental results

The application makespan when the application is executed on a substrate without simulated failures is 1205s ± 40s, serving as the baseline. For these values, the regular cost of this VI without reliability support is $2.95 (short term lease), serving as base cost for analysis.

The first experiment examines the protection of the database node. In this case, the database is the unique component protected, and faults are submitted in accordance with MTBF definition. Table 6.2 summarizes the execution of this scenario. The application makespan increases proportionally to the number of failures detected on the database node. Comparing with the baseline, the application makespan increases by +16%, +26% and +40% with regard to the MTBF values, 60000s, 30000s and 15000s, respectively.

In our experimental setup, we provided reliability by backing up the database 1:1, and the price for all values of the MTBF would be $3.04. However, while 1:1 replication made our proof-of-concept implementation feasible (the current Remus implementation for Xen 3.4 is limited to a 1:1 protection), it does not keep the required reliability at the specified level. To calculate the theoretical price of each VI with the proper reliability support, we compute the number of backup nodes required to provide the reliability level of 99.99% as a function of the MTBF, computed according to [Yeow et al., 2010]. For this scenario, the cost of database protection with reliability level 99.99% increases the VI cost by about 6%, 10%, and 13% for MTBF 60000s, 30000s, and 15000s, respectively (see Table 6.4). If we assume that the backup nodes are selected by the InP from a pool of nodes reserved for this purpose with a long-term lease, then the price of reliability amounts to an increase of 2%, 3%, and 4% for MTBF 60000s, 30000s, and 15000s, respectively (again, see Table 6.4).

Each workflow service has a pre- and post-processing stage where the input data is copied to worker nodes and the results are sent to the database. The more failures happen during these two stages, the more the application makespan increases. In Table 6.3, we present the data transfer time (in seconds) of this scenario. The data transfer time increase dominates when there are more failures detected on database node.

The second experiment analyzes the protection of workers nodes. The MTBF varies according to the failure model presented above. After an MTBF, a random physical machine is crashed. The backup virtual machine is automatically started and continues

MTBF	DB	Increase	CN	Increase
∞	1205s		1205s	
60000s	1401s	16.26%	1208s	0.2%
30000s	1524s	26.47%	1225s	1.7%
15000s	1688s	40.08%	1244s	3.2%

Table 6.2: Execution time and % increase over baseline for critical database protection only (column DB), and for computing nodes protection (column CN).

MTBF	Total data transfer time
∞	165.02s ± 44.30s
60000s	190.20s ± 96.75s
30000s	292.96s ± 115.38s
15000s	299.61s ± 128.26s

Table 6.3: Total data transfer time of application services running with critical database protection scenario.

running the same workflow task. As presented in Table 6.2, the application makespan slightly increases with regard to the number of failures detected on the infrastructure. The delay on the backup node activation is compensated by other parallel executions. Providing reliability for workers nodes (99.9%) dramatically decreases the time to complete the application, from execution time for the 15000s MTBF of 1688s down to 1244s in Table 6.2, a gain of almost 40%. Table 6.5 shows the price increase due to reliability for the different values of the MTBF, assuming that the backup nodes are drawn from the same (short term lease) pool as the rest of the virtual infrastructure, or from a long term lease pool set aside by the InP.

In both cases, database protection and workers protection, the application ran normally, with faults being transparent to the application provider.

			Short term		Long term	
MTBF	p_{FAIL}	n_{r_b}	$P_{G'}$	$P_B/P_{G'}$	$P_{G'}$	$P_B/P_{G'}$
60000s	0.03	2	\$3.13	6%	\$3.01	2%
30000s	0.06	3	\$3.23	10%	\$3.04	3%
15000s	0.12	4	\$3.33	13%	\$3.07	4%

Table 6.4: Price with reliability for database protection (reliability level 99.9%) and fraction of price corresponding to reliability with backup provisioned on short term leases or long term leases.

We also performed the experiments using the task resubmission mechanism (application level) to compare with the VI reliability service. In general, after a failure occurs on a worker node, a new worker node must be provisioned, and the task executed on the failed node has to be relaunched on the new worker node. We minimize the activation

			Short term		Long term	
MTBF	p_{FAIL}	n_{r_b}	$P_{G'}$	$P_B/P_{G'}$	$P_{G'}$	$P_B/P_{G'}$
60000s	0.03	5	\$3.42	16.1%	\$3.10	5.3%
30000s	0.06	8	\$3.71	25.8%	\$3.19	8.4%
15000s	0.12	12	\$4.09	38.7%	\$3.32	12.6%

Table 6.5: Price of reliability for computing node protection (reliability level 99.9%) and fraction of price corresponding to reliability with backup provisioned on short term leases or long term leases.

time of a backup node to zero by reserving, deploying and configuring backup nodes prior to the execution of the Bronze Standard. Hence, the only difference from the previous experiments is the time needed to rework the tasks on the failed worker nodes.

The number of backup nodes for task resubmission mechanism is set to be the same as that in the previous scenario (5, 8, and 12 for MTBF of 60000s, 30000s, 15000s, respectively) so that the cost and amount of resources used are equivalent. Our experimental results show that the application makespan increases significantly in comparing with the virtual infrastructure reliability service, +13.08%, +19.67% and +22.19% with respect to 60000s, 30000s and 15000s of the MTBF, as presented in Table 6.6. The makespan gap would have been more if backup nodes were not pre-allocated and configured. We do not present results otherwise since the time required for reservation, deployment and configuration may vary with the configuration and total utilization of the Grid.

MTBF	Reliability	Resubmission	Increase
60000s	1208s	1366s	+13.08%
30000s	1225s	1466s	+19.67%
15000s	1244s	1520s	+22.19%

Table 6.6: Application makespan with task resubmission mechanism and percentage increased when compared with VI reliability service.

6.7 Conclusions

We have presented a mechanism that introduces transparent reliability support into virtualized infrastructures. The transparency allows virtual infrastructure users to focus on application development and scale reliability requirements at deployment. The InP can provide reliability as a service, and implement reliability transparently from the point of view of the service operator.

Our mechanism relies on VXDL for describing the reliability requirements in a flexible and expressive manner. These requirements are translated into the real number of backup nodes and links that must be provisioned for guaranteeing the reliability level. The allocation algorithm (an extended version of VXAlloc, discussed in Chapter 5) allocates the extended VI (original resources and replicas). During the execution time, a

synchronization mechanism preserves virtual machine states in case of physical node failures. We implemented the mechanism on top of the HIPerNet framework, deployed over the Grid'5000 infrastructure, and demonstrated that it effectively supports reliability and enables the transparent execution of fault-sensitive distributed applications. In particular, our implementation reduces the completion time of the application under a slight increase of the resource cost.

Further work includes the implementation of a $n : k$ reliability ratio within our testbed, in order to fully benefit from the virtualization of reliability. This also involves implementing the sharing of redundant nodes across different virtual infrastructures to minimize the number of such nodes, as described in Section 6.4. In [Yeow et al., 2011], Wai-Leong Yeow et al. have started this work proposing some algorithms to implement $n : k$ reliability based on routing and code construction, highlighting the conditions to support optimal rates for synchronization and recovery.

For the cost benefit, we presented a simple yet promising back-of-the-envelope analysis. We would like to refine the model to better distinguish the economical trade-offs for each of the stakeholders: service customer, service provider and virtual infrastructure provider, in particular when the virtual infrastructure is hosted across different administrative domains.

Seven

Conclusions & Future work

7.1 Conclusions

Reviewing the advances in virtualization techniques, Cloud Computing, and especially the dynamic provisioning of Virtual Infrastructures, we note that the way we use Internet resources is changing. The reservation and provisioning of on-demand resources is becoming a reality as we can currently share the available computing and communication resources guided by pay-as-you-go economic models. Frameworks, such as the HIPerNet, have been designed to manage distributed and virtualized substrates on top of which dynamic virtual infrastructures can be provisioned, paving the road toward control and management of such hybrid entities. As these advances are recent and users are still migrating their applications from traditional computing platforms to the Cloud, a number of obstacles have to be addressed to facilitate the process.

Initially, we observed that the existing tools for describing a compute-and-communicate graph (the virtual infrastructure) did not meet all expectations of infrastructure providers and application architects. To overcome this limitation, we have proposed and developed a language for modeling and describing virtual infrastructures, the VXDL. VXDL has shown its strong potential allowing the specification of complex virtual infrastructures in a user-oriented manner. Various software and projects have been adopting this language for representing virtual infrastructures in different scenarios. In addition, as of writing of this manuscript, an open forum for discussions and improvements is being launched (`http://www.vxdlforum.org/`). This forum will allow developers and infrastructure providers to exchange experiences, suggestions, and extensions, guiding the standardization of VXDL.

Specifying VXDL files for complex distributed applications may be a difficult task. We developed a mechanism that automatically extracts information from workflow-based applications. Workflows are a representative way for describing the data and processes interaction (e.g., dependency and parallelism). Our mechanism combines information from the workflow with user's expertise, which comes from previous workflow executions or micro-benchmarks. The mechanism has been validated by executing a large-scale distributed application were different virtual infrastructure compositions have been compared. In addition to highlighting the real importance of performing an efficient VI composition, the results demonstrated the expressiveness of VXDL, which enabled several specifications corresponding to different strategies.

VIs are comprehensive entities carrying substantial information (e.g., resource capacities varying with time, location dependency, user's expectations). Consequently, the process of allocating these entities is more complex (e.g., search space, multiple objectives, more constraints on nodes and links) than provisioning the network or set of nodes. We addressed these issues by developing an allocation mechanism, the VXAlloc, which has proven to be efficient as it balances both users' and InPs' perspectives. The innovations of this solution were patented by INRIA and LYaTiss.

One of the great advantages of the on-demand model explored by the Cloud is the

provisioning of new services. An InP can develop internal mechanisms and commercialize them as new value-added services. Following this line, we proposed that reliability becomes a service. A user can express the reliability requirements of his application as well as any other configuration. The InP interprets these requirements and guarantees that a reliable virtual infrastructure is provisioned, keeping any substrate failures complete abstract to the user.

The contributions of this thesis aimed at making the dynamic provisioning of virtual infrastructures an easier task. It is important to observe that all contributions are articulated around the VXDL language. The language was the *enabler* for the proposed contributions since: i) it is powerful enough for composing different specifications for an application; ii) it enables a user to interact with the management framework by defining specific allocation's constraints directly within the VI specification; and iii) it has an important participation in the reliability service provisioning, enabling users to define the required reliability level for individual components.

7.2 Perspectives for future work

During this thesis we defined and investigated the dynamic provisioning of virtual infrastructures. However, some questions remain open, pointing to future directions of research. We select a few for discussing:

Exploring the elastic aspect of virtual infrastructures: The performance of an application can vary during the execution time due to changes in workload (increase or decrease), request peaks, among other reasons [Andrzejak et al., 2006, Arnaud and Bouchenak, 2010]. As VIs are malleable entities, they can be *elastic* for accommodating these dynamic variations. In this context, being elastic means to vary the capacity during execution (e.g., provision more bandwidth, memory, or even migrating a node close to user for decreasing the latency in communications).

We will concentrate on this important aspect of virtual infrastructures. As a continuation of the work performed in this thesis, the elasticity requirements (known in advance or on-demand) can be specified using VXDL. In addition, the allocation-problem was formulated with capacities specified as profiles (capacity per time), being capable of allocating elastic aspects known in advance. The allocation mechanism can be extended for adapting the dynamic variation that comes during the execution time.

Monitoring virtual infrastructures: For addressing the elastic aspects of a VI and for controlling the efficient usage of provisioned resources, a monitoring tool and a set of specific metrics should be investigated [Lahmadi et al., 2009]. The monitoring tools should consider sharing of resources between VIs. An InP should have knowledge about this information, following aggregated data, whereas a user should receive only information about his private resources [Clayman et al., 2010].

Moreover, a monitoring tool (or even a service) should give combined information about application and VI performance. A change in application behavior (e.g., peak usage, performance, load) will reflect in a change in VI composition (e.g., bandwidth, memory, storage). The reverse scenario may also occur.

Consequently, research is required for defining metrics and efficient tools for monitoring VIs. The contributions of this thesis are a starting point for developing these tools. Similarly to the reliability service, a monitoring service could be proposed, exploring VXDL as basis for formulating requirements and gathering measurement data.

A Mechanism Design Approach for allocating virtual infrastructures: The allocation of virtual infrastructures over distributed substrates may also address the economic aspects involved in dynamic provisioning. This problem can be modeled following a Mechanism Design Approach [Nisan et al., 2007], where users inform the requirements for composing their VIs while InPs expose their substrate composition and capacities. Then, an *allocation mechanism* solves the allocation problem among these participants, proposing *payment* values for users and provisioning costs for the InP. Users can evaluate the proposed solution based on a *utility function* used to identify if it is feasible. A similar approach can be followed for the InP, which would identify if the proposed solution meets its objectives.

These research lines can also explore VXDL for declaring VI's requirements, and extend the allocation-problem formulation addressed in this thesis.

A

VXDL descriptions of Chapter 3

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ens-lyon.fr/LIP/RESO/Software/vxdl VXDL.xsd">
  <virtualInfrastructure id="VISUPIPE" owner="CARRIOCAS project">
    <user>koslovski</user>
    <startDate>2011-04-21T08:00:00</startDate>
    <totalTime>PT11H</totalTime>
    <vGroup id="Storage cluster" multiplicity="20">
      <vNode id="nodes storage cluster">
        <storage>
          <interval>
            <min>200</min>
          </interval>
          <unit>GB</unit>
        </storage>
      </vNode>
    </vGroup>
    <vGroup id="Filtering cluster" multiplicity="40">
      <vNode id="nodes filtering cluster">
        <memory>
          <interval>
            <min>4</min>
          </interval>
          <unit>GB</unit>
        </memory>
        <cpu>
          <cores>1</cores>
          <frequency>
            <interval>
              <min>2.0</min>
            </interval>
            <unit>GHz</unit>
          </frequency>
        </cpu>
      </vNode>
    </vGroup>
    <vGroup id="Mapping cluster" multiplicity="30">
      <vNode id="nodes mapping cluster">
        <memory>
          <interval>
            <min>4</min>
          </interval>
          <unit>GB</unit>
        </memory>
        <cpu>
          <cores>1</cores>
          <frequency>
            <interval>
              <min>2.0</min>
            </interval>
            <unit>GHz</unit>
          </frequency>
        </cpu>
        <storage>
          <interval>
            <min>80</min>
          </interval>
          <unit>GB</unit>
        </storage>
      </vNode>
    </vGroup>
```

Figure A.1: VXDL description for the VISUPIPE example (part 1 of 4).

```
<vGroup id="Rendering cluster" multiplicity="10">
  <vNode id="nodes rendering cluster">
    <memory>
      <interval>
        <min>8</min>
      </interval>
      <unit>GB</unit>
    </memory>
    <cpu>
      <cores>1</cores>
      <frequency>
        <interval>
          <min>2.0</min>
        </interval>
        <unit>GHz</unit>
      </frequency>
    </cpu>
  </vNode>
</vGroup>
<vGroup id="Visualization cluster" multiplicity="30">
  <vNode id="nodes visualization cluster">
    <location>lyon.france.eu</location>
    <devices>visualization wall</devices>
    <memory>
      <interval>
        <min>8</min>
      </interval>
      <unit>GB</unit>
    </memory>
    <cpu>
      <cores>1</cores>
      <frequency>
        <interval>
          <min>2.0</min>
        </interval>
        <unit>GHz</unit>
      </frequency>
    </cpu>
  </vNode>
</vGroup>
<vLink id="link 1 Latency 0.200 ms">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Storage Cluster</source>
  <destination>Storage Cluster</destination>
</vLink>
<vLink id="link 2 Latency 0.200 ms">
  <latency>
    <interval>
      <max>0.200</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Visualization Cluster</source>
  <destination> Visualization Cluster</destination>
</vLink>
<vLink id="link Latency 0.400 ms">
  <latency>
    <interval>
      <max>0.400</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <bandwidth>
    <forward>
      <interval>
        <min>10.0</min>
      </interval>
      <unit>Mbps</unit>
    </forward>
  </bandwidth>
  <source>Filtering Cluster</source>
  <destination>Filtering Cluster</destination>
</vLink>
```

Figure A.2: VXDL description for the VISUPIPE example (part 2 of 4).


```
<vLink id="link Latency 0.150 ms">
  <latency>
    <interval>
      <max>0.150</max>
    </interval>
    <unit>ms</unit>
  </latency>
  <source>Rendering Cluster</source>
  <destination> Rendering Cluster</destination>
</vLink>
<vLink id="high capacity link 1">
  <bandwidth>
    <forward>
      <interval>
        <min>38.4</min>
      </interval>
      <unit>Gbps</unit>
    </forward>
  </bandwidth>
  <source>Storage Cluster</source>
  <destination>Filtering Cluster</destination>
</vLink>
<vLink id="high capacity link 2">
  <bandwidth>
    <forward>
      <interval>
        <min>38.4</min>
      </interval>
      <unit>Gbps</unit>
    </forward>
  </bandwidth>
  <source>Filtering Cluster</source>
  <destination>Mapping Cluster</destination>
</vLink>
<vLink id="high capacity link 3">
  <bandwidth>
    <forward>
      <interval>
        <min>38.4</min>
      </interval>
      <unit>Gbps</unit>
    </forward>
  </bandwidth>
  <source>Mapping Cluster</source>
  <destination> Rendering Cluster</destination>
</vLink>
<vLink id="high capacity link 4">
  <bandwidth>
    <forward>
      <interval>
        <min>38.4</min>
      </interval>
      <unit>Gbps</unit>
    </forward>
  </bandwidth>
  <source>Rendering Cluster</source>
  <destination>Visualization Cluster</destination>
</vLink>
<virtualTimeline id="VISUPIPE timeline">
  <timeline id="data transfer 1">
    <activate>Storage Cluster</activate>
    <activate>Filtering Cluster</activate>
    <activate>high capacity link 1</activate>
    <totalTime>PT1H</totalTime>
  </timeline>
  <timeline id="Filtering stage">
    <after>data transfer 1</after>
    <activate>Filtering Cluster</activate>
    <totalTime>PT2H</totalTime>
  </timeline>
  <timeline id="data transfer 2">
    <after>Filtering stage</after>
    <activate>Filtering Cluster</activate>
    <activate>Mapping Cluster</activate>
    <activate>high capacity link 2</activate>
    <totalTime>PT1H</totalTime>
  </timeline>
  <timeline id="Mapping stage">
    <after>data transfer 2</after>
    <activate>Mapping Cluster</activate>
    <totalTime>PT1H</totalTime>
  </timeline>
```

Figure A.3: VXDL description for the VISUPIPE example (part 3 of 4).


```
      <timeline id="data transfer 3">
        <after>Mapping stage</after>
        <activate>Mapping Cluster</activate>
        <activate>Rendering Cluster</activate>
        <activate>high capacity link 3</activate>
        <totalTime>PT1H</totalTime>
      </timeline>
      <timeline id="Rendering stage">
        <after>data transfer 3</after>
        <activate>Rendering Cluster</activate>
        <totalTime>PT2H</totalTime>
      </timeline>
      <timeline id="data transfer 4">
        <after>Rendering stage</after>
        <activate>Rendering Cluster</activate>
        <activate>Visualization Cluster</activate>
        <activate>high capacity link 4</activate>
        <totalTime>PT2H</totalTime>
      </timeline>
      <timeline id="Visualization stage">
        <after>data transfer 4</after>
        <activate>Visualization Cluster</activate>
        <totalTime>PT1H</totalTime>
      </timeline>
    </virtualTimeline>
  </virtualInfrastructure>
</description>
```

Figure A.4: VXDL description for the VISUPIPE example (part 4 of 4).

B Publications

International Journals

[1] Fabienne Anhalt, Guilherme Koslovski, and Pascale Vicat-Blanc Primet. Specifying and provisioning Virtual Infrastructures with HIPerNET. *ACM International Journal of Network Management (IJNM) - special issue on Network Virtualization and its Management*, 2010.

[2] Tram Truong Huu, Guilherme Koslovski, Fabienne Anhalt, Pascale Vicat-Blanc Primet, and Johan Montagnat. Joint elastic cloud and virtual network framework for application performance optimization and cost reduction. *Journal of Grid Computing (JoGC)*, 2010.

Conferences with Proceedings

[3] Fabienne Anhalt, Guilherme Koslovski, Marcelo Pasin, Jean-Patrick Gelas, and Pascale Vicat-Blanc Primet. Les Infrastructures Virtuelles à la demande pour un usage flexible de l'Internet. In *JDIR 09 : Journees Doctorales en Informatique et Reseaux*, Belfort, France, February 2009.

[4] Guilherme Koslovski, Sebastien Soudan, Paulo Gonçalves, and Pascale Vicat-Blanc. Locating Virtual Infrastructures: Users and InP Perspectives. In *12th IEEE/IFIP International Symposium on Integrated Network Management - Special Track on Management of Cloud Services and Infrastructures (IM 2011 - STMCSI)*, Dublin, Ireland, May 2011. IEEE.

[5] Guilherme Koslovski, Tram Truong Huu, Johan Montagnat, and Pascale Vicat-Blanc Primet. Executing distributed applications on virtualized infrastructures specified with the VXDL language and managed by the HIPerNET framework. In *First International Conference on Cloud Computing (CLOUDCOMP 2009)*, Munich, Germany, October 2009.

[6] Guilherme Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charão. VXDL: Virtual Resources and Interconnection Networks Description Language. In *The Second International Conference on Networks for Grid Applications (GridNets 2008)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Beijing, China, Oct. 2008. Springer Berlin Heidelberg.

[7] Guilherme Koslovski, Wai-Leong Yeow, Cedric Westphal, Tram Truong Huu, Johan Montagnat, and Pascale Vicat-Blanc Primet. Reliability support in virtual infrastructures. In *2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, Indianapolis, Indiana USA, Nov/Dec 2010. IEEE.

[8] Pascale Vicat-Blanc Primet, Fabienne Anhalt, and Guilherme Koslovski. Exploring the virtual infrastructure service concept in Grid'5000. In *20th ITC Specialist Seminar on Network Virtualization*, Hoi An, Vietnam, May 2009.

[9] Pascale Vicat-Blanc Primet, Jean-Patrick Gelas, Olivier Mornard, Guilherme Koslovski, Vincent Roca, Lionel Giraud, Johan Montagnat, and Tram Truong Huu. A scalable security model for enabling dynamic virtual private execution infrastructures on the internet. In *IEEE/ACM International Conference on Cluster Computing and the Grid (CCGrid2009)*, Shanghai, China, May 2009.

Research Reports and Posters

[10] Fabienne Anhalt, Christophe Blanchet, Romaric Guillier, Tram Truong Huu, Guilherme Koslovski, Johan Montagnat, Pascale Vicat-Blanc Primet, and Vincent Roca. HIPCAL: final report. Research Report, INRIA, 2010.

[11] Fabienne Anhalt, Romaric Guillier, Guilherme Koslovski, and Pascale Vicat-Blanc Primet. HIPerNet: virtual infrastructure manipulation. Practical Session in Grid'5000 Spring School, April 2010.

[12] Guilherme Koslovski and Pascale Vicat-Blanc Primet. Specifying virtual infrastructures using VXDL. In *Rescom Summer School*, June 2009. poster, RESCOM 2009.

[13] Tram Truong Huu, Johan Montagnat, Guilherme Koslovski, Romaric Guillier, and Pascale Vicat-Blanc Primet. Virtual resources allocation on Aladdin/Grid'5000 infrastructure through the HIPerNet middleware. Presentation in Grid'5000 Spring School, April 2010.

[14] Pascale Vicat-Blanc Primet, Jean-Patrick Gelas, Olivier Mornard, Guilherme Koslovski, Vincent Roca, Lionel Giraud, Roberto Podesta, and Victor Iniesta. Hipernet design document. Technical report, INRIA, July 2008. "Delivrable #3 : HIPCAL ANR-06-CIS-005".

[15] Pascale Vicat-Blanc Primet, Guilherme Koslovski, and Fabienne Anhalt. HIPCAL: Combined network and system virtualization. In *Colloque STIC de l'Agence Nationale de la Recherche (ANR)*, January 2010. poster, STIC 2010.

[16] Pascale Vicat-Blanc Primet, Guilherme Koslovski, Fabienne Anhalt, Tram Truong Huu, and Johan Montagnat. Exploring the Virtual Infrastructure as a Service concept with HIPerNet. Research Report RR-7185, INRIA, 2010.

[17] Pascale Vicat-Blanc Primet, Sebastien Soudan, and Guilherme Koslovski. Lyatiss tuner. In *2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2010)*. IEEE, 2010.

Patents

[VXAlloc] Koslovski, Guilherme and Vicat-Blanc Primet, Pascale. VXAlloc Patent - INPI: 10/01626, 2010, LYaTiss, INRIA ENS Lyon - Available: http://www.lyatiss.com

[VXCap] Vicat-Blanc Primet, Pascale and Koslovski, Guilherme and Soudan, Sebastien. VXCap Patent - INPI: 10/01624, 2010, LYaTiss, INRIA ENS Lyon - Available: http://www.lyatiss.com

Software

[VXDLParser] Koslovski, Guilherme and Vicat-Blanc Primet, Pascale. VXDL Parser APPcode: IDDN.FR.001.260009.000. S.P.2009.000.10800, 2009, LYaTiss, INRIA ENS Lyon - Available: http://www.lyatiss.com

[HIPerNet] Vicat-Blanc Primet, Pascale and Koslovski, Guilherme and Anhalt, Fabienne and Soudan, Sebastien and Guillier, Romaric and Martinez, Philippe and Mornard, Olivier and Gelas, Jean-Patrick. HIPerNet APPcode: IDDN.FR.001.260010.000.S.P.2009.000.10700, 2009, LYaTiss, INRIA ENS Lyon - Available: http://www.lyatiss.com

References

Online References

[4WA] The FP7 4WARD project. Available: http://www.4ward-project.eu/.

[AKA] Akamai: the leader in Web application acceleration and performance management, streaming media and content delivery. Available: http://www.akamai.com/.

[AMAa] Amazon Elastic Compute Cloud (EC2). Available: http://aws.amazon.com/ec2/.

[AMAb] Amazon EC2 API Tools. Available: http://aws.amazon.com/developertools/351.

[AMD] AMD Virtualization - Solutions for Business - Reducing Operating Costs while Increasing Business Value. Available: http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx.

[AZUa] Microsoft Windows Azure Platform. Available: http://www.microsoft.com/windowsazure/.

[AZUb] API references for Microsoft Windows Azure. Available: http://msdn.microsoft.com/en-us/library/ff800682.aspx.

[BPO] Microsoft Business Productivity Online Services and Software (BPOS). Available: http://www.microsoft.com/online/.

[CAR] CARRIOCAS project: Calcul Réparti sur Réseau Internet Optique à Capacité Surmultiplieé. Available: http://www.carriocas.org/.

[CON] Condor Project - High Throughput Computing. Available: http://www.cs.wisc.edu/condor/.

[COR] Common Object Request Broker Architecture (CORBA). Available: http://www.omg.org.

[ENO] Enomaly: Elastic Cloud Computing Platform. Available: http://www.enomaly.com.

[Euc] Eucalyptus Systems. Available: http://www.eucalyptus.com/.

[FLE] FlexiScale Public Cloud. Available: http://www.flexiant.com/products/flexiscale/.

[FOR] Force.com. Available: http://www.salesforce.com/platform/.

[GEN] Exploring Networks of the Future project (GENI). Available: http://www.geni.net/.

[GEY] Generalized Architecture for Dynamic Infrastructure Services (FP7 - GEYSERS). Available: http://www.geysers.eu/.

[GOG] Cloud Hosting, Cloud Computing, Hybrid Infrastructure from GoGrid. Available: http://www.gogrid.com.

[GOOa] Google App Engine. Available: http://code.google.com/appengine/.

[GOOb] Google Apps. Available: http://www.google.com/apps/.

[GRI] Aladdin-G5K: ensuring the development of Grid'5000. Available: https://www.grid5000.fr/.

[GWE] ANR-06-MDCA-009 GWENDIA project. Available: http://www.gwendia.polytech.unice.fr.

[HIP] HIPCAL project. Available: http://hipcal.lri.fr/.

[HYP] Microsoft Hyper-V Server. Available: http://www.microsoft.com/brasil/servidores/hyper-v-server/default.mspx.

[IEE] IEEE - Advancing Technology for Humanity. Available: http://www.ieee.org/.

[INT] Virtualization Technologies from Intel. Available: http://www.intel.com/technology/virtualization/.

[KVM] Kernel-Based Virtual Machines. Available: http://www.linux-kvm.org/.

[LIB] libvirt, virtualization API. Available: http://libvirt.org/index.html.

[MA] DIET workflow manager (DIET MA DAG). Available: http://graal.ens-lyon.fr/~diet/workflow.html.

[MOT] Data Intensive Grid-Enabled Workflow Manager (MOTEUR). Available: http://modalis.i3s.unice.fr/moteur2.

[myG] myGrid UK e-Science project. Available: http://www.mygrid.org/.

[NDL] Network Description Language (NDL). Available: http://www.science.uva.nl/research/sne/ndl.

[Net] Cloud ERP, Business Accounting Software, CRM, Ecommerce (NetSuite). Available: http://www.netsuite.com/.

[NML] Network Mark-up Language Working Group (NML-WG). Available: https://forge.gridforum.org/projects/nml-wg.

[NSI] Network Service Interface Working Group (NSI-WG). Available: http://forge.gridforum.org/sf/projects/nsi-wg.

[OCC] Open Cloud Computing Interface Working Group (OCCI-WG). Available: http://forge.gridforum.org/sf/projects/occi-wg.

[Opea] OpenNebula: the Open Source Toolkit for Cloud Computing. Available: http://opennebula.org/.

[OPEb] OpenVZ: container-based virtualization for Linux. Available: http://wiki.openvz.org/.

[QEM] Open Source Processor Emulator (QEMU). Available: http://wiki.qemu.org/.

[RACa] Dedicated Server, Managed Hosting, Web Hosting by Rackspace Hosting. Available: http://www.rackspace.com/.

[RACb] Rackspace Cloud Servers API. Available: http://www.rackspace.com/cloud/cloud_hosting_products/servers/api/.

[RDF] Resource Description Framework (RDF) / W3C Semantic Web Activity. Available: http://www.w3.org/RDF/.

[REN] Le Réseau National de télécommunications pour la Technologie l'Enseignement et la Recherche. Available: http://www.renater.fr/.

[RSp] RSpec Reference - protogeni. Available: http://www.protogeni.net/trac/protogeni/wiki/RSpecReference.

[SAI] Scalable and Adaptive Internet Solutions project (FP7 SAIL). Available: http://www.sail-project.eu/.

[SAL] CRM Salesforce. Available: http://www.salesforce.com/.

[SAP] SAP Business ByDesign. Available: http://www.sap.com/sme/solutions/businessmanagement/businessbydesign/index.epx.

[TER] Cloud Computing without Compromise (3Tera). Available: http://www.3tera.com.

[UML] The User-mode Linux Kernel Home Page. Available: http://user-mode-linux.sourceforge.net/.

[USD] Unified Service Description Language Incubator Group Charter (USDL). Available: http://www.w3.org/2005/Incubator/usdl/charter.

[vCl] VMware vCloud Director: Secure Private Clouds, Infrastructure as a Service. Available: http://www.vmware.com/products/vcloud-director/overview.html.

[Vir] VirtualBox. Available: http://www.virtualbox.org/.

[VMW] VMWare Server. Available: http://www.vmware.com.

[VPN] VPN Technologies: Definitions and Requirements - VPN Consortium. Available: http://www.vpnc.org/vpn-technologies.html.

[VSE] Linux VServers Project. Available: http://linux-vserver.org/.

[XCP] The Xen Cloud Platform (XCP). Available: http://www.xen.org/products/cloudxen.html.

References

[Agapi et al., 2009] Agapi, A., Soudan, S., Pasin, M., Vicat-Blanc Primet, P., and Kielmann, T. (2009). Optimizing deadline-driven bulk data transfers in overlay networks. In *ICCCN 2009 Track on Pervasive Computing and Grid Networking (PCGN)*, San Francisco, USA.

[Ajtai et al., 1992] Ajtai, M., Alon, N., Bruck, J., Cypher, R., Ho, C., Naor, M., and Szemeredi, E. (1992). Fault tolerant graphs, perfect hash functions and disjoint paths. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:693–702.

[Andersen et al., 2001] Andersen, D., Balakrishnan, H., Kaashoek, F., and Morris, R. (2001). Resilient overlay networks. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 131–145, New York, NY, USA. ACM.

[Anderson et al., 2005] Anderson, T., Peterson, L., Shenker, S., and Turner, J. (2005). Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4):34–41.

[Andreozzi et al., 2007] Andreozzi, S., Burke, S., Donno, F., Field, L., ans Jens Jensen Ans Balazs Konya, S. F., Litmaath, M., Manbelli, M., Schopf, J., Viljoen, M., Wilson, A., and Zappi, R. (2007). GLUE Schema Specification. Technical Report 1.3, GLUE Working Group.

[Androutsellis-Theotokis and Spinellis, 2004] Androutsellis-Theotokis, S. and Spinellis, D. (2004). A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371.

[Andrzejak et al., 2006] Andrzejak, A., Reinefeld, A., Schintke, F., and Schütt, T. (2006). On Adaptability in Grid Systems. In Getov, V., Laforenza, D., and Reinefeld, A., editors, *Proceedings of Dagstuhl Seminar 04451: Future Generation Grids*, Springer CoreGRID Series. Springer.

[Anhalt et al., 2010] Anhalt, F., Divakaran, D. M., and Vicat-Blanc Primet, P. (2010). A Virtual Switch Architecture for Hosting Virtual Networks on the Internet. In *11th International Conference on High Performance Switching and Routing*, Dallas, Texas, USA.

[Anjomshoaa et al., 2005] Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., and Savva, A. (2005). Job Submission Description Language (JSDL).

[Arnaud and Bouchenak, 2010] Arnaud, J. and Bouchenak, S. (2010). Adaptive Internet services through performance and availability control. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 444–451, New York, NY, USA. ACM.

[Atwood and Miner, 2008] Atwood, D. and Miner, J. G. (2008). White paper: Reducing Data Center Cost with an Air Economizer. Technical report, Intel.

[Audouin et al., 2009] Audouin, O., Bruno, S., Leclerc, O., Lu, H.-Q., Maibeche, K., Ruggeri, S., Verchere, D., Moal, J. L., Marcot, T., Meuric, J., Tardivel, J.-L., Thual, L., Bilhaut, R., Jouvin, M., Honore, P., Lafoucriere, J.-C., Mathieu, S., Meyer, J., Rodrigues, D., Guillermin, P., Mouton, C., Primet, P., Soudan, S., and Schmutz, A. (2009). CARRIOCAS project: An experimental high bit rate optical network for computing-intensive scientific and industrial applications. *Testbeds and Research Infrastructures for the Development of Networks & Communities, International Conference on*, 0:1–6.

[Barham et al., 2003] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA. ACM.

[Barrios Hernandez et al., 2010] Barrios Hernandez, C., Meneses, E., Denneulin, Y., and Plata, A. (2010). Cloud Computing Security, Before and After an Attack: Risk Mitigation and Forensic Analisys. In *Latin American Conference on High Performance Computing*, Gramado, Brazil.

[Bavier et al., 2004] Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. (2004). Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA. USENIX Association.

[Bavier et al., 2006] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J. (2006). In VINI veritas: realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–14. ACM.

[Benhaddou and Alanqar, 2007] Benhaddou, D. and Alanqar, W. (2007). Layer 1 Virtual Private Networks in Multidomain Next-Generation Networks. *Communications Magazine, IEEE*, 45(4):52–58.

[Bhatia et al., 2008] Bhatia, S., Motiwala, M., Muhlbauer, W., Valancius, V., Bavier, A., Feamster, N., Peterson, L., and Rexford, J. (2008). Hosting Virtual Networks on Commodity Hardware. Technical report, Georgia Tech Computer Science Technical Report GT-CS-07-10.

[Bin Tariq et al., 2009] Bin Tariq, M., Mansy, A., Feamster, N., and Ammar, M. (2009). Characterizing VLAN-induced sharing in a campus network. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 116–121, New York, NY, USA. ACM.

[Bittencourt and Madeira, 2010] Bittencourt, L. F. and Madeira, E. R. M. (2010). Towards the Scheduling of Multiple Workflows on Computational Grids. *Journal of Grid Computing (JOGC)*, 8(3):419441.

[Blythe et al., 2005] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., and Kennedy, K. (2005). Task Scheduling Strategies for Workflow-based Applications in Grids. In *International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 759–767.

[Braun et al., 2001] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. of Parallel and Distributed Computing (JPDC)*, 61(6):810–837.

[Broomhead et al., 2010] Broomhead, T., Cremean, L., Ridoux, J., and Veitch, D. (2010). Virtualize everything but time. In *Usenix Symposium on Operating Systems Design and Implementation (OSDI 2010)*, Vancouver, Canada.

[Butt et al., 2010] Butt, N. F., Chowdhury, M., and Boutaba, R. (2010). Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding. In *Proceedings of the 9th IFIP NETWORKING*. IFIP.

[Buyya, 2009] Buyya, R. (2009). Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:1.

[Cadere et al., 2008] Cadere, C., Barth, D., and Vial, S. (2008). Virtualization and allocation of network service resources using graph embedding. In *Computer and Information Sciences, 2008. ISCIS '08. 23rd International Symposium on*, pages 1–6.

[Calvert et al., 1997] Calvert, K., Doar, M. B., Nexion, A., Zegura, E. W., Tech, G., and Tech, G. (1997). Modeling Internet Topology. *IEEE Communications Magazine*, 35:160–163.

[Campbell et al., 1999] Campbell, A. T., De Meer, H. G., Kounavis, M. E., Miki, K., Vicente, J. B., and Villela, D. (1999). A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29(2):7–23.

[Cappello et al., 2005] Cappello, F., Caron, E., Daydé, M., Desprez, F., Jégou, Y., Primet, P., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quetier, B., and Richard, O. (2005). Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 8 pp.+.

[Carapinha and Jiménez, 2009] Carapinha, J. and Jiménez, J. (2009). Network virtualization: a view from the bottom. In *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, VISA '09, pages 73–80, New York, NY, USA. ACM.

[Caron and Desprez, 2006] Caron, E. and Desprez, F. (2006). DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 20(3):335–352.

[Chien et al., 2004] Chien, A., Casanova, H., suk Kee, Y., and Huang, R. (2004). The Virtual Grid Description Language: vgDL. Technical Report TR0.95, VGrADS Project.

[Chowdhury et al., 2010] Chowdhury, M., Samuel, F., and Boutaba, R. (2010). PolyViNE: Policy-based Virtual Network Embedding Across Multiple Domains. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architecture (VISA)*. ACM.

[Chowdhury and Boutaba, 2010] Chowdhury, N. M. K. and Boutaba, R. (2010). A survey of network virtualization. *Comput. Netw.*, 54(5):862–876.

[Chowdhury and Boutaba, 2009] Chowdhury, N. M. M. K. and Boutaba, R. (2009). Network Virtualization: State of the Art and Research Challenges. *IEEE Communications Magazine*, 47(7).

[Chowdhury et al., 2009] Chowdhury, N. M. M. K., Rahman, M. R., and Boutaba, R. (2009). Virtual Network Embedding with Coordinated Node and Link Mapping. *INFOCOM 2009. 28th IEEE International Conference on Computer Communications. Proceedings.*

[Clark et al., 2005] Clark, C., Fraser, K., Hand, S., Hansen, J. G., Jul, E., Limpach, C., Pratt, I., and Warfield, A. (2005). Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA. USENIX Association.

[Clayman et al., 2010] Clayman, S., Galis, A., Chapman, C., and Toffetti, G. (2010). Monitoring service clouds in the future internet. *Towards the Future Internet - Emerging Trends from European Research*, pages 115–126.

[Cordella et al., 2004] Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372.

[Cully et al., 2008] Cully, B., Lefebvre, G., Meyer, D., Feeley, M., Hutchinson, N., and Warfield, A. (2008). Remus: high availability via asynchronous virtual machine replication. In *Proceedings of USENIX NSDI*, pages 161–174.

[Dang and Altmann, 2009] Dang, M. Q. and Altmann, J. (2009). Resource allocation algorithm for light communication grid-based workflows within an SLA context. *International Journal of Parallel, Emergent and Distributed Systems*, 24(1):31–48.

[Dang and Hsu, 2008] Dang, M. Q. and Hsu, D. F. (2008). Mapping Heavy Communication Grid-Based Workflows Onto Grid Resources Within an SLA Context Using Metaheuristics. *International Journal of High Performance Computing Applications (IJHPCA)*, 22(3):330–346.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113.

[Deelman et al., 2003] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., and Koranda, S. (2003). Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing (JOGC)*, 1(1):9–23.

[der Ham et al., 2007] der Ham, J. V., Grosso, P., der Pol, R. V., Toonk, A., and de Laat, C. (2007). Using the Network Description Language in Optical Networks. In *Tenth IFIP/IEEE Symposium on Integrated Network Management.*

[Dias de Assunção and Buyya, 2009] Dias de Assunção, M. and Buyya, R. (2009). Performance analysis of allocation policies for interGrid resource provisioning. *Information and Software Technology*, 51(1):42 – 55. Special Section - Most Cited Articles in 2002 and Regular Research Papers.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

[Dijkstra et al., 2008] Dijkstra, F., Andree, B., Koymans, K., van der Ham, J., Grosso, P., and de Laat, C. (2008). A multi-layer network model based on itu-t g.805. *Comput. Netw.*, 52:1927–1937.

[Divakaran and Vicat-Blanc Primet, 2007] Divakaran, D. M. and Vicat-Blanc Primet, P. (2007). Channel Provisioning in Grid Overlay Networks. In *Workshop on IP QoS and Traffic Control.*

[DMTF, 2009] DMTF (2009). DMTF white paper DSP2017: Open Virtualization Format Specification (OVF).

[Duffield et al., 1999] Duffield, N. G., Goyal, P., Greenberg, A., Mishra, P., Ramakrishnan, K. K., and Van der Merive, J. E. (1999). A flexible model for resource management in virtual private networks. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 95–108, New York, NY, USA. ACM.

[Dutt and Mahapatra, 1997] Dutt, S. and Mahapatra, N. R. (1997). Node-covering, Error-correcting Codes and Multiprocessors with Very High Average Fault Tolerance. *IEEE/ACM Trans. Comput. Biology Bioinformatics*, 46(9):997–1015.

[Eriksson, 1994] Eriksson, H. (1994). MBONE: the multicast backbone. *Commun. ACM*, 37(8):54–60.

[Fan and Ammar, 2006] Fan, J. and Ammar, M. H. (2006). Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12.

[Farrel and Bryskin, 2005] Farrel, A. and Bryskin, I. (2005). *GMPLS: Architecture and Applications (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Feamster et al., 2007] Feamster, N., Gao, L., and Rexford, J. (2007). How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64.

[Festor et al., 2010] Festor, O., Cherkaoui, O., and Granville, L. Z. (2010). Special issue on virtualization. *International Journal of Network Management*, 20(3):109.

[Foster, 2001] Foster, I. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In Sakellariou, R., Gurd, J., Freeman, L., and Keane, J., editors, *Euro-Par 2001 Parallel Processing*, volume 2150 of *Lecture Notes in Computer Science*, pages 1–4. Springer Berlin / Heidelberg.

[Foster and Kesselman, 1997] Foster, I. and Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128.

[Foster, 2005] Foster, I. T. (2005). Globus Toolkit Version 4: Software for Service-Oriented Systems. In Jin, H., Reed, D. A., and Jiang, W., editors, *NPC*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer.

[Galán et al., 2009] Galán, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., and Vaquero, L. M. (2009). Service specification in cloud environments based on extensions to open standards. In *COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–12, New York, NY, USA. ACM.

[Garcia-Espin et al., 2010] Garcia-Espin, J. A., Riera, J. F., Lopez, E., Figuerola, S., Donadio, P., Buffa, G., Peng, S., Escalona, E., Soudan, S., Anhalt, F., Robinson, P., Antonescu, A.-F., Tzanakaki, A., Anastasopoulos, M., Tovar, A., Jimnez, J., Chen, X., Ngo, C., Ghijsen, M., Demchemko, Y., Landi, G., Lopatowski, L., Gutkowski, J., and Belter, B. (2010). GEYSERS Deliverable D3.1: Functional Description of the Logical Infrastructure Composition Layer (LICL). Available: http://www.geysers.eu/images/stories/deliverables/geysers-deliverable_3.1.pdf.

[GEYSERS, 2011] GEYSERS (2011). White paper: GEYSERS, the Revolution of Infrastructure Provisioning. Available: http://www.geysers.eu/.

[Glatard et al., 2008] Glatard, T., Montagnat, J., Lingrand, D., and Pennec, X. (2008). Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *International Journal of High Performance Computing and Applications (IJHPCA)*, 22(3):347–360.

[Glatard et al., 2006] Glatard, T., Pennec, X., and Montagnat, J. (2006). Performance evaluation of grid-enabled registration algorithms using bronze-standards. In *Proceedings of MICCAI*.

[Guo et al., 2009] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. In *ACM SIGCOMM '09*.

[Guo et al., 2006] Guo, W., Sun, W., Hu, W., and Jin, Y. (2006). Resource Allocation Strategies for Data-Intensive Workflow-Based Applications in Optical Grids. In *10th IEEE Singapore International Conference on Communication Systems (IEEE ICCS 2006)*, pages 1–5.

[Gupta et al., 2001] Gupta, A., Kleinberg, J., Kumar, A., Rastogi, R., and Yener, B. (2001). Provisioning a virtual private network: a network design problem for multicommodity flow. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 389–398, New York, NY, USA. ACM.

[Gupta et al., 2006] Gupta, D., Cherkasova, L., Gardner, R., and Vahdat, A. (2006). Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362, New York, NY, USA. Springer-Verlag New York, Inc.

[Haider et al., 2009] Haider, A., Potter, R., and Nakao, A. (2009). Challenges in Resource Allocation in Network Virtualization. In *20th ITC Specialist Seminar on Network Virtualization*.

[Handley, 2006] Handley, M. (2006). Why the Internet only just works. *BT Technology Journal*, 24:119–129.

[Harary and Hayes, 1996] Harary, F. and Hayes, J. P. (1996). Node Fault Tolerance in Graphs. *Networks*, 27(1):19–23.

[He et al., 2008] He, J., Zhang-Shen, R., Li, Y., Lee, C.-Y., Rexford, J., and Chiang, M. (2008). DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In *CoNEXT '08: Proceedings of the 2008 ACM CoNEXT conference*. ACM.

[Houidi et al., 2011] Houidi, I., Louati, W., Ameur, W. B., and Zeghlache, D. (2011). Virtual network provisioning across multiple substrate networks. *Computer Networks*, 55(4):1011 – 1023. Special Issue on Architectures and Protocols for the Future Internet.

[Houidi et al., 2008] Houidi, I., Louati, W., and Zeghlache, D. (2008). A Distributed Virtual Network Mapping Algorithm. *Communications, 2008. ICC '08. IEEE International Conference on*, pages 5634–5640.

[Houidi et al., 2010] Houidi, I., Louati, W., Zeghlache, D., Papadimitriou, P., and Mathy, L. (2010). Adaptive Virtual Network Provisioning. In *Proceedings of VISA Workshop*.

[Jacobs, 2005] Jacobs, D. (2005). Enterprise Software as Service. *Queue*, 3(6):36–42.

[Jannotti et al., 2000] Jannotti, J., Gifford, D. K., Johnson, K. L., Kaashoek, F. M., and O'Toole, J. W. (2000). Overcast: Reliable Multicasting with an Overlay Network. In *Usenix OSDI Symposium 2000*, pages 197–212.

[Jeong and Colle, 2010] Jeong, S. and Colle, D. (2010). IETF - Network Working Group - Virtual Networks Problem Statement.

[Kamp and Watson, 2000] Kamp, P. H. and Watson, R. N. M. (2000). Jails: Confining the omnipotent root. In *Proceedings of 2nd Intl. SANE Conference*.

[Kangarlou et al., 2009] Kangarlou, A., Eugster, P., and Xu, D. (2009). VNsnap: Taking Snapshots of Virtual Networked Environments with Minimal Downtime. In *Proceedings of IEEE/IFIP DSN*.

[Keller and Rexford, 2010] Keller, E. and Rexford, J. (2010). The "Platform as a Service" Model for Networking. In *USENIX, Internet Network Management Workshop / Workshop on Research on Enterprise Networking (INM/WREN)*.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, 36(1):41–50.

[Kertesz et al., 2007] Kertesz, A., Rodero, I., and Guim, F. (2007). BPDL: A Data Model for Grid Resource Broker Capabilities. Technical Report TR-0074, Institute on Resource Management and Scheduling - CoreGRID.

[Kesselman and Foster, 1998] Kesselman, C. and Foster, I. (1998). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers.

[Kim et al., 2010] Kim, W., Sharma, P., Lee, J., Banerjee, S., Tourrilhes, J., Lee, S.-J., and Yalagandula, P. (2010). Automated and scalable QoS control for network convergence. In *Proceedings of the 2010 INM/WREN*. USENIX Association.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220:671–680.

[Krishnamurthy et al., 2001] Krishnamurthy, B., Wills, C., and Zhang, Y. (2001). On the use and performance of content distribution networks. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, New York, NY, USA. ACM.

[Laganier and Vicat-Blanc Primet, 2005] Laganier, J. and Vicat-Blanc Primet, P. (2005). HIPerNet: A Decentralized Security Infrastructure for Large Scale Grid Environments. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 140–147, Washington, DC, USA. IEEE Computer Society.

[Lahmadi et al., 2009] Lahmadi, A., Andrey, L., and Festor, O. (2009). Performance of Network and Service Monitoring Frameworks. In *11th IFIP/IEEE International Symposium on Integrated Network Management - IM 2009*, Long Island, US.

[Li et al., 2010] Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). CloudCmp: comparing public cloud providers. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 1–14, New York, NY, USA. ACM.

[Lingrand et al., 2009a] Lingrand, D., Montagnat, J., and Glatard, T. (2009a). Modeling user submission strategies on production grids. In *International Symposium on High Performance Distributed Computing(HPDC'09)*, pages 121–130.

[Lingrand et al., 2009b] Lingrand, D., Montagnat, J., Martyniak, J., and Colling, D. (2009b). Analyzing the EGEE production grid workload: application to jobs submission optimization. In *Proceedings of JSSPP Workshop*, volume LNCS 5798, pages 37–58. Springer.

[Lischka and Karl, 2009] Lischka, J. and Karl, H. (2009). A virtual network mapping algorithm based on subgraph isomorphism detection. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88, New York, NY, USA. ACM.

[Liu and Foster, 2003] Liu, C. and Foster, I. (2003). A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science - University of Chicago.

[Lu and Turner, 2006] Lu, J. and Turner, J. (2006). Efficient Mapping of Virtual Networks onto a Shared Substrate. Technical report, Washington University in St. Louis.

[LYaTiss, 2011] LYaTiss (2011). White paper: Orchestrating the Cloud: Integrating Server, Networking and Storage Virtualization to Maximize the Performance and Elasticity of a Cloud-Based Infrastructure. Available: http://www.lyatiss.com/.

[Maheswaran et al., 1999] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., and Freund, R. F. (1999). Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Eighth Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, San Juan , Puerto Rico. IEEE Computer Society.

[Mandal et al., 2005] Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., and Johnsson, L. (2005). Scheduling strategies for mapping application workflows onto the grid. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC'05)*, pages 125–134, Washington, DC, USA. IEEE Computer Society.

[McKeown et al., 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74.

[Mell and Grance, 2009] Mell, P. and Grance, T. (2009). The NIST Definition of Cloud Computing. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

[Menth et al., 2009] Menth, M., Duelli, M., Martin, R., and Milbrandt, J. (2009). Resilience Analysis of Packet-Switched Communication Networks. *Networking, IEEE/ACM Transactions on*, PP(99):1.

[Microsoft, 2009] Microsoft (2009). White paper: Application Virtualization Cost Reduction Study.

[Mysore et al., 2009] Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *ACM SIGCOMM '09*.

[Nagarajan et al., 2008] Nagarajan, A. B., Mueller, F., Engelmann, C., and Scott, S. L. (2008). Proactive Fault Tolerance for HPC with Xen Virtualization. In *Proceedings of ACM ICS*.

[Niebert et al., 2008] Niebert, N., Khayat, I., Baucke, S., Keller, R., Rembarz, R., and Sachs, J. (2008). Network Virtualization: A Viable Path Towards the Future Internet. *Wireless Personal Communications*, 45(4):511–520.

[Nisan et al., 2007] Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007). *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA.

[Oinn et al., 2007] Oinn, T., Li, P., Kell, D. B., Goble, C., Gooderis, A., Greenwood, M., Hull, D., Stevens, R., Turi, D., and Zhao, J. (2007). *Taverna/myGrid: Aligning a Workflow System with the Life Sciences Community*, chapter 19, pages 300–319. Springer-Verlag.

[Oppenheimer et al., 2005] Oppenheimer, D., Albrecht, J., Patterson, D., and Vahdat, A. (2005). Design and implementation tradeoffs for wide-area resource discovery. In *HPDC*.

[Popek and Goldberg, 1973] Popek, G. J. and Goldberg, R. P. (1973). Formal requirements for virtualizable third generation architectures. In *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles*, page 121, New York, NY, USA. ACM.

[Pras et al., 1999] Pras, A., van, B.-J. B., and Sprenkels, R. (1999). Introduction to TMN. Technical Report TR-CTIT-99-09 - Centre for Telematics and Information Technology, University of Twente.

[Ramakrishnan et al., 2009] Ramakrishnan, L., Nurmi, D., Mandal, A., Koelbel, C., Gannon, D., Huang, T., Kee, Y.-S., Obertelli, G., Thyagaraja, K., Wolski, R., YarKhan, A., and Zagorodnov, D. (2009). VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC09)*.

[Raman et al., 1998] Raman, R., Livny, M., and Solomon, M. H. (1998). Matchmaking: Distributed Resource Management for High Throughput Computing. In *HPDC*, pages 140–.

[Renault et al., 2010] Renault, É., Louati, W., Houidi, I., and Medhioub, H. (2010). A framework to describe and search for virtual resource objects. In *Future Generation Information Technology - Second International Conference (FGIT)*, Lecture Notes in Computer Science, pages 208–219. Springer.

[Rezmerita et al., 2006] Rezmerita, A., Morlier, T., Néri, V., and Cappello, F. (2006). Private Virtual Cluster: Infrastructure and Protocol for Instant Grids. In *Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference*, volume 4128 of *Lecture Notes in Computer Science*, pages 393–404, Dresden, Germany. Springer.

[Ricci et al., 2003] Ricci, R., Alfeld, C., and Lepreau, J. (2003). A Solver for the Network Testbed Mapping Problem. *SIGCOMM Computer Communications Review*, 33:30–44.

[Ricci et al., 2006] Ricci, R., Oppenheimer, D., Lepreau, J., and Vahdat, A. (2006). Lessons from resource allocators for large-scale multiuser testbeds. *SIGOPS Oper. Syst. Rev.*, 40(1):25–32.

[Rimal et al., 2011] Rimal, B., Jukan, A., Katsaros, D., and Goeleven, Y. (2011). Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach. *Journal of Grid Computing*, 9:3–26. 10.1007/s10723-010-9171-y.

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

[Rosenberg and Mateos, 2010] Rosenberg, J. and Mateos, A. (2010). *The Cloud at Your Service*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

[Rosenblum and Garfinkel, 2005] Rosenblum, M. and Garfinkel, T. (2005). Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5):39–47.

[Sakellariou et al., 2005] Sakellariou, R., Zhao, H., Tsiakkouri, E., and Dikaiakos, M. D. (2005). Scheduling Workflows with Budget Constraints. In *CoreGRID Integration Workshop (CGIW2005)*, pages 347–357, Pisa, Italy. Springer-Verlag.

[Schlosser et al., 2011] Schlosser, D., Duelli, M., and Goll, S. (2011). Performance Comparison of Hardware Virtualization Platforms. In *Proceedings of the IFIP/TC6 NETWORKING 2011*. IFIP.

[Schoo et al., 2010] Schoo, P., Fusenig, V., Souza, V., Melo, M., Murray, P., Debar, H., Medhioub, H., and Zeghlache, D. (2010). Challenges for Cloud Networking Security. In *Mobile Networks and Management*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg.

[Schrijver, 1998] Schrijver, A. (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons.

[Senkul and Toroslu, 2005] Senkul, P. and Toroslu, I. H. (2005). An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399–422.

[Sherwood et al., 2010] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G. (2010). Can the Production Network Be the Testbed? In *USENIX Symposium on Operating Systems Design and Implmentation (OSDI)*.

[Silva et al., 2008] Silva, J. N., Veiga, L., and Ferreira, P. (2008). Heuristic for resources allocation on utility computing infrastructures. In *6th International Workshop on Middleware for Grid Computing (MGC 2008)*, pages 1–6. ACM.

[Sincoskie and Cotton, 1988] Sincoskie, W. and Cotton, C. (1988). Extended bridge algorithms for large networks. *Network, IEEE*, 2(1):16 –24.

[Subramanian et al., 2004] Subramanian, L., Stoica, I., Balakrishnan, H., and Katz, R. H. (2004). OverQos: an overlay based architecture for enhancing internet QoS. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 6–6, Berkeley, CA, USA. USENIX Association.

[Sugerman et al., 2001] Sugerman, J., Venkitachalam, G., and Lim, B.-H. (2001). Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of 2001 Usenix Annual Technical Conference*, pages 1–14. Usenix Assoc.

[Tamura et al., 2008] Tamura, Y., Sato, K., Kihara, S., and Moriai, S. (2008). Kemari: VM Synchronization for Fault Tolerance. In *USENIX '08 Poster Session*.

[Tennenhouse and Wetherall, 2002] Tennenhouse, D. L. and Wetherall, D. J. (2002). Towards an Active Network Architecture. In *DANCE '02: Proceedings of the 2002 DARPA Active Networks Conference and Exposition*, page 2, Washington, DC, USA. IEEE Computer Society.

[Topcuoglu et al., 2002] Topcuoglu, H., Hariri, S., and Min-You, W. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *International Journal of Supercomputer Applications (IJSA)*, 13(3):260–274.

[Truong Huu and Montagnat, 2010] Truong Huu, T. and Montagnat, J. (2010). Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In *2nd International Symposium on Cloud Computing*, Melbourne, Australia.

[Verchere et al., 2008] Verchere, D., Audouin, O., Berde, B., Chiosi, A., Douville, R., Pouyllau, H., Primet, P., Pasin, M., Soudan, S., Marcot, T., Piperaud, V., Theillaud, R., Hong, D., Barth, D., Cadere, C., Reinhart, V., and Tomasik, J. (2008). Automatic Network Services Aligned with Grid Application Requirements in CARRIOCAS Project. In *Networks for Grid Applications*, volume 2 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 196–205. Springer Berlin Heidelberg.

[Vicat-Blanc et al., 2011] Vicat-Blanc, P., Soudan, S., Guillier, R., and Goglin, B. (2011). *Cluster and Computing Networks*. ISTE Ltd and John Wiley & Sons Inc.

[Vicat-Blanc Primet et al., 2009] Vicat-Blanc Primet, P., Soudan, S., and Verchere, D. q. (2009). Virtualizing and scheduling optical network infrastructure for emerging IT services. *Optical Networks for the Future Internet (special issue of Journal of Optical Communications and Networking (JOCN))*, 1(2):A121–A132.

[VMWare, 2009] VMWare (2009). White paper: The Business Value of Virtualization.

[VMWare, 2010] VMWare (2010). White paper: Understanding Full Virtualization, Paravirtualization, and Hardware Assist.

[Waldspurger, 2002] Waldspurger, C. A. (2002). Memory Resource Management in VMware ESX Server. In *Proceedings of OSDI '02*.

[Wang and Ng, 2010] Wang, G. and Ng, T. S. E. (2010). The impact of virtualization on network performance of amazon EC2 data center. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1163–1171, Piscataway, NJ, USA. IEEE Press.

[West, 2000] West, D. B. (2000). *Introduction to Graph Theory (2nd Edition)*. Prentice Hall.

[Whitaker et al., 2002] Whitaker, A., Shaw, M., and Gribble, S. (2002). Denali: Lightweight virtual machines for distributed and networked applications.

[White et al., 2002] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA. USENIX Association.

[Xiao et al., 2007] Xiao, Z., Chang, H., and Yi, Y. (2007). *Optimization of Workflow Resources Allocation with Cost Constraint*, pages 647–656. Springer Berlin / Heidelberg.

[Xu Chen, 2009] Xu Chen, Z. Morley Mao, K. V. d. M. (2009). ShadowNet: A Platform for Rapid and Safe Network Evolution. In *Proceedings of USENIX Anual Technical Conference (USENIX'09)*.

[Yeow et al., 2010] Yeow, W.-L., Westphal, C., and Kozat, U. C. (2010). Designing and Embedding Reliable Virtual Infrastructures. In *Proceedings of VISA Workshop*.

[Yeow et al., 2011] Yeow, W.-L., Westphal, C., and Kozat, U. C. (2011). Highly Available Virtual Machines with Network Coding. In *Proceedings IEEE INFOCOM'11 mini conferece*, Shanghai, China. IEEE.

[Yu and Buyya, 2005] Yu, J. and Buyya, R. (2005). A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing (JOGC)*, 3(3-4):171 – 200.

[Yu et al., 2008] Yu, M., Yi, Y., Rexford, J., and Chiang, M. (2008). Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29.

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

[Zhao and Sakellariou, 2006] Zhao, H. and Sakellariou, R. (2006). Scheduling Multiple DAGs onto Heterogeneous Systems. In *15th Heterogeneous Computing Workshop (HCW 2006)*, Rhodes Island, Greece.

[Zhu and Ammar, 2006] Zhu, Y. and Ammar, M. (2006). Algorithms for Assigning Substrate Network Resources to Virtual Network Components. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12.

Standards, Recommandations & RFCs

[802.1Q, 2005] 802.1Q (2005). IEEE Standard for Local and Metropolitan Area Network - Virtual Bridge Local Area Networks. Available: http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf.

[Andersson and Madsen, 2005] Andersson, L. and Madsen, T. (2005). Provider Provisioned Virtual Private Network (VPN) Terminology (RFC4026). Available: http://www.ietf.org/rfc/rfc4026.txt.

[Andersson and Rosen, 2006] Andersson, L. and Rosen, E. (2006). Framework for Layer 2 Virtual Private Networks (L2VPNs) (RFC4664). Available: http://www.ietf.org/rfc/rfc4664.txt.

[Augustyn and Serbest, 2006] Augustyn, W. and Serbest, Y. (2006). Service Requirements for Layer 2 - Provider-Provisioned Virtual Private Networks (RFC4665). Available: http://www.ietf.org/rfc/rfc4665.txt.

[Baugher et al., 1999] Baugher, M., Weis, B., Hardjono, T., and Harney, H. (1999). BGP/MPLS VPNs (RFC2547). Available: http://www.ietf.org/rfc/rfc2547.txt.

[Bell et al., 1999] Bell, E., Smith, A., Langille, P., Rijhsinghani, A., and McCloghrie, K. (1999). Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions (RFC2674). Available: http://tools.ietf.org/html/rfc2674.

[Berger, 2003] Berger, L. (2003). Generalized Multi-Protocol Label Switching (GMPLS) - Signaling Functional Description (RFC3471). Available: http://www.ietf.org/rfc/rfc3471.txt.

[Biron and Malhotra, 2001] Biron, P. V. and Malhotra, A. (2001). W3C Recommendation: XML Schema Part 2: Datatypes. Available: http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[Callon and Suzuki, 2005] Callon, R. and Suzuki, M. (2005). A Framework for Layer 3 - Provider Provisioned Virtual Private Networks (PPVPNs) (RFC4110). Available: http://www.rfc-editor.org/rfc/rfc4110.txt.

[Carugi and McDysan, 2005] Carugi, M. and McDysan, D. (2005). Service Requirements for Layer 3 - Provider Provisioned Virtual Private Networks (PPVPNs) (RFC4031). Available: http://www.ietf.org/rfc/rfc4031.txt.

[CIM, 1999] CIM (1999). Common Information Model (CIM) Standards. Available: http://www.dmtf.org/standards/cim/.

[Crocker and Overell, 1997] Crocker, D. and Overell, P. (1997). Augmented BNF for Syntax Specifications: ABNF (RFC2234). Available: http://www.ietf.org/rfc/rfc2234.txt.

[EBNF, 1996] EBNF (1996). ISO/IEC 14977 : 1996(E) - Information technology - Syntactic metalanguage - Extended BNF. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=26153.

[Ellison et al., 1999] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., and Ylonen, T. (1999). SPKI Certificate Theory (RFC2693). Available: http://www.ietf.org/rfc/rfc2693.txt.

[Fedyk et al., 2008] Fedyk, D., Rekhter, Y., Papadimitriou, D., Rabbat, R., and Berger, L. (2008). Layer 1 VPN Basic Mode (RFC5251). Available: http://tools.ietf.org/html/rfc5251.

[Fielding et al., 1999] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1 (RFC2616). Available: http://www.ietf.org/rfc/rfc2616.txt.

[Gregorio and de Hora, 2007] Gregorio, J. and de Hora, B. (2007). The Atom Publishing Protocol (RFC5023). Available: http://www.ietf.org/rfc/rfc5023.txt.

[Gudgin et al., 2007] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F., Karmarkar, A., and Lafon, Y. (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[Hamzeh et al., 1999] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and Zorn, G. (1999). Point-to-Point Tunneling Protocol (PPTP) (RFC2637). Available: http://www.ietf.org/rfc/rfc2637.txt.

[ISO, 1994] ISO (1994). ISO/IEC 7498-1:1994 - Information Technology – Open Systems Interconnection – Basic Reference model. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20269.

[ITU-T, 1995] ITU-T (1995). Maintenance Telecommunications Management Network (M3020). Available: http://eu.sabotage.org/www/ITU/M/M3020e.pdf.

[Kent and Atkinson, 1998] Kent, S. and Atkinson, R. (1998). Security Architecture for the Internet Protocol (RFC2401). Available: http://tools.ietf.org/html/rfc2401.

[Kent and Seo, 2005] Kent, S. and Seo, K. (2005). Security Architecture for the Internet Protocol (RFC4301). Available: http://www.ietf.org/rfc/rfc4301.

[Kompella and Rekhter, 2005] Kompella, K. and Rekhter, Y. (2005). Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS) (RFC4202). Available: http://www.ietf.org/rfc/rfc4202.txt.

[Moskowitz and Nikander, 2006] Moskowitz, R. and Nikander, P. (2006). Host Identity Protocol (HIP) Architecture (RFC4423). Available: http://www.rfc-editor.org/rfc/rfc4423.txt.

[MPI, 1995] MPI (1995). The Message Passing Interface (MPI) standard. Available: http://www.mcs.anl.gov/research/projects/mpi/.

[Nichols et al., 1998] Nichols, K., Blake, S., Baker, F., and Black, D. (1998). Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers (RFC2474). Available: http://www.ietf.org/rfc/rfc2474.txt.

[Rosen and Rekhter, 2006] Rosen, E. and Rekhter, Y. (2006). BGP/MPLS IP Virtual Private Networks (VPNs) (RFC4364). Available: http://www.rfc-editor.org/rfc/rfc4364.txt.

[Rosen et al., 2001] Rosen, E., Viswanathan, A., and Callon, R. (2001). Multiprotocol Label Switching Architecture (RFC3031). Available: http://www.ietf.org/rfc/rfc3031.txt.

[SQL, 1992] SQL (1992). ISO/IEC 9075:1992 - Information technology - Database Language SQL. Available: http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16662.

[Takeda, 2007] Takeda, T. (2007). Framework and Requirements for Layer 1 Virtual Private Networks (RFC4847). Available: http://www.ietf.org/rfc/rfc4847.txt.

[Townsley et al., 1999] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and Palter, B. (1999). Layer Two Tunneling Protocol (L2TP) (RFC2661). Available: http://tools.ietf.org/search/rfc2661.

[Valencia et al., 1998] Valencia, A., Littlewood, M., and Kolar, T. (1998). Cisco Layer Two Forwarding (Protocol) L2F (RFC2341). Available: http://tools.ietf.org/html/rfc2341.

