Doctoral Thesis

Università degli Studi di Napoli Federico II

Architectures and Algorithms for Resource Management in Virtualized Cloud Data Centers

Author:
Antonio Marotta

Supervisor:
Prof. Stefano Avallone

Co-advisor:
Prof. Giorgio Ventre

Coordinator:
Prof. Franco Garofalo

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Department of Electric Engineering and Information Technologies

March 2015

UNIVERSITY OF NAPLES FEDERICO II

Abstract

Computer Science
Department of Electric Engineering and Information Technologies

Doctor of Philosophy

Architectures and Algorithms for Resource Management in Virtualized Cloud Data Centers

by Antonio Marotta

Cloud Computing has raised a great interest over the last years as it represents an enabling technology for flexible and ubiquitous access over the network to a set of shared computing resources. Cloud paradigm leverages the instruments of the virtualization technique, such as the *VM Live Migration*, which can be exploited in order to achieve multiple objectives. For instance, it could be used for hardware maintenance purposes or to avoid over-load and under-load in the resources utilization. Another typical use of the VM migration is the consolidation of the workload into a smaller number of physical hosts: this is one of the techniques aimed at increasing the energy efficiency of the IT infrastructure, which represents the motivation behind the birth of the *Green Computing*. However, despite all the advantages that come from the application of the cloud paradigm, there is some reluctance in its adoption for mission critical infrastructures because of the security pitfalls it still exhibits, as well as the lack of mechanisms intended at increasing isolation and protection from internal and external threats. This thesis is intended at exploring both the described aspects: on one hand, an architecture for enhancing security of a virtualized critical infrastructure is designed and provided with a mitigation strategy based on the Software Defined Networking approach and live migration. On the other hand, migrations are used to propose two VM consolidation strategies with the objective of minimizing the overall infrastructure power consumption.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

VM	**V**irtual **M**achine
VMM	**V**irtual **M**achine **M**onitor
LVM	**L**ogical **V**olume **M**anager
VLAN	**V**irtual **L**ocal **A**rea **N**etwork
ATM	**A**ir **T**raffic **M**anagement
GRE	**G**eneral **R**outing **E**ncapsulation
LAN	**L**ocal **A**rea **N**etwork
HAC	**H**igh **A**vailability **C**luster
HTC	**H**igh **T**hroughput **C**luster
HPC	**H**igh **P**erformance **C**luster
SaaS	**S**ervice **a**s **a** **S**ervice
PaaS	**P**latform **a**s **a** **S**ervice
IaaS	**I**nfrastructure **a**s **a** **S**ervice
API	**A**pplication **P**rogramming **I**nterface
NIST	**N**ational **I**nstitute of **S**ecurity and **T**echnologies
SLA	**S**ervice **L**evel **A**greement
SOA	**S**ervice **O**riented **A**rchitecture
DCIM	**D**ata **C**enter **I**nfrastructure **M**anagement
DCiE	**D**ata **C**enter **I**nfrastructure **E**ffectiveness
PuE	**P**ower **u**sage **E**ffectiveness
DCeP	**D**ata **C**enter **E**nergy **P**ractitioner
CuE	**C**arbon **u**sage **E**ffectiveness
DCPM	**D**ata **C**enter **P**redictive **M**odeling
ATC	**A**ir **T**raffic **C**ontrol
MILP	**M**ixed **I**nteger **P**rogramming **M**odel

EC2	**E**lastic **S**ervices
OS	**O**perating **S**ystem
NFS	**N**etwork **F**ile **S**ystem
KPI	**K**ey **P**erformance **I**ndicator
CII	**C**ritical **I**information **I**nfrastructure
ICT	**I**nformation and **C**ommunication **T**echnologies
ACC	**A**rea **C**ontrol **C**entre
MPLS	**M**ulti **P**rotocol **L**abel **S**witching
SDN	**S**oftware **D**efined **N**etworking
TLS	**T**ransport **L**ayer **S**ecurity
OVSDB	**O**pen**v****S**witch **D**ata**B**ase
LLDP	**L**ogical **L**ink **D**iscovery **P**rotocol
VXLAN	**V**irtual e**X**tensible **L**ocal **A**rea **N**etwork
VTEP	**V**irtual **T**unnel **E**nd**P**oint
CORE	**C**ommon **O**pen **R**eserach **E**mulator
NIDS	**N**etwork **I**ntrusion **D**etection **S**ystem
SESAR	**S**ingle **E**uropean **S**ky
SecRAM	**O**pen**v****S**witch
IA	**I**mpact **A**rea
DVS	**D**ynamic **V**oltage **S**caling
BPP	**B**in **P**acking **P**roblem
SSAP	**S**tatic **S**server **A**llocation **P**roblem
FFD	**F**irst **F**it **D**ecreasing
NPE	**N**etwork **P**ower **E**ffectiveness
PoE	**P**ower **o**n **E**thernet

Chapter 1

Cloud Computing: Opportunities and Challenges

1.1 Virtualization Technique and its Impact on Computer Systems

Critical services are increasingly relying on heterogeneous multiple applications and resources in order to satisfy a growing and multidisciplinary demand. With the objective of providing users with distributed and very intensive applications, one of the most promising and efficient techniques to consolidate system's utilization on one hand, and to lower power, electricity charges and space costs in data centers on the other, is without any doubts the virtualization [1]. This concept was introduced in the 1960s by IBM: a Virtual Machine (VM) was intended as an exact software reproduction of a real machine and all of its subsystems. In the next two decades the costs of hardware decreased and, consequently, the interest in the virtualization mechanisms became very poor. The first technical definition of VM was provided in an article by Gerald J. Popek and Robert P. Goldberg in 1974 [2], through the introduction of the concept of *Virtual Machine Monitor* (VMM). This is defined as the execution environment for multiple VMs, characterized by a set of properties to satisfy:

- *Equivalence* ensures an execution environment which is substantially identical to the real machine's one for the VM applications.

- *Efficiency* guarantees a high efficiency in the execution of the VM programs. When possible, the VMM must allow the direct execution of the VM application instructions: unprivileged instructions should be executed by hardware without the involvement of the VMM.

- *Control of resources* ensures the stability and security of the entire system. The VMM should be provided with full control of the hardware resources: the VM applications and the operating system should not access them in a privileged way.

The advent of *Personal Computing* and the need to execute specific applications on different hardware platforms and operating systems were the motivations of the advancements in the virtualization field. The tangible benefits of virtualization consist in simplifying the provisioning of IT resources and applications, in addition to the simplicity users can install, configure and access them with. One of the primary purposes of all the various forms of virtualization is to make the utilization of system resources very efficient. During the years it has been proven that, by committing hardware and IT equipment to dedicate applications, the resulting resources utilization is under a cost-effective threshold. Hence, the chance to share the same physical resources amongst several VMs has become very popular. Among the types of virtualization there are:

- **Process Level Virtualization** which consists in the virtualization of individual application processes. It allows the deployment of applications through the execution of a VM for each process without changing the host OS;

- **System Level Virtualization** is the technique that enables the emulation of a full real physical system with all its devices, such as CPU, memory, disk, network interfaces and so on;

- **Operating System Level Virtualization** uses the host OS kernel to instantiate user space dedicated to the management of multiple guest OSes, without the need for a hypervisor;

- **Resource Virtualization** enables a guest OS to use specific resources of a host system.

1.1.1 Server Virtualization

As described in the previous section, the System Level Virtualization allows a perfect reproduction of a real entire physical system. The idea behind this technique is to share the same resources among different users that can create their own VMs. This kind of virtualization is also referred to as *Server Virtualization*: a software version of physical resources is exposed to VMs. The main actors of the server virtualization are:

- **Host OS** (kernel), namely is the operating system which is installed on a computer system and can access the physical resources through its kernel. This is the core part of the OS and it is responsible for providing the running processes with secured and controlled access to physical resources, through a hardware abstraction, also known as *Hardware Abstraction Layer*. This layer lowers the complexity, since it consists in an uniform interface to the underlying physical resources;
- **Guest OS**, which is an operating system running in a virtual environment offered by a VM: it has access (through its kernel) to hardware resources which are dynamically allocated by the hypervisor or a host OS depending on the virtualization technique that is used;
- **Virtual Machine** which is the virtualized representation of a physical machine;
- **Hypervisor** or **Virtual Machine Monitor**, the basic component which provides the software layer that allows the abstraction of physical hardware resources. This component allows to execute, monitor and manage a huge number of different operating systems or instances of the same OS that can coexist on the same hardware components.

The advantages coming from the system level virtualization are:

- *isolation*, which ensures that a guest OS crash does not have any effect on the other VMs running on the physical nodes and on the host OS;
- *resource consolidation*, for packing the workload in a smaller number of physical machines;
- *hardware abstraction*, that enables the execution of incompatible or legacy applications without the need to change the hardware platform;

- *resources migration*, that offers the possibility of transferring a running VM between two physical hosts in order, for example, to ensure host maintenance without interrupting active applications.

The most important component of the system level virtualization is the hypervisor that, as explained before, allows to execute different VMs on the same system. Hypervisors are grouped into two main categories:

- *Type 1 - native or bare metal*
  Native hypervisors are designed to directly control and monitor the hardware resources of the physical node and the guest OS. The virtualization capabilities are integrated in the node's architecture, in order to offer a lightweight operating system which is created above its hardware. For this kind of hypervisor it is needed to provide the VMM with all the necessary drivers to interact with the devices.
- *Type 2 - hosted*
  This type of hypervisor is designed to interact with the installed host OS; hence, it consists in a distinct software layer, which is added as a typical application and it does not replace the OS which is installed on the physical node. A hosted VMM operates in user space and access the hardware resources via the system calls of the host OS.

The second type of hypervisor is characterized by the drivers compatibility and easiness with regards to the installation, since it relies on the host OS for device management and other services (such as scheduling for instance). Anyway the hosted hypervisors show lower performances then the native ones.
According to the access mode to the physical resources and the interactions between the VM and the VMM, the system level virtualization can be categorized as:

- **Full Virtualization**: the VMs use the same interface or, in other words, the same instructions set of the physical architecture;
- **Para Virtualization**: the VMM presents a different interface from the one offered by the hardware architecture. This is a technique that allows the hypervisor to provide the VM with an optimized version of the hardware physical resources.

Therefore it requires modifications for the guest OS running in the VM, which consist in the optimization of specific system calls and the involvement of the hypervisor in the execution of critical system calls. These latter are exclusively managed and executed by the hypervisor which performs on behalf of the guest OS.

- **Emulation**, which consists in the ability to execute programs, which were compiled for a certain instructions set, on a computer system equipped with a different set. Each instruction is emulated for allowing operating systems with different architectures to run them without modifications. Emulation techniques offer the advantage of interoperability among heterogeneous environments; on the other hand, they are also affected by efficiency and performance implications.

The main difference between the two approaches lies in the exposition of the optimized hardware resources to the VM. Para Virtualization, however, reduces flexibility, due to the fact that the guest OS needs adaptation. Therefore this techniques is applicable only to operating systems that support it, thus limiting the compatibility.

1.1.2 Storage Virtualization

The storage virtualization has the objective of overcoming the heterogeneity of storage systems and to provide applications with a single point of access to shared memorization resources. Storage virtualization allows to reduce the traditional separation between an application or a service and the underlying storage systems. Through virtualization the storage can reside anywhere and on any type of devices and they can be also replicated for dependability purposes. For example, systems can have access to a shared storage that is somewhere in the network. This approach lowers the number of storage devices which is needed and the amount of power consumption: as a direct consequence, the operational and administrative costs for back up and archival storage can be drastically reduced. With storage virtualization technologies, it is possible to use software tools in order to create an optimized and logically centralized view of the storage environment that aggregates components, such as disks, controllers and storage networks. This elements can be efficiently shared among the various applications. The storage virtualization offers several advantages, such as high availability, fail-over and disaster recovery. There

are three types of storage virtualization, implemented according to the infrastructural level:

- *Host-based* virtualization is implemented through the Logical Volume Manager (LVM), which groups more physical volumes to provide applications with a larger and flexible pool of storage.
- *Network-based* is used to virtualize storage through a network connection that will be used by users in order to access it through a specific protocol.
- *Controller-based* virtualization enables the flexible management of the storage array system of a company. This type of virtualization can extend all the features of the controller to external memory devices. Data can be migrated and the replication can take place between different storage devices.

1.1.3 Network Virtualization

Network Virtualization is a technology layer that allows to create one or more virtual networks from a single physical network: they can be characterized by an independent topology from of the underlying physical one. Technologies that have been used to implement network virtualization are:

- *Virtual Local Area Networks* (VLANs) allow a group of computers, which are connected to a LAN switch, to be isolated at the layer 2 of the ISO/OSI stack, by creating different and separated logical groups.
- *Active and programmable networks* are physical networks on which there is the possibility to run code with the aim of programming the behaviour of the devices.
- *Asynchronous Transfer Mode* (ATM) networks represented for a long time an important aspect of the network virtualization, namely, the connection virtualization. The ATM virtual circuits are paths that provide isolation of shared connecting resources, through the reservation of ATM nodes along specific routes.

A virtual logical topology, which is created above the physical one, is also referred to as *Overlay Network*. The nodes of an overlay network are connected through virtual

links that correspond to paths in the underlying network. This type of logical networks is usually implemented at the application level: overlay networks have no geographical restrictions and are flexible and adaptable to changes.
Several organizations are adopting network virtualization based technologies in order to meld the data-link and IP approaches. The motivation is to be found in the typical data link technologies scaling issues: the approach of creating virtual L2 networks across IP networks is often referred to as *L2 over L3* and it represents an abstraction layer between the physical and virtual networks. Network customers may access as many level 2 networks as they like, while the underlying physical networking is run, for instance, on multiple data networks, connected across layer 3 networks. This technique tries to combine the benefits coming from the two layers: the scalability, the fast convergence times and the bandwidth allocation of L3 technologies and the simplicity in the configuration of L2 networks. L2 over L3 is obtained through encapsulation techniques, such as General Routing Encapsulation (GRE), Virtual eXtensible Virtual LAN, or by using OpenFlow. In an OpenFlow network, switches do not use traditional L2 or L3 protocols to determine the traversed links. Instead, an OpenFlow controller configures the switches, by creating forwarding paths and programming the switches' hardware tables. The controller has full knowledge of the physical and virtual network topology and orchestrates the provisioning of all the virtual networks across the complete set of connected OpenFlow-enabled devices.

1.2 Recent Evolutions of the Computing Paradigm

The Information Technology field has been experiencing major changes over the past four decades: the introduction of the corporate mainframes allowed the business to reduce the number of employees, needed for manual processing of transactions. The era of Personal Computing has provided users with the capability of directly managing their activities, thanks to the availability of data and applications on their personal computers. This reduced the time to access a single and centralized computation resource and it gave the chance to work in a parallel way. Finally the era of *Network Computing* has improved the level of transparency of the information between multiple groups within a company and it has allowed the exchange of data between companies. Each of these revolutions has brought new economies of scale: for example, personal computing had the effect

to lower the cost to bear for a single transaction; the network computing increased the number of users able to access computing resources and, in general, the business value, by also bringing down the cost of the network bandwidth.

1.2.1 Cluster Computing

Cluster computing can be defined as an inter-connection of stand-alone computers working together as a single, integrated computing resource [3]. One of the first and most famous example of cluster was *ARCnet* [4], a system developed by DataPoint in 1977 as a high-speed local area network (LAN). This cluster system had the objective of supporting not only the parallel computing, but also the possibility to share peripherals. In 1983, Digital Equipment Corporation developed a cluster solution, known as *VAX-cluster* [5], that consisted in a collection of multiple computers connected via a serial link. The main purpose of this cluster solution was to provide operators with the access to the computing and memorization resources as if they were a single system. In those years the development of Unix-based workstations in companies and universities was increasing more and more, while the estimated resources usage during the day was very low. Both the academic and industrial fields were experimenting the need of a *middleware* capable of hiding the heterogeneity of the hardware resources and exploiting the available interconnected computing resources in order to solve complex problems. In this context new technologies were born, such as, *Parallel Virtual Machine* and *Message Passing Interface*, with the aim of supporting the development of cluster-oriented applications which are independent from the underlying physical architectures. In the 1990s the cluster-based solutions became very popular and they were massively used for realizing systems demanding for high availability. The possible classification is in terms of:

- the structure, which can be homogeneous or heterogeneous (from the hardware and/or operating systems view);
- the installation (e.g. glass-house, campus etc.);
- the supported application type

- *High Availability Clusters* (HAC), which are characterized by the redundancy of the components in order to avoid a single point of failure. Depending on their behaviour in case of fault, they can be distinguished in:
  - *Active-Passive*, if during normal operations at least one node (more generally a component of the infrastructure) is in a passive state (standby). If the active node undergoes a failure, the standby one becomes active, taking over the master in delivering the service;
  - *Active-Active*, if all the nodes are active and perform tasks related to the same service. If a node is a fault state, the remaining active nodes take over it. The main advantage lies in a better load distribution between the nodes of the infrastructure, without resources wastage. Anyway, on the other side, the management of the nodes is not transparent to the applications that need to deal with the access to shared resources (e.g. memory areas).

  A high availability cluster is called *shared nothing* if each node has a dedicated storage: the advantage is that there is no required mechanism for accessing the storage system and, as a consequence, no protocol overhead. The bottleneck is represented by higher costs for the storage resources, because they are not distributed and optimally used. The other possibility is *shared all*, where all the nodes have access to a shared storage, that can be organized in different way and accessed through several protocols.
- *High Throughput Clusters/ High Performance Clusters* (HTC/HPC) are computing systems aimed at testing and running large parallel-processing codes, visualization and scientific applications. HPC systems allows to execute a single instance of parallel software over many processors, while HTC systems ensure the execution of multiple independent software instances on multiple processors at the same time.
- *Load Balancing Clusters* are used to scale up the performance of server-based applications, such as a Web or mail server, by distributing client's requests across multiple servers. Load balancers are entities receiving incoming requests and dispatching them to a specific server according to a chosen policy (e.g round robin, least connection, load-based and so on).

1.2.2 Grid Computing

The term *Grid* was coined in the 1990s: it denoted a distributed computing infrastructure for advanced science and engineering applications [6]. During the years it has rapidly involved multiple applicative domains, ranging from advanced networking to artificial intelligence. The reason behind the constitution of a grid-based infrastructure is the need for solving a real and very complex scientific problem, though coordinated resource sharing and collaborative problem-solving. A Grid can defined as a stack of hardware and software layers, intended at offering a dependable, consistent and non-expensive access to very powerful computing resources for requesting users. Grid Computing is an approach to build computational infrastructures which are is endowed with:

- a distributed access coordination function, thanks to which users are granted the possibility to use resources for a specified time interval;
- open-source and general-purpose protocols, APIs and libraries;
- mechanisms to assure a contracted level of quality to users that require it.

Grid Computing was born with the objective to share very expensive resources, such as data, network devices and CPU in a *Virtual Organization*. This is the combination of different institutions, research centers and/or individuals which often have a unique goal and define a common set of sharing rules. Such an organization is formed with the intention of creating a collaboration, in which computing resources can be geographically distributed and accessed. One of the benefits of the grid is, indeed, the exploitation of under-utilized resources by means of a dynamic combination of nodes and applications. Users can access the shared infrastructure through a set of clear rules. The key point of grids lies in the ability to renegotiate the way the organization can share resources. There is a clear need for policies and access mechanisms: for instance, authentication must be deployed in combination with authorization in order to verify if the user accessing a specific resource has enough permissions.
One of the de facto standards for Grid Computing implementation is the Globus Toolkit [7], which defines protocols and the middleware layer to guarantee a controlled access to the resources which are shared in a VO. The tasks which are addressed by Globus involve resources discovery, provisioning, management, security and jobs scheduling. The grid stack consists can be considered according to five layers:

- the fabric layer, which defines the access modes to different types of resources, among which there can be compute, storage, networking devices etc.;
- the connectivity layer, which consists in the core communication and authentication protocols. For example, the *Grid Security Infrastructure* [8] provides every grid transaction with a desired security level;
- the resource layer, which is made of protocols for the publication, discovery, negotiation, monitoring and accounting of the operations on shared resources. The *Grid Resource Access and Management* [9] is the protocol that assures the scheduling of computing resources to users and it is also used for monitoring and control tasks;
- the collective layer, which is used to monitor the interactions with a well-defined set of resources and for the discovery service of VO resources;
- the application layer, which is characterized by any user application which is deployed by using the low-layers protocols and APIs.

Grids are characterized by a geographical distribution of devices and a great amount of available data: this enables the realization of complex experiments which would result in very high costs for the single organization. Grid and Cloud computing paradigms basically share the same underlying concept which is, the *Utility Computing*, namely a service provisioning model through which computing resources are used by a customer as needed. This has the objective of maximizing the utilization and bringing down the relative costs. Anyway the two approaches are quite different in terms of their purposes and resources distribution.

1.2.3 Cloud Computing

Cloud Computing is a model for enabling flexible and ubiquitous network access to a pool of shared computing resources. A plenty of different definitions of this paradigm exists:

- the Focus Group defines it *as a model that provides users with the convenient, on demand and network-based access to a shared pool of configurable computing resources (e.g., networks, servers, storage element, applications and services) which*

can be made available and released with a minimum management effort and/or interaction with a service provider [10];

- *Cloud computing is a paradigm to provide services that are delivered and consumed on demand, at any time, via an access network, by exploiting any connected device that uses this technology.*

The term cloud can be used if the paradigm which consumer can access different type of resources through is characterized by all these properties [11]:

- *On-demand and self-service*
  Users can automatically acquire resources according to their needs at any moment. All the available resources can be requested and used when users or applications require.

- *Resource pooling*
  Cloud providers manage pooled infrastructure resources which are ready for use in order to serve multiple users. They are dynamically allocated or re-allocated on the basis of customer's demand. Computing resources are shared among multiple users but they are granted with a multi-tenant isolation for security purposes. The resources pool is accessed with a full independence from the location, so that the customer does not know the physical or geographical location of the resources. Examples of pooled resources can include storage, optical disks, network bandwidth and VMs.

- *Rapid elasticity*
  Resources need to be acquired and released in an elastic way, in order to achieve scalability in relation to the service demand. The user is provided with the illusion of being able to access unlimited resources. There must be the possibility of acquiring resources through a fast and automatic mechanism and to release them when they are not needed any more.

- *Measured service*
  The resources requested by a customer should be measurable according to a set of quantitative parameters in order for the cloud system to control and optimize their use through continuous measurement of performance indicators. The measures

should be available to the users via APIs for transparency purposes and for allowing rapid modification of QoS requests. Besides these measures are useful to the provider in order to realize accounting and billing if there is an economic deal.

- *Broad network access*

  The customer must be able to access resources through a broadband network and with standard mechanisms that enable their use from heterogeneous clients (smart-phones, tablet, laptop, workstation etc.).

As introduced in the previous section cloud service provisioning model shows some differences in relation to Grid Computing: this latter has the objective of sharing resource to solve very complex and computational expensive problems, while cloud is a way to provide a full stack for network-based application development, testing and deployment. Therefore, in the grid paradigm the focus is on the processes to share and schedule the utilization of resources for different users; cloud is fundamentally distinguished by the establishment of a pay-per-user relation between a customer and a provider. Besides grids are non-commercial and publicly founded by multiple domains, while, instead, cloud offerings are often commercial and single-domain.
Cloud Computing nowadays has involved concepts coming from very disparate cultural fields and people are using such a technology by mingling aspects from different backgrounds and cultures such as technological, economic, socio-logic and legal environments. This underlines the great success cloud has been acquiring during the last years. This paradigm is able to improve the efficiency level, as it allows to bring down the necessary costs for the purchase and maintenance of computing resources. Cloud Computing produces elasticity since it gives the chance to exploit the capabilities of shared IT resources through the network. Thanks to these properties, customers can avoid the risk of over-provisioning resources when there is not a clear formalization of the capacity and the demand. Cloud technology is, therefore, a great innovation opportunity for companies and also for the research field.

1.3 Cloud Computing Technology

1.3.1 Possible Classifications

Cloud infrastructures can be classified concerning the service and the deployment model they implement. From the service model perspective, they are classified depending on the abstraction level of the resources which are used by the final user. The basic classification involves three different layers:

- *Software as a Service* (SaaS) is the delivery of application-layer services which are accessible regardless of the location and the type of used devices. If there is a commercial deal, the customer pays the license or rent the application which is managed and orchestrated by the provider. SaaS provides a self-contained operating environment which is used to deliver the entire user experience, including the content, its presentation and the business logic. The customer should be able to use cloud-based services from any kind of devices: he has the illusion to access the environment anywhere, anytime, anyhow. Examples of SaaS are GMail, Google Apps, Youtube, Flickr, Facebook, LinkedIn and Picasa.

- *Platform as a Service* (PaaS) is the delivery of basic development services such as operating systems, middleware, languages, database technologies and the runtime environments necessary to build and run applications. In the PaaS model, the consumer is given the ability to deploy applications on cloud infrastructures, by using programming languages, libraries, services and tools supported by the provider. As for the SaaS model, the consumer does not handle the underlying cloud infrastructure but he has control over installed applications and configuration settings of the host environment. Examples are Google's Application Engine, Microsoft Azure, Salesforce and AWS Beanstalk.

- *Infrastructure as a Service* (IaaS) is the delivery of infrastructure resources related to processing power, storage, network and other basic elements. This layer is also known as *Resource Cloud*, since it represents the lowest level of abstraction. The aim of the IaaS Cloud paradigm is to provide IT resources as services which are delivered through the network, by hiding in such a way the sophistication of the underlying infrastructure and by guaranteeing their dynamic allocation based on

the current load. The customer is charged in a pay-per-use basis, depending on the type of infrastructural resource he has requested. IaaS provides a set of Application Programming Interfaces (APIs) which allow the management and other forms of interactions with the infrastructure. Examples are Amazon EC2, Amazon S3, Rackspace Cloud Servers and Flexiscale.

This basic classification has been extended and it has assumed very different granularities, according to the specific service which is offered through the cloud paradigm. Hence, if any service (X) is offered by means of an universal access, in a scalable and elastic way and in a pay-as-you-go basis, it is possible to use the term *XaaS*.
The other classification is based on the deployment or implementation model:

- *Private Cloud*, in which the cloud infrastructure is used exclusively by the organization who holds it. This type of cloud infrastructure is ideal for companies and organizations that must comply with a series of regulations and which desire to efficiently manage their resources.

- *Public Cloud*, in which the cloud infrastructure is accessible via the network and it is property of the organization that offers the cloud services. This type of infrastructure is ideal for companies and organizations which choose to hit a new market section or decide to outsource ICT technologies.

- *Hybrid Cloud*, in which the cloud infrastructure is regarded as a composition of two or more deployment models that coexist within the same organization. This type of deployment is ideal, for example, to e-commerce sites: in order to accommodate the traffic fluctuations, a public cloud platform can be used for flexible order processing. However, it is necessary to respect privacy and security for online transactions: this is more controllable in a private cloud.

- *Community Cloud* is used by a specific community of consumers belonging to an organization which is concerned with security requirements, policy and compliance considerations. It may be owned or managed directly by the organization, by a third party or by a combination of them. Community clouds are often created as a federation of private clouds.

1.3.2 Reference Architecture

The cloud layer stack can be graphically illustrated thanks to the reference model which has been proposed by the National Institute of Standards and Technologies [12]. It consists in a representation of the cloud definition and properties and it is a generic architecture that does not refer to a particular technological solution. As shown in Figure 1.1, a set of functions and activities are defined with regards to the development of a cloud solution.

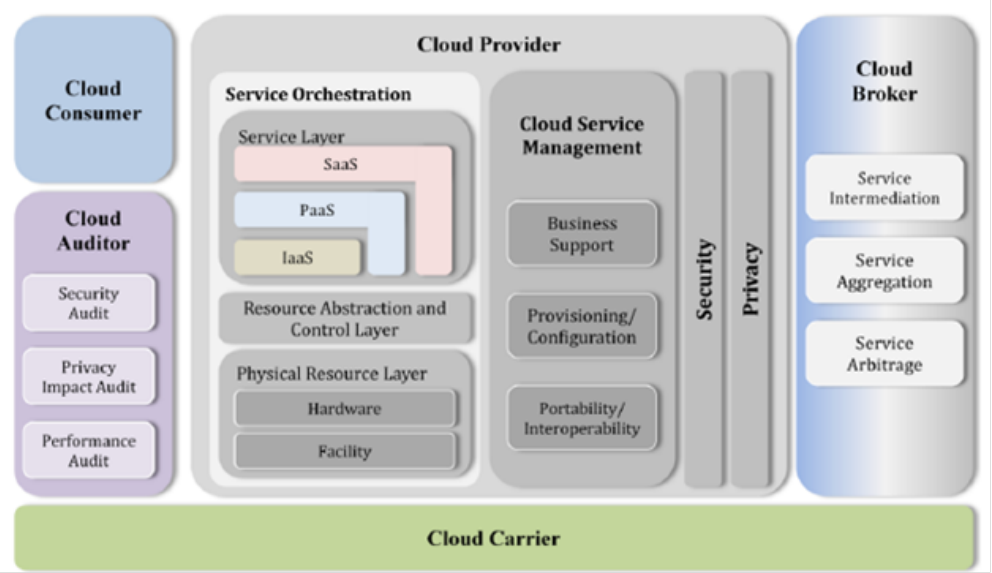


FIGURE 1.1: Cloud Reference Architecture

This architecture enables the understanding of the relationships among all the elements of a cloud infrastructure to the aim of identifying the role of each element and the responsibilities. The institute has identified five key roles in the cloud:

- the *Cloud Consumer* is the entity that wants to use a cloud enabled service. It can be an individual or organization, which establishes an economic relationship with a provider;
- the *Cloud Provider* is basically who provides the cloud based services and is responsible for managing all the infrastructure for service delivery. Among the activities under its supervision there are:
  - Service deployment;
  - Service orchestration;

  - Cloud services management;
  - Security and privacy.

  For SaaS, the provider is in charge of the management of the software platform which users are able to access through the cloud. In the case of PaaS, the provider has full access to the tools that allow service deployment in the cloud but it has no vision of low-level underlying physical infrastructure. In the IaaS the vision is completely different because the provider has to handle all the physical resources and OSes. Cloud providers usually have a service catalogue which the customer can select one or more services from. The relationship between provider and consumer is often sanctioned by a sort of Service Level Agreement (SLA) contract that regards the quality of service for the system performance;

- the *Cloud Auditor* is responsible for monitoring and verifying cloud services. It provides an independent and third-party opinion on the quality of service offered by the provider. The evaluation of a cloud auditor can be in terms of performance, security and availability. The focus is on how to monitor the services and to identify quantitative metrics for different kinds of measured properties;

- the *Cloud Broker* negotiates the needs of the consumer with the provider, in order to overcome problems, such as the management complexity and the heterogeneity of services. The broker offers in particular:

  - service intermediation, which is the process in which the broker interfaces with the cloud provider, instead of the consumer, to enrich the service offered by the provider and add some capabilities. For instance, the broker could add security technologies that were not initially contemplated by the provider;
  - service aggregation, thanks to which the cloud broker may aggregate multiple services offered by the provider by creating a logical single access point;
  - service arbitrage, which is similar to aggregation, with the difference that services are dynamically added when the customer needs them;

- the *Cloud Carrier* is the entity that guarantees the use of services via a secured and stable network connection. It interacts with the provider to provision connectivity to services that will be offered to clients. This role is usually covered by network operators or telecommunication agencies, that have to deals with the management

of the network devices in order to ensure a set of SLA requirements between a provider and consumer.

A possible scenario involves the interaction between a consumer and a provider that may be mediated by a cloud broker. The consumer can request a particular service the broker can actually deliver as a composition of multiple services offered by different providers or by an individual provider. Otherwise the interaction between the provider and the consumer can be direct without any broker. When the consumer decides to rely on a cloud provider, it establishes a SLA contract. In turn, the provider decides to sign a contract with a carrier to ensure a specific quality of service. The auditor is the entity that has to make sure everything that has been agreed between a consumer and a provider is actually respected.

1.3.3 Cloud Benefits and Risks

Cloud enables the ubiquitous, convenient and on demand network access to a shared set of configurable computing resources, which can be quickly acquired and released with minimal effort and interaction with the service provider. Cloud model allows organizations to bring down the costs for capital investment (CapEx, capital expenditures), required to purchase computer systems, communication systems (servers, storage systems, routers and so on) and the operational cost (due for example to generators and air conditioning systems). In addition to these costs, companies also face the OpEx, namely the operating expenses, required to manage products, systems or business activities. Cloud can lower the ingress barriers for small organizations and start-up, by also creating new professional figures, such as the cloud architect.
As explained before, the relationship between a provider and a consumer is characterized by a SLA contract: the service and the resources are delivered and measured in terms of appropriate indicators, the so-called key performance indicators. Among them there are the temporal indicators, which involve, for example, the service response time, the throughput, the latency of transactions and so on. The cost indicators deal with CapEx and OpEx, and with the consolidation of IT resources; finally the quality indicators measure user's satisfaction and the perceived quality of experience. They basically indicate how much the user appreciates the interface of the services and the interactions with them. When an economic deal is established between a provider and

a consumer, there is often no possibility to renegotiate agreed SLA parameters. The definition of a contract with a provider is not something immediate: services are usually offered in composition with multiple providers, and it is not very simple to negotiate SLAs with multiple providers. Another problem is also linked to the penalties and the legal protection, when some of the parameters are not respected. A technical issue lies in the mechanisms for monitoring SLA-oriented parameters like, for example, the ones regarding security. Besides cloud services do not often offer a full transparency to the final user.

When approaching cloud solutions, one of the main concerns should be the risk of lock-in: services that promote data portability, open formats and standards should be preferred, in order to facilitate the transition from one provider to another one. There is the need to adopt international standards to ensure a switching off among different providers. The technological evolution of cloud has been so rapid, that most of the existing providers built their own services relying on ad-hoc protocol and/or APIs. Therefore companies are often tightly coupled with a non standard solution and, therefore, they are unlikely to migrate to another provider that may offer better service conditions. In that case there is also a risk for the customer's activities if the provider goes out of business. In the cloud standardization activities there are different organizations working to the same objective. Three macro-areas have been identified:

- *General*

  This area includes activities aimed at developing and defining cloud taxonomy, use cases and service requirements. Since cloud involves a wide range of technical and business elements, it is crucial to define standard technical terms in order to promote standardization.

- *Cloud configuration management*

  This area includes the standardization process for the implementation technologies, the architecture, the APIs and security. Cloud APIs are basilar to allow interoperability between different environments and vendors: developers can benefit from web servers standards and the Service Oriented Architecture (SOA) when designing functional and management aspects. For instance, the *Cloud Security Alliance* is defining the best practices in thirteen areas, which are related to cloud governance and management.

- *Inter Cloud Federation*

  This area is intended at the definition of standard interactions needed for the federation of cloud solutions. To this aim communication protocols between different cloud providers are necessary. The inter-cloud federation is, in fact, the interconnection of two or more cloud service providers that can work together in order to balance the traffic load and to offer more scalable computing power.

The deployment model should be selected with the awareness that moving from private clouds to public ones, the control on data is gradually lost. It is possible that some information may require special security measures due to their nature (e.g. health, genetic, financial data). In such cases, due to risks like potential loss or unauthorized access, consumers should carefully decide if it is better to use cloud services or to maintain in-house processing of such data types. Besides customers should know whether data will remain in the physical availability of the provider, or if the service is designed on technologies provided by a third-party provider. As an example, a cloud application cab be offered by a SaaS provider which uses a storage service offered by a third party company. Another risk is related to data persistence policies: the cloud service provider needs to specify the timing for data persistence and the techniques used to delete user's information in the contract clauses.
In conclusions, the analysis of Cloud Computing solutions should be conducted by taking into consideration several factors, benefits and risks: an economic analysis can also be useful to compare the cost reduction with the adoption of a cloud solution against the on premise approach. The cloud paradigm also leads to energy savings which result in a reduction of costs. The trade-off to be investigated lies in the economic benefit and the achievable service level, in terms of reliability and performance, which can not be easily measurable and transparent to the consumer.

1.4 Cloud Computing Security Threats

Besides the classical computer systems security issues, cloud is characterized by known and well-studied threats [13] involving different areas, such as:

- Organizational risks and policy, among which there are the lack of governance and lock-in. Data and applications portability must be integrated in the risk management process. An organization could decide to change cloud provider because of an unacceptable increase in the costs at contract renewal or a decrease in the service quality. Besides a provider can cease business operations or just close some services.

- Legal risks which relate to the compliance to national laws, regulations and certifications, the lack of audit mechanisms and data protection. For example, the customer should be aware, in the case of public clouds, if the provider transfers sensitive data on servers located in countries that do not have the same privacy standards or copyright laws. The provider should clarify be the used protection measures and how they are managed. The problem is exacerbated by the use of federated clouds and the exchange of data between data centers in different countries.

- Technological risks which were identified by the Cloud Security Alliance [14]:

  – Data leakage

    Unauthorized access to sensitive data stored in the cloud can lead to threats like data modifications or disclosure. The management of authentication and authorization mechanisms is crucial, in the sense that sensitive resources should be protected from unauthorized users. When the user entrusts his data to the cloud, he is not aware of their location and the way they will be treated. Data leakage can also arise due to vulnerabilities in the cloud APIs, which users can access data through. Remedies could lie in strong access control, encryption and protection of the data integrity and effective key generation. Another practical solution to this type of problems can be found in the application of advanced encryption techniques, such as the one based on attributes: the decision of which users can decipher a text, is taken on the basis of attributes and policies associated with both the text and the user. Also data persistence must be addressed: when a contract between a provider and a customer ends, data must be effectively and completely destroyed. Hence, techniques for locating data in the cloud and erasing/destroying them when needed should be available.

- Data loss

  Data stored in the cloud can be lost due to different reasons, not only because of malicious attacks. Any accidental deletion by the cloud service provider or a physical catastrophe could lead to the permanent loss of customer's data. If a customer encrypts data before uploading it to the cloud and then loses the encryption keys, data can be lost. Remedies for this threat is a strong key management and procedures for data back-up and replication.

- Account or service hijacking

  Attacks methods involve phishing, fraud, and software vulnerabilities exploitation. The attacker has the aim of intercepting credentials in order to be able to eavesdrop sensitive transactions and manipulate data. Providers can leverage strong two-factors authentication techniques and deploy proactive monitoring to detect unauthorized activities.

- Insecure interfaces and APIs

  Cloud interfaces provide the user with the ability to orchestrate, provision and manage all the implemented services. They constitute the unique entry point for users to interact with the platform: if the exposed APIs are vulnerable, there may arise several security threats, such as: clear text authentication, anonymous access, reusable token and weak access controls. Security and availability of the cloud services rely upon the security of these basic APIs. This risk can be lowered thanks to better authentication and authorization mechanisms. Service interfaces must be designed to protect the consumer from malicious attacks by applying authentication, access control, encryption and monitoring.

- Denial of Service

  This attack aims at inhibiting legitimate cloud users' access to data and/or applications. The objective is to make resources unavailable, by stressing system resources such as processor power, memory, disk space or network bandwidth. DoS attacks take advantage of vulnerabilities in the applications, databases, or other cloud resources. When there is such an attack, the consumer faces a service outage, and a higher computing power, that in some cases translates in huge costs.

– Malicious insiders

The convergence of different IT services under a single management domain causes the increase of this kind of threat. The provider does not specify how employees will have access to physical assets and there is a partial lack of transparency in the monitoring activities. The nature of Cloud Computing makes the realization of incident notification and re-mediation very difficult: monitoring alerts, incident indicators coming from intrusion detection systems and firewalls, in addition to the sources that need to be monitored can increase exponentially in a cloud environment. There is the need for an incident response strategy, namely the *Security Information and Event Management*, that is able to identify the source of the issue and to notify the event. If possible the system should also mitigate the effects of the alarm. Other remedies involve the specification of strict requirements on human resources as part of legal contracts and the guarantee of full transparency for the client.

– Insider attacks

Although the great possibilities that come along with the application of the cloud paradigm, new security challenges related to the virtualization technique must also be taken into account. Before even securing the running VM and the customer data, it is needed to be sure about the integrity of the virtual image which is about to be spawned in the cloud infrastructure. If an insider attacker gets access to the location where the VM images are stored, he has the chance to modify the way VMs will behave according to his malicious intention. This is proposed in [15] where an architecture for image access control, filtering, maintenance and scanning is proposed. The access control layer is responsible for granting access to a VM image: each image has an owner that can decide to provide a third party with a permission grant. The filtering module is useful to extract sensitive information from the VMs, while the tracking mechanism allows to record all the operations which were executed in the VM. Finally the image repository can be periodically scanned to verify if there are vulnerabilities after their publication with the aim of finding malicious software. Another important task is the verification of the guest VMs integrity and the behaviour of the components

of the platform. This is aimed at monitoring the activities in the cloud platform in order to detect unusual actions, while granting full transparency to the virtual guests [16]. The proposed architecture is able to monitor running VMs in a non-intrusive manner: it is basically designed as a lightweight middleware between kernel and hypervisor to guarantee full transparency to the VM. Sensitive data structures are monitored in order to identify malicious attempts of modifying the way the VM will behave, by periodically computing a checksum on the data to protect.

- Abuse and nefarious use of Cloud Computing

  The providers offer the illusion of unlimited compute, network, and storage capacity to final users. Some commercial providers even offer free limited trial periods. By exploiting anonymity, spammers, malicious code authors, and other criminals can be able to access computing resource to conduct their activities. Typical attacks can be key cracking, distributed denial of service, bot-nets and so on.

- Unknown risk profile

  Many organizations adopt cloud technologies with the promise of cost reductions, operational efficiency and improved security. On the other hand, indeed, if organizations do not really understand risks and operational responsibilities, such as incident response, encryption and security monitoring, they can incur into different issues. The monitoring of SLA performance indicators should be clear and software should be automatically scanned and patched for vulnerabilities.

- Shared technology issues

  Cloud vendors deliver their services in a scalable way, by sharing computing resources, network bandwidth and so on. Sometimes the underlying components of the infrastructure are not designed to offer a good level of isolation. The hypervisor mediates access between a guest OS and the physical resources. However, also hypervisors exhibit vulnerabilities that enable, for example, an operating system to gain inappropriate control levels on the underlying platform. A defence strategy involving compute, storage and network security enforcement is needed.

Cloud security issues may assume different emphasis with respect to the service model and the cloud deployment scheme. As an example, private cloud is characterized by fewer security issues in comparison to public clouds. Here, indeed, network and data are stored and managed by third parties and off premises. In a private cloud environment, security mechanisms used for the physical network can be replicated in the virtualized environment and fall are under the control of the company itself. However, severe issues hold for both the deployment models: insider attacks can come from legitimate users. In addition to this, all the deployments and the business models inherits the vulnerabilities and threats typical of the virtualization technology.

1.5 Green Computing

The computing paradigms described in section 1.2 have been marked by a growing awareness of the green concern, as shown by the birth of *Green Computing*. Cloud paradigm was also characterized by the introduction of this aspects, as proven by the aim of melding the elastic resources provisioning with the energy efficiency. The motivations of the increasing interest lie in the fact that data centers are experiencing the need to drastically reduce the energy consumption. This is a double-face issue: the first one is related to an environmental concern and coincides with the arising problem of CO_2 emissions. The second one essentially deals with economic reasons: it was estimated that the total energy bill for data centers doubles every five years. The importance of the green problem was also stressed by the U.S. Environmental Protection Agency report [17]: it has found out that the daily power consumption of a typical data center is equivalent to the monthly power consumption of thousands of homes. The principal sources of energy consumption are:

- servers and storage which account for 50%;
- the room cooling system (34%);
- networking (7%);
- conversion (7%) and lightning (2%).

A definition of **Green Computing** can be found in [18]:
it is defined as the study and practice of designing, manufacturing, using and disposing of computers, servers, and associated subsystems, such as monitors, printers, storage devices, and networking and communications systems efficiently and effectively with minimal or no impact on the environment.
The awareness in the data centers energy consumption has become the main purpose of this new computing model. The focus is on how to lower power consumption costs and, on the other side, to maximize the system's efficiency. This objectives need to be pursued during the initial phase of the system design, in order to use techniques and models which increase the green benefits. This process imposes new challenges to the system engineers: the design of the architecture must involve green aspects regarding the interactions among the system components, the network devices and the software stack. The main focus of green techniques is the measurement of the system energy cost, and its relation with the power consumption model of the hardware components. They strictly depend upon the overall power consumption models, which are often inaccurate for complex computer systems: in these cases, the energy-aware decisions can reveal as sub-optimal. Methods to evaluate the power consumption of a generic system can regard direct hardware measurements, theory computing and simulators.

1.5.1 Power Consumption in Data Centers and Energy Efficient Metrics

Data center typical infrastructure has been distinguished by a revolution which was fostered by the success of new computing paradigms and network topologies. Data centers are aimed at hosting and managing a set of different types of hardware resources, in order to provide services and applications. The classic architecture of data centers follows the so-called *three tier* paradigm, which is characterized by a collection of servers at the bottom level, which are grouped into racks. Each rack is equipped with a Top of Rack switch, that is often replicated in order to increase the reliability of communications. In the upper level, one or more aggregate switches are in turn connected to core switches. The traditional tree hierarchy based architectures are being extended for overcoming the typical scalability limits and increasing the network throughput. Among the objectives which are often pursued by data center owners we can mention, on one side, the reduction of the operational costs, manual maintenance and the carbon emissions, and on the

other one, the increase of the devices lifetime. Some of these purposes are addressed by the *Data Center Infrastructure Management* (DCIM) layer, which tends to include all the assets and physical resources in the traditional management procedures, such as IT asset life-cycle management and facilities monitoring. Among data center typical assets, there are critical equipments (also known as IT equipments), which are necessary to the service delivery and non critical equipments which are essential to the correct operation of the first category of devices (e.g. power units, cooling equipment and generators). Both the equipment types are very energy demanding and they can be characterized by inefficiency: regarding the IT devices, reasons of a poor energy efficiency can be found in the utilization of old servers and power supply units that consume too much power, or in the unbalanced ratio between the physical resources and the actual needs. In particular, the second aspect has been the main motivation behind the massive application of the virtualization technique: it can increase energy efficiency, which is often expressed as the ratio $\frac{Gbps}{Watt}$ and refers to the achieved energy saving to satisfy a determined computational load. Due to the complexity of the data center infrastructure, the energy efficiency cannot be independent from the input workload and the environmental conditions. The evolution of the DCIM has promoted the birth of many associated data center performance management and measurement metrics, such as the:

- *Data Centre Infrastructure Effectiveness* (DCiE), which is defined as the fraction of the IT equipment power supply and the total data center power:

$$DCiE = \frac{IT\ Equipment\ Power}{Total\ Facility\ Power} \tag{1.1}$$

  where the total facility power is the measured one at the incoming data center meter, while the IT equipment power is the one consumed by IT resources. For instance a DCiE of 0.60 indicates that the 60% of the total incoming power is used for non critical equipment (cooling and power utility), while only 40% is involved in the real computational processing;

- *Power Usage Effectiveness* (PuE) [19], that is a measure of the efficiency through which a data center uses energy, or in other words, how much energy is used by the computing equipment (in contrast to cooling and other overheads). It is the inverse of the DCiE:

$$PuE = \frac{1}{DCiE} \tag{1.2}$$

PuE has an ideal value of 1.0, which implies that all the used energy goes to the IT equipment;

- *Data Center Energy Practitioner* (DCeP) [20], that is a measure of the productivity of the data center, in contrast to the PuE and DCiE that do not take into account the useful work. This is intended in terms of effective tasks of the data center, such as transactions per second. This metric is defined as:

$$DCeP = \frac{Useful\ Work}{Total\ Required\ Energy\ to\ produce\ the\ work} \tag{1.3}$$

- *Carbon Usage Effectiveness* (CuE) [21], which is a green sustainability metric defined as:

$$CuE = \frac{Total\ CO_2\ emissions\ caused\ by\ Total\ Data\ Center\ Energy}{IT\ Equipment\ Energy} \tag{1.4}$$

where the denominator is the same value as the numerator of the PuE metric and the numerator is the total carbon emissions caused by total data center energy. The CuE unit is kilograms of CO_2 over kilowatt-hour (kWh);

- *Data Center Predictive Model* (DCPM), which is the ability to forecast the performance of a data center in the future, e.g. its energy use, energy efficiency or cost.

1.5.2 Energy Efficient Techniques

The application of energy-aware techniques can offer multiple benefits in terms of environmental protection and costs reduction. A computer system is a clear example of different subsystems that interact among them: it can be characterized by energy state changes, which energy-aware decisions can rely on, in order to reduce the overall power consumption. To this aim, plenty of information is needed, such as application run-time data and monitored power consumption states of the sub-components. Energy reducing strategies can be designed according to the way systems and applications respond to state changes and by adjusting the behaviour of the subcomponents. One approach is to directly control the power-saving aspects of some components when an under-utilized threshold is activated: a system may hibernate components such as CPU and RAM or

lower the voltage by reducing performances. Applications can take advantage of monitored power consumption states by applying energy-efficient policies. There are different research papers which introduce energy-awareness management tasks in the operating system, hardware abstraction and resource allocation [22]. Energy-efficiency has also interested networking architecture and protocols design. This task is also referred to as *Green Networking*: the usual approaches deal with techniques for dynamically adapting network devices processing capabilities and available bandwidth to meet traffic load and services requirements, or standby-wake-up modes [23].

By focusing on IT equipments, among the strategies that can help to achieve a high energy efficiency there are:

- identification of old-fashioned servers which reduce the overall efficiency and are not used for any crucial activity anymore;
- migration to more efficient resources. For instance by using blade servers, the benefits arising from their deployment are related to:
  - a single power supply for all the blades;
  - less heat generation in comparison to traditional servers thanks to shared cooling and power;
  - a reduced cost for the network devices and wiring, since the blade provides a bus (or more buses) which all the servers can send network traffic to;
- the use of multi-core processors which allow a better energy consumption thanks to the introduction of power states;
- energy-proportional computing, whose aim is to use servers at a threshold close to the maximum capacity. Several works in literature point out that the actual servers' utilization is under a cost-effective percentage. Some techniques address this problem by leveraging the introduction in modern CPUs of the so-called *P-states* (performance states), which allow clock rate reduction and *T-states*(throttling states), a technique that automatically adjusts the frequency of a microprocessor. With the enabled p-states, the power consumption can be reduced at low workload and, the more states are used, the more this techniques can achieve efficiency.

Among the planned strategies for increasing energy efficiency, data centers are massively exploring the virtualization and its consolidation advantages to face the IT equipments under-utilization. This mechanism allows to create isolated virtual environments: it allows to drastically reduce the number of used servers and to avoid the commitment of a single server to a dedicated activity. The task scheduling decisions can be also based on green aspects: for example, it is possible to specify policies, in order to consolidate the workload on a subset of more efficient servers, or by realizing a load balancing of the requests across physical hosts.

1.6 VM Live Migration

Virtualization-enabled techniques, such as *VM Live Migration*, can help in obtaining energy efficiency. For instance, migration can be used to centralize the computational load in data centers, which are endowed with a better cooling system or placed areas with low temperatures or countries with low electricity prices. Migration algorithms can also be used to consolidate workload in data centers with high renewable power.
Live migration is a facility implemented by all the hypervisors which enables the transfer of a running virtual machine from a source host to a destination one, assuming a shared storage for the VM images, with almost negligible downtime for the applications. The first one is the so-called "**pre-copy**" phase, during which the VM still runs on the source server, while its memory is iteratively copied to the destination. The algorithm is divided in "rounds", that are the intervals of time needed to transfer a determined number of memory pages. In the first round all the RAM memory of the VM is copied to the destination site: during this transfer, some memory pages could have been written by the running applications. That is why other rounds are necessary to complete the live migration process. The second stage is called "**pre-copy termination**" phase: the iterative copy can be stopped when a condition is met, such as the number of maximum iterations, the amount of memory already copied or the number of pages that were written in the previous round. The third step is the "**stop-and-copy**" phase, which implies the suspension of the VM at the source host, in order to send the last dirty pages over the network and to resume its execution at the target site. VM live migration can be evaluated according to different parameters:

- the *migration time*, that is the time required by the three-phase algorithm for copying the VM memory and its state from source to destination;
- the *down-time*, which refers to the time needed to perform the stop-and-copy stage and represents the interval of time in which the VM is not up;
- the *energy overhead* due to the network transfer of the VM related data.

1.7 Cloud Computing and Mission-Critical Infrastructures

Over the last years, IT technologies are increasingly becoming widespread thanks to the availability of bandwidth and the decreasing hardware costs. This is proved by the fact that applications from disparate domains are relying on digital services provided through IT technologies. In addition to this, digital transformation is causing the traditional business models and processes to be renewed, due to the success of cutting-edge paradigms, fostering ubiquitous and low-cost computing. IT-enabled processes and services are extremely linked to everyday lives: the infrastructures delivering mission critical services are assuming a great importance in the security hardening, because their failures translate in loss of money, unfulfilled crucial activities and, often, in human lives hazards. Within the expanding scope of mission-critical applications and the fulfilment of their expected service level requirements, companies are increasingly facing the problem of adopting new paradigms, in order to execute them in a cost-effective way, without the need to increase data center resources expenditures. With the technological progress, mission critical systems cannot be isolated anymore and they should be be part of agile, flexible and low-cost platforms. They are also characterized by arising security threats, for which an holistic cyber-security approach is needed. On one hand new technologies are often characterized by a great number of benefits in terms of cost-effectiveness and, on the other, they may emphasize the risk of sensitive data loss and, as a consequence, they can have a disruptive impact on mission-critical functions. Several organizations are adopting virtualization and cloud technologies, with the aim of reducing power consumption and increasing the overall resources efficiency. Anyway when it comes to the application of virtualization technologies to mission critical applications, service levels, business continuity plans and and risk management strategies must be carefully taken into account.

This thesis aimed at exploring the two described aspects of the cloud paradigm: on one hand, with the proposal of a cloud based architecture for enhancing the infrastructural security and, on the other, by investigating new energy-efficient models and algorithms. As a mission critical infrastructure, a real an Air Traffic Control (ATC) system was considered and deployed on a cloud platform. The investigation started with the selection of a well-suited cloud IaaS platform for such a system: by having in mind the requirements, the nature of critical infrastructures and the key performance indicators of cloud services, some evaluation parameters were identified. After analysing the identified parameters, an experimental evaluation of two cloud open-source projects was performed in **Chapter 2**. Since the dependability requirements of mission critical systems demands for automatic disaster and recovery mechanisms, the possibility of exploiting them in a cloud-based infrastructure was investigated. To the aim of giving a response to the need for security mechanisms, in **Chapter 3**, an architecture for a multi-layer security is discussed and properly designed. It is provided with a mitigation strategy that leverages, on one hand, virtualization-based mechanisms, such as the VM Live Migration, and on the other, a Software Defined Networking approach to guarantee dynamic network reconfiguration.

The green enabling techniques the thesis focus on is also based on the VM migration: this is used to increase the overall cost-efficiency and the resources utilization. One of the techniques proposed in literature for improving resource utilization in a cloud-based system is the *VMs Consolidation Problem*, which consists in finding, given the current allocation, the minimum set of migrations, defining a new allocation scheme, with the objective of minimizing the number of used servers. The problem was extended in **Chapter 4** to take into account the different efficiency models of the servers. Considered a set of VMs, given the current allocation, the amount of resource available at each server and the one requested by the VMs and the power consumption model of the servers, the problem consists in finding the set of migrations defining a new set of active nodes that minimizes the linear combination of the power consumption of the resulting active servers and the number of migrations. The problem was also formulated through a Mixed Integer Programming Model (MILP), in order to have an idea of the optimal solution, and a Simulated Annealing based heuristic was also proposed not to limit the scalability of the resolution. The algorithm is based on a simple idea, which consists in the measurement of the VM live migration effectiveness: in order to have a

balance between the nodes consolidation and the overall power consumption, the computed value takes into account the convenience of migrating a VM from the source node, the willingness of the destination one for accepting the VM and the difference in the power consumption after the VM migration to the new node. Anyway, since the typical data center power consumption cannot exclude networking devices, the network topology was also introduced in the proposed model with the objective of achieving a trade-off between different objectives. These are the minimization of the power consumption of the used servers, the network power consumption and the number of migrations. Each one of these objectives can be emphasised by using different weights in order to give more importance to one of them. An optimization model for this second problem was proposed, besides a heuristic to solve it in a reasonable time was properly implemented.

Chapter 2

A Cloud Platforms Assessment for Mission-Critical Infrastructures

2.1 Context and Motivations

Critical infrastructures are more and more linked to everyday life: they basically consist in different subsystems that interact with each other with the aim of accomplishing sensitive services. The complexity of such infrastructures is managed through plenty of computer systems and their life-cycle from design to implementation is often handled through human routine maintenance procedures. Therefore the administrators of complex systems are starving for technologies enabling automatic processes for configuring a huge number of nodes. Nowadays cloud paradigm has become a key technology, which both academia and industry focus their research interests on. Industrial experiences show that the IaaS cloud layer can be exploited to create virtualized testbeds realizing critical services, with the objective to reproduce real operational environments in house. When dealing with real production systems, virtualization can enable several benefits in terms of:

- the chance to reproduce real world scenarios aimed at performing distributed testing campaigns. In order for the system to be operational, a very exhaustive experimental campaign is needed to guarantee that every single requirement is met with

the expected level. Industry can leverage a distributed and virtualized testbed to bring down the costs needed to realize the testing campaigns;

- the availability of automatic procedures to implement backup or disaster and recovery of the entire testbed. Cloud platform are increasingly adopting mechanisms to provide the user with recovery mechanisms, since service disruption in a cloud platform can be very common;
- the possibility to configure and manage the testbed components and their versioning through automatic mechanisms.

The mission-critical infrastructure that was used as use case is a real Air Traffic Control (ATC) system: for such an application, Cloud Computing is a cutting-edge technology for setting up an Extended Enterprise Private platform, with the purpose of geographically connecting distributed ATC centers for dependability reasons, e.g., by realizing a fail-over configuration among centers in order to increase the overall system availability. ATC systems are very demanding from the requirements point of view and software-intensive. In order to virtualize such a system, a cloud platform is needed to guarantee:

- the reproducibility of complex hardware configurations, which should achieve performances as nearest as possible to the one obtainable in the operational situations. This is very important, for example, for the network configuration. The different nodes can be provided with different different dependability techniques: as an example, all the links in the network can be in a redundant configuration, namely *bonding*, to dynamically respond to network outages;
- a minimum impact on the applications, which are very resources demanding. To this aim, the requirements needed by the cloud applications show be very minimal with regards to the CPU and RAM specifications;
- low resources provisioning time in order to realize fast fail-over techniques and disaster and recovery techniques with the minimum service disruption.

That is the motivation why an initial scouting phase was conducted and a group of key features was established in order to restrict the number of IaaS platforms to compare. The objective of this stage was to identify the project meeting the described

requirements. Afterwards the selected projects were compared by taking into account the resources stress and the resources provisioning time. In particular the technical key requirements that were taken into account for the scouting process are:

- full open-source license;
- the support for KVM [24] technology;
- the support for OpenvSwitch [25] technology for handling the creation of the virtual network interfaces;
- a strong community and then
  - bug reporting;
  - continuous software maintenance;
  - updating process and new releases;
- fully downloadable packages without limitations;
- industrial support and existing experiences.

2.2 IaaS Platforms Performance Evaluation

The evaluation of Cloud Computing IaaS platforms is not a simple task, due to the fact that the available projects are intrinsically different. In most works of the literature, only functional and qualitative characteristics are often considered as discriminating factors. Among them there are customizability, security and network-related issues. Semploski *et al.* [26] provide a critical comparison between three open-source projects that offer Infrastructure-as-a-Service, namely OpenNebula, Eucalyptus [27] and Nimbus [28]. The authors present the mechanisms needed of the spawning of a new VM and the differences between them and a raw level comparison according to the provided features. In [29] a process for implementing a custom benchmark, useful to understand the performance of various cloud IaaS offerings (in a pay-per-use model), is presented. The authors face the lack of precise indicators to assess the performance of the same VM running on different platforms and the inapplicability of hardware benchmarking. They show the results of the proposed benchmarking suite of applications by considering two instance types (a

cheap and an expensive configuration) and some factors, such as the data-center location and the provided performance level both in a short-term and a long term utilization. In [30] a performance comparison between Eucalyptus and OpenNebula is conducted. The workload consists of the software packages needed to run a web application very similar to Wikipedia [31] web service and a benchmarking application which emulates an online stock system. They attempted to compare the platforms in terms of:

- provisioning time of VMs;
- elapsed time of batch processing;
- throughput of web transactions;

by considering several configurations differing for the location of VMs images (local disk or NFS storage) and their allocation type (using dense or sparse files). The experiments prove that the best results for the three parameters are not always achieved with the same configuration.

In [32], the authors argue that classical benchmarking techniques, like TPC-W, are not suitable for the characteristics of the cloud platforms, which lie in scalability, pay-per-use model and fault tolerance. For the purpose of addressing the lack of a standard benchmarking technique for Cloud Computing, they suggest their static metrics to assess the dynamism and the elasticity which are typical of the cloud paradigm. Ming *et al.* [33] propose an analysis of the VM *start time*, which they consider as the time needed to find an allocation for the virtual appliance and to copy, boot and configure all the required resources. The authors compare the start time of three different cloud providers (Rackspace [34], Amazon EC2 [35] and Microsoft Azure [36]). The experiments aim at evaluating how the start time is influenced by different parameters, such as the date and time, the size of the image, the virtual hardware template and the data center location. They demonstrate in their findings that the start time is linearly dependent with the OS size and it is also related to the instance type, while it is quite constant when requesting a pool of virtual resources. VMmark [37] is a benchmark to assess the stress load of the physical nodes' resources, such as RAM, CPU utilization and I/O load, by deploying a set of VMs on top of VMware [38] technology. Also Casazza *et al.* [39], in their vConsolidate project, suggest a benchmarking for evaluating the performance of different workloads consolidated in cloud hosting servers.

2.3 Platforms Under Evaluation

Starting from the requirements described in section 2.1, the following evaluation was restricted to two different projects, namely OpenNebula and OpenStack. Two testbeds with the same hardware specifications ware installed and properly configured, with the purpose of performing an experimental evaluation of these platforms. In the next section some key features and the structure of the considered platforms are briefly explained.

2.3.1 OpenStack

OpenStack is an open IaaS platform, whose objective is to offer a collection of different services which can be deployed in a fully distributed and scalable fashion. It started as a project under the name of OpenStack Austin in 2010 and it has been characterized by different releases (the latest stable one is called Juno and it is the tenth one). OpenStack offers a number of services which exchange messages through an asynchronous communication, among which there are:

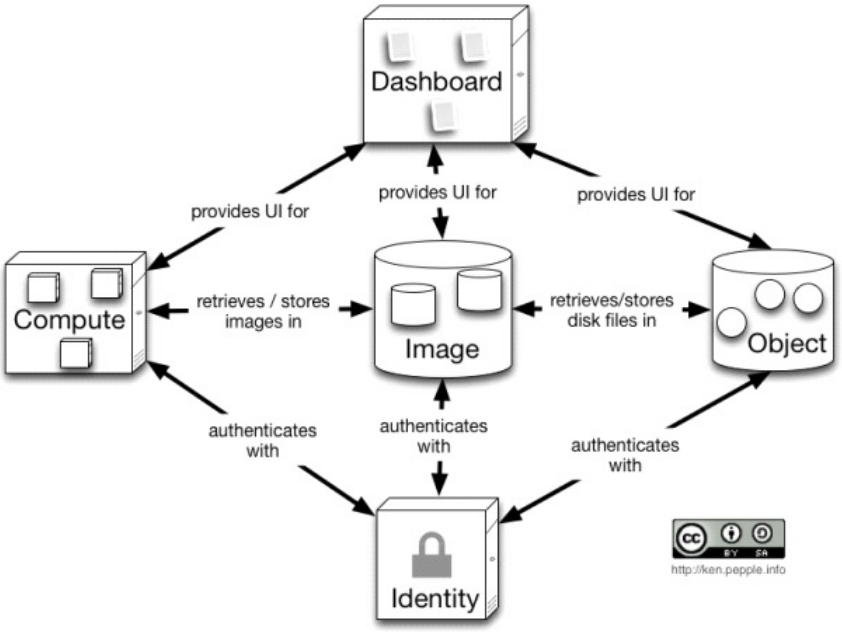


FIGURE 2.1: OpenStack Services

- **OpenStack Compute (Nova)** is the main service that allows to initialize a set of virtual instances and it is responsible for the network configuration for each

instance. Nova is the service that manages the entire life-cycle of the instances within the cloud infrastructure. It is completely agnostic to the type of hypervisor that provides the virtualization services and it interacts with it via the exposed APIs. Among the subcomponents of Nova, there are:

  - *Nova-API*, namely the interface to the outside world which is the only way users have to interact with the cloud infrastructure. This service also allows to manage virtual appliances from Amazon, by using the Elastic Cloud (EC2) APIs. This component is designed as a RESTful web server and it communicates with the other components of the infrastructure via a message queue based protocol. In particular asynchronous request-response calls are used and callbacks are used as a trigger, which is activated when the output for a specific request is available.
  - *Nova-Compute* is the service which handles all the life-cycle of a virtual appliance, by interacting with the configured hypervisor.
  - *Nova-Network* is responsible for the configuration of the virtual network of the VMs: it allocates IP addresses and creates networks according to the chosen types, such as flat networks, VLAN based networks and so on.
  - *Nova-scheduler* acts as a dispatcher of the Nova-API calls for the creation of a VM. The scheduler chooses a compute server from a pool of available resources through an algorithm which is based on configurable parameters.

- **OpenStack Object Storage (Swift)** is the service used to store objects ensuring redundancy and fail-over features. An object is a storage entity with meta-data that represents and describes it. When a file is loaded in the object storage, it is not subjected to either encryption or compression. Swift allows to store, retrieve files, back up data or as an archive for developing applications that require the integration with a storage system. Among the functions implemented by Swift there are:

  - secure storage for a large number of objects;
  - redundancy;
  - data container (applications and VM images);
  - scalability e backup;

Among the Swift components, there is the container that manages a list of objects that are stored in a particular structure. This can be seen as a folder as in a filesystem, although the difference lies in the fact that the containers can not be nested. A container needs to be created before starting uploading any file. The object component has the responsibility to store, search and delete objects. The Proxy is the node the consumers interact with by using the exposed APIs. The proxy receives requests to upload files, change meta-data or create containers. The Ring contains information about the physical location of the objects stored in Swift. Finally the Account is the server that manages the list of containers.

- **OpenStack Image Service (Glance)** is the service that allows to store, search and obtain original VM images. It can be used in combination with the Object Storage to stock images or it can use a special interface with the Amazon storage solution, namely S3. In particular Glance can use as storage for the images:
  - the local filesystem (default);
  - OpenStack Object Store;
  - S3 storage;
  - HTTP (read-only).
- **OpenStack Volume (Cinder)** is the service which handles the creation of additional volume partitions, which can be created on the fly and dynamically exposed to the VMs through the iSCSI [40] protocol. Volumes are allocated as block storage resources that can be attached to instances as a secondary storage or they can be used as the root storage to boot instances. A snapshot is intended as a read-only point in time copy of a volume, that can be created from one that is currently in use. A volume is requested via Cinder, which creates a logical volume into a particular group. This volume is then attached to an instance by creating a unique iSCSI session with the compute node.
- **Dashboard (Horizon)** provides users with a web-based platform to access and manage all the implemented services. It can be used to manage instances, mount volumes for instances, create a key pair, manipulate containers and so on. Among the features available through Horizon there are:
  - Instance Management: create or terminate instance, view console logs and connect through VNC etc.;

  - Access and Security Management: create security groups, manage key-pairs, assign floating IPs, etc.;
  - Image Management: edit or delete images;
  - Manage users, quotas and usage for project;.
  - User Management: create user, assign roles, etc.;
  - Volume Management: creating volumes and snapshots;
  - Object Store Manipulation: create, delete containers and objects.

- **Keystone** provides authentication and authorization procedures based on policies for all the OpenStack services. Authentication determines if a given request is actually coming from the user who requested the service, along with the provided account information. There are two types of authentication: the first one is based on the account (username-password), the other is based on token. Authorization is the process through which an authenticated user can be granted the access to a particular service. This service also handles the user management functionalities by keeping track of users and what they are permitted to do. A user can be assigned different roles in different tenants, which can be thought of as a project, group or organization. Keystone provides a token and a service catalogue containing information about the endpoints of the services a user is authorized to access to.

- **Network (Neutron)** is the service which handles the creation of a network for the virtual appliances and it can be used as a substitute for Nova-network service. The structure of Neutron is very modular, since it enables to use a specific plugin which defines the interface with a networking driver (e.g. Linux Bridge).

2.3.2 OpenNebula

OpenNebula (One) was born as a research project in 2005 with the main purpose of designing and implementing an efficient and scalable management platform for VMs on large-scale distributed infrastructures. The *One* platform assumes that the underlying physical infrastructure is built up in line with the classical cluster-like architecture, where the basic components are: front-end, hosts and datastores. The front-end node is where the platform installation has been accomplished: it is responsible for the execution of the centralized services (monitoring and management, scheduling, web interface and

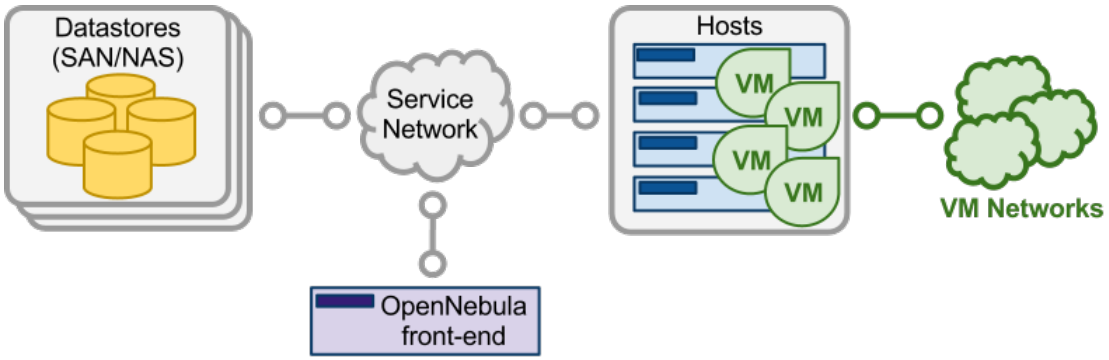


FIGURE 2.2: OpenNebula Architecture

so on). Hosts are the hypervisor-enabled nodes providing the resources needed by the running VMs, while datastores are the storage areas holding the VM disk images. The last versions of OpenNebula have been integrated with a series of applications which are oriented to the automation of configuration tasks and the resources scaling. These applications make the adoption of such a platform very attractive for large testbeds when there is the need to create on-demand testing scenarios, consisting in a large number of virtualized systems that interact through different networks. These applications, called respectively *OneFlow* and *OneGate*, are included in the latest available version. Among the advanced features offered by the platform, there are also:

- the implementation of High Availability (HA) architectures that deal with events of faults occurring on the VMs, on the host nodes or the front-end;
- the ability to aggregate heterogeneous cluster with each other, as well as the ability to create, in a physical data center, a virtual data center associated to particular security policies and different QoS levels.

The OpenNebula architecture is characterized by the installation of a single software module on a node that takes the name of manager, which also hosts the web interface for managing the features offered by the platform and for defining, instantiating and monitoring virtualized resources. The other nodes are the so-called hosts, which are running the VMs: on these hosts there is not any installed software module, even if they have to be properly configured to be included in an OpenNebula cluster. For the creation of a VM, OpenNebula uses a template-based approach, that makes it more flexible when

configuring virtual resources: in particular it is possible to specify parameters and low-level instructions to the hypervisor in a specific section of the template. Regarding the management of the storage areas, OpenNebula divides them into three main categories:

- *System Datastore*, that contains the images of the disks associated with the VMs running on the nodes. These images may be, for example, whole OS copy or simple file system links.
- *Images Datastore*, which is designed to contain the original images used to create the disks representing the running VMs. This datastore can be of different types in relation to the underlying technology:
  - file-system, used to store images in a file format. The images are loaded into a shared directory mounted on the cluster hosts;
  - Logical Volume Manager, to store images using LVM volumes instead of file formats;
  - specific storage solutions, such as GlusterFS, Ceph, or VMFS (VMware specific);
- Files Datastore, which can store files useful, for instance, in the context section of a VM template.

For each datastore different transfer mechanisms can be used: this allows one host to simultaneously access storage areas managed with different transfer modes. Among them there is, for instance, the shared mode, thanks to which the datastore is exported as a shared file system to hosts (e.g. via NFS). Otherwise images can be copied into the hosts' datastore via ssh.

2.4 Functional Assessment

In order to realize a functional assessment of the two platforms, the discriminating factors which were taken into account are related to the the reproduction of the real system and the design of recovery strategies among distributed data centers. Therefore, these parameters were considered:

- the support for distributed storage technologies, needed for WAN-optimized data replication;
- federation capabilities for the management of multiple data centers, which are geographically distributed;
- VM configuration flexibility;
- network configuration flexibility;
- easiness of installation.

OpenNebula and OpenStack have quite the same capabilities in terms of storage and virtualization technologies support, as shown in Table 2.1.

TABLE 2.1: Storage Technologies Support

Storage	NFS	GlusterFS	CEPH	iSCSI	VMFS
OpenStack	Yes	Yes	Yes	Yes	Yes
OpenStack	Yes	Yes	Yes	Yes	Yes

TABLE 2.2: Features

	OpenNebula	**OpenStack**
Installation	moderate	complex
VM/Network customization	high	moderate
Federation capability	zones	cells (exp. authentication)

Anyway they are quite different when dealing with virtual resources configuration as proved by Table 2.2. OpenNebula platform is simpler to install and configure, since it is characterized by a monolithic structure, while OpenStack requires the installation and configuration for each of the described services. This aspect emphasizes the complexity of this platform that increasingly tends to increase. On the other hand, OpenStack is characterized by a large community: this is an important feature when supporting and bug reporting activities are needed. It also offers a range of additional interesting services, which are, however, out-of-scope with respect to the porting of an ATC system on a virtualization platform.

The flexibility and simplicity in the VMs and virtual networks configuration based on the use of templates rather than plug-in settings represented a very important factor

for the considered use-case. Besides, the ability to exploit features offered by low-level hypervisor configuration parameters gave the chance to overcome some issues, such as VMs graphics peculiar characteristics.

2.5 Provisioning and Scheduling Time

SMICloud [41] is a framework based on the publicly available service measurement indexes for comparing and ranking cloud providers according to the service level agreed with the client and the perceived QoS. The framework receives the client's requests along with the application deployment requirements. By monitoring the performance of the cloud services, it is able to compute the ranking factors for all the providers which are listed in a service catalogue. The Cloud Service Measurement Index Consortium [42] has identified a set of business oriented key performance indicators (KPIs) that can be exploited in order to compare all the cloud services (IaaS, PaaS, SaaS, etc.). The framework includes both critical business and technical features and it has a hierarchical structure: in the top level, there are seven categories, each with a certain number of attributes. For the experimental evaluation of the platforms, a specific category, namely the *agility*, was taken into account. The *elasticity* is among the attributes included in the this category and it is crucial to the performance analysis of the cloud platforms: it can be interpreted as the ability of a service to adapt itself to the changes and the requirements issued by the applicant. It is also attached to the scalability of the service, that is a measure of the increase in its performances when additional resources are granted to the system. By having in mind the characteristics of the IaaS cloud paradigm and its application in the context of mission critical infrastructures, the elasticity attribute was evaluated in terms of the *provisioning time*. In general, the provisioning interval can be described as the time needed to activate or deactivate a generic resource. More in details, the provisioning time can be viewed as a service response time: it is the time interval starting with the request for a specific service and ending with its availability. In this case, the service consists in the creation of a VM in the cloud platform, according to a well-known hardware template. Thus, it is possible to consider an average response time for the creation of the virtual appliance as follows:

$$\frac{1}{N}\sum_{k=1}^{N} T_k \tag{2.1}$$

where, considered N subsequent requests of the same VM, T_K stands for the time between the client's request and the instant in which the resource is considered in an *active* status by the monitoring process of the platform. This is a state in which the VM disk was copied in the instances folder and its creation ended successfully, even if the VM is still in the booting phase, so it is not ready to serve the first user's request.

2.5.1 Analysis of the Provisioning Time

The time needed to deploy VM k can be divided in three different terms, as explained in the following formula:

$$T_k = T_k^r + T_k^s + T_k^e \tag{2.2}$$

where

1. T_k^r takes into account the issue of the creation request by using the APIs of the platforms and instant in which the command starts to be processed.
2. T_k^s is the time it takes for the platforms to select one of the available physical hosts with enough resources to host VM k. This time is dependent on the complexity of the scheduling algorithm implemented by the platform and on the number of the configured compute hosts.
3. T_k^e embodies the processing time including all the tasks necessary to spawn and configure the VM. It can vary according to the type of storage which has been chosen for holding the VM images: all the IaaS platforms allow to use different kinds of storage solutions. It has been proved in [30] that different configurations, such as local storage or NFS, can lead to diverse provisioning times. This factor is also influenced by the network configuration of the virtual appliance.

In order to obtain the provisioning and the scheduling times a Java-based application was developed and deployed on the controller node. Basically, the application exploits the APIs of the platform to perform the following tasks:

- constructing one of the virtual hardware flavors which were taken into consideration, according to a binary string that represents all the configuration parameters;
- requesting a new VM with the selected service offering;

- obtaining the status of the deployed VM;
- computing the time between the issue of the creation request and the ready status response.

2.5.2 CPU and RAM Stress

In order to compare the resources stress of the considered cloud platforms, the CPU and RAM utilization of the compute hosts, hosting the maximum number of VMs, each one executing the stress command [43], were evaluated. As first step, the number of VMs with the same virtual hardware template that can be instantiated on a single physical host, without over-provisioning CPU and RAM, was investigated. The possibility to over-provision the available resources can be enabled or disabled in the configuration parameters of the platforms. The objective was to reach a 100% CPU utilization on each VM, by balancing between user/system and I/O processes. The stress utility allows to specify the kind of stressing operation through some parameters, such as:

- cpu, which describes the number of forked processes executing a sqrt() command (user level);
- vm, the number of forked processes executing malloc()/free() commands (system level);
  - vm-bytes is the number of bytes, expressed in MB, which every vm worker has to allocate;
- hdd, the number of forked processes executing a write() command (I/O level);
  - vm-keep specifies to re-dirty memory, instead of freeing and reallocating it.

A combination of the three parameters was used with the purpose of achieving a 0% time in which CPU is idle and a trade-off between user, system and I/O CPU utilization, by setting these parameters:

- cpu 1;
- vm 1 and vm-bytes 350M;
- hdd 500 and vm-keep.

2.5.3 Virtual Machine Deployment

OpenStack

Regarding the elaboration time for the VM deployment, OpenStack only gives the chance to create *ephemeral* virtual instances, which means that the VM image no longer exists after the user deletes it. As a consequence, it is possible to create multiple VMs from the same image. The only way to obtain persistent images is to spawn an image from a volume previously created through the block service (Cinder). When the user creates a new VM from an image stored in the Glance repository for the first time, the deployment deals with these steps:

- request handling;
- virtual network configuration;
- image creation, which in turn is divided into the following sub-steps:
    1. the image is copied from the repository to a directory (called _base) of the VMs datastore;
    2. this image is then converted from its original format to raw;
    3. the image is copied again obtaining a different version for each requested flavor (this can be referred to as "flavor image");
    4. the image is resized using qemu-img resize according to the specifications of the requested flavor;
    5. the filesystem of the image is checked and resized to the specific filesystem;
    6. an instance disk is finally created through qemu-img create by specifying qcow2 as output format.

When the user needs a secondary (ephemeral) disk for the VM, the additional steps concern the creation of a disk which is formatted according to a particular filesystem and then attached to the VM. All the subsequent requests of VMs with the same image and flavor are not affected by steps from 1 to 5. Instead, when creating a VM with the same image but a different flavor, only steps 1 and 2 are skipped.
Prior to create the VM, the scheduler needs to know where to allocate it. OpenStack uses the scheduler to select a host where to spawn the new VM. The cloud administrator

can decide to deploy a specific type of scheduler by choosing among different policies, such as:

- filter (the default one), which defines a number of filters the compute nodes have to satisfy and computes a ranking score for each of them;
- chance, which chooses random one of the available hosts with enough resources;
- simple, which adopts a spread first policy, by selecting the least loaded node (it is no longer supported in the newer versions of OpenStack).

OpenNebula

OpenNebula creates and handles two datastores, which stand for the storage areas holding the VM disks. The original VM images are registered in a datastore, which is referred to as *images datastore*, whereas those related to the running VMs are stored into the *system datastore*. OpenNebula does offer the possibility to create a persistent image: by creating a VM with a persistent disk, every modification of the instance is preserved even if it is destroyed. Obviously, only one VM at a time can be created starting from a persistent image. Non-persistent images, instead, are deleted when the VM is cancelled. OpenNebula entrusts part of the VM creation to a centralized Transfer Manager which is pre-configured with various scripts, among which one of them is executed according to the chosen storage model (NFS shared in our case). This component is in charge of handling the transfer of an image between a source (the front-end) and a destination (a compute host). In particular the actions executed by the TM when creating a non-persistent/persistent image are:

- clone (non-persistent), which creates an exact copy of a disk from the images datastore to the system datastore of the target host;
- ln (persistent), which creates a symbolic link in the target system datastore that points to the source image.

When a deployment request for a non-persistent VM arrives to the TM, a clone script is executed: it copies the selected image it in the system datastore of the target host. On the other hand, if the user requests a VM with a persistent image, a ln script is executed and a VM root disk is created as symbolic link pointing to the original image.

In both the cases, after the VM disk is created, the virtual guest is deployed by using virsh. The second step deals with the network configuration which basically consists in:

- adding a new port to the virtual switch;
- defining a treatment for the packets that pass through this port by establishing two flows in the virtual switch.

As argued before, the other factor that can influence the provisioning time is the scheduling algorithm. In OpenNebula f pre-defined scheduling policies are available, such as packing, striping and load-aware, in addition to the opportunity of specifying customized rank expressions. In the investigation, the *packing policy*, offered by the platform was used: it allows to use the whole available CPU and RAM of each physical host. The platform also defines a configurable time value (scheduling interval) between two scheduling actions, fixed by default to 30 seconds. This parameter was set to $1s$ in order to minimize the whole idle time during consecutive allocations of VMs.

2.6 Experimental Evaluation

2.6.1 Configuration Parameters

Before setting up our experimental campaign, the parameters involved in the creation of a virtual machine were tuned, in order to evaluate their influence on the provisioning and scheduling time measures. This allowed to neglect the variability of some parameters and to set a specific combination of values. In particular these parameters were considered:

- the server load, as concerns the number of VMs already instantiated on the host;
- over-provisioning, that indicates if the amount of RAM and CPU physical resources can be virtually exceeded;
- the virtual machine image:
  1. type (qcow2 and raw);
  2. size;
  3. persistence;

- the scheduling algorithm;
- the instance type, also referred to as "flavor" or virtual machine template, through which the user can assign:
  1. the number of vCPUs (expressed as CPU slots);
  2. the amount of RAM (expressed in MB);
  3. an optional ephemeral secondary storage;
  4. the number of virtual network interfaces.

TABLE 2.3: Fixed Parameters

Host Load	**Scheduling Algorithm**	**OverProvisioning**
No VM	Fill First Scheduler	Disabled

TABLE 2.4: VM Flavor

VM Instance Type		
vCPU	**RAM(MB)**	**vNIC**
1	512	1

TABLE 2.5: Variable Parameters

Secondary Storage
Yes/No

TABLE 2.6: VM Allocation

VM Allocation			
OpenNebula		OpenStack	
Persistent	Non-Persistent	1^{st} Allocation	2^{nd} Allocation

The first experiments showed no appreciable variation in the results due to the server load, the over-provisioning option and the VM flavor, so they were considered as fixed. The picked values are shown in table 2.3. Since the aim was to obtain the relevance of the basic steps of the elaboration time (T_k^e) on the provisioning interval, the variability of the time measures due to different sizes and formats of the VM image was not investigated. The dependence of the provisioning time on these parameters was well studied in [30]

and [33]. The variability analysis of the copy of the image was of poor interest in the experimental evaluation: the provisioning time, indeed, is obviously affected by the copy of the image, which is in turn dependent on the network load of the switch connecting the clustered nodes and the NFS storage server. For this reason in the experimental campaign, only one case was considered, which involves the copy of the image, by fixing the size and the format.

2.6.2 Experimental Scenarios

As explained in the previous sections, the two platforms are intrinsically different when spawning a new VM. When a user requests a VM with a non-persistent image in OpenNebula, the deployment basically consists in copying the original image into the target system datastore and creating the VM by using virsh. On the other hand, the first time a user deploys a VM in OpenStack, a full copy of the image stored in the Glance backend is needed; then the image is converted and resized according to the flavor, created using qemu-img and spawned. In this perspective the time it takes to have a running VM can be comparable with the deployment of a non-persistent image in OpenNebula. Instead, if the VM is deployed starting from a persistent image in OpenNebula, a symbolic link to the original image is generated (without copying it). In OpenStack, after the first creation of a VM, all the subsequent requests for the same image does not involve its copy. Here is why the provisioning time in the latter case can be compared to the one achieved with the deployment of a persistent image in OpenNebula. By having this analysis in mind, three different cases were taken into account in the investigation:

1. case 1 involves the deployment of a persistent qcow2 image with a virtual disk size of 1.7 GB (effective size of 8.0 GB) in OpenNebula without a secondary disk and the allocation subsequent to the first one of the same VM type in OpenStack;

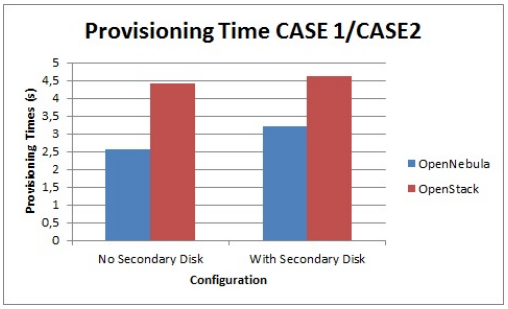
2. case 2 differs from case 1 because of adding a secondary ephemeral disk to the instance;

3. case 3 deals with the creation of a non-persistent image without a secondary disk in OpenNebula and the first allocation of such a VM in OpenStack.

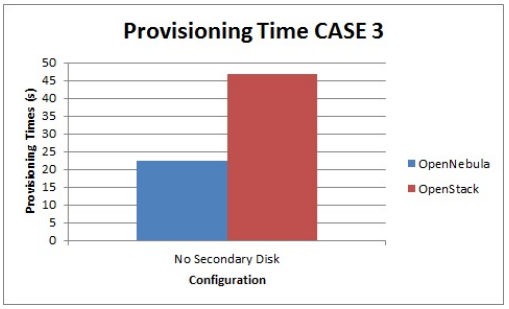
2.6.3 Results

2.6.3.1 VM Provisioning Time

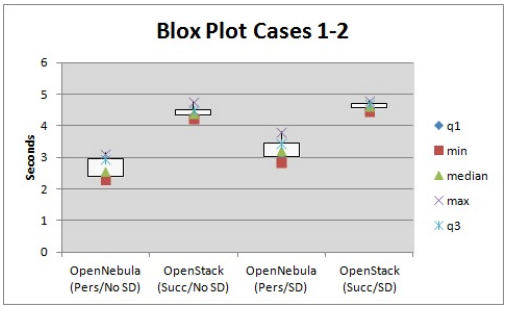
As remarked by figure 2.3(a), the creation of an additional ephemeral disk does not have a significant impact on the provisioning time in both the platforms. Therefore, in the analysis of the remaining case, the presence of the secondary disk was neglected. In the first considered case (the deployment of a VM with a persistent image in OpenNebula and the second creation of the same VM in OpenStack) OpenNebula outperforms OpenStack: for the latter, the mean provisioning time is about 42% longer (1,85 seconds). By default, OpenStack gives the chance of injecting metadata and an asymmetric key (for SSH password-less access) into the VM: anyway in our analysis this possibility was avoided and no firewall for the VMs was used. In figure 2.3(c) the variability of the obtained measures is illustrated: OpenStack has more stable results compared to OpenNebula, and this is due to a greater impact of the hosts monitoring process. The deep analysis of the logs of the platforms allowed to obtain the principal contributions of the provisioning time. As shown in figure 2.3(e), four factors were identified: the time to issue the VM request, the communication overhead in the platform, the scheduling time and the creation of the VM. In both the platforms the creation of the VM accounts for about 80% of the entire time and the presence of different modules in OpenStack also allowed to recognize a 12% additional overhead, which is related to the communication with the scheduler and the compute module. Nevertheless, the time it takes for OpenStack to spawn a new VM is longer with respect to OpenNebula and it is the reason why the influence in percentage terms of the VM deployment tasks (figure 2.3(f)) was investigated. OpenStack requires about 39% of the whole time for handling the VM creation and interacting with the other services, such as Glance and Quantum to manage the image and to get network-related information respectively. The management of the virtual networking (basically OpenvSwitch and tun/tap drivers) and the virtualization drivers accounts for about 30%, just as the creation of the VM image. OpenNebula, on the contrary, shows a simpler internal architecture and employs fewer commands for the management of the network and virtualization drivers. Anyway it should be observed that it was not possible to estimate the additional overhead in the OpenNebula platform, because of the smaller amount of data which can be deduced from the logs. The greatest impact on the provisioning time is related to the interaction with KVM and



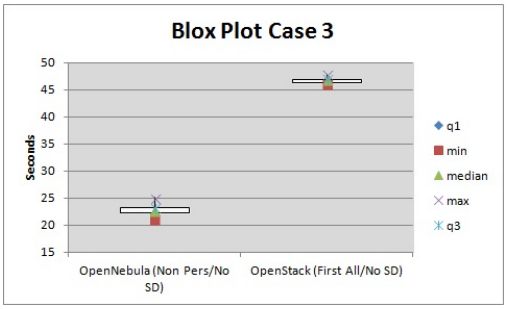
(A) Cases I-II



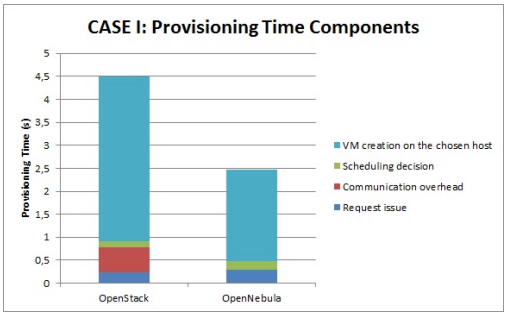
(B) Case III



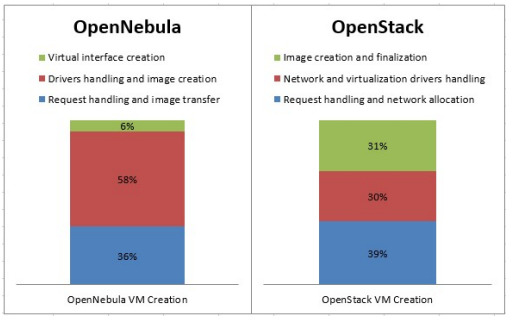
(C) Cases I-II



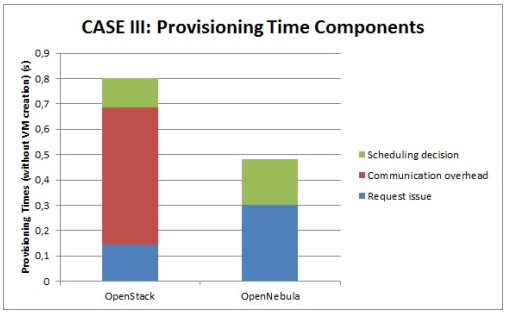
(D) Case III



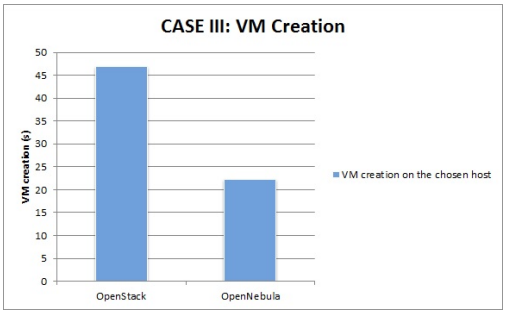
(E) Provisioning Components CASE I



(F) VM Creation CASE I



(G) Provisioning Components CASE III



(H) VM Creation CASE III

FIGURE 2.3: Provisioning Times Results

OpenvSwitch drivers and the execution of the VM deployment script.

Figures 2.3(b), 2.3(d) and 2.3(g) show the provisioning times of the third case, which deals with the creation of a non-persistent virtual appliance in OpenNebula and the

first allocation of the same VM in OpenStack. The time needed for the creation of the instance are prominently different for the two platforms: the time it takes for OpenStack almost doubles the one achieved in OpenNebula (2.3(h)). This time takes into account the copy of the whole image, which is executed between two directories, both exported via NFS protocol. Anyway the time required by OpenStack is double because the image, which is qcow2, is converted to raw and this conversion demands for about 20 seconds additional time.

2.6.3.2 VM Scheduling Time

In figure 2.4 the mean values for the scheduling times in both the platforms are presented: the scheduling algorithm implemented by OpenNebula is affected by slightly longer times with respect to OpenStack.

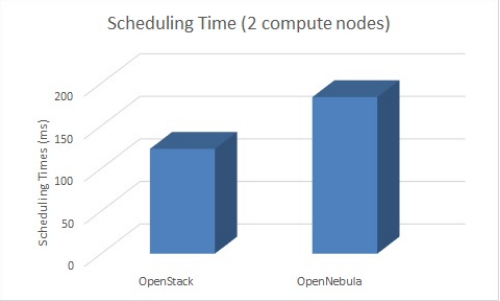


FIGURE 2.4: Scheduling Times Results

Nevertheless these results were expected by comparing the complexity of the algorithms which are implemented by the two platforms. The OpenStack scheduling algorithm consists in two stages: the first one has the goal of selecting hosts according to the filters specified in the Nova configuration file. RAM and CPU filters were configured with an "allocation_ratio" of 1.0, in order to disable over-provisioning. Afterwards, all the hosts that did not pass the filtering phase are excluded from the final decision. In the second phase there is a weighting process of all the filtered hosts according to the specified cost functions. The default one returns the amount of free RAM of the host and has a weight, which represents the behaviour of the algorithm (fill-first 1.0 or spread-first -1.0). So for each cost function, every host obtains a score and the final ranking is calculated as follows:

$$rank_i = \sum_j w_j s_{ij} \quad \forall i \tag{2.3}$$

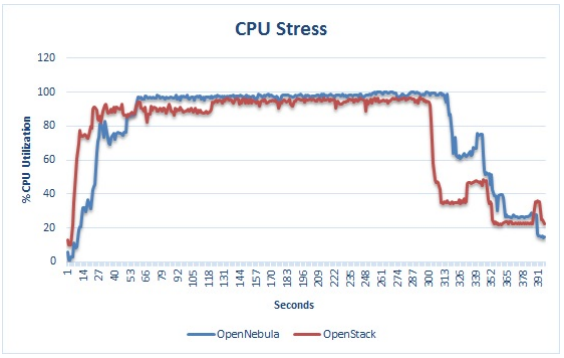
where s_{ij} is the score of node i related to cost function j and w_j is the weight of the j-th function. Besides the default cost function, another one for CPU slots usage was also included in order to choose a target with the largest combined utilization of CPU and RAM.
OpenNebula uses a match making scheduler which implements the rank scheduling policy. The allocation decision is taken after these steps:

- VM filtering, in which the VMs that request more storage than the available amount are filtered out and assume a pending state;
- host filtering, during which the servers that do not meet VM requirements or do not have enough resources (CPU and RAM) are filtered;
- hosts ranking, that is the evaluation of the remaining hosts, using a ranking score policy obtained through the information periodically collected by the monitor driver.

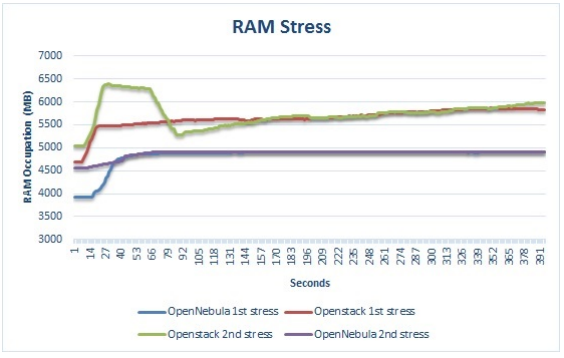
Regarding the ranking factors, OpenNebula allows to configure the policy, by using a predefined or a custom one. The packing policy was chosen: it has the objective of minimizing VMs fragmentation and the number of active nodes, by selecting the node with more VMs running on it. Anyway it should be pointed out that the scheduling time accounts on the provisioning interval for about 2,78% in OpenStack and about 7,15% in OpenNebula.

2.6.3.3 CPU and RAM Stress

The experiments aimed at evaluating the impact in terms of CPU and RAM utilization when stressing the two platforms. The maximum number of VMs was deployed and the linux stress command was used on each of them, by applying the parameters explained before and a duration interval set to 300 seconds. In figure 2.5(a), it can be noticed that, during almost all the stress time, the level of CPU utilization remains quite stable for both the platforms, even if the mean percentage achieved by OpenNebula is 97,45%, while for OpenStack it is slightly inferior, by settling on 93,2%. Figure 2.5(b) exhibits, instead, the occupation of RAM (in MB) after the allocation of 8 virtual machines and during the stress test. The graph compares two (almost consecutive) stress tests: in the



(A) CPU Stress



(B) RAM Stress

FIGURE 2.5: Stress Results

OpenNebula case, during the stress, the curves mostly overlap and they are characterized by a mean value of 4895 MB. OpenStack experiments shows a higher RAM occupation (5694 MB as mean value) and the second test exhibits a 10% increment in the pick RAM occupation value with respect to the first experiment.

2.6.3.4 Analysis of the Results

The functional assessment was intended at verifying which platform has more flexibility in the creation of virtual resources. It has been pointed out that OpenNebula gives the chance to easily configure VM characteristics by using a very modular template. This was of great help when creating VMs for legacy nodes characterized by specific hardware or graphics features. Besides in OpenNebula is much more simpler to create a persistent VM image which was the best fit option to the considered use case; OpenStack only allows to request ephemeral VMs by default, even if it has a procedure for making the image persistent. The results achieved for the evaluation of the provisioning time also indicate that OpenNebula is a very promising project, even if the results are not to be

granted for exhaustive. Anyway, by analysing the results of the assessment, OpenNebula platform was chosen as preferred cloud platform.

Chapter 3

A Cloud Based Architecture for the Security of Mission-Critical Infrastructures

3.1 Context and Motivations

The development of the industrialized countries is the result of the advanced services they are able to offer to their citizens by means of complex infrastructure, whose failure in terms of partial or total unavailability could have a catastrophic impact on both the economic field and on human lives. Critical Information Infrastructures (CIIs) are intertwined in many modern human activities such as communications, health, tourism and finance, and they rely on computer-based systems to exchange, aggregate and extract significant amounts of sensitive data. As a result, the act of preserving CIIs availability, integrity and confidentiality has a great impact on our lives: the unavailability of a given CII, due to accidents or malicious attacks, can affect the life of a nation. Transportation systems, electricity supply chain, telecommunications and military infrastructures are just some examples of such kind of critical systems. The political and economic relevance these infrastructures have achieved during the years makes them particularly vulnerable to failures due to unintentional, for instance natural disasters or technical problems, or intentional failures, such as malicious events or terrorist attacks. Next to recent computing evolutions, critical business services are characterized by an increasing digitalization

process. As a consequence, mission critical systems are no longer isolated and demand for flexible, agile and low-cost platforms to deliver their services, satisfy customer's experience and improve cyber-security. Nevertheless the obvious benefits coming from the application of the cloud paradigm, one of the main obstacles in the deployment of legacy critical infrastructures in the cloud is the lack of dependability assurance and solutions to increase the protection against external and internal threats. The security issues of the cloud paradigm can be considered at different levels: at the IaaS layer, users should be protected from each other when using the same cloud platform. Indeed, the mechanism that allows to obtain the first level of isolation and protection among users leveraging shared resources is the virtualization. However, not all types of virtualization allow to obtain an adequate protection level: network, for example, can be a potential vulnerability, in the sense that, if the virtual network is not properly set up and secured, the risk of providing potential attackers with the access to sensitive data becomes very probable. Several proposed approaches use an holistic approach in order to improve security in relation to different infrastructural layers, such as data, application and network layers. The nature of mission and safety critical applications really stresses out the lack of robust security solutions and the need for automatic disaster and recovery procedures. This is the reason why an architecture for implementing security mechanisms at different layers in a cloud environment was properly designed and implemented. The main idea is to provide this architecture with agile mitigation strategies in the event of attacks, by leveraging from one side virtualization mechanisms, such as the migration of virtual resources, and on the other, the dynamic reconfiguration made possible by the Software Defined Networking approach. By considering the same case study that was introduced in **Chapter 2**, a security risk evaluation methodology was applied. The starting point was the identification of the distribution of the system's vulnerabilities over the assets; then, the evaluation of the likelihood and the impact of the built threat scenario in terms of confidentiality, integrity and availability were computed. The threat scenario is intended as the exploitation of multiple vulnerabilities over the assets, along with the attacker's required skills to launch the attack. The threat scenario was used as proof of concept of the designed architecture in order to verify the attack detection and the mitigation strategy.

3.2 A Case Study: a Real Air Traffic Control System

An Air Traffic Control (ATC) system is a mission, business and safety critical infrastructure: it can be defined as mission critical, because a threat in the system could lead to a failure in the accomplishment of the mission, which coincides in this case with the flight from a source airport to a destination one. It is business critical since a failure in the system produces a huge loss of money, because people should be refunded if a flight does not reach its destination or it is cancelled. Finally a problem in the infrastructure could have effects on human lives: for instance, if the collision between two routes is not correctly detected, a very potential disaster could occur. The Air Traffic Management (ATM) systems represent a suitable example of complex System of Systems: they consist in a large number of heterogeneous hardware and software components that are typically spread over different ATC centres within a single country. A national Information and Communication Technology (ICT) Service Provider is usually responsible for guaranteeing confidentiality, integrity and availability (CIA) of the ATM data exchanged across a wide area network (WAN); whereas the Air National Service Provider is responsible for both safety and security of ATM functionalities (as foreseen in the regulation EC 1035/2011 [44]). To improve the robustness of the whole ATM system against possible damages, such kinds of systems are often built up in redundant configuration over a distributed infrastructure. ATC systems are very demanding and software-intensive: they are safety critical, highly distributed and hard real time. Among the dimensional architectural requirements of this type of systems there are:

- ultra-high availability (6 nines);
- high performance;
- modifiability;
- scalability;
- usability.

In the ATC field, ATC centers belonging to the same system are often deployed over different cities in a given country, either for fault tolerance purposes and remote connection needs at country level. ATC Units provide the all the traffic control related services

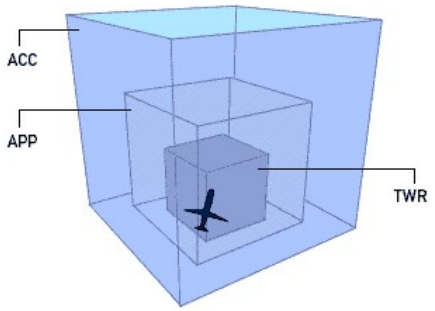


FIGURE 3.1: ATC Areas

and they also responsible for Flight Information and Alerting Services to pilots. In order to protect air traffic flows operating between airports, it is established a Control Zone, which can be defined as a volume of controlled airspace which extends from the surface of an airport to a specified upper limit. In particular, as shown in Figure 3.1, there are:

- the *Tower*, which consists in the air traffic zone which is part of the airspace surrounding an aerodrome. The control service of that area is in charge of administrating and assuring security to all the ground movements and to the traffic in the immediate proximity of an aerodrome;
- the *Approach Control*, which is the facility responsible for the control service in a bigger air volume which is interested by take-off and landing procedures;
- more control zones are part of a control area, which is a more extended volume of controlled airspace. The *Area Control Centre* (ACC) is the facility responsible for the control service of that volume. After take-off, ACC has the task of ensuring the proper distance and separation both vertically and horizontally among aircrafts and to ensure the criteria of fluency and safety. When approaching the destination airport, the ACC gives back control to the flight control tower of the airport itself or if there is a fly over, the control is passed to another neighbouring ACC.

3.2.1 The Area Control Center

The Area Control Center (ACC) is the main operative center in an ATC system. ACC is the infrastructure responsible for controlling flights in a particular volume of airspace at high altitudes between approach and departure airports. From the hardware perspective, the ACC system is composed by a set of cooperating computers (central units)

hosting the applications dealing with surveillance and flight data (for detecting any conflicts) and with auxiliary information (useful for recording services). Central units are also connected to other external systems for transmission and reception of ATC significant information. Consoles operating as controllers workstations are connected to the central units, as shown in Figure 3.2.

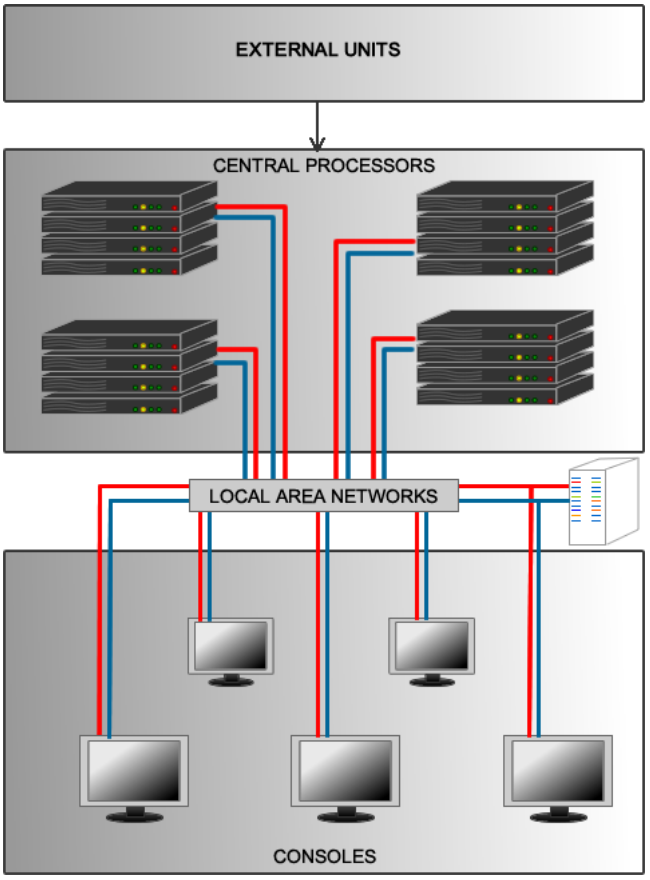


FIGURE 3.2: ATC System

The system is characterized by different subsystems which interact with each other and also with other external data sources, such as weather, radar stations, external coordination units and satellites. The function performed by the system is the management of the life-cycle of flight plans: it has to undertake all the necessary actions to ensure flight safety during the cruise phase. All the nodes belonging to the various subsystems can communicate with each other via different VLANs in order to exchange application related messages. Each component is equipped with:

- a middleware module that deals with the configuration of the entire system. In fact every node has to be aligned with the main system's database and it is needed to verify if the component is correctly working. The middleware is not to be intended

as a separate component, because it is inherent to every single node, or in other words, it is intrinsically part of the software installed on it;

- a supervisor module that:
  - handles the role of each node. Indeed, every component is replicated and can assume one of two roles, namely master or slave, in order to increase the total availability of the overall system;
  - is in charge of properly initializing the software components of the node itself;
  - coordinates a service for the transmission and reception of messages through different VLANs using the UDP protocol. Each application can declare a certain set of ports which it desires to receive messages from, by following the typical publisher-subscribe pattern.

The ATC system can be divided in four fundamental macro-components:

- *Radar and Surveillance Data Processing*, namely the components that receive raw data from radars and process them according to a common standard, by creating a unique mixed data, the so-called system track. Basically these nodes compute the routes the planes are currently following: the system tracks represent the input to other macro-blocks, which handle the graphical view.
- *Operational display*, which consists in all the nodes responsible for displaying information and defining a graphical interface to the human operators. They basically represent the support for the human operator that can take decisions on the basis of the graphical visualization of the air traffic data, such as radar tracks, flight list, alarms and weather information.
- *Flight Plan Processing*, namely the nodes that deal with the management of the flight plan, characterized by a set of operations, such as insert, edit and delete. The macro-area also involves components which are responsible for the long or short term detection of emergency situations. The components can predict, detect and prevent hazards like, for example, situations in which aircraft tracks are not separated by a minimum threshold or flying below a certain altitude, or if they are infringing a protected air volume. They are also capable of detecting deviations of the current system track from the computed and expected one, namely the trajectory prediction.

- *Control and Recording*, namely the components that are in charge of recording all the information passing through the network which are related to the air traffic control procedures. In particular they have a double function: besides recording and realizing a back-up of the data, they are also used for playback. Indeed the recorded flight information can be replayed in order to reproduce old traffic scenarios or for fast view data reproduction.

3.3 The Proposed Architecture

The selection of the cloud platform, discussed in the previous chapter, was the preliminary step for the implementation of the testbed and the design of an architecture characterized by attack mitigation strategies. The architecture is provided with a mechanism that leverages the dynamic nature of the Software Defined Networking paradigm on one hand, and the use of a virtualization-based technique to protect virtual appliances, namely the VM Live Migration. This mechanism can be exploited when one of the virtual nodes is compromised and a malicious activity is detected, in order to mitigate the effect of the attack. Since the objective is to provide different levels of security, the proposed architecture has to be analysed considering three different layers, as shown in figure 3.3. Cloud layer shows two data centers that are geographically connected through a private backbone network. Each data center has its own IaaS layer and there is a node that is responsible for the management of the overall architecture. Moreover, in order to further increase network security in the connection between the distributed data-centers, a mechanism that splits packets into parts and then redirects them to disjoint paths is used, so that an intruder is not able to reconstruct the flowing traffic. This is done by using a traffic engineering mechanism based on the MultiProtocol Label Switching (MPLS) technology [45]. The confidentiality is guaranteed because if a malicious user is able to intercept the traffic between two nodes of the network passing through different paths, he should be aware of the splitting mechanism in order to reconstruct the original message from the parts he has collected. The mechanism is able to ensure the confidentiality, without the overhead due to the classical cryptography.
In the virtualization layer, the view is independent from a particular platform implemented in one of two data centers. Each physical machine is hosting a virtual switch to which all interfaces of the VMs are connected. All the traffic the VMs are exchanging

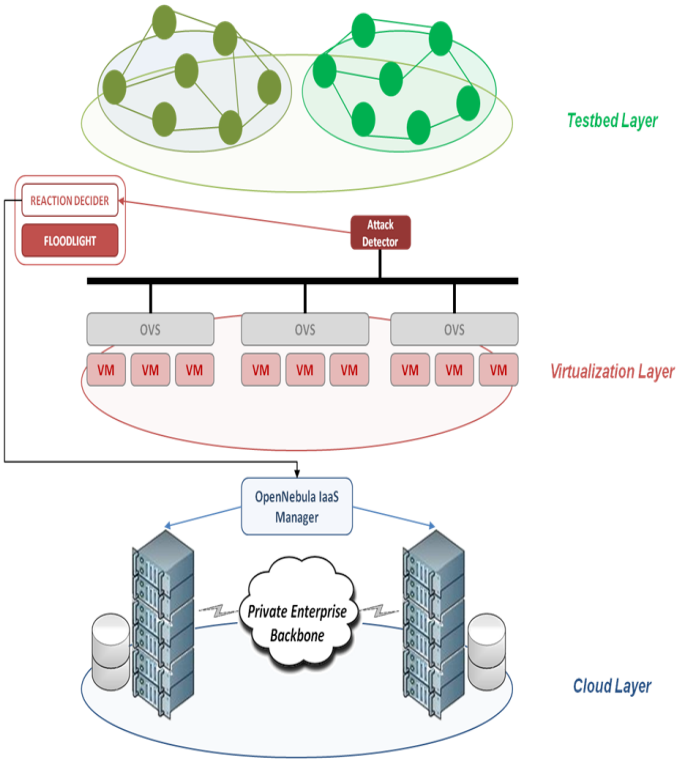


FIGURE 3.3: Architecture Layers

is checked and compared with some well known malicious traffic patterns, in order to identify possible attacks. A possible threat is detected by means of a "Network Intrusion Detection System": when a malicious pattern is recognized, an alarm is generated. After the system has identified the type of alarm, the best countermeasure is chosen on the basis of the level of the attack severity. A mitigation strategy can be triggered in the case of an attack against a node: the attacked machine is migrated in another cloud platform, hosted in a second data center, in order to make sure that the attacker is no longer able to access to the VM. Anyway after the successful migration of the node to the new location, there is the need to let legitimate VMs reach the migrated virtual appliance, by properly and dynamically reconfiguring the network flows.

3.3.1 Software Defined Networking

The reconfiguration mechanism described in the previous section is realized thanks to the Software Defined Networking (SDN) paradigm: it has rapidly changed the perspective of doing network research, by giving the chance to easily design and test new network protocols or forwarding algorithms. This paradigm is based on a sharp distinction between the infrastructure layer, composed by network devices, and the control layer where the

network intelligence is deployed, namely the *Controller*. Network devices are no longer close boxes, but they become programmable and their behaviour is logically centralized through a global view of the network. One of the most successful implementation of this paradigm is OpenFlow, which is basically the interface between the two layers. It allows to control and define traffic management strategies to be performed by the switches in the infrastructure. The data-path of an OpenFlow switch consists in a "flow table" which keeps track of the flows traversing it. A flow is identified through a match in the OpenFlow terminology, which is a set of network packet related fields. Each of them can be set to a specific value or just wild-carded, in order to create different flow granularities. Every row of the flow table is the association of a match with a particular action (or group of actions): all the possible actions are defined by the protocol specifications. They represent the way packets matching with a particular flow will be treated. An action can be also composite: as an example, a switch can insert a VLAN tag in the packet and then decide to forward it through a specific output port. An OpenFlow switch consists in one or more flow tables and a group-table. Within the flow table, flows are cached for fast transmission. More in details the structure of a row is like this:

- **match fields**, defining the key used to search for a flow to be associated with an incoming packet;
- **counters**, which represent the number of packets that are associated with a given flow. There are counters for each table, flow, and port number of the queue;
- **instructions**, which are used to modify the actions list.

Match Fields	Counters	Instructions

Table 3.1: A Flow Table Row

The OpenFlow switch processes the incoming packets through a pipeline consisting in one or more flow tables. In the presence of a pipeline, each table is numbered starting from zero: for incoming packets, the switch tries to find a compatible entry in the first table. If the search is successful, then:

- the counters are updated;

- the instructions associated with the flow are executed. An instruction could simply pass the packet to the second table in the pipeline. More complex instructions can add some actions to the set, or write meta-data in the packet that can be used as matching fields in the next tables of the pipeline. If there is no instruction to process the packet the next table, then the pipeline is blocked and the related action set is executed.

If there are overlapping flows and a packet matches more entries in a table, the row with the highest value of priority is chosen. When you have a table-miss, it means there is no compatible flow and the behaviour depends on the configuration of the flow table: the switch can send the packet to the controller or simply drop the packet.
OpenFlow is also attracting a great interest in cloud platforms as a leading technology for implementing *Networking as a Service*. The need to virtualize networks has become the main focus of the cloud community: being the hypervisor the heart of the server virtualization, it is needed to find networking solutions which allow to reach the same level of abstraction with physical network resources. Indeed, OpenFlow can be used to guarantee the programmability of the virtual networking layer. The application of this dynamic approach also enables the deployment of security policies and their enforcement at the edge of the network. New applications can be designed on the controller to apply security policies when a packet generated by a malicious VM matches a learned dangerous traffic pattern.

3.3.2 OpenFlow-based Security Architectures

The dynamic nature of cloud enabled systems makes the traditional security solutions inefficient, raising the need to find new approaches for protecting the network infrastructure from different kind of attacks. OpenFlow can be considered as a enabling technology for the dynamic application of security policies in the network at a low cost. Some recent works exploit the potential of OpenFlow-based platforms to implement security techniques. In [46], the authors argue that the SDN paradigm can make the implementation of traffic anomaly detection easier by using the well-known NOX [47] OpenFlow controller in SOHO (small office/home office) networks. They implement four different anomaly detection algorithms as applications on the controller and show that they are able to work at line rate without affecting the home-network related traffic. The results

point out the programmability of the SDN, in the sense that it allows to implement line-rate detection of network vulnerabilities exploitation and also risk mitigation: once the anomaly is detected, a fast reaction can be spread all over the network. Braga *et al.* [48] face a known security problem, which is the Distributed Denial of Service. Their proposal aims at minimizing the overhead due to the extraction of network features used in the detection process. They built an application on top of a NOX controller which consists in three different modules:

1. the *Flow Collector* which is in charge of periodically requesting flows to the OpenFlow switches through the secure channel;
2. the *Flow Extractor* which is able to obtain the features involved in the classification of normal/malicious traffic;
3. the *Classifier* that receives the features selected by the extractor module and uses a self-organizing map (an unsupervised classifier) to detect potential DDoS attacks.

The work shows the great flexibility of implementing a detector by using a standard interface (the OpenFlow protocol) to obtain information about flows in the network. Results underline that the detection and false alarm rates are close to the other approaches, but at the same time it is assured a low overhead rate. This is due to the fact that this approach does not have to collect every packet directed to the victim node to obtain the information needed by the classification process. Wang *et al.* [49] suggest a flexible security management architecture for large-scale production networks to overcome the drawbacks of the existing static solutions, by considering the peculiarities of data center networks. The main aims of such an architecture are:

- service insertion, which means the possibility of introducing new network services without efforts (such as protocol identification or a load balancer);
- scalability: to accomplish the need of avoiding a single point of failure, the goal is to assure that introducing more security elements, it is possible to maintain wire-speed performance and to increase reliability of the system.

The proposed architecture is composed by a central control component which has the view of the global state of the network and of the designed security policies; besides one

or more security elements which are responsible for detecting anomalies in the traffic patterns. The controller knows which traffic flows (according to the global policies) have to be redirected to a determined security element by simply installing rules in the flow tables of the edge switch. If the security element raises an alarm after the detection process, it can send a message to the controller, which instructs the edge switch to drop all the packets that match the malicious flow. This method represents a powerful way of deciding which security boxes need to be activated in order to implement different security mechanisms or even other services, such as load balancing. However, a distributed architecture requires a deep analysis to verify that the introduced latency (caused by the presence of the security elements) does not affect user experience. The last interesting work about OpenFlow-based security solutions can be found in [50]. A NetServ node can be considered as a programmable node which can allow to design and implement various network services and different types of devices, such as a router or a switch. It can obviously use OpenFlow as data plane: the switch can be connected to NetServ nodes which represent processing units. In the proposed experiment, the authors use a flow-based intrusion detection, instead of a deep packet inspection technique. It is a two steps process: the first one, namely flow exporting, consists in creating flows from observed traffic. The second one, namely flow collection, keeps memory of the flows for the subsequent monitoring phase.

3.4 The Components of the Architecture

In Figure 3.4, all the components and their interactions are graphically described. The cloud platform is provided with the capability of reconfiguring itself to mitigate the effects of a raised security alert.

The traffic produced by VMs is sniffed with the objective of finding malicious patterns: when there is a match with a specific rule, the intrusion detection agent sends an alert through a *Transport Layer Security* (TLS) based socket to a *Mitigation Server*. Upon the reception of an alarm, the control is entrusted to the *Alarm Handler*, which is in charge of parsing the information which are contained in the alert, in order classify the severity level of the attack. On the basis of the classification, the Alarm Handler can adopt a specific countermeasure with the aim of mitigating the effects of the attack. Therefore a mitigation and recovery strategy for the involved nodes is triggered. It

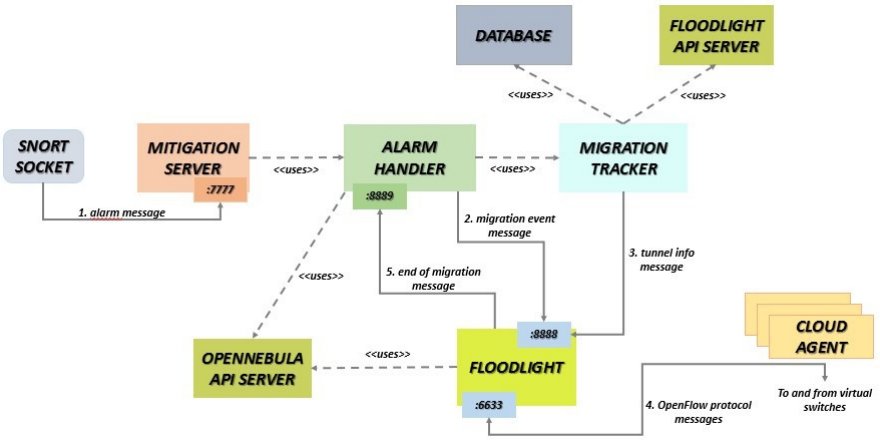


FIGURE 3.4: Architecture Components

consists in dynamically activating a hot migration of the virtual guest under attack in a remote data center belonging to the same cloud infrastructure, by interacting with the cloud platform manager. The information about the VM migration is handled by the *Migration Tracker*, which keeps track about the migration process and detects the endpoints of the tunnel to automatically create in the virtual switches of the cloud hosts. In fact, once the migration process is terminated, it is needed is to guarantee the location transparency of the virtual appliance, so that a legitimate machine can access the services hosted on the attacked node, without being aware of the migration process. The virtual switches are connected to an OpenFlow controller, named *Floodlight*, which is in charge of populating the switches' flow tables according to the activated modules. Floodlight is also responsible for the flow reconfiguration when a migration due to a security event occurs. All the components communicate through a secure channel, by realizing a mutual authentication. On one side, the server sends its own certificate which is determined from the key-store: this typically stores the identity information about a subject, namely the certificate along with its private key and the certificate chain. On the other side, the client verifies if this certificate can be trusted: if the server's certificate or the one of its Certificate Authority are found in the trust-store, then the server is authenticated. The trust-store holds the certificates of trusted Certifying Authorities. If the client authentication is also enabled at server side, the server requests for client's certificate: hence, he sends its own certificate which is found from its key-store. The server mutually verifies if the client's certificate can be trusted. If so then the client is authenticated and the mutual authentication took place.

3.4.1 OpenvSwitch

The virtual switches are based on OpenvSwitch, which is a multilayer, open-source, Apache 2.0 licensed technology designed to run in extremely distributed environments, as well as commercial products. Among the features it supports, there are:

- Link Aggregation Control Protocol, which allows to aggregate different physical ports in order to obtain one logical channel;
- Bidirectional Forwarding Detection, which is a network protocol providing low-overhead detection of faults even on physical media that do not support failure detection, such as Ethernet or virtual circuits;
- Spanning Tree Protocol, used to determine redundant routes by using L2 technologies;
- Network Interface Cards bonding configuration for source-MAC load balancing and active backup;
- OpenFlow protocol support and QoS traffic policing;
- tunnelling protocols, such as VXLAN, IPsec and GRE.

OpenvSwitch is fully supported by almost all cloud platforms. It consists in a kernel module and a series of utilities in the user space. The core components, as shown in figure 3.5, are:

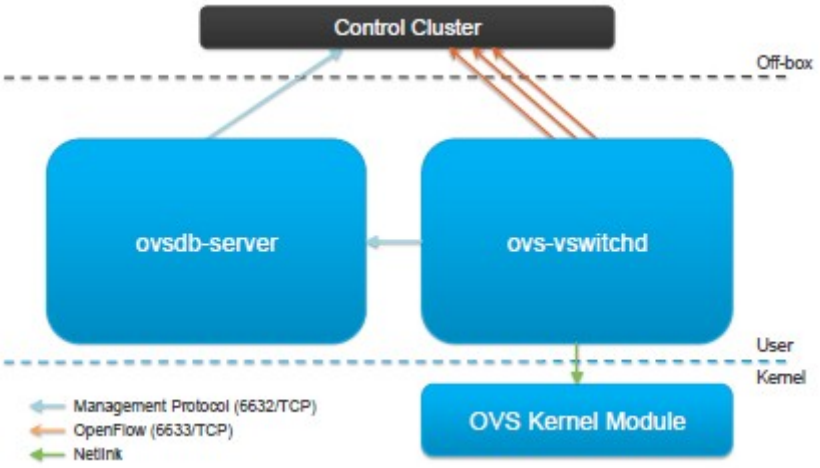


FIGURE 3.5: OpenvSwitch Core Components

- ovsdb-server, namely the database that holds switch-level configurations, such as bridges, interfaces, tunnel definitions and controller addresses. All the configuration is stored on disk and survives a reboot. This component speaks the *OpenvSwitch Database Management Protocol* (OVSDB), that is an OpenFlow configuration protocol, designed to manage OpenvSwitch implementations. There is also a very high-level interface for retrieving all the configuration data from the database.

- ovs-vswitched is the core component in the system that communicates to:

  - the outside world using OpenFlow;
  - the ovsdb-server using the OVSDB protocol;
  - the kernel module over netlink;
  - the system through netdevabstract interface.

  It supports multiple independent bridges and provides a packet classifier with efficient flow lookup. This core component is responsible for implementing mirroring, bonding, and VLANs through modifications of the same flow table exposed through OpenFlow. Besides it checks flow counters to handle flow expiration and stats.

The classical *linux bridge* uses a very simple forwarding mechanism which is based on matching MAC addresses of the hosts. All the operations needed to forward packets are done in the kernel space. OpenvSwitch uses a different forwarding schema that is derived from the SDN paradigm: the decision on how to treat packets is taken in the user space. Hence, the first packet of a flow is processed by ovs-vswitched in the user space, while all the subsequent packets match the cached flows in the kernel module for fast transmission. The kernel module is, indeed, the component that implements switching and tunnelling actions. If a packet is comparable to a flow, the virtual switch simply executes the related actions and updates statistics, otherwise the packet is sent to the user-space. For the virtual networking, in the cloud platform a virtual switch is configured and connected to a physical interface: this was properly set-up to forward VLAN-tagged traffic. All the virtual NICs are plugged into the virtual switch, in order to enable the communication of different VMs over distributed physical nodes. The structure of a virtual switch is shown in Figure 3.6.

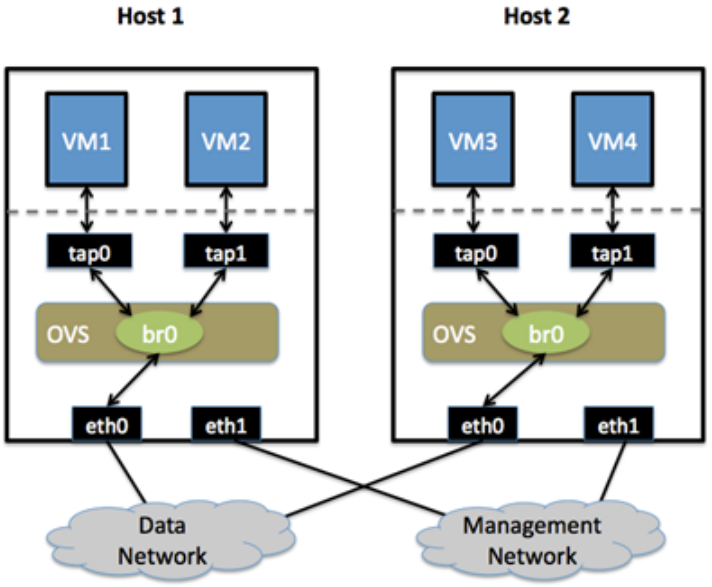


FIGURE 3.6: OpenvSwitch Bridge

Among the different alternative approaches for the networking mechanism used by the cloud platforms, OpenvSwitch shows a great flexibility and several interesting features. For instance, it enables the implementation of bonding, in order to guarantee redundancy over network links. OVS is also equipped with utilities for control and monitoring of the exchanged data, as well as QoS rules to meet specific requirements. Since OVS also speaks the OpenFlow protocol, the forwarding decisions can be externally taken by a controller.

3.4.2 The OpenFlow Controller: Floodlight

OpenFlow switches are basically dummy network devices which are programmed by an external controller in charge of implementing the policies which packets are forwarded through. The selection of an open-source solution among several available OpenFlow controllers was based on a performance comparison, which was accomplished through OFlops [51], namely OpenFlow Operations Per Second. This latter is composed by two software packages:

- OFlops, a particular controller that allows to benchmark multiple features of the switches;

- Cbench (Controller benchmarker), that generates packet-in events for the controller by emulating switches' connections. It is able to calculate the maximum packet-in message generation rate, the delay between packet arrival and packet-in event and the processing delay.

Flow-mod/s	Nox Hub	Nox Switch	Nox Learning switch	Trema	Floodlight
Min	7427	19225	7127	52207	56368
Max	7440	20109	7147	56204	57535
Avg	7435	19916	7137	54333	57142
Stdev	3,55	299,31	6,84	1285	407,81

FIGURE 3.7: OpenFlow Controllers Evaluation

In Table 3.7, the results in terms of Flow-mod messages per second are shown: this message is used by the controller to install, modify or delete a flow in the switch flow table. In the comparison, only open-source controllers were taken into account, in addition to other features, such as the customization of the controller modules, the availability of RESTful APIs and the development support. From the evaluation process, Floodlight, a Java event-based Controller, released under the Apache license, was selected. It is developed by an open community and it is also characterized by the availability of a huge collection of applications built on top of it. The controller is endowed with very powerful APIs, which represent the interface for developing applications by leveraging Floodlight supported features. The API server is available at port 8080 of the controller. Any application, written in any language, is able to retrieve information or just to take advantages of the services, by sending HTTP REST commands to the controller. There is also a graphical dashboard which allows to have a global view of the network, switches state and flow tables. Floodlight has an extremely modular structure: the different applications can be activated by loading the appropriate module in the properties file.

3.4.2.1 Learning Switch

One of the controller's basic applications is the one which defines the behaviour of the switch data path: when a packet arrives to an OpenFlow switch and it does not match

any of the flows in the table, a *OFPacketIn* message is generated and sent to Floodlight. The *Learning Switch* application is in charge of handling this message, by registering itself as a listener for this type of OF messages. It basically reproduces the way switches learn the correspondence between ports and MAC addresses. The OpenFlow packet-in message received by the controller contains the packet that the switch was not able to handle and the reason why the message was generated by the switch. In this particular case, the reason is a table miss for the packet. Hence, given the the packet source and destination MAC addresses, if the source one is unicast, the controller holds the correspondence between the source MAC address and the port which the message arrived from, in relation to the switch that sent the packet-in message. In this way, the controller knows which output port should be used to send packets destined to that specific MAC address. Besides, if for the same switch Floodlight has learned a mapping between the destination MAC and an output port, the information to instruct that switch on how to treat the packet is available. In order to do this, a flow table entry which matches on the source and destination MAC, the data-link type (the protocol type in the next header) and the IP source and destination addresses is created. This is realized by creating an *OFMatch* object: this consists in the set of fields which the mapping between a packet and a flow is based on. OpenFlow uses a 32-bit wild-cards field and Floodlight has a convenience API that allows to easily construct a wild-card bit-mask. By starting with all wild-carded bits, it is possible to set the appropriate meaningful bits, by specifying the desired fields and specific values. As an example, a match based on IP source and destination masks is created like this:

- match.setNetworkSource(IPv4.toIPv4Address("192.168.12.0"))
- match.setNetworkDestination(IPv4.toIPv4Address("10.0.10.0"))
- match.setWildcards(Wildcards.FULL.withNwSrcMask(24).withNwDstMask(8))

The first step is needed to specify the IP net-mask and, then, it is possible to set the appropriate wild-card bits on the match, so that only parts of the IP addresses of the incoming packets will be matched.
The controller uses a *Flow Mod* message, to create a new flow in the switch flow table. The next time a packet matches the previously created rule, the switch will be able to directly execute the actions that are associated to that flow. This protocol message can

assume different semantics: it can be used to edit the switch flow table, by using one of these options:

- *ADD*, for inserting a new flow in the table with the specified match and action set;
- *MODIFY*, for changing the action set associated to a particular match;
- *DELETE*, for erasing a flow by indicating the match.

In order to install flows, the controller sends a message to the switch by using the first option. Besides, a reverse flow for the incoming packets is also built and another *Flow Mod* message is sent to the switch. On the other side, if for that switch a mapping between the destination MAC address and an output port has not been learned yet, the controller will force the switch to flood the packet.
The learning switch application was modified in order to enable VLAN based communication among the VMs controlled by OpenNebula. Therefore the controller has to check if the packet source and destination MACs are known by the OpenNebula manager. If so, Floodlight needs to understand it the attachment points, which the MAC addresses are associated to, belong to the same OpenFlow switch. If this condition holds, it means that the packet is exchanged between two VMs that are controlled by the same OF switch and the action set associated to the match will contain only one action, that is:

- *OFActionOutput*, which forces the switch to send the packet through the output port which was retrieved from the learning table associated to that switch.

Otherwise, the communication is between two VMs which are on different nodes, and since the virtual networking is based on the VLAN technology, it is needed to get the VLAN identifier. This can be done by using the OpenNebula APIs: it is possible to get the virtual network which the MAC addresses belong to and to query for the VLAN tag. When the controller knows all the information, an action set is created: it contains two actions, that are:

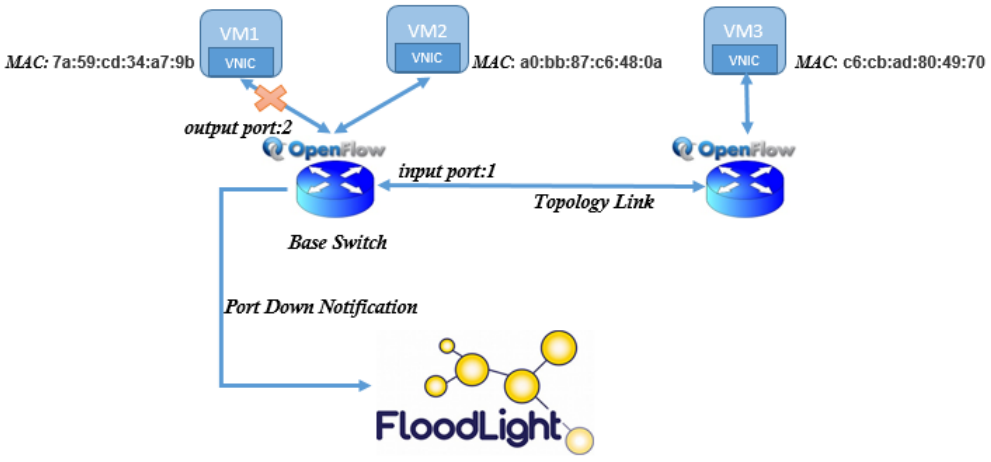
- an *OFActionVirtualLanIdentifier*, which is used to insert a VLAN tag in the packet;
- *OFActionOutput* for forwarding the packets towards the obtained port.

For the reverse flows all the match fields are simply in reverse order and, in case of VLAN based communication, an *OFActionStripVirtualLanIdentifier* action is used to strip the VLAN tag from the packet, before sending it to the destination port.

3.4.2.2 Flow Reconciliation

Flows need to be reconfigured when certain events occur, such as a link or port that goes down, by triggering a notification to the controller. Also in the event of VM migrations, the flows adjustment is handled by a specific module which needs to be enabled on the controller, namely the *Flow Reconciliation*. This is an application which is activated when a notification with a *PORT_DOWN* event is sent by a switch: it means that a port was deleted from that switch because, for example, a VM is migrated and its attachment points are deleted from the source switch and added to the destination one. This application is used to delete invalid flows across a network when a port or a link goes down. It was extended in order to respond to a migration which is issued by the Alarm Handler due to a security alarm. The controller first verifies if it has received a message from the Alarm Handler: if there was no such event, the application simply deletes the invalid flows in the switch that represents the source of the port down notification and also in all the neighbouring switches, potentially caching flows towards the problematic port. The controller knows all the links among the switches which were established in the topology through a specific application, namely the *Link Discovery Service*. It uses the Logical Link Discovery Protocol (LLDP) to detect links among the switches: a LLDP packet has a fixed source MAC and the broadcast address as destination one. The controller can detect two different types of links, namely "direct" or "broadcast": a direct link is established if a LLDP packet is forwarded through one port and the same packet is received on another port: this means the two ports are directly connected. A broadcast link is detected if a broadcast packet is sent out through a port and received on another. The creation of this link means that there is another L2 switch between the ports which is not controlled by Floodlight. The invalid flows, pointing to the downed port are queried by using an OpenFlow protocol message called *OFStatisticsRequest* and setting as request type *FLOW*. The request that will be sent to the OF switch is *OFFlowStatisticsRequest*: this object has to be populated with a match that, in this case, will have all wild-carded fields, except the output port. After the controller has obtained the problematic flows, it creates a map data structure that stores the ingress port of

the invalid flows and the list of matches with actions that forward packets towards the downed port, as shown in Figure 3.8. The invalid flows are deleted from the base switch that generated the notification: this is accomplished by using a *Flow Mod* OpenFlow message, in which the *DELETE* command, a wild-carded match and the deleted port as output port are specified.



Ingress Port	Matches
1	[match1: input port: 1, DL Destination: "7a:59:cd:34:a7:9b", DL Source: "c6:cb:ad:80:49:70", ether type:0x800]
	[match2: input port: 1, DL Destination: "7a:59:cd:34:a7:9b", DL Source: "a0:bb:87:c6:48:0a" , ether type:0x800]

FIGURE 3.8: Port Down Interactions

All the topology links are scanned in order to obtain the potential switches storing flows towards the invalidated port. By analysing a link, if its destination port is equal to the previously stored port in the map structure, a neighbour switch has been found, or in other words a connection between the base switch and another one was detected. Therefore another map structure is created for keeping track of the link source port (the output port for the neighbouring switch) and the matches that were previously determined. The controller now holds a list of neighbour switches that need to be controlled for invalid flows to be reconfigured. For each of them the flows towards the output port (the key of the map) are requested and compared with the stored matches. If there is a correspondence, the matches are saved together with the input ports. For each of the matches, the controller knows the current switch, whose flow table needs to be modified because there are flows directed to the deleted port, but the information about the new location of the VM is also needed. This data can be retrieved by another Floodlight component, which listens from port change events. When an *ADD* event occurs, this means that a new port was added in a switch that generated the notification.

When the controller listener receives a security message and there is such a change in a switch, the data-path identifier of the device is registered and the port is registered in a list. The controller now can obtain the information about the new switch that handles the ports of the migrated virtual machine. Since the VM has migrated in another data-center which is connected through a L3 network to the original one the flows are modified in order to be redirected to VXLAN tunnels. The components which is responsible for keeping track of the VM-ports to tunnel endpoints is the Migration Tracker. At this point the flows towards the migrated VM can be modified on the neighbour switch, by using the OpenFlow message type *Flow Mod*.

3.4.2.3 VxLAN

In order to ensure the continuity of data-link communication through a wide area network, the reachability of the VM is handled by means of a tunnelling technology, namely *Virtual eXtensible Local Area Network* (VXLAN). It is an encapsulation technology to forward L2 traffic over a L3 network, that creates an overlay network which is called VXLAN segment. In this overlay network, VMs can communicate even if they are connected through a geographical network as if they were in a LAN. Only VMs belonging to the same overlay segment can interact and each segment is identified through a 24-bit identifier, namely the *VXLAN Network Identifier*. Hence, there is the possibility to create up to 16 million VXLAN segments in the same domain. The encapsulation tunnel is created between two *Virtual Tunnel Endpoints* (VTEPs): they are responsible for creating the additional header containing the segment identifier and realizing and MAC-in-UDP encapsulation. The endpoints of the tunnel are collocated within the hypervisor on the server that hosts the VMs. Anyway, a VTEP can also be on a physical switch or server and it could be implemented in software or hardware. When considering VMs unicast communication, a VM, that wants to communicate over L2 with another one hosted by a different server, sends a MAC frame to the destination. If the two VMs are connected through a VXLAN segment, the tunnel endpoint obtains the identifier of the overlay network the VMs belong to and it verifies if the destination MAC is associated to the same segment. If so, the switch needs to know which is the endpoint controlling the destination MAC. The original packet is then encapsulated: an outer header comprising the endpoint's MAC and IP, along with the VXLAN header are added, as in Figure 3.9. The frame is then forwarded by the source endpoint, and

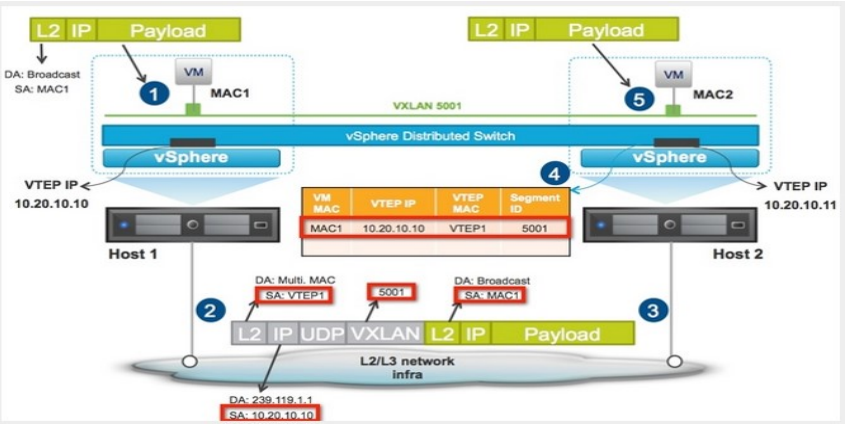


FIGURE 3.9: VXLAN Encapsulation

on the other side, the tunnel destination checks for the validity of the VXLAN segment identifier and if there is a VM matching the packet's inner destination MAC address. The frame is, then, de-encapsulated and forwarded to the VM which is completely unaware about the encapsulation. Besides the destination endpoint is also able to learn the mapping between the inner source MAC, which belongs to the source VM, and the outer IP address, which belongs to the endpoint controlling that VM. OpenvSwitch supports UDP encapsulation, even if there is no implemented control plane: this one is in charge of defining the correspondence MAC-to-VTEP, or in other words, between VMs and the endpoints that manage them. VXLAN solves this problem via IP Multicast: a source VM that wants to communicate with another one sends an ARP request in broadcast to the IP multicast address associated with the VNI. In this way all the other endpoints will learn the correspondence between the source VM MAC address and the one of the endpoint. In the proposed architecture, OpenvSwitch is used to dynamically create tunnels VXLAN and the OpenFlow Controller to handle the VM-to-VTEP association.

3.4.3 Network Emulator: CORE

The geographical connection between the two data centers is emulated through *Common Open Emulator Research* (CORE), an efficient and scalable network virtualization environment. Core allows to emulate a virtual network, which can also be connected to physical network devices. Core is able to emulate a wide area network with specific characteristics, such as bandwidth, delay and packet loss rate. There are two basic components: a python daemon which is responsible for emulating virtual network nodes by using virtualization and processing packet manipulation and a graphical interface. The

emulator has a very modular architecture, since the nodes can also be created from other external network simulators and emulators. A network node is a very lightweight VM, which can be created by using either Linux network name-space virtualization (namely Linux containers) and Linux Ethernet bridging, or FreeBSD jails. The emulation scenario can be created through the graphical interface or by means of Python scripting. The interface enables to draw nodes and network devices which will be emulated once the scenario is executed; on the other hand, a script can be written to configure the scenario by importing CORE Python modules. After executing the emulation, the user can interact with the nodes and check for statistics. Among the components that can be created in an emulated network there are:

- Links.
- Network-layer virtual nodes, such as router which runs Quagga [52] to forward packets; host which stands for an emulated server machine and PRouter that represents a physical router.
- Links-layer nodes, such as hub that forwards incoming packets to every connected node; switch that forwards incoming packets to attached hosts using a MAC based hash table; RJ45, thanks to which emulated nodes can be linked to real physical machine interfaces in order for real networks to be connected to the emulation scenario; finally tunnel, that allows to connect more than one emulation scenario by using the GRE protocol.

3.4.4 A Network Intrusion Detection System

Snort [53] is a packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS), which is able to perform well-known patterns matching and to detect different kinds of attacks, such as buffer overflows, port scans, common gateway interface attacks and so on. Snort is able to decode the application related data of a packet, in order to recognize hostile activities that can be characterized by unique detection fingerprints. Snort consists in three main subcomponents, namely the packet decoder, the detection engine and the logging and alerting subsystem. The decode engine is organized in different subroutines which are related to all the protocols of the stack, starting from the data link layer and ending with the application one. The

detection engine is in charge of recognizing traffic patterns that fit with the pre-loaded rules. Snort is extremely extensible, in the sense that it allows a full customization of the rules. A rule follows a very simple syntax; as as example, a rule can be like this:

alert tcp any any → HOME_NET 80 (flags: S; msg: "TCP SYN Flood"; flow: stateless; threshold: type both, track by_src, count 1000, seconds 10; sid:10001; rev:1;)

The first field represents the action to undertake when traffic matches that rule: the available actions are pass, log, or alert. If a pass action is specified, the packet is simply dropped. The administrator has the ability of to enabling different logging and alerting modes. Logging is useful, for instance, to store information about the packets that generated the alarm in a human-readable format or in a binary one, with the aim of helping the data analysis. Alerts are a notification of the event that can be sent to the syslog, unix-sockets or just written in a textual file. The other fields are: protocol, the source network and port, the operator that specifies the direction of the traffic and the destination network and port. In the example rule an alert is generated when a packet coming from any source address and port is directed to a host in the home network specified in the configuration file on the port 80, with the flag SYN active. The message is just a textual string that will be written in the alert. The threshold filter represents the motivation why the alarm is raised: the type both alerts once per time interval after c occurrences of the event and then ignores any additional events during the interval. The occurrence rate can be tracked either by source or destination IP address. Count represents the number of matching rule events over a time interval of s seconds that will cause the alert generation. The sid keyword is an unique identifier for the rule that can be exploited by the plugin to simply recognize it; instead rev is used for the identification of the rule revision.

3.4.5 The Mitigation Server and the Alarm Handler

The Mitigation Server listens on a secure channel to receive alarm messages generated by the Network Intrusion Detection System. Snort is configured to generate alerts which are sent to a unix socket. This latter is programmed to send a specific message to the Mitigation Server, that contains:

- a textual information about the alarm, that allows to select a countermeasure;

- the transport protocol, the source and destination IP addresses and the source and destination ports of the packets that generated the alarm;
- an attack classification field;
- a time-stamp, which refers to the time the alert was generated and can be used to distinguish among different attempts to launch the same kind of attack.

The message is sent over a secure TLS channel to the server: upon the reception of this message, the Mitigation Server gives the control to the Alarm Handler. This component is in charge of:

- parsing the message to extract all the information needed to identify the VM under attack. This is done by using the exposed cloud APIs: in particular, given the IP address of the virtual appliance, it is possible to retrieve low-level information from the IaaS platform, such as:
  - the VM name and identifier;
  - the VM network interfaces and MACs;
  - the physical node hosting the VM and the cluster id where the server is located.

  The OpenNebula APIs expose a huge number of XML-RPC based methods: in order to be able to use them, as first step it is needed to set up the authorization string and the endpoint, which represents where the OpenNebula API server is deployed. At this point, it is possible to create an object representation, that can be a pooled resource (e.g. HostPool) or a single element (e.g. VirtualNetwork). OpenNebula APIs give the chance to use an object representation for all the physical and virtual resources, such as servers, VMs, networks and so on, and to perform different actions on them. The method *info* is used to access the information contained within the queried objects, by exploiting their attributes;
- determining the new cluster, that is the new data-center where the VM will be migrated. The Alarm Handler then uses an algorithm to select the physical server, characterized by the highest combination of used resources (considering CPU and RAM), which can host the migrating VM;

- sends a non-blocking message to Floodlight to report a security event that will be managed by enabling the migration of the attacked virtual resource;
- interacting with the Migration Tracker in order to inform this component about the migrating VM and the data which was retrieved from the cloud platform;
- setting the migration status as "in process";
- triggering the live migration of the attacked machine by using the appropriate API of the cloud platform;
- blocking the source of the attack by setting a rule to discard all packets coming from the IP of the attacker using the APIs of Floodlight;
- logs the event;
- blocking itself to to receive a massage determining the end of the migration by Floodlight: this is used to indicate a successful migration and the reconciliation of the network flows.

3.4.6 The VM Migration Tracker

This component is in charge of requesting the creation of VXLAN based tunnels between the switches having flows towards the migrated machine and the switch on the new hosting server. Every OpenFlow switch is identified through a *datapath identifier* that can be queried through the APIs exposed by Floodlight: after obtained all the switches' identifiers, for each of them it is possible to request all the cached flows with a match containing the MAC(s) of the virtual appliance that is about to be migrated. It is possible to build an OpenFlow message which corresponds to statistics request: if the switches do contain flows, the tracker knows the switches holding flows towards the migrating VM. For these switches the Migration Tracker needs to verify if a VTEP, or in other words, a tunnel endpoint exists: for convenience it coincides with the base switch where the port notification was sent. Hence the component needs to verify if a VXLAN tunnel among the endpoints that connect the servers in the distributed data centers was already set-up: this information is stored in a database table, where for each of the created tunnels there are the endpoints addresses, the tunnel identifier and the key. If there is no installed tunnel, a new one has to be created: the components sends messages

to the corresponding endpoints over the secure channel, with the objective of ordering the creation in the virtual switch of a VXLAN port with the obtained information. Otherwise a tunnel has already been set up and the related data is retrieved from the database: the tracker just interacts with Floodlight for informing it about the endpoints and the tunnel identifier.

3.5 A Proof of Concept of the Architecture

3.5.1 The SecRAM Methodology

The virtualized testbed allowed to realize a vulnerability assessment that was useful to:

- apply a methodology for the identification of the risks on the operational assets of the ATM system;
- build a real threat scenario with the identified vulnerabilities;
- test the mitigation strategy of the cloud based architecture when the threat is identified.

Regarding the first two points, due to the lack of shared risk assessment methodologies in the ATM community, the Single European Sky ATM Research (SESAR) [54] defined a new methodology, namely the *ATM Security Risk Assessment Methodology* (SecRAM) [55]. It represents the foundation for the application of cost-effective, proportionate and reliable security measures within each part of an ATM system. The methodology consists in three stages as shown in Figure 3.10, namely the risk identification, the risk evaluation and finally the risk treatment.

- The risk identification is the process of finding, listing and characterizing elements of risk. It consists in identifying and evaluating the assets to be protected (namely *Primary* and *Supporting Assets*) and in building the threat scenario defined as a combination of a threat over a supporting asset within the considered environment.
- The risk evaluation is the process of assigning values to the likelihood and impacts of a risk.

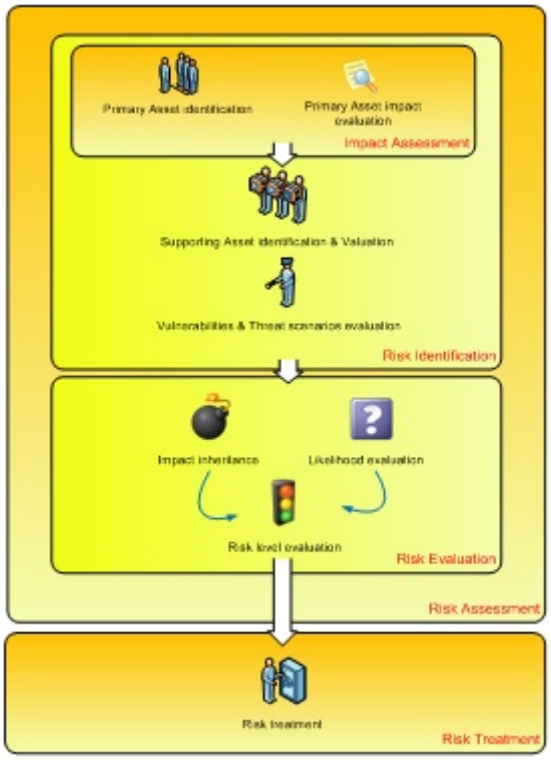


FIGURE 3.10: SecRAM Methodology

- The risk treatment is the process of selecting and implementing measures to modify risk.

Primary assets are the intangible functions, processes, activities, information and services which need to be protected; supporting assets are entities which contain or encapsulate the primary assets. As an example, if a primary asset consists in sensitive information then the supporting asset could be the hard disk on which the information resides. Supporting assets have vulnerabilities that are exploitable by threats aiming at impairing primary assets. After having identified them, all the primary assets can be linked with at least one supporting asset and vice-versa.

For each primary asset, it is needed to identify the required level of confidentiality (C), integrity (I) and availability (A). This evaluation is represented through a number ranging from 1 to 5: the impact is evaluated in terms of loss of each of the security requirements, for each of the primary assets on every impact area (IA). All the impact areas characterizing an ATC system are described in Table 3.11.

As said before, supporting assets are the system components which have vulnerabilities that are exploitable, with the aim of impairing the primary assets. The risk evaluation

	5	4	3	2	1
IMPACT AREAS	**Catastrophic**	**Critical**	**Severe**	**Minor**	**No impact / NA**
IA1:PERSONNEL	Fatalities	Multiple Severe injuries	Severe injuries	Minor injuries	No injuries
IA2:CAPACITY	Loss of 60%-100% capacity	Loss of 60%-30% capacity	Loss of 30%-10% capacity	Loss of up to 10% capacity	No capacity loss
IA3:PERFORMANCE	Major quality abuse that makes multiple major systems inoperable	Major quality abuse that makes major system inoperable	Severe quality abuse that makes systems partially inoperable	Minor system quality abuse	No quality abuse
IA4:ECONOMIC	Bankruptcy or loss of all income	Serious loss of income	Large loss of income	Minor loss of income	No effect
IA5:BRANDING	Government & international attention	National attention	Complaints and local attention	Minor complaints	No impact
IA6:REGULATORY	Multiple major regulatory infractions	Major regulatory infraction	Multiple minor regulatory infractions	Minor regulatory infraction	No impact
IA7:ENVIRONMENT	Widespread or catastrophic impact on environment	Severe pollution with long term impact on environment	Severe pollution with noticeable impact on environment	Short Term impact on environment	Insignificant

FIGURE 3.11: Impact Areas

is based on the computation of the impact and the likelihood of a threat scenario. This one is built by identifying:

- for each supporting asset the relevant threats;
- for each threat the targeted criteria (confidentiality, integrity and availability).

A threat scenario has a specific likelihood of occurrence and will have a specific impact (depending on the target criteria).
Finally, the risk treatment consists in one of these decisions for the threat scenario:

- Accept (or tolerate) the risk, which means that no further action is needed. The risk level is considered low enough to be accepted.
- Reduce (or treat) the risk to a new level through the selection of controls so that the residual risk can be reassessed as being acceptable.
- Avoid (or terminate) the risk, which means that if the risk is considered too high and the counter-measures to reduce it too onerous, then the project can decide to withdraw the activity or change its nature so that the risk is not present anymore.

- Transfer the risk, which means that the project decides that the risk should be transferred to another party that can effectively manage it. Once the risk treatment plan has been defined, residual risks need to be determined. This involves an update or re-iteration of the risk assessment, taking into account the expected effects of the proposed risk treatment. After this important activity (risk treatment), it is needed to consider the acceptance of the risk. In fact, the risk treatment plan is fundamental to assess the risk in order to meet the acceptance criteria.

3.5.1.1 Application of the Methodology

The methodology was applied to one of the test-bed components, namely the one responsible of receiving radar tracks from multiple sources and mixing them thanks to a correlation service. The information that comes from this correlation process is given as output to the presentation subsystem.

Risk Identification

The first step of the risk identification is to list and evaluate the primary and supporting assets. By taking into account the nature of the component, the application of the methodology was carried out with the most critical supporting and primary assets, which are respectively:

- PA1: surveillance data;
- PA2: correlation service;
- SA: correlation manager.

The choice fell on the component itself as a secondary asset, since it has a fundamental functionality in the system; instead, the primary assets are the surveillance data and the correlation service. Their security pitfalls could bring to:

- excessive workload for the Air Traffic Control Officer;
- a wrong correlation process and subsequently to an erroneous procedure of the ATC controller, with catastrophic impacts.

For every primary asset, the impact level in terms of loss of confidentiality, integrity and availability on each of the security impact areas (shown in are shown in Table 3.11) was assessed. The results of the impact assessment are shown in Table 3.12.

		IA 1	IA 2	IA 3	IA 4	IA 5	IA 6	IA 7	Overall impact	Effect
Surveillance data PA1	**Loss of C**	1	1	1	1	1	1	1	1	N/A
	Loss of I	4	5	3	1	1	2	1	5	Tracking computing could be wrong due to erroneous information about flight position
	Loss of A	4	5	4	1	1	3	1	5	Tracking computing is not possible because of no information about flight position
Correlation Service PA2	**Loss of C**	1	1	1	1	1	1	1	1	N/A
	Loss of I	5	5	5	4	1	1	1	5	Correlated data could be corrupted
	Loss of A	5	5	5	4	1	1	1	5	Impact on area control center and/or airport systems system responsible for receiving and displaying radar data; direct impact on ATC operations

FIGURE 3.12: Primary Asset Evaluation

For every impact area a value ranging from 1 (no impact) to 5 (catastrophic impact) is used to specify the impact severity: the overall impact is defined as the maximum impact level among all the areas. Next to the overall impact value, an effect for each of the security principles is also given. The next step is the identification of vulnerabilities, namely the security breaches, that can be exploited with an interest related to different impact areas. One or more vulnerabilities can be exploited by a threat scenario, that is defined as a combination of an attacker and his resources, motivation and objectives. As explained before, only supporting assets have vulnerabilities exploitable by threats. In this phase for each previously identified supporting asset, it is needed to recognize relevant threats, and for each one the targeted criteria (C, I, A). The threat scenarios were obtained from a preliminary vulnerability assessment, whose results are collected in the next section. The considered threats are:

- Malware;
- Eavesdropping;
- Unauthorized access;
- Corruption of data;
- Deleting data;
- Human error;

- Denial of service;
- Fraudulent copying of software.

The overall link between primary and supporting assets, the relevant threats for the supporting asset and their impact on each criteria is shown in 3.13.

Supporting Asset	Threats	PA1			PA2		
		C	I	A	C	I	A
		1	5	5	1	5	5
Correlation Manager	Malware	x	x		x	x	x
	Eavesdropping	x			x		
	Unauthorized access	x	x		x	x	x
	Corruption of data		x			x	x
	Deleting data			x		x	x
	Human Error	x	x			x	
	Fraudulent copying of software		x		x		
	Denial of service			x			x

FIGURE 3.13: Threat Scenarios Evaluation

The threat scenario evaluation represents the last step for the risk identification: the impact on the CIA criteria that is caused by the threat exploitation on the supporting asset is evaluated. The risk evaluation consists in obtaining the impact and the likelihood of occurrence for each of the threat scenarios taken into account. The threat scenario impact is obtained from Table 3.13, by selecting the maximum impact of the target criteria. This value is the so-called "inherited impact". The "likelihood" is, instead the evaluation of the chance of a threat scenario occurring: it is computed by considering the existing security countermeasures and the security controls. It can be a value ranging from 1 to 5: its meaning is explained in 3.14(a).

Likelihood	Qualitative and quantitative interpretation
5. Frequent	high chance that the scenario occurs in a short term.
4. Probable	high chance that the scenario occurs in a medium term.
3.Occasional	high chance that the scenario occurs during the life time of the project.
2. Remote	a low chance that the scenario occurs during the life time of the project.
1.Improbable	very little/no chance that the scenario occurs during the life time of the project.

(A) Likelihood Significance

Threats	Time	Skills	Target knowl.	Target avail.	Likelihood
Malware	x	x	x		2
Eavesdropping		x			4
Unauthorized access			x		4
Data corruption	x	x	x		2
Deleting of data			x		4
Human error			x	x	3
Fraudulent copying of sw	x		x	x	2
Denial of service			x		4

(B) Likelihood Values

FIGURE 3.14: Likelihood Computation

Supporting assets	Threats	Inherited Impact	Likelihood
Correlation Manager	Malware	5	2
	Eavesdropping	1	4
	Unauthorized access	5	4
	Data corruption	5	2
	Deleting of data	5	4
	Human error	5	3
	Fraudulent copying of sw	5	2
	Denial of service	5	4

FIGURE 3.15: Supporting Asset Evaluation

In order to accomplish this evaluation, some features were taken into account, such as the time and skills required to prepare the attack, the knowledge of the attack target and, finally, the time window in which the target needs to be available as explained in Table 3.14(b). The risk evaluation process is shown in 3.15.

3.5.1.2 Vulnerability Analysis and Threat Scenario

Since the SecRAM methodology does provide vulnerability analysis, there was a preliminary aimed at identifying the vulnerabilities affecting the entire system and, particularly, the considered supporting. The recognised vulnerabilities were useful to build a threat scenario, representing the trigger for the mitigation strategy. Hence, the first step was the vulnerability analysis through a penetration testing tool with the objective of finding eventual security breaches. After launching the vulnerability scanning process, all the found vulnerabilities were collected. The inherent possible threats were built the percentage of their occurrences in the overall system was computed. Figure 3.16 shows the results. One or more of these vulnerabilities can be exploited to create a threat scenario. For example if a remote login service is in execution on the machine that consists in the supporting asset, an attacker could use a brute force attack to guess the credentials used to access it, by accomplishing an unauthorized access (a threat impacting integrity and confidentiality). This could lead to delete data, with impact on service availability, or to fraudulent copy of software, with impact on integrity and confidentiality. In particular we exploit two main vulnerabilities to build a threat scenario which is described in the next paragraph, namely:

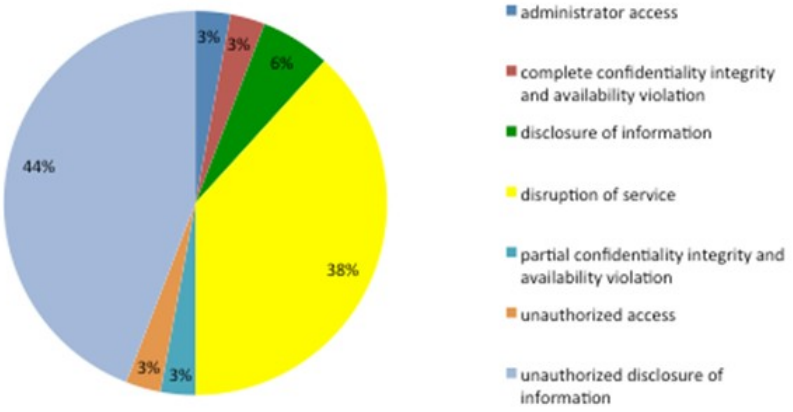


FIGURE 3.16: Vulnerabilities Distribution

- unsecured access control;
- implementation flaw.

3.6 Experimental Evaluation

3.6.1 A Threat Scenario

An attack could be represented as a graph, in which all the steps needed by an intruder to achieve a malicious goal on a system are illustrated. In [56], an algorithm for automatically generating attack graphs is presented: this can be useful to the system administrator to decide which set of countermeasures could be undertaken to protect it. The attack scenario that was built can be described as a multi-stage graph: the starting point for an attacker is the local access to the network the targeted component is connected to. The attacker could gain information about the machine from the network, by simply using a network scanning tool. Collected data can regard the nodes' host-names, their operating systems, open ports, known running applications and so on. The scan process could be used, for example, to find out that the target node has a secure shell application listening on port 22. The attacker now knows all the information needed to perform a classical attack, such as a password guessing, with the objective of gaining unprivileged access to the machine. If the attacker is able to obtain unprivileged access, this could be exploited to collect application-oriented information. For instance, given the identified component, the attacker could be interested in impairing the supporting asset functionality. In order to do this, the attacker is willing to learn the ports the

application expects to receive raw radar data from to launch a Denial of Service attack. The node is flooded with a huge amount of data with the aim of saturating the machine's resources and making it unavailable.

This attack is exploited by generating UDP-based traffic towards the discovered ports both without a semantic meaning for the application and legitimate traffic. The machine under attack was also monitored in terms of resources consumed by every single process. By stressing the application with legitimate radar tracks which were re-played at different rates, it is possible to verify that the application is provided with a pool overflow recovery procedure. In the case of attacks with meaningless data, the service representing the interface between the component and the local networks generates an exception. By monitoring the resources held by the application and the network traffic with an intrusion detection system, an alarm can be issued as combination of the critical CPU and RAM utilization and an alert generated by the detector. When the alarm is risen an opportunistic mitigation strategy can be activated.

3.6.2 Testbed Set-Up

The experimental test-bed was created by exploiting one OpenNebula Controller and four different compute nodes: two of of these servers belong to a data center and the remaining ones are in a second data center.

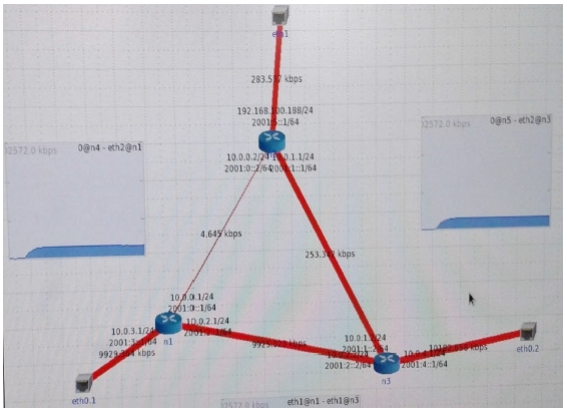


FIGURE 3.17: Emulator Topology

The connection between the two data centers is emulated through a simple scenario which is shown in Figure 3.17. For the sake of simplicity, the network topology is built up with three routers. The same number of RJ45 components is used: they are employed to connect the emulated routers to the physical machine's real interfaces. This latter

is where the emulator was properly installed and configured. All the links between the routers are set to a limited bandwidth, which is 100 Mbps; besides the other parameters that were chosen are a 0% packet loss and no extra delay.

3.6.3 Evaluation Parameters and Results

The built threat scenario was used for an experimental campaign aimed at verifying the correctness of the functionalities of the overall architecture and at obtaining some quantitative parameters. The considered parameters are:

- *Attack reaction time*, that represents the system's latency, or in other words, the overall response time to realize the attack mitigation. It is the period of time between the receipt of the alarm and the conclusion of the mitigation strategy. This time interval takes into account various components, such as the time required to implement all the sub-operations that take place before executing the VM migration, and the time needed to migrate the VM;
- *Flow reconciliation time* defined as time to modify flows towards the target VM. This parameter can be considered as the time necessary to re-conciliate the flows (one incoming and the other outgoing) related to a VM with a single network interface.

Regarding the VM live migration, the authors in [57] [58] presented a possible model for evaluating all the parameters characterizing VM live migration. The traffic volume generated by the migration, denoted with V_{mig}, can be decomposed in the amount of traffic of each iteration. Therefore, by supposing that the algorithm terminates after N rounds, the traffic volume is:

$$V_{mig} = \sum_{i=0}^{n} V_{mig,i} \tag{3.1}$$

and

$$V_{mig,i} = \begin{cases} V_{mem} & \text{if } i = 0 \\ d \cdot l \cdot t_{i-1} & \text{for all rounds } i \not\equiv 0 \end{cases} \tag{3.2}$$

where V_{mem} is the RAM memory of the VM, d is the application dirtying rate, in other words, the memory usage pattern of the VMs applications, l is the page dimension and

t_{i-1} is the duration of the last round. The total migration time, given the migration volume and the available network bandwidth, is:

$$T_{mig} = \frac{V_{mig}}{b} \tag{3.3}$$

Instead the downtime implies the suspension of the execution of the VM, so that the VMM is finally able to transmit the left-over dirty pages over a network with bandwidth b.

$$T_{down} = \frac{d \cdot l \cdot t_n}{b} \tag{3.4}$$

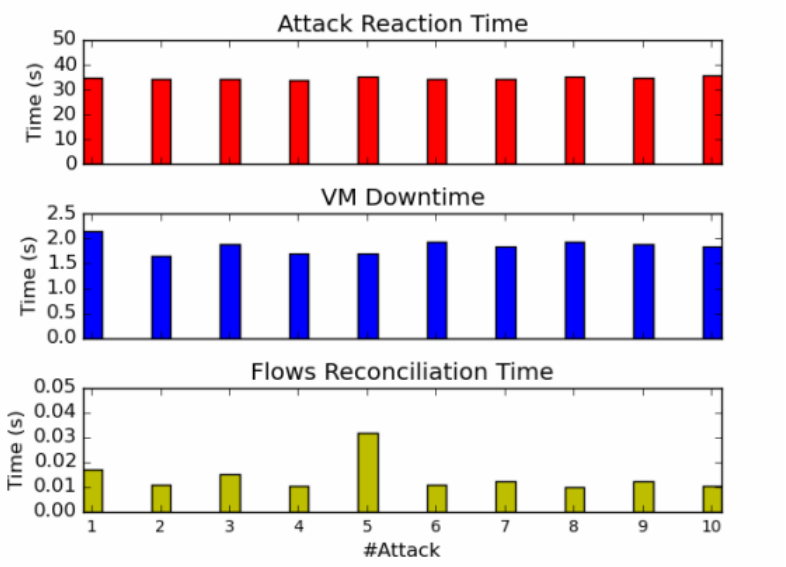
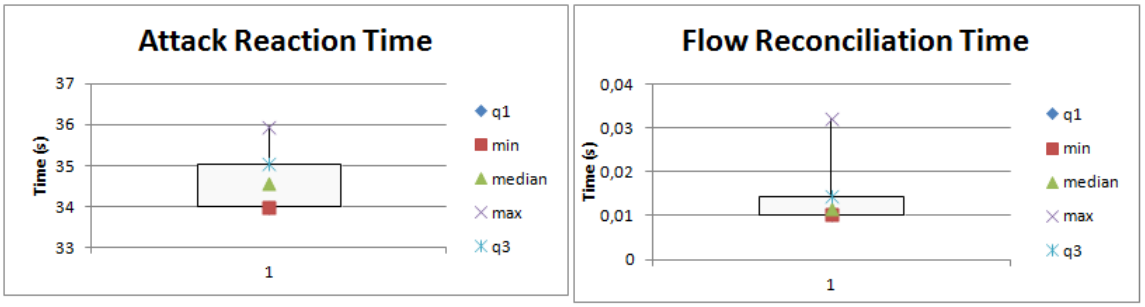


FIGURE 3.18: Attack Results



(A) Total Attack Reaction Time (B) Flow Reconciliation Time

FIGURE 3.19: Results Variability

The results were obtained by launching ten DoS attacks against a VM with 1 virtual CPU and 2GB of RAM and by computing the described parameters. In Figure 3.18 the

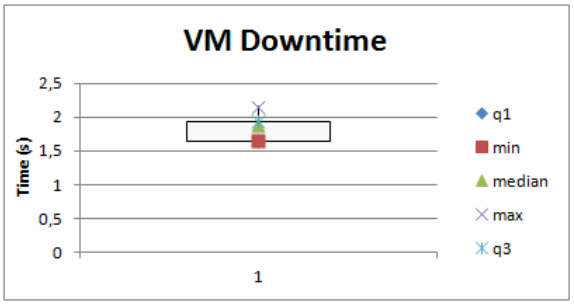


FIGURE 3.20: VM Downtime Variability

results for the different attacks against the same VM are shown: the red bars indicate the total response time of the system which is in a range between 33.986s and 35.95s, with a mean value of 34.71. The VM downtime, or in other words, the time interval in which the VM is not up, is between 1.65s and a worst case of 2.15s, with a mean value of 1.86s. The time needed by the reconciliation module to reconfigure network flows is in a range bewteen 102ms and 320ms. The variability of the results is shown in the box plots in figures 3.19 and 3.20. Obviously the greatest variability is related to the overall attack reaction time, which depends on several components interacting among them to realize the mitigation strategy. Anyway the variability is slightly greater than 1 second.

Chapter 4

Algorithms for Energy-Efficient Cloud Resources Management

4.1 Context and Motivations

The cost-efficiency of a cloud provider depends on its ability to over-subscribe capacity by leveraging the smoothing effect without degrading the quality of service [59]. This citation embodies one of the main objectives in cloud-based data centers that is the attempt to find the right compromise between power consumption and the perceived quality of service, which has to be as nearest as possible to the end-user's expected one. With the advent of virtualization and Cloud Computing, one of the main goal to obtain has become the assurance of a high resources utilization which can also results in a high level of energy-efficiency. The objective of introducing energy-awareness in the IT resources management of virtualized data center has been pursued by plenty of research papers in the scientific community. A great number of techniques have been proposed to increase the energy efficiency of different types of systems in relation to disparate scopes, such as:

1. Hardware

   Most recent CPUs offer the capability of dynamically changing the supply voltage at a cost of a performance degradation: this technique is also known as *Dynamic Voltage Scaling* (DVS). The dynamic scaling of the frequency/voltage is often applied to servers which are characterized by an unbalanced resources utilization, by

lowering the supply voltage at a cost of a performance degradation. This technique allows to achieve a trade off between power consumption and processing. In [60] a joint dynamic voltage scaling for processors and communication links is presented. The authors define an algorithm for real-time applications that computes an efficient routing of communication events by determining a task allocation among heterogeneous processors and communication links, with the objective of maximizing energy savings on one hand, and meeting all real-time constraints on the other.

2. Application

   By examining some inherent characteristics of the applications and the produced workloads, it is possible to design schedulers with the ability of achieving a good energy reduction. Application-oriented scheduling choices can be triggered not only for energy savings, but also for security or resiliency reasons: for instance, replicated nodes belonging to the same system are likely to be deployed over two different physical nodes. Scheduling decisions can be taken in combination with hardware characteristics, like in [61], where a new approach to meld intra and inter-task voltage scheduling is discussed in order to achieve energy savings in hard real-time systems with uncertain task execution time. The authors compute the optimal voltage scheduling by considering multiple concurrent tasks with the earliest deadline first policy on an ideal processor with an unlimited frequencies set. The solution is then adapted to a finite range of frequencies and finally improved by taking into account energy overheads of frequency switching for scheduling and energy reduction.

3. Network

   Scheduling and allocation decisions can be also taken with the aim of minimizing the utilization of network bandwidth and devices. As an example, it is convenient to allocate VMs exchanging a great amount of traffic among them on physical servers which are hosted in the same rack, since the traffic is not sent to the physical switches. This is useful for minimizing the communication overhead over the network: this idea is taken into account in [62]. Network related efforts to reduce the power consumption (which accounts for around 20% of the total in big data centers [63]) can imply different approaches [64]:

- re-engineering of the internal design and architecture of the network devices, in order to lower the complexity and introduce more energy-efficient technologies. These can consist in new silicon circuits, memory technologies, like Ternary Content-Addressable Memory used for packet processing engines, and novel technologies for network links (e.g. pure optical switching architectures);
- dynamic adaptation of the scheduling processing of the devices or link bandwidth to the traffic load. This can be achieved by performing dynamic voltage scaling and idle logic by handling the dynamic trade-off between packet routing and power consumption;
- sleeping/standby techniques which are used to detect unused devices elements, such as ports that can assume a stand-by mode, and by waking up them only when needed.

Also in the cloud paradigm, there is a growing interest in the green aspect: the combination of the elastic resource utilization and the cost-efficiency is also referred to as *Green Cloud Computing*. Cloud big players, for example, are investigating the possibility to introduce green oriented scheduling choices in the VMs allocation task, in order to always select the most efficient server for a new request. In this chapter, the VM consolidation problem is analysed and two different models for power efficient resources management are presented, in addition to a promising heuristic to solve them.

4.2 The VMs Allocation Problem

The problem of packing a number of VMs into a set of physical nodes has its origins in the multidimensional, multiple-choice, multi-constrained *Bin Packing Problem* (BPP) [65]. The problem is multidimensional because the item (the VM) needs to be packed in a physical node according to the available capacity of different resources. For the VM allocation problem, the dimensions can be the CPU utilization, expressed in MHz or in cores, RAM memory and network bandwidth. The possibility to have multiple choices is due to the fact that a single VM can be allocated to more than one physical node (bin); finally this choice could be subject to different constraints. The original problem is formulated like this:

Given a set of bins with a certain capacity and a number of items, each characterized by a well-known size, the objective of the problem is to find the mapping between items and bins which minimizes the number of used bins, without overloading the available capacity of the bins.

It is well-known that the basic bin packing problem is NP-hard. The problem can be easily extended to the specific case of the VMs allocation to the physical nodes.

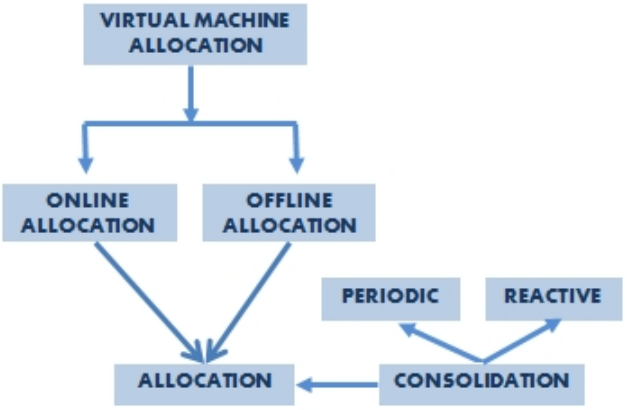


FIGURE 4.1: Possible Approaches

As shown in figure 4.1, the classification of the VMs placement problem can be considered according to two different versions: in the on-line algorithms, the problem consists in finding an allocation for incoming VMs as soon as one request arrives. Hence, the online algorithms just process one VM at a time. In the on-line version, the aim is to take a good decision without being conscious about the future requests and by only analysing the current load. This is the usual approach in the Infrastructure-as-a-Service (IaaS) platforms, whose objective is to satisfy a client's request as soon as possible. All the scheduling algorithms used in the most common IaaS platforms uses an on-line version which can achieve different objectives defined through policies. Online algorithms cannot guarantee the optimal solution, since the main issue is that they cannot guess when the input will end (how many items will be packed into the bins). The simplest way to find a solution for the online version of the problem is to rely on greedy algorithms, which have the property of quickly converge to a good solution, by making a locally optimal choice at each step. This is basically derived from the substructure of the optimal solution of a problem, which generally consists in the combination of the optimal solutions of its sub-problems. In the off-line version, instead, the list of VMs and their resource demands are well-known inputs for the algorithm. Obviously the off-line algorithm can

find a better solution than the on-line ones because the algorithm has the view of all the items to pack. This allows to take allocation decisions which can easily reach a smarter solution.

4.2.1 Problem Formulation and Model

The VMs allocation problem is often referred to as *Static Server Allocation Problem* (SSAP), as found in [66]. The formulation takes into consideration m services, n servers and k different resources, such as CPU cores, RAM memory units and bandwidth usage. Let us suppose that each service requires a certain amount of resource k, while every server has a total amount of that resource and it is characterized by an activation cost (for example the amount of money to pay for every hourly usage). The activation of a server is denoted with the decision variable y_i, which holds 1 if the server is executing at least one service. The cost for activating it is c_i. If the service j is hosted by the server i, the variable x_{ij} holds the value 1, otherwise it is set to 0. The objective is to find an allocation for each service by minimizing the total cost due to the active servers and by not exceeding the amount of required resources.

$$min \quad f = \sum_{i}^{n} c_i \cdot y_i \tag{4.1}$$

The constraints are expressed like this:

- each service should be hosted by at most one server;

$$\sum_{i}^{n} x_{ij} = 1 \quad \forall j \tag{4.2}$$

- all the services running on each server should not use more resources than the available ones;

$$\sum_{j}^{m} r_{jk} \cdot x_{ij} \leq s_{ik} \cdot y_i \quad \forall i \quad \forall k \tag{4.3}$$

  where r_{jk} is the amount of resource k required by the service j and s_{ik} is the amount offered by server i.

Other typical constraints of the problems can be:

1. the number of services on a particular node i or a subset of nodes should not exceed a certain threshold;
$$\sum_{j}^{m} x_{ij} \leq n_i \tag{4.4}$$

2. a subset of service S must be allocated to different servers for specific reasons (e.g. for resiliency replicated services should be on different physical nodes);

$$\sum_{j \in S} x_{ij} \leq 1 \quad \forall i \tag{4.5}$$

3. a subset of servers R should be used for the allocation of a particular service or group of services J (for instance for security reasons).

$$\sum_{i \in R} x_{ij} = 1 \quad \forall j \in J \tag{4.6}$$

The problem is static because there is the assumption that every service use a constant amount of each resource during all its life-time, or in other words, it has a stable workload. A generalization of the problem is the *SSAP with variable workload*, in which time is another dimension. Time is divided in intervals t, $t \in [0...\tau]$: this set is also known as *planning period*. In that case the amount of resource k requested by the service j is related to the time interval t. Hence, the budget constraints becomes:

$$\sum_{j}^{m} r_{jkt} \cdot x_{ij} \leq s_{ik} \cdot y_i \quad \forall i \quad \forall k \quad \forall t \in [0...\tau] \tag{4.7}$$

4.2.1.1 Greedy Techniques for On-line and Off-line Allocation

As explained in section 4.2, the on-line version of the BBP is often faced with simple greedy techniques which do not guarantee to find the optimal solution, but they have the nice property of rapidly finding a very good solution. Examples of algorithms for the online bin packing problem are:

- First-Fit, which is a very straightforward greedy approximation algorithm. For each new item, the allocation policy consists in sequentially scanning all the available bins in order to pack the item in the first bin which has enough capacity to host it. If no bin is found, it opens a new bin and puts the item within it. It is simple to show this

algorithm achieves an approximation factor of 2, or in other words, the number of bins used by this algorithm is no more than twice the optimal number of bins. This is due to the fact that two bins cannot be at most half full: such a possibility implies that, at some point, one bin was at most half full and a new one was used to pack an item of size at most $\frac{V}{2}$, where V is the size of the bins. But since the first one has at least a space of $\frac{V}{2}$, the algorithm will not open a new bin for any item whose size is at most $\frac{V}{2}$. Only after the bin fills with more than $\frac{V}{2}$ or if an item larger than $\frac{V}{2}$ arrives, the algorithm may open a new bin. The algorithm pseudo-code for the first-fit VM allocation algorithm is shown in 1.

Algorithm 1 First Fit

```
Input: nodes_list, VM
Output: allocated, node
Set allocated to false
Get nodes_list size
for each node in nodes_list do
    Get current node
    if VM requested CPU ≤ current node available CPU AND VM requested
        RAM ≤ current node available RAM then
        Allocate VM on current node
        Set allocated to true
    end if
    if allocated is true then
        Return the node where the VM was allocated
    else
        Return an error
    end if
end for
```

- Best-Fit allocates a new item by scanning all the bins and packing it in the one with the tightest capacity needed to allocate it. Therefore the selected bin will be the fullest after packing the new item. The performance of the best fit policy is the same as the first-fit. The pseudo-code for the Best-Fit VM allocation algorithm is shown in 2.

- Next-Fit uses a simpler allocation policy: if the item fits in the bin where the last item was packed, use it; otherwise a new bin is used. The approximation factor is the same as First-Fit: by considering the optimal solution for packing a number Y of items in a set of B bins, it was demonstrated that the Next-Fit algorithm never uses more than $2 \cdot B$ bins.

Algorithm 2 Best Fit

Input: nodes_list, VM
Output: allocated, node
Set boolean allocated to false
Get nodes_list size
Get the first node
Set the minimum amount of CPU and RAM to the values of the remaining CPU and RAM of the first node
Set current node to the first one
for each node starting from 2 to node_list size **do**
    Get current node
    Get the remaining CPU and RAM of the current node
    if the current CPU $\leq$ minimum CPU AND current RAM $\leq$ minimum RAM AND the allocation of the VM to the current node is possible **then**
        Set allocated to true
        Update current node
        Update minimum CPU and RAM
    end if
end for
if allocated is true **then**
    Allocate the VM to the selected node
    Return the node where the VM was allocated
else
    Return an error
end if

The off-line algorithms can find better solutions, since they have a complete view of all the items: hence, it is possible to sort the item list in increasing or decreasing order and then apply the first fit or best fit policy.

Algorithm 3 First-Fit Decreasing

Input: list_nodes, list_vms
Output: mapping
Order the VMs list in decreasing size of requesting CPU and RAM
for each VM in the sorted list **do**
    Get the current VM
    Call first_fit (VM, list_nodes)
    Udpate mapping
end for

The best-fit decreasing and first-fit decreasing strategies are among the simplest heuristic algorithms for solving the bin packing problem. It was demonstrated that they use no more than $\frac{11}{9} \cdot OPT + 1$ bins (where OPT is the number of bins given by the optimal solution). The simpler of these, the First Fit Decreasing (FFD) strategy, sorts the items in decreasing order of their sizes, and then it inserts each item into the first bin in the

list with sufficient remaining space. The pseudo-code for the first-fit decreasing VM allocation algorithm is shown in 3.

4.3 A Generalization of the Problem

An additional VMs placement problem is however enabled by the live migration mechanism implemented by common hypervisors. Indeed, the fact that VMs are created and destroyed on demand and the resources they request may change over time might lead to an inefficient utilization of resources, especially from the power consumption viewpoint. In virtualized data centers, it could be convenient to consolidate the infrastructure in order to keep the allocation of VMs up-to-date and to minimize the number of active servers. Thus, the live migration mechanism can be employed to reallocate some VMs on different physical hosts, in order to optimize the resource utilization. Hence a new version of the VM placement problem, which considers the current allocation of VMs and determines the set of migrations to perform with the aim of minimizing the number of active nodes. This is a task which can be triggered every interval of time or by events: in the first case, one of the issues is to choose the right compromise between the time the consolidation needs to be activated and the server usage pattern. A long period can lead to sub-optimal allocation and overloading situations, while a too short one can lead to a huge number of useless migrations. Other techniques fall in the reactive category: in this case the algorithm is executed when a threshold condition for the utilization of the server's resources is met.

4.3.1 Related Work

A huge number of papers deal with the problem of reducing the typical energy consumption of a cloud-based data center, by either approaching the allocation problem and/or the consolidation one. Enacloud [67] is an on-line framework for VM allocation and consolidation in cloud environments which is activated when an event occurs, such as workload arrive, departure and resize. The idea behind the proposed recursive algorithm is that small VM workloads are more likely to be inserted into left gaps. The objective is to avoid that the allocation of a new workload leads to open a new box. That is why the algorithm attempts to allocate the packed workloads which are smaller

than the requested one with it. In [68] two exact algorithms aimed at energy-aware VMs allocation and consolidation are presented. They are combined in order to reach a common objective that is the minimization of the data center energy consumption. The authors propose an Integer Linear Programming model for the allocation problem which is solved through an extension of the best-fit heuristic. The VMs are ordered in decreasing order of energy consumption and the algorithm tries to allocate the most energy demanding VMs first. Besides, a model for the exact migration algorithm is also built with the goal of maximizing the number of idle servers in the infrastructure. Then the two approaches are merged together: the exact migration model is solved when two or more VMs are destroyed. Anyway the authors face the migration problem through a linear solver, thus limiting the scalability of their approach.

Also Beloglazov *et al.* [69] split the problem of energy-aware allocation into two parts: new VM requests are allocated through a modification of the best-fit decreasing heuristic; then the obtained allocation scheme is optimized thanks to two thresholds that signal under-loading or overloading situations. The first threshold is used to implement green consolidation, while the second one is employed to reduce hot spots in the utilization of the resources of the servers. The algorithm is aimed at minimizing the number of migrations, by selecting as migrating VM the one that can reduce the utilization of resources in order to avoid overload. If the host utilization falls under a low threshold, the algorithm aims at migrating all of its VMs in order to switch it off. The paper only considers CPU as the most impacting resource for energy consumption and a threshold based approach for consolidation may not be suitable for a very dynamic environment.

In [70], the authors focus on consolidation: given an allocation scheme, the problem is to generate a mapping which guarantees not only the minimum number of used servers, but also the smallest set of migrations. Their framework is based on the idea of reallocating a group of VMs only if their migrations are useful to switch off some of the servers, otherwise no migration is activated. The rationale behind the algorithm is to associate both the VM and the physical node with a bi-dimensional score based on CPU and memory utilization. Anyway they only considered the static resource utilization of both the VMs and the servers and no energy consumption parameter.

The work in [71] presents an ILP model to solve the problem of reallocating VMs over predefined intervals of time, by also taking into account the migration overhead. The authors show some experimental results based on a real data set, considering VMs with stable workload applications, and they point out the percentage of energy reduction

(between 45% and 55%) and for different values of migration overhead both at source and destination sites. Anyway the proposed problem is solved directly using the ILP model, thus limiting the scalability of the approach. A heuristic for large scale problems is only left as future work. Xiao *et al.* [72], face the problem of detecting and solving the underload and overload of servers while assuring the benefits of green cloud computing. The work also introduces a technique to estimate the future resource needs of virtual machines and the skewness which is a measure of the unevenness in the usage of different types of resources. The consolidation algorithms are triggered according to the server's temperature: it represents a way to evaluate overload or underload situations in the utilization of a combination of resources. The hot spot mitigation is triggered when the temperature is higher then a warm threshold, while the consolidation is used if the server has a temperature below the green computing threshold.

The paper in [73] introduces a technique to perform dynamic virtual machine consolidation in a completely distributed fashion, based on a peer-to-peer network of physical machines by constructing a neighbourhood for each node. This avoids a single point of failure and improves the scalability of the system. The authors also proposes a consolidation algorithm which relies on ant colony optimization, a probabilistic technique for solving computational problems, and they compare it with other well-known algorithms in terms of packing efficiency and number of triggered migrations. Also this work does not take into account the different characteristics of the physical server, such as their energy efficiency.

4.4 Power Efficient VMs Consolidation Problem

Here a new version of the VMs re-allocation problem is proposed, namely the *Power Efficient VMs Consolidation*. Suppose that the cloud environment consists in a set J of n_{tot} physical machines, of which only a subset ($J^o_{act} \subseteq J$) is active before the consolidation, as they are hosting at least one VM. Each server j is characterized by an initial power consumption before the consolidation, P^o_j. We tackle the following problem:

Considered a set of m VMs, given the current allocation of each VM to a physical node, the amount of resource available at each server, the amount of resource requested by the VMs, the power consumption model of the server, the problem consists in finding

the set Mig of migrations defining a new set of active nodes J^N_{act}, that minimizes the linear combination of the power consumption of the resulting active servers (P^N_j) and the number of migrations:

$$f = \alpha \cdot \frac{\sum_{j \in J^N_{act}} P^N_{act}}{\sum_{j \in J^o_{act}} P^o_{act}} + \beta \cdot \frac{|Mig|}{m} \tag{4.8}$$

by guaranteeing that the resources used at each physical server do not exceed the available amount when migrating a VM.

4.4.1 Server Power Modeling

In [74] and [75] the energy consumption of a physical machine is defined as the power consumption due to the utilization of its resources over time. Let us consider P resources and Q tasks that request a certain amount of each resource, the overall utilization of a specific resource can be computed as:

$$U_q = \sum_{p=0}^{P-1} u_{qp} \tag{4.9}$$

where u_{qp} is the utilization of the resource p issued by the task q. Therefore the total utilization over all the considered resources is:

$$U_{tot} = \sum_{q=0}^{Q-1} U_q \tag{4.10}$$

Several papers in literature showed the most impacting resource on power consumption is the CPU; then, if this simplification is taken into account, the power consumption of a server j can be defined as:

$$P_j(t) = P_{idle,j} + (P_{max,j} - P_{idle,j}) \cdot U_{cpu,j}(t) \tag{4.11}$$

where $U_{cpu,j}$ is a value between 0 and 1. Basically the power consumption consists in two factors: the first one is the "static component" ($P_{idle,j}$), that is the consumption due to the elementary physical components (transistors), while the second one is strictly dependent from the utilization of the macro-systems of the host (RAM, CPU, network interfaces and so on).

4.5 A Mixed Integer Linear Model

The consolidation problem can be modelled through a Mixed Integer Linear Programming (MILP) model, whose parameters are presented in Table 4.1. The overall model is shown in Table 4.2.

Input parametes:	
x^o_{jk}	holds 1 if the VM k is allocated to node j before the consolidation
s_{ij}	is the amount of resource i available at node j
r_{ik}	is the amount of resource i needed to allocate VM k
η	is a value between 0 and 1 that takes into account the overhead for migrating any VM
$P_{idle,j}$	is the idle power consumption of node j
$P_{max,j}$	is the maximum power consumption of node j
Decision variables:	
y_j	is 1 if node j is active after the consolidation, 0 otherwise
P_{y_j}	is the current power consumption of node j after the consolidation
$z^{<-}_{jk}$	is 1 if VM k migrates towards node j, 0 otherwise
$z^{->}_{jk}$	is 1 if VM k migrates from node j, 0 otherwise
x^N_{jk}	is 1 if VM k is allocated to node j after the consolidation, 0 otherwise

TABLE 4.1: Model Parameters

Each server is associated with a binary decision variable y_j, $j \in [1..n]$, which holds 1 if the server is active and 0 otherwise. We consider m VMs, identified with the subscript k, ranging from 1 to m. A migration of a VM k from a node j is represented with the binary decision variable $z^{->}_{jk}$, while if it is migrating to j, the variable $z^{<-}_{jk}$ is used. The objective (4.8) to minimize the linear combination of the new overall power consumption and the number of migrations necessary to the consolidation can be expressed as in (4.12). Two variables that indicate one single VM migration is used and that is why there is a division by two in the count of the number of migrations. The overall power consumption and the number of migrations are normalized, in order to obtain

comparable values.

The MILP Model:

$$\min f = \alpha \cdot \frac{\sum_{j=1}^{n} y_j \cdot P_{y_j}}{\sum_{j=1}^{n} P_{init,y_j}} + \beta \cdot \frac{\sum_{j=1}^{n} \sum_{k=1}^{m} \frac{z_{jk}^{->} + z_{jk}^{<-}}{2}}{m} \tag{4.12}$$

s.t.

$$P_{y_j} = P_{idle,j} + (P_{max,j} - P_{idle,j}) \cdot \frac{\sum_{k=1}^{m} x_{jk}^{N} \cdot r_{ik}}{s_{ij}} \quad i = CPU \tag{4.13}$$

$$\sum_{j=1}^{n} x_{jk}^{N} = 1 \quad \forall k \tag{4.14}$$

$$y_j \leq \sum_{k=1}^{m} x_{jk}^{N} \quad \forall j \tag{4.15}$$

$$y_j \geq x_{jk}^{N} \quad \forall j, \forall k \tag{4.16}$$

$$\sum_{k=1}^{m} ((r_{ik} \cdot x_{jk}^{o}) + (r_{ik} \cdot z_{jk}^{<-}) - (r_{ik} \cdot z_{jk}^{->})) \leq \eta \cdot (s_{ij} \cdot y_j) \quad \forall j, \forall i \tag{4.17}$$

$$x_{jk}^{o} - z_{jk}^{->} \geq 0 \quad \forall j, \forall k \tag{4.18}$$

$$x_{jk}^{N} + z_{jk}^{->} \leq 1 \quad \forall j, \forall k \tag{4.19}$$

$$x_{jk}^{o} - x_{jk}^{N} - z_{jk}^{->} \leq 0 \quad \forall j, \forall k \tag{4.20}$$

$$x_{jk}^{N} - z_{jk}^{<-} \geq 0 \quad \forall j, \forall k \tag{4.21}$$

$$x_{jk}^{o} + z_{jk}^{<-} \leq 1 \quad \forall j, \forall k \tag{4.22}$$

$$-x_{jk}^{o} + x_{jk}^{N} - z_{jk}^{<-} \leq 0 \quad \forall j, \forall k \tag{4.23}$$

TABLE 4.2: Problem Model

We normalize the power consumption to the sum of the initial powers of the active servers before the consolidation, and the number of migrations to the number of VMs. It is assumed that every node can have a different known power consumption model. Therefore the power consumption can be computed through eq. (4.13). The binary

decision variable x^N_{jk} represents the allocation of the VM k to the node j after the consolidation, while instead, x^o_{jk} is the current mapping. After the consolidation process, by looking at the new allocation, each VM should be allocated to at most one physical server. This constraint is assured in the equalities (4.14). Besides, every physical node is considered active if, and only if, at least one VM is running on it (4.15-9). Then we have the so-called budget constraint: for each server j and for every considered resource i, the amount of resources held by the old assignment and the resources needed or freed by the VMs migrating, should not exceed the maximum available amount (4.17). The parameter η (between 0 and 1) is used to take into consideration the additional migration overhead. If a VM k is migrated from j, $z^{->}_{jk}$ is equal to one:

x^o_{jk}	x^N_{jk}	$z^{->}_{jk}$	**acceptable**
0	0	0	yes
0	0	1	no
0	1	0	yes
0	1	1	no
1	0	0	no
1	0	1	yes
1	1	0	yes
1	1	1	no

(A) Possible Combinations for $z^{->}_{jk}$

x^o_{jk}	x^N_{jk}	$z^{<-}_{jk}$	**acceptable**
0	0	0	yes
0	0	1	no
0	1	0	no
0	1	1	yes
1	0	0	yes
1	0	1	no
1	1	0	yes
1	1	1	no

(B) Possible Combinations for $z^{<-}_{jk}$

FIGURE 4.2: Possible combinations

$$z^{->}_{jk} = 1 \quad => \quad x^o_{jk} = 1 \qquad AND \qquad x^N_{jk} = 0 \tag{4.24}$$

this necessarily implies that k was allocated to j before the consolidation ($x^o_{jk} = 1$), and in the new mapping it is not assigned to j anymore ($x^N_{jk} = 0$). Instead, if VM k is not migrated from the server j ($z^{->}_{jk}$ is equal to 0), it is possible that it was allocated to j before the consolidation and, afterwards, it is still assigned to it, or it was not allocated to j. Anyway in this case, if $z^{->}_{jk} = 0$, it cannot happen that $x^o_{jk} = 1$ and $x^N_{jk} = 0$. All the possible combinations and the acceptable ones are shown in Table 4.2(a) and (b). By considering the other variable, $z^{<-}_{jk}$:

$$z^{<-}_{jk} = 1 \quad => \quad x^o_{jk} = 0 \qquad AND \qquad x^N_{jk} = 1 \tag{4.25}$$

if k migrates to y, this implies that the VM was not allocated to the node before the consolidation, and afterwards it is assigned to it. The constraints that assure the avoidance of unacceptable combinations are 4.18-12-13. The two decision variables that define the old and new allocation should be consistent also if you are taking into account the migration of k towards j (4.21-15-16).

4.6 A Simulated Annealing based Algorithm for Power-Efficient VMs Consolidation

4.6.1 Desirability of a VM Migration

The formulated model has the objective to minimize the linear combination of the power consumption of the active servers normalized to the total initial power and the number of migration normalized to the number of VMs, as it follows:

$$min \quad f = \alpha \cdot \frac{\sum_{j \in J^N_{act}} P_j}{total\_init\_power} + \beta \cdot \frac{num\_migs}{num\_VMs} \tag{4.26}$$

where P_j is the power consumption of the active server j after the consolidation, the denominator in the first ratio is the overall consumption of the initial nodes and num_migs is the number of migrations needed to perform the consolidation. Since the saving in terms of power is permanent in contrast to the temporary consumption due to a live-migration, there are two weights in (4.12), α and β, which stress out the relative importance between the number of hibernated hosts and the number of VM re-allocations. The algorithm is based on a very simple idea: since the aim is to have a balance between the consolidation of the nodes and the overall power consumption, in order to evaluate the possible migrations, a value that is defined as the "desirability" of migrating a VM k from the node j to another server h is taken into account. This value is computed as:

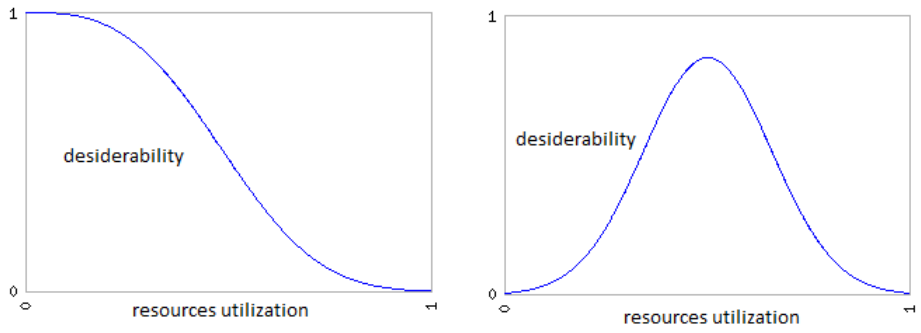
$$D^k_{jh} = \frac{D_j(u_j) + D_h(u_h)}{2} + \triangle_P \tag{4.27}$$

where the first parameter is the arithmetic mean of the convenience of migrating the VM k from the node j and the willingness of the node h of accepting the VM; the second one represents the difference in the power consumption after the migration of the VM to the new node. A migration of a VM from a source node is convenient

when the server has a low resources utilization, or in other words, it could be a good candidate to be consolidated. A migration to a specific node should be leveraged when the destination has a medium or high utilization, as long as it can accept new VMs without being overloaded. We obtain the desirability values for the source and the destination through two functions that express the attractiveness of the VM move, given the resource utilization. The two functions are chosen such that the migration of the VM has a high total desirability if the utilization of j is low and the one of h is medium: this means it could lead to consolidate a new server. Then, it is possible to evaluate a VM migration from a source to a destination, by computing the desirability for the nodes as it follows:

$$D_{src}(u) = e^{-a\cdot(u)^3} \tag{4.28}$$

$$D_{dst}(u) = b \cdot e^{-c\cdot(u-1/2)^2} \tag{4.29}$$



(A) Source Node (B) Destination Node

FIGURE 4.3: Desirability Functions

For the source node, we use a sigmoidal-like shaped function, in order to have high values when the utilization is low and vice-versa; the graph is shown in Figure 4.3(a). For the destination node, instead, we use a sort of gaussian function (4.3)(b) which has this property: if the destination has a medium or high utilization, the desirability of the move is high; on the contrary the migration is discouraged. The resource utilization is computed as the medium utilization over the considered resources (CPU and RAM). The power difference produced by the migration, $\triangle_P$, is computed by taking into account the power model for each of the physical servers: here, we assume that all the nodes can be characterized by a different power consumption and therefore they can have different levels of energy efficiency. Thus, given the old and new power consumption both at the

source and the destination nodes, the power delta is defined as:

$$\triangle_P = \frac{|P_{src}^{old} - P_{src}^{new}|}{P_{dst}^{new} - P_{dst}^{old}} \tag{4.30}$$

Basically, among all the possible migrations, we do not consider the most attractive ones based solely on the resource utilization and the deriving desirability values, but also the ones producing the smaller increase in the power consumption.

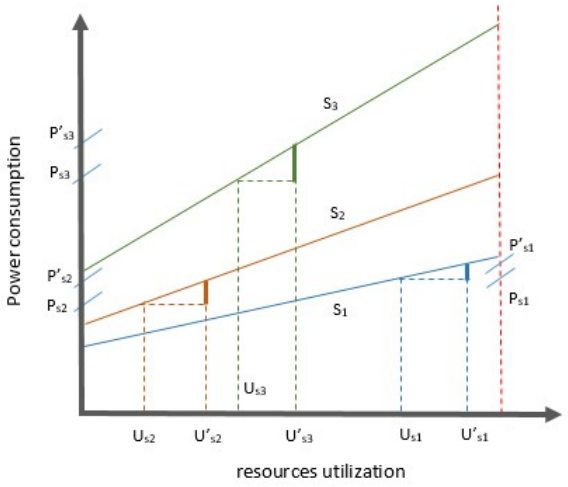


FIGURE 4.4: Power Deltas

This is explained in figure 4.4, where three linear power models of just as many nodes, are drawn: by considering the new resource utilization after accepting the migrating VM, the new values of power are computed. The power deltas are obtained as the difference between the new and the old value of the power consumption. So, when considering a migration, the reduction of power consumption should be as large as possible at the source node (the numerator in the equation 4.30), while the increase at the destination should be as small as possible (the denominator in 4.30), in order to always select the most energy-efficient servers.

4.6.2 Heuristic Description

The algorithm is basically derived from the meta-heuristic *Simulated Annealing*, an optimization technique capable of finding a good approximation to the global optimum of a given function in a large search space. This is a meta-heuristic because it is inspired by nature behaviour: the annealing in metallurgy is a technique involving heating and

controlled cooling of a material to increase the size of its crystals and reduce their defects. The algorithm accepts as input parameters the current allocation, the list of VMs with their static requirements for RAM and CPU, the active nodes with the remaining amounts of each resource, the initial value of temperature and a cooling factor for decreasing the temperature. As output, it computes the list of migrations with its size, the number of consolidated nodes and the overall power. The algorithm sets the initial value of the objective function, given by the sum of the power consumption of all the active nodes. As long as the temperature is greater than a certain low threshold, in the two for cycles the value of desirability for every possible VM re-allocation (assumed that this migration was not accepted before, or it was refused or it undoes a previous migration) is computed according to (4.27) (lines 3-15). If there is no possible migration the main cycle is broken. Then the evaluated migrations are ordered in decreasing order; the best valued re-allocation is chosen and triggered (lines 16-20). The allocation is then updated accordingly: if the source node was consolidated (it has no VMs running on it), it is erased from the list of active nodes and the consolidation is indicated through a boolean variable. The number of migrations is increased by one and the new value of the objective function is obtained according to (4.12) (lines 21-27). If it is the first iteration, the migration is unconditionally accepted; if the source node was not consolidated then the variable unsuccessful is increased by one. Otherwise the difference between the new value of objective function and the old one is computed. If this difference is negative or the source node was consolidated, it means that we had an improvement for the objective function. There is also need to take into account the case in which power was increased due to the migration, but a node was switched off thanks to it. Therefore the migration is accepted and the variable unsuccessful, which takes into account the number of migrations that were accepted by probability and that gave no improvements to the objective function, is set to zero. If the difference is positive, we can accept a worsening in the value of the objective function. This is the typical approach of the Simulated Annealing to avoid a local minimum. The degradation is acceptable with a certain probability, having this property: it is a decreasing function of the difference between the new and the old values of the objective function, and also of the temperature. This parameter is chosen so that it is sufficiently high to assure great values of probability at the beginning of the algorithm. But then the temperature is decreased more and more because of a cooling factor, and also due to the number of accepted migrations thanks

Algorithm 4 Simulated Annealing based Consolidation

```
Input: current_mapping, vms, active_nodes, initial_temp, cooling
Output: mig_list, cons_nodes, overall_power
Set the current objective function according to eq. 4.26
while Probability is greater than a threshold do
    Set the boolean consolidated parameter to false
    for Each VM i do
        Get VM hosting node
        for Each physical_node j do
            if VM i hosting node is different from node j AND allocation of VM i
                to j is possible then
                if If the migration is not an already accepted one OR a reverse one OR
                    a refused one then
                    Evaluate the desirability of the migration and store it
                end if
            end if
        end for
    end for
    if There is no possible migration then
        Break while cycle
    end if
    Order migrations in desirability decreasing order
    Select the migration corresponding to the greatest value of desirability
    Perform the migration and adjust the allocation accordingly
    if  Source node has no other VMs then
        Remove it from the active_nodes list
        Set the boolean consolidated parameter to true
    end if
    Increase mig_list size by 1
    Compute the new objective function value according to eq 4.26
    if This is the first iteration then
        Accept the migration
        if consolidated==false then
            Increase unsuccessful by 1
        end if
    else
        Compute the difference between new and current objective function values
        if The difference is less than 0 OR consolidated==true then
            Accept the migration
            Unsuccessful = 0
        else
            Update the current temperature
            Evaluate the probability of acceptance
            if The migration was accepted by probability then
                Increase acc_by_prob by 1
                Increase unsuccessful by 1
            end if
        end if
    end if
```

Algorithm 5 Simulated Annealing based Consolidation Cont.d

```
    if The migration was accepted then
        Update the current objective function value with the new one
        Add the migration in the list
    else
        Undo the migration
        Save the migration in the list of refused migrations
    end if
    if unsuccessful is greater than 0 then
        Remove the last unsuccessful migrations from the list
    end if
end while
```

to the probability, according to this law:

$$temp = temp \cdot cooling \cdot (1 - \frac{acc\_by\_prob}{num\_vms}) \tag{4.31}$$

The acceptance probability is obtained through this function:

$$prob = e^{(-\frac{delta}{temp})} \tag{4.32}$$

where *delta* is the difference between the values of the objective function. The outcome of the evaluation of the probability is obtained through the roulette method. If the evaluation is positive, the variables *acc_by_prob* and $unsuccessful$ are increased by one (lines 28-46). In the end, if the migration was accepted, the current value of function is substituted with the new one and the migration is inserted into the list. Otherwise the migration is undone and it is inserted into the list of refused migrations. If the algorithm terminates and the variable $unsuccessful$ is greater than 0, it is possible to delete the last $unsuccessful$ migrations from the list, since they were accepted by probability, but they are not convenient to the minimization of the objective function (lines 47-57).

4.6.2.1 A Simple Example

To explain how the algorithm works in practice, suppose to have the following scenario. Let us consider two physical servers, s_1 and s_2. For convenience decimal values are used for the CPU and RAM parameters of both the physical servers and the VMs and we define this kind of conversion: a value of 0.1 for CPU means 1 core, while 0.1 for RAM is equal to 512 MB basic memory unit. For the sake of simplicity, linear power models

which are proposed in a cloud simulator, namely SimCloud [76], are used. The authors define the linear model as a vector, in which the first parameter is the consumption of the server when it is fully utilized, while the second is the percentage of max_power, defining the static component. Given the utilization, the power consumption is basically computed with equation (4.11). The values of CPU, RAM and the power models for servers and VMs are shown in Tables 4.3 and 4.4 respectively.

TABLE 4.3: Example PHs Parameters

Host	CPU spec	RAM spec	Power Model
s1	1.2 (12 cores)	2.8 (14336 MB)	(170 W, 0.3)
s2	2.0 (20 cores)	3.0 (15360 MB)	(270 W, 0.3)

TABLE 4.4: Example VMs Parameters

Virtual Machine	CPU spec	RAM spec
m1	0.5 (5 cores)	0.4 (2048 MB)
m2	0.3 (3 cores)	0.3 (1536 MB)
m3	0.2 (2 cores)	0.3 (1536 MB)

For the proposed use case, no over-provisioning between the virtual and the physical resources and a stable workload for the VMs are considered. For the desirability functions, these parameters are chosen: $a = 6$, $b = 0.85$ and $c = 20$. Instead the weights α and β in the objective function are set to 0.9 and 0.1 respectively. Considered a mapping between the VMs and the servers, $map = (s_1, m_1)$, (s_2, m_2), (s_2, m_3), the node s_1 has a power consumption of 100.98W, while s_2 128.25W. The allocation scheme can be obviously consolidated: an aggressive consolidation algorithm could choose to migrate m_1 to s_2, thus freeing one node with only one migration, by achieving a total power consumption of 175.50W. According to our algorithm, we evaluate every single possible migration, thanks to the desirability functions and the power deltas:

$$
\begin{aligned}
m_1: \quad & s_1 -> s_2 \quad des = 0.53 \quad delta = 1.06 \\
m_2: \quad & s_2 -> s_1 \quad des = 0.63 \quad delta = 0.95 \\
m_3: \quad & s_2 -> s_1 \quad des = 0.63 \quad delta = 0.99
\end{aligned}
$$

The most efficient migration is $m_3 : s_2 -> s_1$, with a total desirability of 1.62. The current value of objective function is equal to 1 multiplied by α (0.9). By triggering the selected migration, we obtain a new value of 0.934 and, since it is the first iteration, the migration is accepted. In the second iteration the best rated migration is $m_2 : s_2 -> s_1$, with a total desirability of 1.81. This migration leads to an objective function of 0.655: since we had a reduction of 0.279 (and a node was consolidated), the migration is accepted. As there is not any further possible migration, the algorithm stops with 1 consolidated node, 2 migrations and a total power consumption of 149.77W. Therefore with an additional migration, we were able to save 25.73W.

4.7 Data Centers Network Energy Efficiency

Big data centers which provide cloud enabled services and applications are characterized by an increasing demand of bandwidth with a resulting need for network power awareness. Classical network infrastructures are based on the old tree hierarchy of switches and redundant paths among them. Over the last years, they have become not sustainable for cloud players and new architectures have been proposed with the aim of increasing network efficiency. This latter is traditionally expressed in terms of parameters such as network throughput, equipment capital expenditure (CapEx), network diameter and so on. However, the typical evaluation of the network efficiency has been also revolutionized, by stressing out the importance of the power consumption of the network devices. In [77], a new parameter is taken under examination, namely the *Network Power Effectiveness* (NPE), which can be defined as the ratio between the aggregate network throughput and the total network power consumption. This value can be expressed in *bps per Watt* and it clearly reflects the trade-off between throughput and power consumption. Along with novel network architectures, new techniques for increasing energy efficiency are also emerging; among them, we can find:

- *Power over Ethernet* (PoE), which is a technology of ad-hoc systems with a cable that can provide both data connection and electrical power. PoE can bring many advantages, such as time and cost savings for installing electrical power cables, and flexibility, since elements which can be powered through this technique (as

an example wireless access points) can be located wherever they are needed and easily re-positioned, if required.

- *Sleep on Idle and Wake on Arrival*, which are implemented in commercial switches and allow to put network interfaces (or the entire switch) in sleep-mode when there is no traffic flowing, and to wake-up a link, only when a packet arrives. Research papers argue that the traffic load in regular data centers is very day-time dependent: in [78], it is shown that the average link utilization in the connections to the aggregate switches is only 8% of the capacity for 95% of the time, while in the core layer the utilization is between 20% and 40%. Therefore it is possible to rely on these techniques to dynamically switch-off under-utilized network devices and links. Also protocols can benefit from energy efficiency: for instance, the IEEE 802az [79] standard is aimed at achieving energy reduction in the Ethernet based communications, by activating a link only when real data are sent. When there is no effective data transmissions, the standard uses a signalling protocol in order to save energy: the sender can flag that there is no transmission through the *low power idle* and the link can become idle. The sender can periodically use some signals, so that the link does not remain quiescent for too long without a refresh. Finally, when the transmitter wishes to re-activate the link, it sends normal signals.

- *Power aware routing and rate adaptation*, which introduce power consumption awareness in the devices responsible for routing functionalities. Rate adaptation is useful to adapt the port transmission rate to the traffic load. Anyway, since networking devices are not energy proportional, in the sense that the energy is not proportional to the load, this techniques cannot achieve a sensible energy reduction. Results in [80] show that network devices overhead is 85% of the total power consumption of a fully active one.

4.7.1 Related Work

The paper [77] proposes an interesting study of the network power effectiveness of the most common data centers architectures. The authors evaluate the network power consumption, by applying both the classical and power-aware routing algorithms to well-known topologies. Besides they assess the impacts on the results of different parameters,

such as the topology size, load, traffic patterns and so on. VMFlow [81] proposes a *Network Aware VM Placement* problem formulation, which is solved though a greedy heuristic. This latter considers one single traffic demand at a time and jointly computes a VM placement and a path for the flow with the objective of minimizing the total network power. The algorithm selects a traffic demand from a decreasing ordered list and an allocation is determined among the servers that have enough resources to host the source and destination VMs. After the allocation, the algorithm computes the path that causes the minimum increase of energy consumption. The objective is to guarantee the highest number of VM allocations and, consequently, of traffic demands satisfaction. The authors in [82] proposes a network power manager that is able to switch-off unused switches and/or links and to route traffic flows over the active networking elements. Anyway they do only focus on the networking power efficiency, without taking into account the possibility of VMs replacement enabled by live migration. The authors use heuristic to locally determine the power state of each switch and link. The paper [83] presents a formulation for the flow assignment problem in a data center network; besides an evaluation of the path consolidation algorithms, based on the characteristics of the topology, is shown. Given the total power consumption over the network elements in a time frame, the problem is formulated as a classical Multi-Commodity Flow Problem, with the aim of minimizing the consumption derived from the used switches. The paper [84] presents a network-aware power manager, which is able to optimize VMs placement, by also minimizing the total power consumption, deriving from the used switches and links needed to satisfy the traffic demands. The authors formulate a combinatorial optimization problem, in which they do not take into account different power models for the servers and switches nor the number of migrations in the objective function. Since the deriving problem is NP hard, they decompose it into three well-known problems: one is used for solving traffic aware VM grouping (Balanced Minimum K-Cut), the second is for the VM group to rack mapping (Quadratic Assignment Problem) and the third for VMs traffic flow routing (Multi-Commodity Flow Problem).

4.7.2 A Cross-Layer Power Efficient Resource Management Problem

The typical total data center power consumption cannot exclude networking devices, such as switches and routers. The proposed model for the consolidation problem is here extended in order to take management decisions within a virtualized data center

with the objective of achieving a trade-off between different objectives, such as the minimization of the used servers, the total network power consumption or the number of migrations. Each one of these objectives can be emphasised by using different weights in order to give more importance to one of them. The new problem also takes into account the network topology and the VMs traffic demands. The *Cross-Layer Power Efficient Resource Management Problem* is defined as it follows:

Given a set of physical servers J, each one packed in a specific rack, a subset of J indicating which servers are active (J^o_{act}) in the sense that they host at least one VM before the consolidation, the current power consumption of each server P^o_j, the current allocation of a set of m VMs to the physical nodes, a set of switches connected among them in a particular topology, along with their initial power consumption Ps^o_{act}, a set of gateways which are connected to a virtual sink representing Internet, the matrix of traffic demands, the connectivity matrix and the capacity one (expressed in Mbps), the objective is to find the set of migrations Mig, defining a new set of active nodes J^N_{act} and a new set of active switches S^N_{act}, that minimizes the linear combination of the power consumption of the resulting active servers and switches and the number of necessary migrations:

$$f = \alpha \cdot \frac{\sum_{j \in J^N_{act}} P^N_{act}}{\sum_{j \in J^o_{act}} P^o_{act}} + \beta \cdot \frac{|Mig|}{m} + \gamma \cdot \frac{\sum_{s \in S^N_{act}} Ps^N_{act}}{\sum_{s \in S^o_{act}} Ps^o_{act}} \tag{4.33}$$

by satisfying all the traffic demands of the VMs, by not exceeding the available resource amount at each server and by not overcoming the available capacity of the links.

A simple example topology where there are 5 switches, 2 gateways, 8 physical hosts, 2 racks and 14 VMs is depicted in Figure 4.5. In this simple topology each rack has only a bidirectional link towards one switch and each switch in one level is connected to all the other switches in the upper one. All the parameters of the problem are explained in table 4.5. The set of nodes V is just the union between the switches, the gateways, the racks and the virtual sink. This latter is considered as a "special" VM, the $(k+1)th$, which belongs by default to the $v-th$ node in the topology. The association of each VM to a node (a rack or the virtual sink) is defined through two matrix, namely X and W, where the first one defines which server the VM is allocated to, while the second indicates the topology node the server is associated to. The VMs traffic demands are expressed with a matrix D with dimension Kx$(K+1)$: each VM can forward traffic towards the virtual sink or another VM. Given a traffic demand, $d_{k,l}$, the source of the flow is the rack, in which the server hosting VM k is packed, while the destination can be

Parameters:

Input:

K	*set of VMs* $k = 1...K$
J	*set of physical servers* $j = 1...J$
J^o_{act}	*set of active servers before the consolidation* $\subseteq J$
X^o	*current allocation between VMs and active servers* KxJ^o_{act}
P^o_j	*power consumption of the active servers before the consolidation*
S	*set of switches* $s = 1...S$
S^o_{act}	*set of active switches before the consolidation* $\subseteq S$
P^o_s	*power consumption of active switches before the consolidation*
$P_{s,idle}$	*set of idle power consumption of the switches* $s = 1...S$
G	*set of gateways* $g = 1...G$
R	*set of racks* $r = 1...R$
t	*virtual sink representing Internet*
V	*set of nodes* $G \cup S \cup R \cup t$
W	*represents the allocation of each server to a node in the topology* $(J+1)xV$
C	*connectivity matrix* VxV
B	*links capacity matrix* VxV
D	*VMs traffic demands matrix* $Kx(K+1)$

Output:

X^N	*new allocation between VMs and servers after the consolidation* KxJ^N_{act}
J^N_{act}	*set of active servers after the consolidation* $\subseteq J$
P^N_j	*power consumption of the active servers after the consolidation*
S^N_{act}	*set of active switches after the consolidation* $\subseteq S$
P^N_s	*power consumption of the active switches after the consolidation*
$F^{k,l}_{v,u}$	*traffic flow between VM k and l on the link between node v and u*
$F^{tot}_{v,u}$	*total traffic flow between on the link between node v and u*
$H^{k,l}_{v,u}$	*is true if the link u-v is used for the traffic between VM k and l*
$H^{tot}_{v,u}$	*is true if the link u-v is active*

TABLE 4.5: Extended Model Parameters

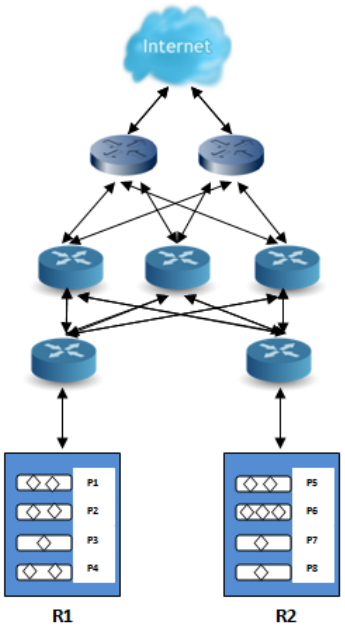


FIGURE 4.5: Topology

either the rack in which VM l is allocated or the virtual sink if $l = t$. Each link between two nodes is represented through a value 1 in the connectivity matrix C, which has a null diagonal and a dimension VxV, where V is the cardinality of the nodes set. A link is also associated with a maximum capacity defined by the matrix B. Each gateway is connected to the virtual sink, which correspond to the external Internet.

4.7.2.1 Switch Power Consumption

A switch is characterized by a chassis, namely the frame for mounting its circuit components and containing a certain number of line cards which provide the network interfaces. The power consumption of the switch [85] can be defined as:

$$P = P_{ch} + n_{cards} \cdot P_{cards} + \sum_{r=1}^{max} n_p^r \cdot P_p^r \cdot u_p \tag{4.34}$$

where P_{ch} is the chassis power consumption, n^{cards} is the number of line cards, P^{cards} is their the power consumption, n_p^{rate} is the number of ports operating at a specified rate, P_p^{rate} is their power consumption and, finally, u_p is the utilization of the port p. This

one can be defined as:

$$u_p = \frac{1}{TC_p} \cdot \int_t^{t+T} B_p(t)\,\mathrm{d}t. \tag{4.35}$$

where T is the observation time period, C_p is the capacity of the port and $B_p(t)$ is its instantaneous throughput at a determined rate. For the sake of simplicity, the power consumption of the switch is computed as:

$$P = P^{tot} + \sum_{port \in Q_{act}} P_{port} \tag{4.36}$$

where P^{tot} incorporates the first two addends of equation 4.34, P_{port} is the power consumption of each active port (Q_{act} is the set of active ports).

4.7.3 Problem Model

The model is characterized by the input and decision variables explained in Table 4.6, besides the ones of Table 4.1, while the overall model is shown in Table 4.7. The constraints of the consolidation problem defined in Section 4.5 are not reported for the sake of brevity, but they are also contemplated in this new model. The model aims at minimizing the linear combination of three parameters:

- the power consumption of the active servers after the consolidation over the initial one (the total power due to the current allocation);
- the number of migrations over the number of VMs;
- the power consumption of the active switches after the consolidation over the initial one;

The first constraint (4.38) simply computes the switch power consumption as explained in eq. (4.36). The constraints (4.39) is the so-called flow conservation constraint: by considering each node and by looking at the traffic demand between k_1 and k_2, if the node coincides with the rack which the VM k_1 is associated to, the difference between the entering flow and the exiting one should be equal to the opposite of the demand itself, since the traffic is generated at the source. In this case the difference between the two sums in the second term of (4.39) is equal to -1. For each intermediate node, this difference should be equal to zero, because all the flows that enter the node should

Input variables:	
$P_{idle,s}$	is the constant power consumption of the switch s due to the chassis and the line cards
$P_{s,v}$	is the constant power consumption of the link connecting the switch s to the node v
d_{k_1,k_2}	is the traffic demand in Mbps between VM k_1 and VM k_2 (or t)
c_{v_1,v_2}	is equal to 1 is there is a link between v_1 and v_2, 0 otherwise
b_{v_1,v_2}	is equal to the capacity (in Mbps) of the link between v_1 and v_2
$x_{j,k}$	is equal to 1 if the VM k is allocated to the server j, 0 otherwise
$w_{j,v}$	is equal to 1 if the server j is allocated to the node v, 0 otherwise
Decision variables:	
a_s	is equal to 1 if the switch s is active, 0 otherwise
$f_{v_1,v_2}^{k_1,k_2}$	is the flow amount of traffic (in Mbps) between VM k_1 and VM k_2 (or t) on the link connecting v_1 and v_2
F_{v_1,v_2}	is the total flow amount of traffic (in Mbps) on the link connecting v_1 and v_2
$h_{v_1,v_2}^{k_1,k_2}$	is equal to 1 if there is traffic (in Mbps) between VM k_1 and VM k_2 (or t) on the link connecting v_1 and v_2
H_{v_1,v_2}	is equal to 1 if the link connecting v_1 and v_2 is active (if there is any traffic passing through it)
$P_{final,s}$	is the final power consumption of the switch s

TABLE 4.6: Input and Output Variables

also be completely forwarded through it. For the intermediate nodes, the difference in the sums of second term of (4.39) is equal to 0. If the node is the destination of the flow, namely the rack which the VM k_2 belongs to or the virtual sink t, the difference in the entering and exiting flows should be equal to the amount of the traffic demand. In this third case, the difference in the sums of second term of (4.39) is equal to 1. The constraint (4.40) is the indivisible flow constraint: if the flow between k_1 and k_2 enters an intermediate node v_1, it has to be forwarded completely through one exiting

The MILP Model for the combined problem:

$$\min f = \alpha \cdot \frac{\sum_{j=1}^{n} y_j \cdot P_{y_j}}{\sum_{j=1}^{n} P_{init,y_j}} + \beta \cdot \frac{\sum_{j=1}^{n}\sum_{k=1}^{m} \frac{z_{jk}^{->} + z_{jk}^{<-}}{2}}{m} + \gamma \cdot \frac{\sum_{s=1}^{S} a_s \cdot P_{final,s}}{\sum_{s=1}^{S} P_{init,s}} \quad (4.37)$$

s.t.

$$P_{final,s} = P_{idle,s} + \sum_{v}(H_{s,v} \cdot P_{s,v}) \quad \forall s \quad (4.38)$$

$$\sum_{v_2} f_{v_2,v_1}^{k_1,k_2} \cdot c_{v_2,v_1} - \sum_{v_2} f_{v_1,v_2}^{k_1,k_2} \cdot c_{v_1,v_2} = d_{k_1,k_2} \cdot (\sum_{j} x_{j,k_2} \cdot w_{j,v_1} - \sum_{j} x_{j,k_1} \cdot w_{j,v_1}) \quad \forall k_1, k_2 \quad \forall v_1 \quad (4.39)$$

$$\sum_{v_2} h_{v_2,v_1}^{k_1,k_2} - \sum_{v_2} h_{v_1,v_2}^{k_1,k_2} = 0 \quad \forall v1 : v1 = intermediate \quad \forall k_1, k_2 \quad (4.40)$$

$$f_{v_1,v_2}^{k_1,k_2} \leq b_{v_1,v_2} \cdot h_{v_1,v_2}^{k_1,k_2} \quad \forall v_1, v_2 \quad \forall k_1, k_2 \quad (4.41)$$

$$f_{v_1,v_2}^{k_1,k_2} \geq 0 \quad \forall v_1, v_2 \quad \forall k_1, k_2 \quad (4.42)$$

$$h_{v_1,v_2}^{k_1,k_2} \leq f_{v_1,v_2}^{k_1,k_2} \cdot c_{v_1,v_2} \quad \forall v1, v2 \quad \forall k1, k2 \quad (4.43)$$

$$F_{v_1,v_2} = \sum_{k_1,k_2} (f_{v_1,v_2}^{k_1,k_2} + f_{v_2,v_1}^{k_1,k_2}) \quad \forall v_1, v_2 \quad (4.44)$$

$$F_{v_1,v_2} \leq b_{v_1,v_2} \quad \forall v_1, v_2 \quad (4.45)$$

$$H_{v_1,v_2} \leq \sum_{k_1,k_2} h_{v_1,v_2}^{k_1,k_2} \quad \forall v_1, v_2 \quad (4.46)$$

$$H_{v_1,v_2} \geq h_{v_1,v_2}^{k_1,k_2} \quad \forall v_1, v_2 \quad \forall k_1, k_2 \quad (4.47)$$

$$a_s \leq \sum_{v2} H_{s,v2} + \sum_{v1} H_{v1,s} \quad \forall s \quad (4.48)$$

$$a_s \geq H_{v1,s} \quad \forall s \quad \forall v1 \quad (4.49)$$

$$a_s \geq H_{s,v2} \quad \forall s \quad \forall v2 \quad (4.50)$$

TABLE 4.7: Combined Problem Model

link. This means that the traffic flows cannot be split in the network. This is true for the source and destination nodes (both have one connecting link). The inequalities in (4.41) indicate that if a link is not used for transporting the flow between k_1 and k_2, the variable $f_{v_1,v_2}^{k_1,k_2}$, should be zero. If the link is used for forwarding the flow between k_1 and k_2, the capacity of that link should not be exceeded. Besides every flow on each link should be not negative (4.42). If the flow of two VMs on a specific link is equal to zero or there is no connection between the nodes, $h_{v_1,v_2}^{k_1,k_2}$ should be equal to zero. This is expressed by the constraint in (4.43). The equalities in (4.44) simply computes for each link the total flow which is forwarded along it. The inequalities in (4.45) represent the network budget constraint: for each link between two nodes, by summing all the flows passing through it, the resulting amount of traffic should not exceed the total link capacity. An active link is expressed through the constraint (4.46-47): if there is at least one flow forwarded through the link, it should be active; otherwise if there is no traffic on that link, it should be considered as idle. The same kind of constraint is also applied to the switches (4.48-49-50): if all of its links are idle, the switch can be shut off; on the contrary, if one of the ports is active, the switch should be active.

4.7.4 The Combined Heuristic

Given the current allocation of the VMs to the servers, the initial solution for the flow-routing problem is obtained through a modified version of the Dijkstra algorithm, whose pseudo-code is shown in 6. Let us consider a list of nodes, a traffic flow between a source and a destination, the links capacity matrix, the cost of each link and the connectivity matrix: it is needed to compute a minimum cost path between the source and the destination nodes. Each link between a switch s and v is characterized by a cost which can be linked to the power consumption as:

$$cost(s, v) = P(s) + P(v) + P(s, v) \tag{4.51}$$

Therefore the cost for using that link is the cost for activating the switch s and v (their idle power consumption) and the power consumed by sending traffic over the link s-v. At the end of the algorithm, very node i in the topology will be associated with two labels:

- $f(i)$, which indicates the total cost of the path, or in other words, is the sum of the costs of the links on the path to node i;
- $J(i)$, which denotes the node that precedes i in the minimum cost path.

Algorithm 6 Cost-Aware Minimum Path

```
Input: source, destination, flow, nodes_list, conn_matrix, costs_matrix, capacity_matrix
Output: path
T = all the nodes ∈ nodes_list
S = ∅
for i=0 to size of T do
    node = T.get(i)
    if node == source then
        f.put(source, 0)
        prec.put(source, 0);
    else
        if source and node are connected and there is enough capacity between them
          to accommodate flow then
            f.put(node, costs[source][node])
            pred.put(node,source)
        else
            f.put(node, ∞);
        end if
    end if
end for
for i ∈ T do
    Find j in T such as f(j) = min f(i) and i ∈ T
    Remove j from T
    Add j in S
    for k=0 to size of T do
        current = T.get(k)
        if j and current are connected AND the capacity between them is enough to
          accommodate flow AND f.get(curr) ≥ f.get(j) + costs[j][current] then
            f.put(current, f.get(j)+costs[j][current]);
            pred.put(current, j);
        end if
    end for
end for
path = Reconstruct_path(source, destination, prec)
```

There are two different sets, S and T, that contains the nodes that were labelled and the ones without labels respectively. The set T is initially set all the nodes in the list, while S is the empty set. For every node belonging to T, if the current one coincides with the source, the label representing the cost to reach it is set to zero, like its predecessor. Otherwise if the source and the current node are connected and there is enough capacity

to accommodate the flow, the predecessor of the node becomes the source and the f label is just the cost between the source and that node. If the previous cases do not hold, the current node cannot be reached from the source, and the cost is set to infinite. Then for each node in T, the algorithm finds the one holding the minimum value of f label: this node, j, is removed from T and added to S. Given the node j, for each remaining one in T, if j and the current one (k) are connected, the capacity is sufficient to forward the flow and the cost to get to k is greater than the cost to reach j in addition to the cost between j and k, then the cost and predecessor labels of k are updated. At the end, the algorithm simply reconstructs the total path, by looking at the predecessor of each node.

The heuristic uses the cost-aware minimum path algorithm to identify the best power consuming paths for all the traffic demands. The algorithm, then, evaluates the desirability of each possible VM migration. When considering a migration, it is needed to verify if the re-positioning causes the VM to move from one rack to another. In that case, indeed, the paths for the traffic flows of the migrating VM must be updated. The desirability takes into account a value which represents a reconfiguration cost, in order to encourage not only the migrations which result in better resource utilization and power consumption, but also the ones have a minimum disruption over the paths that were established through the Dijkstra algorithm. The cost is set to the inverse of the number of flows the VM exchanges with other VMs that need a path re-computation. Instead, if the VM migrates from one server to another within the same rack, none of its established paths needs to be recalculated: hence, the cost is set to zero. The migration with the highest value of desirability is triggered: if this re-positioning cause the update of some paths, all the resources held by the old path are released and the cost-aware minimum path algorithm is used again. The impact on the total network consumption is part of the objective function: the migration is accepted with the same probability law that was introduced in Section 4.6.2; if it is refused the original path is restored.

4.8 Experimental Evaluation

4.8.1 Parameters

The consolidation algorithms can be evaluated by considering different parameters. Since the objective is to minimize the number of used server, they can be compared in terms of active servers or, equally, the number of consolidated servers, that can be expressed as it follows:

$$cons\_servers = initiat\_active\_servers - new\_active\_servers \tag{4.52}$$

Besides the other parameters that can be taken into account are the number of migrations needed to consolidate the servers, the total power consumption due to the active servers after the consolidation and the total network power consumption. In [70] a metric was also introduced for evaluating how effective the migrations are: this is the *migration efficiency*, which can be defined as:

$$me = \frac{cons\_servers}{migs} \cdot 100 \tag{4.53}$$

This value represents how much every single VM migration contributes in percentage to the objective of releasing one node. For example, a migration efficiency of 100% indicates that every single migration leads to one hibernated node; instead an efficiency of 30% means that each migration is useful to consolidate a 30% of a single server.
Another discriminating parameter for the comparison is the *packing efficiency*, which is a measure of how much the algorithm was able to consolidate the servers and it is defined as it follows:

$$pe = \frac{cons\_servers}{initial\_active\_servers} \tag{4.54}$$

Another group of indicators is related to the resources utilization. It is possible to compute the average CPU utilization, which is the ratio of used CPU to the total available amount, the average RAM utilization, namely the ratio of used RAM to the total available one and the average VMs per server.

4.8.2 Algorithms

4.8.2.1 First Fit Decreasing Consolidation

This algorithm [70] is an adaptation of the first-fit decreasing policy which is applied in order to compute the list of migrations needed to consolidate as many servers as possible. The algorithm is based on a simple idea: every VM is associated to a value representing the linear combination of the resources which are needed by the virtual appliance, compared to the total available ones. This value represents how much amount of resources the VM is requesting. The algorithm computes a value, namely α, which is the ratio between the total amount of CPU which is requested by all the VMs and the sum between the overall requested CPU and RAM, as it follows:

$$\alpha = \frac{\sum_{k \in K} r_k^{cpu}}{\sum_{k \in K} (r_k^{cpu} + r_k^{mem})} \tag{4.55}$$

where r_k^{cpu} and r_k^{ram} are respectively the CPU and the RAM required by the VM k. The *score* of a VM, that is a value representing the quantity of required resources, is calculated as it follows:

$$score_k = \alpha \cdot r_k^{cpu} + (1 - \alpha) \cdot r_k^{cpu} \tag{4.56}$$

The list of VMs is ordered by descending score, in order to consider more expensive VMs at the beginning. Then the algorithm checks if each VM can be migrated to each physical server: if the migration is possible, it is performed and another VM is taken from the list. The algorithm computes the number of used servers, the number of migrations and the released nodes.

4.8.2.2 Sercon

The idea behind Sercon [70] is to take into account a group of migrations if, and only if, they lead to a server consolidation, otherwise they are rejected. The algorithm uses the same concepts introduced in previous section: both the VMs and the physical nodes are ordered by using the score, which represents how much a server is loaded, or how many resources a VM needs. After obtaining the scores, the nodes are ordered by decreasing values and the least loaded one is chosen as a candidate for the consolidation. Given

this node, for each of its allocated VMs, the score is also evaluated: the VMs are then ordered, so that the first one is the most expensive in terms of requesting resources. The algorithm tries to reallocate every VMs on the selected node, with the objective of hibernating it. If each VM can be migrated away from the node, the list of migrations is updated accordingly; otherwise if any of the migrations is not possible, all the VM moves are discarded and the subsequent node in the list is taken into consideration as a new candidate for the consolidation attempt. If the node was consolidated the variable indicating the unsuccessful attempts is set to zero; otherwise it is incremented by one. The algorithm stops if the unsuccessful attempts are greater than the maximum number which is given as input or if the migration efficiency is greater than an input threshold.

4.8.3 Results

The consolidation algorithm was implemented in Java, by creating a simulator for the data center resources and it was compared with First Fit Decreasing Consolidation and Sercon. The values for CPU, RAM and power models of the servers are randomly taken from the sets of values specified in tables 4.8 and 4.9, for servers and VMs respectively.

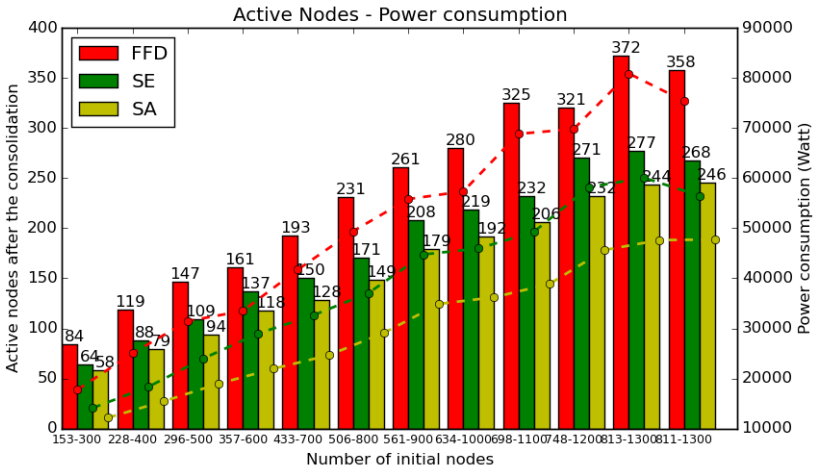
TABLE 4.8: Servers Experimental Values

CPU cores	4, 6, 8, 12, 16, 18, 20, 24, 32
RAM (GB)	2, 4, 6, 8, 10, 12, 16, 20, 32
Power (idle %)	0.2, 0.3, 0.4
Power (max W)	160, 180, 190, 220, 240, 260, 270, 280, 290

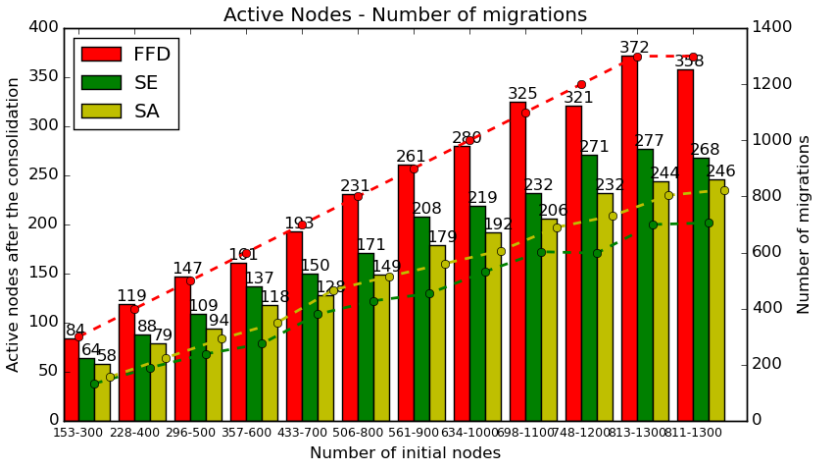
TABLE 4.9: VMs Experimental Values

CPU cores	1, 2, 3, 4, 5, 6, 7, 8
RAM (GB)	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Given the number of physical servers and VMs, the initial allocation is computed through an algorithm that selects for every VM a random node: if it has enough resources, the VM gets allocated on it. This is why, by considering an initial number of servers, only a subset of them will be actually active. For active nodes, it is meant servers hosting at least one VM. In Figure 4.6(a), the x axis shows the number of active nodes before



(A) Active Nodes - Power Consumption



(B) Active Nodes - Migrations

FIGURE 4.6: Results

the consolidation and the number of VMs. There are two y axes: the one on the left represents the number of active nodes after the consolidation (the bars), while the one on the right shows the overall achieved power consumption (the dotted lines). Figure 4.6(b), instead, shows on the right y axis the number of migrations needed to perform the consolidation. It turns out that the proposed heuristic is able to always guarantee the minimum number of active servers and also the lowest overall power consumption. More in details, the proposed algorithm is able to obtain a reduction between 27% and 37% in the number of active nodes, and between 31% and 44% in the power consumption, in comparison to FFD. Instead, by comparing it with Sercon, a reduction in a range of 9%-17% for the number of active nodes and of 14%-24% for the power consumption was obtained. Some experiments were also performed with other allocation algorithms used for creating a mapping scheme, namely best-fit and first-fit. Figure 4.7 represents

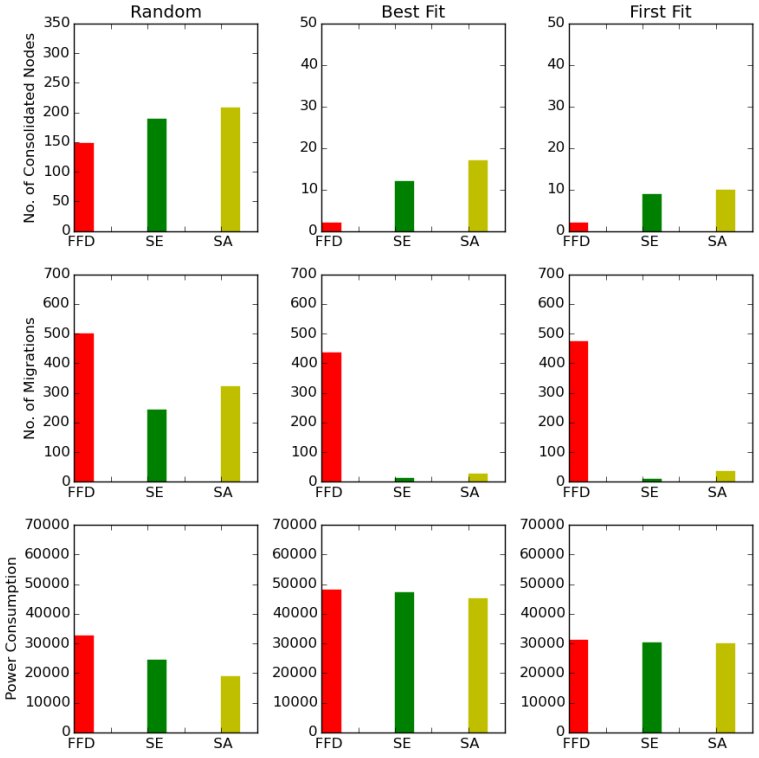


FIGURE 4.7: Results By Allocation Policies

in each column the number of consolidated nodes, the number of migrations and the power consumption achieved by running the consolidation algorithms for each of the cited allocation policies (random, best and first). For the experiment a number of 400 initial used servers was used (302 active before the consolidation for random-fit, 249 for best-fit and 149 for first-fit) and 500 VMs. The graphs in the first row show the number of consolidated nodes: by applying the best-fit and first-fit policies, the consolidated nodes are fewer than the number achieved with the random allocation, but the proposed algorithm still outperforms the other two. Regarding the migration efficiency Sercon achieves values in a range between 67% and 78%, FFD between 23% and 36%, while the proposed algorithm between 61% and 73%. The values achieved by Sercon are slightly higher due to the fact that, by average, it produces less migrations because it does not consider the server's power consumption. The packing efficiency is better (range between 1.64 and 2.40) for the proposed algorithm because it can pack a bigger number of VMs in a smaller set of servers. Sercon, instead, achieves a packing efficiency in a range between 1.39 and 2.01, while FFD between 0.82 and 2.35.

The VM consolidation problem was also solved through IBM CPLEX [86], in order to quantify the distance from the optimum solution. By considering one instance of the

problem I, the quality of the solution produced by the heuristic is computed as:

$$gap = \frac{|OPT(I) - HEU(I)|}{OPT(I)} \cdot 100 \tag{4.57}$$

where $OPT(I)$ is the optimum solution for the instance and $HEU(I)$ is the one obtained through the heuristic algorithm.

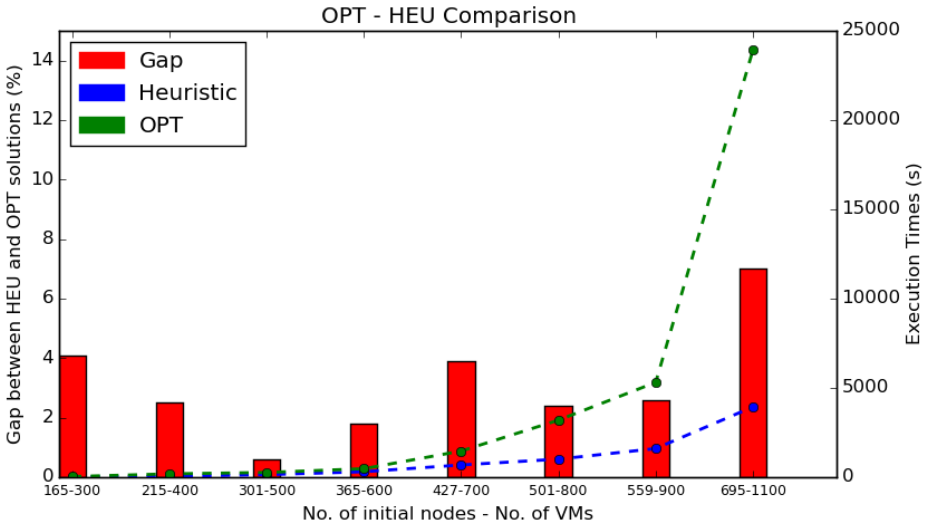


FIGURE 4.8: Times Comparison

In order to limit the processing time of the CPLEX optimizer, a relative tolerance of 0,01% was employed by using two CPLEX parameters, namely *epagap* and *epgap*. They are used to instruct the optimizer to stop the execution as soon as it has found a feasible integer solution proved to be within one percent of the optimal one. In Figure 4.8, a comparison in terms of gap (expressed in percentage) between the value of the objective function obtained by the heuristic and the optimal one is showed. Besides in the same graph also the execution times are considered. In the figure, there are two axes: the one on the left denotes the gap, whose values are represented through bars, and on the right axis the execution times are showed. From the figure, it is possible to underline how CPLEX execution times begin to exponentially explode from the eighth x-coordinate. The values shows that the worsening of the heuristic objective function values are included in a range between 0.6% and 7%, while the execution time reduction is within 47,13% and 83,53%. The exact results (for a limited data set) in terms of the values of the objective function, the active servers after the consolidation, the number of VMs migrations and the total servers power consumption are shown in Table 4.10. The consolidation algorithm was also extended in order to solve the problem that was

TABLE 4.10: Optimal Solution Comparison

init PHs - active PHs - VMs	OPT active	SA active	OPT Migs	SA Migs	OPT Power	SA Power
300-216-400	53	54	221	217	10542.75W	10659.95W
400-286-500	68	74	288	284	13362.57W	13821.92W
500-344-600	89	99	348	331	17712.99W	18590.54W
600-419-700	102	109	424	416	21034.57W	21751.23W
700-478-800	120	133	498	470	23661.33W	24587.97W

defined in section 4.7. In Figure 4.9, the original consolidation algorithm is compared with the cross-layer heuristic. The network topology is characterized by the number of hierarchy levels, the number of switches at each level and the number of racks. The set of parameters that where considered is:

- $cooling\_factor = 0.9999$ and $initial\_temp = 0.01$;
- $num\_servers = 150$ and $num\_VMs = 300$;
- $num\_nodes\_level = \{15, 10, 6, 6, 3, 1\}$, $levels = 6$ and $racks = 15$;

In the graphs, the number of active nodes, the number of migrations and the total server consumption are compared for the basic consolidation algorithm (that does not takes into account the network power consumption), by using $\alpha = 0.9$ and $\beta = 0.1$ and the combined heuristic, by setting:

- $\alpha = 0.05$ and $\beta = 0.05$ and $\gamma = 0.90$
- $\alpha = 0.90$ and $\beta = 0.05$ and $\gamma = 0.05$

In the first case γ is set to 0.90, therefore in the objective function the network power consumption assumes more importance over the other two objectives. The second set of parameters is characterized by a higher value of α, in order to promote the minimization of the servers power consumption.

By considering the number of active servers, the top-left graph shows that the original

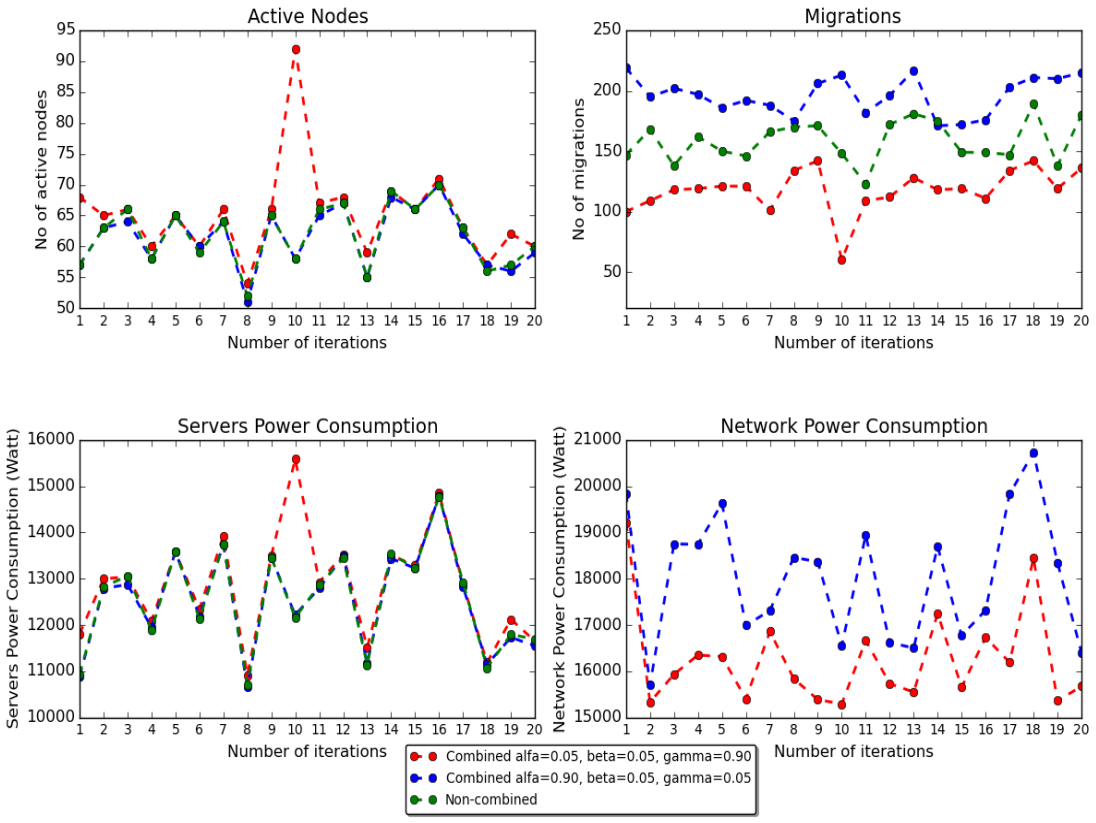


FIGURE 4.9: Algorithms Comparison

consolidation algorithm and the cross-layer heuristic that gives more importance to the server consolidation achieve quite the same results, as proved by the fact that the lines almost overlap. They also have a mean value of 61 active servers after the consolidation, while the cross-layer algorithm with a higher γ value has a mean of 65 active nodes. These results reflect also in the bottom-left graph, which is related to the servers' total consumption: the lowest mean power consumption is achieved by the original consolidation algorithm. With regard to the number of migrations, the algorithm that always achieves the minimum number is the cross-layer heuristic with a higher γ value. The bottom-right graph is related to the network power consumption: in this case only the two different versions of the cross-layer algorithm are compared because the original one does not take into account the network topology. As expected, when the weight associated to the network power consumption (γ) is higher, the algorithm assures the minimum power consumption. The mean values for the three cases are shown in Table 4.11. By looking at the mean results, despite a 7% worsening in the number of active servers compared to the other two cases, the cross-layer algorithm with a higher γ value

achieves a better result in the number of migrations (26% fewer migrations in comparison to the original algorithm, and 40% if considering the other case). Besides it also assures a 10% reduction in the network power consumption.

TABLE 4.11: Mean Values

	Average active servers	**Average migrations**	**Average servers power**	**Average net power**
Non Combined	61	158	12547W	-
Combined $\alpha = 0.05$	65	117	12861W	16260W
Combined $\alpha = 0.90$	61	196	12527W	18026W

Chapter 5

Conclusions

Computing paradigms have been experiencing several changes over the past four decades: each computing paradigm revolution was the reason behind new economies of scale. For example, *Personal Computing* gave the chance to avoid the delay in the access to a single workstation and to make transactions parallel, while *Network Computing* increased the number of users able to exploit distributed computing resources. The increasing complex nature of scientific problems was the reason for the need of a *cluster middleware* capable of hiding the heterogeneity of the hardware resources, in order to use the available interconnected computing resources as if they were one single centralized resource. An additional computing evolution was guided by the massive geographical distribution of very powerful computing resources. Here, the focus is on the software layers that allow a coordinated mechanism for resource sharing and collaborative problem-solving. This is the inspiration behind the birth of *Grid Computing*. Modern computing paradigms are extremely linked to the *Utility Computing*: the underlying concept is a service provisioning model that allows users to exploit a computing resource when needed and according to the specified requirements, with the objective of maximizing the utilization and bringing down the costs. In particular, *Cloud Computing* is able to provide the customer with a generic service which is offered through an universal access, in a scalable and elastic way and in a pay-as-you-go basis. Cloud explosion has been also fostered by the success of the virtualization techniques that enable the consolidation of the system's utilization, the reduction of the computing power, electricity charges and space costs in data centers and a heterogeneous execution environment. New technologies are also distinguished by a growing *green awareness*. The power consumption of a typical data center is becoming

more and more a concern in terms of a double-face motivation: the first one is related to an environmental problem and coincides with the arising CO_2 emissions; the second one is related to the costs a company needs to face for the electricity consumption. The combination of the cloud typical elastic resources provisioning and the energy efficiency encouraged by *Green Computing* is explored in lots of new mechanisms and techniques developed with the aim of providing the system with a lower power consumption.

Next to these recent evolutions, critical business services are characterized by an increasing digitalization, together with higher expected levels of quality and unbearable costs for unavailability and downtime. Critical Information Infrastructures (CIIs) are fundamental in many modern human activities such as communications, health, tourism and finance: they rely on computer-based systems to exchange, aggregate and extract significant amounts of sensitive data. Mission critical systems are no longer isolated and they demand for flexible, agile and low-cost platforms to deliver their services, satisfy customer's experience and improve cyber-security. In fact, the political and economic relevance of these infrastructures makes them particularly vulnerable to failures due to intentional failures, such as malicious events or terrorist attacks. The application of a cutting-edge technology, like Cloud Computing, to complex and mission critical infrastructures can lead to lots of operational benefits, even if there is still some reluctance in its adoption in critical industry. On one hand, a private cloud based solution can be employed, for example, to reproduce real world scenarios with the aim of performing distributed testing campaigns, or to create automatic procedures for backup mechanisms. On the other, the problems that arise with the virtualization of these systems, demanding for very strict requirements, is that there is no enough guarantee of the same dependability levels which can be achieved in the real operational environments. Another issue of the cloud paradigm is that it is difficult to assure a required level of availability: this is basically due to the lack of validation techniques for cloud services and infrastructure assessment. For example this is true when considering the fact that virtualization adds a layer which can be prone to different failure modes. Also data store consistency assume a central role if you consider a cloud infrastructure that is shared among different data centers and there is a failure in the master one. These directions probably represent the research efforts that will give a certain answer to the applicability of the cloud technology to such sensitive infrastructures. Besides, the virtualization layer may cause new issues, related to other possible attack surfaces. These threats potentially make the risk of providing potential attackers with the access to sensitive

data very probable. The nature of mission and safety critical applications really stresses out the lack of robust security solutions and the need for automatic recovery strategies. This thesis investigated the two described directions, by adopting a private cloud solution for the implementation of a mission critical infrastructure and by leveraging virtualization techniques for designing power efficient resources management strategies. The case study that was considered deals with a real world Area Control Center, an Air Traffic Control system, for which missing dependability and security targets would result in human and financial losses. The center represents the facility responsible for the control service of an airspace volume which extends from the surface of the airport to a specified upper limit. The investigation started from the practical installation, experimentation and configuration of different cloud solutions, with the aim of building a suitable Extended Private Enterprise Cloud platform for the considered case scenario. The first objective was the identification of application-specific requirements for the cloud platform: they consist in the reproducibility of real hardware configurations, a minimum impact on the applications and a low resources provisioning time. These requirements assume a great importance in the realization of fast fail-over techniques by guaranteeing a minimum service disruption. Cloud services can be ranked according to different key performance indicators: by considering the nature of critical services and the need for recovery mechanisms, two parameters useful for measuring the elasticity attribute were taken into consideration. This property can be interpreted as the ability of a service to adapt itself to the changes and the requirements issued by the applicant. The evaluation was based on the *provisioning time* and the *scheduling time*: the first one is the time needed to activate or deactivate a generic resource. More in details, if the resource consists in a VM, it can be defined as the time interval needed by the platform to spawn and make a VM available to the user. The second one is the time needed to make a decision on where to allocate a new VM and it can be considered as one of the parameters of the provisioning time. The assessment was restricted to two open-source cloud projects, namely OpenStack and OpenNebula, which were the best responding ones to the specified requirements. After having identified the cases in which the creation of a VM in one platform is comparable to the other, the impact of the different parameters involved in the definition of a VM template on the provisioning time was evaluated. Besides an experimental campaign for comparing the two platforms was carried out. The results show that, despite the great success of OpenStack, OpenNebula offer slightly

better values in terms of provisioning time. In the evaluation, another aim was to identify the significant contributions and their percentage impact on the provisioning time: it turns out that OpenStack has a non-negligible overhead due to the communication between the various services. Besides the management interface with the hypervisor drivers and with the virtual network module has the greatest impact. The results on the scheduling times show, instead, that the time related to the choice of a physical host has a little percentage impact for both the platforms. OpenNebula exhibits a slightly higher scheduling time because of a greater complexity in the implemented scheduling algorithm. The selection of the platform was the starting point for the implementation of a multi-layers enhancing security architecture. The idea was to provide this architecture with agile mitigation strategies in case of attacks, by exploiting from one side the virtualization potentialities, such as the Live Migration of virtual resources, and on the other one, the dynamic reconfiguration, made possible thanks to the Software Defined Networking approach. After identified the components of the architecture, a vulnerability assessment of the real ATM scenario was performed as a proof of concept, and a well-known risk evaluation methodology was applied on the ATM system's operational assets. This was a preliminary step to build a real threat scenario, which laverages the identified vulnerabilities and was used as a trigger for the realized mitigation strategy.

VM Live Migration was also exploited to extend of the so-called *VMs Consolidation Problem*, which is used to achieve energy consumption savings in virtualized environments. Since VMs are created and destroyed on demand and the resources they request may change over time might, the resources utilization can be very inefficient especially from the power consumption viewpoint. It is convenient to consolidate the infrastructure in order to keep the allocation of VMs up-to-date and to minimize the number of active servers, by using the live migration mechanism. Two models for obtaining a consolidation decision that achieves the minimization of the linear combination of the consumption of the used servers, the number of migrations and the network power consumption are proposed, by using the Mixed Integer Linear Programming. The problems are faced by using a Simulated Annealing based heuristic, which is based on a very simple idea: since the aim is to have a balance between the consolidation of the nodes and the overall power consumption, in order to evaluate the possible migrations, a value that is defined as the "**desirability**" of migrating a VM from a source node to a destination one is taken into account. This value consists in the arithmetic mean of the

convenience of migrating the VM from the source and the willingness of the destination one of accepting the VM, plus the difference in the power consumption after the migration of the VM to the new node. The algorithm was implemented, by creating a data center resource simulator and its evaluation compared to other algorithms found in the literature.

Acknowledgements

The conclusion of a three-years PhD is an achievement characterized by proud and joy. The potential of life often lies in the chance we are given to prove ourselves that we are able to defeat our weaknesses and to grow as person, as worker, as world citizen. The PhD was an amazing occasion to learn methods, techniques and models used in the scientific research but also to apply them in industry real use cases. I was blessed with the opportunity of meeting people from all over the world, share ideas, get in touch with very different minds: this was very inspiring to my personal growth. For this very important goal, I want to truly thank my advisor, prof. Stefano Avallone, who believed in me, by offering me the chance to leverage this very professional enhancing experience. His competence, technical and personal advices have meant a lot to me and have helped me in the scientific results I have been able to achieve during the last three years. His passion to work was very stimulating and it was a source of new stimulus to push my limits. I want to also thank my co-advisor, prof. Giorgio Ventre, which was a solid reference point during all the PhD period. I also want to thank three people I had the chance to work with. A thank is dedicated to Vittorio Manetti, which I personally admire for his passion in research, his clarity and his availability. His very inspiring ideas were a great helping hand for my activities and his advices were always very wise. I also want to thank Luigi Battaglia, who was the person which I learned more from in terms of technical stuff: he has been always present for helping me and I have been so lucky to work with a very prepared and experienced person. The last (but not least) thank goes to Gabriella Carrozza: her deep dedication and passion for work should be a model for every person. She is not afraid of taking new risks and challenges and I want to thank her for her creativity: it was a root for new ideas and new research directions. Obviously a truly thank is also devoted to my parents and my sister: all their sacrifices gave me the opportunity of reaching this very important title.

Bibliography

[1] A.A. Semnanian, J. Pham, B. Englert, and Xiaolong Wu. Virtualization technology and its impact on computer hardware architecture. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 719–724, April 2011. doi: 10.1109/ITNG.2011.127.

[2] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974. ISSN 0001-0782. doi: 10.1145/361011.361073. URL http://doi.acm.org/10.1145/361011.361073.

[3] Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, Rasit Eskicioglu, Peter Graham, and Frank Sommers. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies, chapter 16*, pages 521–551, 2006.

[4] D. Provan. Transmitting IP traffic over ARCNET networks. RFC 1201 (Standard), February 1991. URL http://www.ietf.org/rfc/rfc1201.txt.

[5] O.C. Ibe, R.C. Howe, and K.S. Trivedi. Approximate availability analysis of vax-cluster systems. *Reliability, IEEE Transactions on*, 38(1):146–152, Apr 1989. ISSN 0018-9529. doi: 10.1109/24.24588.

[6] I. Foster. The anatomy of the grid: enabling scalable virtual organizations. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 6–7, 2001. doi: 10.1109/CCGRID.2001.923162.

[7] Ian T. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol.*, 21(4):513–520, 2006.

[8] Globus toolkit version 4 grid security infrastructure: A standards perspective the globus security team 1, 2004.

[9] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[10] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[11] Nick Antonopoulos and Lee Gillam. *Cloud Computing: Principles, Systems and Applications*. Springer Publishing Company, Incorporated, 1st edition, 2010. ISBN 1849962405, 9781849962407.

[12] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. CreateSpace Independent Publishing Platform, USA, 2012. ISBN 1478168021, 9781478168027.

[13] Rohit Bhadauria and Sugata Sanyal. Article: Survey on security issues in cloud computing and associated mitigation techniques. *International Journal of Computer Applications*, 47(18):47–66, June 2012. Full text available.

[14] Cloud Security Alliance. Top threats to cloud computing v1.0, March 2010.

[15] G.S. Bindra, P.K. Singh, K.K. Kandwal, and S. Khanna. Cloud security: Analysis and risk management of vm images. In *Information and Automation (ICIA), 2012 International Conference on*, pages 646–651, June 2012. doi: 10.1109/ICInfA.2012.6246757.

[16] Flavio Lombardi and Roberto Di Pietro. Secure virtualization for cloud computing. *J. Netw. Comput. Appl.*, 34(4):1113–1122, July 2011. ISSN 1084-8045. doi: 10.1016/j.jnca.2010.06.008. URL http://dx.doi.org/10.1016/j.jnca.2010.06.008.

[17] U.S. Environmental Protection Agency. Report to congress on server and data center energy efficiency public law 109-431, 2007.

[18] Qilin Li and Mingtian Zhou. The survey and future evolution of green computing. In *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*, pages 230–233, Aug 2011. doi: 10.1109/GreenCom.2011.47.

[19] E. Jaureguialzo. Pue: The green grid metric for evaluating the energy efficiency in dc (data center). measurement method using the power demand. In *Telecommunications Energy Conference (INTELEC), 2011 IEEE 33rd International*, pages 1–8, Oct 2011. doi: 10.1109/INTLEC.2011.6099718.

[20] The Green Grid. A framework for data center energy productivity, .

[21] The Green Grid. Carbon usage effectiveness (cue): A green grid data center sustainability metric, .

[22] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: Managing energy as a first class operating system resource. *SIGARCH Comput. Archit. News*, 30(5):123–132, October 2002. ISSN 0163-5964. doi: 10.1145/635506.605411. URL http://doi.acm.org/10.1145/635506.605411.

[23] R. Bolla, R. Bruschi, Franco Davoli, and F. Cucchietti. Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys Tutorials, IEEE*, 13(2):223–244, Second 2011. ISSN 1553-877X. doi: 10.1109/SURV.2011.071410.00073.

[24] Avi Kivity. kvm: the Linux virtual machine monitor. In *OLS '07: The 2007 Ottawa Linux Symposium*, pages 225–230, July 2007.

[25] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. e.a.: Extending networking into the virtualization layer. In *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009.*

[26] P. Sempolinski and D. Thain. A comparison and critique of eucalyptus, opennebula and nimbus. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 417–426, 2010. doi: 10.1109/CloudCom.2010.42.

[27] Eucaliptus web site. URL http://www.eucalyptus.com/.

[28] Nimbus web site. URL http://www.nimbusproject.org/.

[29] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann. What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 484–491, 2011. doi: 10.1109/CLOUD.2011.80.

[30] Y. Ueda and T. Nakatani. Performance variations of two open-source cloud platforms. In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pages 1–10, 2010. doi: 10.1109/IISWC.2010.5650280.

[31] Wikipedia web site. URL http://en.wikipedia.org/wiki/Main_Page.

[32] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. How is the weather tomorrow?: towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, DBTest '09, pages 9:1–9:6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-706-6. doi: 10.1145/1594156.1594168.

[33] Ming Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430, 2012. doi: 10.1109/CLOUD.2012.103.

[34] Rackspace web site. URL http://www.rackspace.com/.

[35] Amazon ec2 web site. URL http://aws.amazon.com/ec2/.

[36] Microsoft azure web site. URL http://www.windowsazure.com/.

[37] P. Smith L. Roderick E. Zamost V. Makhija, B. Herndon and J. Anderson. VMmark: A scalable benchmark for virtualized systems. Technical report, VMware Inc, CA, September 2006.

[38] Wmware virtualization web site. URL http://www.vmware.com/.

[39] Jeffrey P. Casazza, Michael Greenfield, and Kan Shi. Redefining server performance characterization for virtualization benchmarking. *Intel Technology Journal*, 10(3): 243–251, August 2006. ISSN 1535-766X. doi: http://dx.doi.org/10.1535/itj.1003.

[40] S. Aiken, D. Grunwald, A.R. Pleszkun, and J. Willeke. A performance analysis of the iscsi protocol. In *Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 123–134, April 2003. doi: 10.1109/MASS.2003.1194849.

[41] S.K. Garg, S. Versteeg, and R. Buyya. Smicloud: A framework for comparing and ranking cloud services. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 210–218, 2011. doi: 10.1109/UCC.2011.36.

[42] The cloud service measurement index consortium. URL http://csmic.org/.

[43] Stress utility repository for centos. URL http://www.stresslinux.org/sl/.

[44] Commission implementing regulation (eu) no 1035/2011, official journal of the european union, 2011.

[45] Stefano Avallone, V. Manetti, M. Mariano, and S.P. Romano. A splitting infrastructure for load balancing and security in an mpls network. In *Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007. 3rd International Conference on*, pages 1–6, May 2007. doi: 10.1109/TRIDENTCOM.2007.4444681.

[46] SyedAkbar Mehdi, Junaid Khalid, and SyedAli Khayam. Revisiting traffic anomaly detection using software defined networking. In Robin Sommer, Davide Balzarotti, and Gregor Maier, editors, *Recent Advances in Intrusion Detection*, volume 6961

of *Lecture Notes in Computer Science*, pages 161–180. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23643-3. doi: 10.1007/978-3-642-23644-0_9. URL http://dx.doi.org/10.1007/978-3-642-23644-0_9.

[47] Nox web site. URL http://www.noxrepo.org/.

[48] R. Braga, E. Mota, and A. Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415, Oct 2010. doi: 10.1109/LCN.2010.5735752.

[49] Kai Wang, Yaxuan Qi, Baohua Yang, Yibo Xue, and Jun Li. Livesec: Towards effective security management in large-scale production networks. In *Proceedings of the 2012 32Nd International Conference on Distributed Computing Systems Workshops*, ICDCSW '12, pages 451–460, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4686-5. doi: 10.1109/ICDCSW.2012.87. URL http://dx.doi.org/10.1109/ICDCSW.2012.87.

[50] E. Maccherani, M. Femminella, J.W. Lee, R. Francescangeli, J. Janak, G. Reali, and H. Schulzrinne. Extending the netserv autonomic management capabilities using openflow. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 582–585, April 2012. doi: 10.1109/NOMS.2012.6211961.

[51] Oflops: User manual. URL http://archive.openflow.org/wk/images/3/3e/Manual.pdf.

[52] P. Jakma and D. Lamparter. Introduction to the quagga routing suite. *Network, IEEE*, 28(2):42–48, March 2014. ISSN 0890-8044. doi: 10.1109/MNET.2014.6786612.

[53] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1039834.1039864.

[54] Esar official web site:. URL http://www.sesarju.eu/.

[55] Sesar atm security risk assessment methodology, sesar wp16.2 – atm security, 2003.

[56] S. Jha, O. Sheyner, and J. Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63, 2002. doi: 10.1109/CSFW.2002.1021806.

[57] Anja Strunk. Costs of Virtual Machine Live Migration: A Survey. *2012 IEEE Eighth World Congress on Services*, pages 323–329, June 2012. doi: 10.1109/SERVICES.2012.23.

[58] Haikun Liu, Cheng-zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and Energy Modeling for Live Migration of Virtual Machines. pages 171–181.

[59] D. Breitgand and A. Epstein. Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds. In *INFOCOM, 2012 Proceedings IEEE*, pages 2861–2865, 2012. doi: 10.1109/INFCOM.2012.6195716.

[60] Jiong Luo, N.K. Jha, and Li-Shiuan Peh. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(4):427–437, April 2007. ISSN 1063-8210. doi: 10.1109/TVLSI.2007.893660.

[61] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Dynamic voltage scaling for multitasking real-time systems with uncertain execution time. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(8):1467–1478, 2008. ISSN 0278-0070. doi: 10.1109/TCAD.2008.925778.

[62] J. Sonnek, J. Greensky, R. Reutiman, and A. Chandra. Starling: Minimizing communication overhead in virtualized computing platforms using decentralized affinity-aware migration. In *Parallel Processing (ICPP), 2010 39th International Conference on*, pages 228–237, 2010. doi: 10.1109/ICPP.2010.30.

[63] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008. ISSN 0146-4833. doi: 10.1145/1496091.1496103. URL http://doi.acm.org/10.1145/1496091.1496103.

[64] R. Bolla, R. Bruschi, Franco Davoli, and F. Cucchietti. Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys Tutorials, IEEE*, 13(2):223–244, Second 2011. ISSN 1553-877X. doi: 10.1109/SURV.2011.071410.00073.

[65] L. T. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and Development*, 21(5):443–448, 1977. ISSN 0018-8646. doi: 10.1147/rd.215.0443.

[66] Benjamin Speitkamp and Martin Bichler. A mathematical programming approach for server consolidation problems in virtualized data centers. *IEEE Trans. Serv. Comput.*, 3(4):266–278, October 2010. ISSN 1939-1374. doi: 10.1109/TSC.2010.25. URL http://dx.doi.org/10.1109/TSC.2010.25.

[67] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. EnaCloud: An Energy-Saving Application Live Placement Approach for Cloud Computing Environments. *2009 IEEE International Conference on Cloud Computing*, pages 17–24, 2009. doi: 10.1109/CLOUD.2009.72.

[68] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy Efficient VM Scheduling for Cloud Data Centers: Exact Allocation and Migration Algorithms. *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 671–678, May 2013. doi: 10.1109/CCGrid.2013.89.

[69] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 28(5):755–768, May 2012. ISSN 0167739X. doi: 10.1016/j.future.2011.04.017.

[70] Aziz Murtazaev and Sangyoon Oh. Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review*, 28(3):212, 2011. ISSN 0256-4602. doi: 10.4103/0256-4602.81230.

[71] T. Setzer and a. Wolke. Virtual machine re-assignment considering migration overhead. *2012 IEEE Network Operations and Management Symposium*, pages 631–634, April 2012. doi: 10.1109/NOMS.2012.6211973.

[72] Zhen Xiao, Senior Member, Weijia Song, and Qi Chen. Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment. 24(6):1107–1117, 2013.

[73] E. Feller, C. Morin, and A. Esnault. A case for fully decentralized dynamic vm consolidation in clouds. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 26–33, 2012. doi: 10.1109/CloudCom.2012.6427585.

[74] Young Choon Lee and Albert Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, (2):268–280, March 2012. ISSN 0920-8542. doi: 10.1007/s11227-010-0421-3.

[75] Nguyen Quang-Hung, Pham Dac Nien, Nguyen Hoai Nam, Nguyen Huynh Tuong, and Nam Thoai. A Genetic Algorithm for Power-Aware Virtual Machine Allocation in Private Cloud. *ICT-EurAsia'13*, LNCS 7804:183–191, February 2013.

[76] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurr. Comput. : Pract. Exper.*, 24

(13):1397–1420, September 2012. ISSN 1532-0626. doi: 10.1002/cpe.1867. URL http://dx.doi.org/10.1002/cpe.1867.

[77] Y. Shang, D. Li, M. Xu, and J. Zhu. On the network power effectiveness of data center architectures. *Computers, IEEE Transactions on*, PP(99):1–1, 2015. ISSN 0018-9340. doi: 10.1109/TC.2015.2389808.

[78] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99, January 2010. ISSN 0146-4833. doi: 10.1145/1672308.1672325. URL http://doi.acm.org/10.1145/1672308.1672325.

[79] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, and J.A. Maestro. Ieee 802.3az: the road to energy efficient ethernet. *Communications Magazine, IEEE*, 48(11):50–56, November 2010. ISSN 0163-6804. doi: 10.1109/MCOM.2010.5621967.

[80] A.P. Bianzino, C. Chaudet, F. Larroca, D. Rossi, and J. Rougier. Energy-aware routing: A reality check. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1422–1427, Dec 2010. doi: 10.1109/GLOCOMW.2010.5700172.

[81] Vijay Mann, Avinash Kumar, Partha Dutta, and Shivkumar Kalyanaraman. Vmflow: Leveraging vm mobility to reduce network power costs in data centers. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I*, NETWORKING'11, pages 198–211, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20756-3. URL http://dl.acm.org/citation.cfm?id=2008780.2008800.

[82] Maruti Gupta and Suresh Singh. Greening of the internet. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 19–26, New York, NY, USA, 2003. ACM. ISBN 1-58113-735-4. doi: 10.1145/863955.863959. URL http://doi.acm.org/10.1145/863955.863959.

[83] M.A. Adnan and R. Gupta. Path consolidation for dynamic right-sizing of data center networks. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 581–588, June 2013. doi: 10.1109/CLOUD.2013.106.

[84] Weiwei Fang, Xiangmin Liang, Shengxin Li, Luca Chiaraviglio, and Naixue Xiong. Vmplanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers. *Comput. Netw.*, 57(1):179–196, January 2013. ISSN 1389-1286. doi: 10.1016/j.comnet.2012.09.008. URL http://dx.doi.org/10.1016/j.comnet.2012.09.008.

[85] Dejene Boru, Dzmitry Kliazovich, Fabrizio Granelli, Pascal Bouvry, and AlbertY. Zomaya. Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1):385–402, 2015. ISSN 1386-7857. doi: 10.1007/s10586-014-0404-x. URL http://dx.doi.org/10.1007/s10586-014-0404-x.

[86] Ibm cplex. URL http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.