UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Efficient and Elastic Management of Computing Infrastructures

Gestión Elástica y Eficiente de Infraestructuras de Cómputo

Dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in the subject of Computer Science

October 2015

Author: Carlos de Alfonso Laguna
Advisor: Ignacio Blanquer Espert
Advisor: Germán Moltó Martinez

Acknowledgements

En primer lugar quiero dar las gracias a mis padres Lola y Ramón, por haberme animado siempre a continuar trabajando. Gracias a ellos he ido avanzando en mi vida y he llegado hasta aquí. Están tan orgullosos, que sólo por oírles hablar de mí, merece la pena seguir haciendo cosas. Muchas gracias papás, espero seguir consiguiendo que os sintáis orgullosos de mí.

Quiero dar las gracias a Mar porque siempre está a mi lado y por haberme apoyado tanto para terminar este trabajo. Ha estado todo el tiempo ahí animándome, diciéndome lo poco que me iba a costar terminarlo, y que no me preocupara, porque iba a quedar bien. Muchas gracias mi amor, espero que sigas siempre junto a mí.

También tengo que agradecer la labor de Miguel Caballer y Fernando Alvarruíz, porque con ellos empezó esta tesis y porque han aportado mucho trabajo a ella. Y agradecer a Nacho Blanquer y a Germán Moltó, que se hayan ido incorporando al desarrollo del trabajo, y se hayan implicado de tal forma que han acabado siendo mis directores. Muchas gracias a todos, espero poder continuar trabajando con vosotros.

Y finalmente, quiero dar las gracias a Vicente Hernández, porque él es el artífice de que hace años empezase con esto de la investigación y sé que, aunque no me haya podido acompañar hasta aquí, se habría sentido orgulloso al ver que por fin he terminado. Muchas gracias, Vicente.

Abstract

Modern data centers integrate a lot of computer and electronic devices. However, some reports state that the mean usage of a typical data center is around 50% of its peak capacity, and the mean usage of each server is between 10% and 50%. A lot of energy is destined to power on computer hardware that most of the time remains idle. Therefore, it would be possible to save energy simply by powering off those parts from the data center that are not actually used, and powering them on again as they are needed.

Most data centers have computing clusters that are used for intensive computing, recently evolving towards an on-premises Cloud service model. Despite the use of low consuming components, higher energy savings can be achieved by dynamically adapting the system to the actual workload. The main approach in this case is the usage of energy saving criteria for scheduling the jobs or the virtual machines into the working nodes. The aim is to power off idle servers automatically. But it is necessary to schedule the power management of the servers in order to minimize the impact on the end users and their applications.

The objective of this thesis is the elastic and efficient management of cluster infrastructures, with the aim of reducing the costs associated to idle components. This objective is addressed by automating the power management of the working nodes in a computing cluster, and also proactive stimulating the load distribution to achieve idle resources that could be powered off by means of memory overcommitment and live migration of virtual machines. Moreover, this automation is of interest for virtual clusters, as they also suffer from the same problems. While in physical clusters idle working nodes waste energy, in the case of virtual clusters that are built from virtual machines, the idle working nodes can waste money in commercial Clouds or computational resources in an on-premises Cloud.

Resumen

En los Centros de Procesos de Datos (CPD) existe una gran concentración de dispositivos informáticos y de equipamiento electrónico. Sin embargo, algunos estudios han mostrado que la utilización media de los CPD está en torno al 50%, y que la utilización media de los servidores se encuentra entre el 10% y el 50%. Estos datos evidencian que existe una gran cantidad de energía destinada a alimentar equipamiento ocioso, y que podríamos conseguir un ahorro energético simplemente apagando los componentes que no se estén utilizando.

En muchos CPD suele haber clusters de computadores que se utilizan para computación de altas prestaciones y para la creación de Clouds privados. Si bien se ha tratado de ahorrar energía utilizando componentes de bajo consumo, también es posible conseguirlo adaptando los sistemas a la carga de trabajo en cada momento. En los últimos años han surgido trabajos que investigan la aplicación de criterios energéticos a la hora de seleccionar en qué servidor, de entre los que forman un cluster, se debe ejecutar un trabajo o alojar una máquina virtual. En muchos casos se trata de conseguir equipos ociosos que puedan ser apagados, pero habitualmente se asume que dicho apagado se hace de forma automática, y que los equipos se encienden de nuevo cuando son necesarios. Sin embargo, es necesario hacer una planificación de encendido y apagado de máquinas para minimizar el impacto en el usuario final.

En esta tesis nos planteamos la gestión elástica y eficiente de infrastructuras de cálculo tipo cluster, con el objetivo de reducir los costes asociados a los componentes ociosos. Para abordar este problema nos planteamos la automatización del encendido y apagado de máquinas en los clusters, así como la aplicación de técnicas de migración en vivo y de sobreaprovisionamiento de memoria para estimular la obtención de equipos ociosos que puedan ser apagados. Además, esta automatización es de interés para los clusters virtuales, puesto que también sufren el problema de los componentes ociosos, sólo que en este caso están compuestos, en lugar de equipos físicos que gastan energía, por máquinas virtuales que gastan dinero en un proveedor Cloud comercial o recursos en un Cloud privado.

Resum

En els Centres de Processament de Dades (CPD) hi ha una gran concentració de dispositius informàtics i d'equipament electrònic. No obstant això, alguns estudis han mostrat que la utilització mitjana dels CPD està entorn del 50%, i que la utilització mitjana dels servidors es troba entre el 10% i el 50%. Estes dades evidencien que hi ha una gran quantitat d'energia destinada a alimentar equipament ociós, i que podríem aconseguir un estalvi energètic simplement apagant els components que no s'estiguen utilitzant.

En molts CPD sol haver-hi clusters de computadors que s'utilitzen per a computació d'altes prestacions i per a la creació de Clouds privats. Si bé s'ha tractat d'estalviar energia utilitzant components de baix consum, també és possible aconseguir-ho adaptant els sistemes a la càrrega de treball en cada moment. En els últims anys han sorgit treballs que investiguen l'aplicació de criteris energètics a l'hora de seleccionar en quin servidor, d'entre els que formen un cluster, s'ha d'executar un treball o allotjar una màquina virtual. En molts casos es tracta d'aconseguir equips ociosos que puguen ser apagats, però habitualment s'assumix que l'apagat es fa de forma automàtica, i que els equips s'encenen novament quan són necessaris. No obstant això, és necessari fer una planificació d'encesa i apagat de màquines per a minimitzar l'impacte en l'usuari final.

En esta tesi ens plantegem la gestió elàstica i eficient d'infrastructuras de càlcul tipus cluster, amb l'objectiu de reduir els costos associats als components ociosos. Per a abordar este problema ens plantegem l'automatització de l'encesa i apagat de màquines en els clusters, així com l'aplicació de tècniques de migració en viu i de sobreaprovisionament de memòria per a estimular l'obtenció d'equips ociosos que puguen ser apagats. A més, esta automatització és d'interés per als clusters virtuals, ja que també patixen el problema dels components ociosos, encara que en este cas estan compostos per, en compte d'equips físics que gasten energia, per màquines virtuals que gasten diners en un proveïdor Cloud comercial o recursos en un Cloud privat.

Contents

Chapter 1

Introduction and Objectives

In modern societies, it is very common to apply Information and Communication Technologies (ICT) to almost every daily action. Computer equipment is widely used in one way or another in our workplace (computers, point of sale terminal, robots, etc.), and some voices have started to claim about the problems associated to the big scale of such amount of devices. There are initiatives that focus on the problem that entail too many devices accessing data networks (e.g. Internet, 4G networks, etc.), the waste management of all the devices that are being replaced by new ones with the most advanced features, etc. But one of the main problems is the huge need of electric power to run such big amount of devices, which will increase over time, since giving more functionality to those devices usually implies that they will need more energy.

When we focus on the data centers, where a lot of computer devices and electronic units are needed for their common functions, maintenance, network connection, etc., we face a problem that it is not possible to ignore. The problem is yet more evident when we notice that the huge amount of energy used for the facility is translated into a huge amount of money to pay for the energy. The energy waste cannot be considered only an environmental concern, but also an economical problem for the organizations that own the data centers. This problem has gained importance as the price of the energy has been raising for the last years as seen in Figure 1.1.

From a simple point of view, a data center consists of a set of servers. There are different sizes of data centers that range from a reduced set of servers (e.g. in a small enterprise), to facilities from big providers where there are dozens of thousands of servers. It is usual that the principal providers keep the number of

[1] Sources for data: `http://ec.europa.eu/eurostat/web/energy/data/main-tables` and `http://www.eia.gov/forecasts/steo/tables/?tableNumber=8`

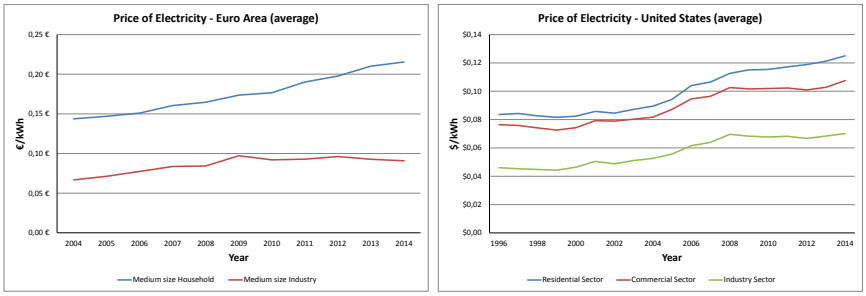


Figure 1.1: Average price of energy in the Euro Area and the United States. According to the available data[1], the price has been raising in both geographical areas for all sectors, except in the last two years for the industry sector in the Euro Area.

servers that they own in secret. In 2013, during a talk, Microsoft filtered that they own more than one million of servers, while Google "is bigger" [16]. In 2009 it was estimated that Google owned near one million of servers, and other big providers such as Facebook, Yahoo, Amazon, IBM or HP own hundreds of thousands of servers each one, hosted in their data centers [92]. In the context of eScience in Spain, there exist different national data centers. One of the most important data centers in Spain is the *Barcelona Supercomputing Center* (BSC) that owns more than 3000 servers as of 2014[29]. There are also other smaller data centers such as the *Centro de Supercomputación de Castilla y León* (CSCL) that owns around 300 servers [50], the supercomputer *Margerit* in the *Centro de Supercomputación y Visualización de Madrid* that is built from around 300 servers[2], the supercomputer *Altamira* hosted in the *University of Cantabria* that accounts with more than 240 servers[3], or the *Centro de Supercomputación de Galicia* (CESGA) that owns around 150 servers [30].

In the end, the new era for ICT and the increase in the amount of new facilities for computation have soared the demand of energy for data centers around the world, in the last years. In 2008 a report stated that the fraction of energy needed to power this kind of facilities was about 0.5% of the total energy consumed in the world. It was also comparable to the energy consumption of a whole country such as Argentina or The Netherlands [53]. This report also predicted that, if no action was undertaken, such percentage would increase to 2% of the total by 2020. This is such a big problem that in 2008, the emission of CO_2 related to data centers were also comparable to the emission of CO_2 of the whole aviation industry [114]. In a later report [70], an analysis of the data *a posteriori* showed a slowdown in the increase of the energetic demand, which was associated to the recession and the economic downturn of 2008. Nevertheless the report situated the consumption of data centers in 1.3% of the total energy in the world, in 2010.

[2] `http://www.cesvima.upm.es/infrastructure/hpc`
[3] `http://web.unican.es/unidades/scti/servicio-santander-de-supercomputacion`

All those data not only refer to the consumption of the servers, but also to the facilities that are needed to host them. It is important to take into account that the total energy needed for the data center is not only destinated to power the servers, but also to other equipment that range from the Uninterruptible Power Supplies (UPSs) to cooling machines that counteract the heat dissipated by the electronic devices (e.g. servers, switches, displays, etc.).

Each data center has a lot of servers, but some reports state that the mean usage of typical data centers is around 50% from the peak capacity to which they were dimensioned [55], and the mean usage of each server is between 10% and 50% of its capacity [110]. Such data uncover that it exists a lot of energy that is dedicated to keep powered on a set of computer hardware that most of the time is idle. Therefore, it would be possible to save energy simply by powering off those parts from the data center that are not being used, and powering them on again as they are needed.

In this context arises the need of containing the energetic consumption of computer equipment. The two main approaches to energy saving in computers are basically the Static Power Management (SPM) techniques and the Dynamic Power Management (DPM) techniques [121]. The SPM techniques are related to the electronic components as individuals, their consumption, their efficiency, etc., and the main action in this sense is to use higher efficiency components with lower power requirements. Most of the developments of SPM techniques have been encouraged by the massive usage of mobile devices (e.g. laptops, smartphones, tablets, wearables, etc.). These kind of devices is usually powered by batteries with reduced sizes. Due to its size, the batteries have also a relatively low capacity. In this sense, the reduction in the consumption of the components of these devices has been the immediate answer to extend the service life of the batteries. The servers and other components in the data centers have benefited from these advances, as the electronic components have been enhanced and the consumption has been reduced. Some examples of such kind of enhancements are: (1) the power sources are now more efficient, while a few years ago, its efficiency was barely a 80% [79] [63]; (2) the Solid State Drives (SSD) are commonly used, thus eliminating the consumption associated to the mechanical parts of the conventional disks; (3) the DDR memories of new generations are now more efficient than previous versions, and are also appearing low consuming variants of the DDR memories; (4) in the last years, the low consuming versions of the CPUs have been generalized; or (5) some technologies such as the Dynamic Voltage and Frequency Scaling (DVFS) or Power Gating (PG) are included in processors, to enable the synchronization between the Operating System (OS) and the processor, in order to reduce power consumption.

On the other side, we find the DPM techniques, that are related to the adaptation of the system to the workload. In this sense we can find techniques that range from selectively shutting down cores or other parts in the CPU, or varying the voltage

or frequency of the clock (i.e. using the DVFS and PG features of the processors, if available), to the usage of energy criteria for the selection of the server where a job is being executed.

Probably one of the most common servers arrangement in a data center is the *cluster*. A computing cluster is a set of servers that are interconnected by one or several networks, working together to solve one or more computational problems. The cluster is usually managed by a Batch Queueing System (BQS) for the case of High Performance Computing (HPC) systems, or Cloud Management Platforms(CMPs) in the case of an on-premises Cloud. These two systems are referred in general as the Local Resource Management System (LRMS). In this way, users request the execution of a job and then, the LRMS selects which subset of the internal servers will be in charge of running the job. Finally, the LRMS will carry out the tasks which correspond to the lifecycle of the job (e.g. copying the input files to the servers, starting the application, monitoring the execution, etc.). The DPM techniques have been widely applied in computing cluster facilities. The main approach in that case is the usage of energy saving criteria for scheduling the jobs into the working nodes, but an additional step consists of putting into low consuming mode (e.g. suspension, hibernation, power off) the nodes that are not being used.

In the last years, the DPM techniques have gained importance due to the advances of the virtualization technology and the emergence of Cloud Computing (the Cloud)[25]. By using virtualization techniques, a Virtual Machine (VM) is delivered to the user, but while it will behave like a physical one, the VM runs on top of the hardware of a physical machine with the help of a hypervisor (see Figure 1.2). Moreover, multiple VMs may share the physical hardware. The virtualization technology was seen as an opportunity to reduce the number of physical servers of an organization, by converting these physical servers into virtual servers and hosting them into a reduced number of physical servers. This way, it is possible to profit from the resources of physical servers that were dedicated to specific tasks, but were idle most of the time. In such case, the physical resources can be shared among different virtual servers, and those other physical servers that have been freed could be powered off (thus saving the energy). The migration from physical servers to virtual servers in a reduced set of physical servers has been generally denominated as *server consolidation*, and it has been possible due to products such as Xen[34], KVM[73], VMWare[126], Virtual Box[104], or Microsoft Hyper-V[90].

The server consolidation was usually performed manually by the system administrator, but the Cloud introduced a new paradigm for the distribution and consumption of resources through the Internet. This technology enabled the usage of

[4]Image inspired in media found on `http://www.vmware.com/virtualization/virtualization-basics/how-virtualization-works`

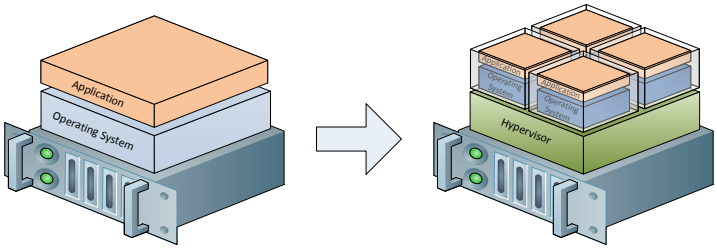


Figure 1.2: Virtualization of physical servers. With the help of hypervisors it is possible to migrate multiple physical servers to virtual machines that share the same hardware[4].

computational infrastructures, storage resources, applications, etc. on demand, in a dynamic way, in a pay-per-use basis, through the Internet [14]. According to the most accepted definition of Cloud computing provided by the NIST (National Institute of Standards and Technology) it "is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [88]. In the end, the Cloud enables the possibility of having storage space, a fully functional set of virtual machines, different applications, etc. according to the specific needs at each moment, and paying only for what is being used, in the case of using public Cloud providers. From the point of view of the end-users, the Cloud enables them to access to virtually any kind of resources in an agile and easy way. From the point of view of the providers, the Cloud is an innovative mechanism to provide services through Internet.

Providing different resources in the Cloud introduce different requirements from the users and problems for the providers on how to deliver the requested resources. This is why the Cloud is usually classified according to the type of resource that is delivered to the end-user. The most common classification differentiates three layers or categories: (i) Infrastructure as a Service (IaaS), where the users deploy running VMs; (ii) Platform as a Service (PaaS), where the developers can use existing components to build his applications (e.g. web server, synchronization queues, database servers, etc.); and (iii) Software as a Service (SaaS), where the user can use applications that are provided from the Cloud.

The Cloud introduces new opportunities. On the one hand, it is possible to outsource the data center to the Cloud, thus saving the costs associated to building such kind of facilities (e.g. acquiring the physical placement, buying the equipment, hiring computer administrators, installing servers, etc.). On the other hand, it is possible to get immediate access to the computational resources, and thus avoiding the delay introduced by building the data center. This also enables to reduce the time-to-market when starting a new business. But the Cloud also in-

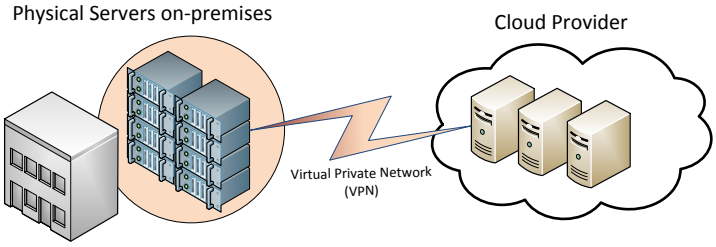


Figure 1.3: It is possible to use *cloud bursting* to increase the size of the data center, by creating some VMs that are connected to the on-premises servers.

troduces the opportunity of dynamically increasing the capacity of a data center if needed. Data centers have been traditionally dimensioned to handle peaks of workload, but as seen before, the mean usage of the data center is far from those peaks. The Cloud enables to have a modest data center and to use the ability of *Cloud bursting* (see Figure 1.3) techniques to increase the computational capacity in the Cloud when a peak on the demand happens. Then the owner not only saves the costs associated to the additional computer equipment and its maintenance, but also the energy dedicated to power that extra equipment.

Anyway, fully outsourcing the data center to the Cloud is not a solution by itself. The public Cloud providers have a pay-per-use business model, which means that they charge a fee for each fraction of time that a VM is running in the Cloud, whether it is being used or not. If we run a computing cluster in the Cloud, we have the risk of having idle VMs that instead of wasting energy that cost money, are directly wasting money. Here we can see that saving costs in the Cloud has a parallelism with energy saving of physical servers. In both cases the idle machines are wasting money. In the end, the minimum energy ever consumed by a server is the residual energy needed when the server is off but capable of being remotely powered on back again (e.g. the energy needed to keep the server in stand-by mode, the energy needed to power the network card to be able to use Wake-on-LAN, the energy needed for the Intelligent Platform Management Interface (IPMI) service, etc.). On the other side, the minimum money ever needed to have a VM ready to be deployed in the Cloud is the amount of money that is needed when the VM is not running but it is possible to start it if needed (i.e. the money needed to keep the storage available, the reservation of IP addresses in some providers, etc.).

In any case, the services provided by Cloud providers are backed by the data centers of those providers, and they use virtualization techniques to provide their services. As an example, when a storage space is requested, the provider does not deliver a physical disk for each user. Instead, they create a virtual storage space that coexists with other virtual storage spaces from other users, in one or

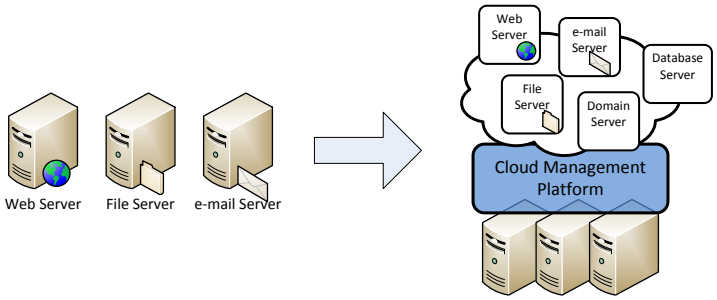


Figure 1.4: The data center can be re-arranged into a Cloud scheme, to create an on-premises Cloud. Then, it is possible to migrate the physical servers to virtual machines, to share the physical infrastructure for additional VMs.

more physical devices. This is the case of the machines requested to a Cloud provider, since the user is not provided with access to a physical machine (as it was happening prior to the Cloud). Instead, a VM is usually created for the user, and the VM typically coexists with other VMs from other users, in the same physical machine.

Many organizations are reluctant to outsource their data, their processes and their machines to other providers, even when they still want to profit from the features of the Cloud. These organizations not only consider that they lose control over those resources, but they also have the risk of incurring on legal issues due to the lack of knowledge about their physical placement. That is why some organizations that already owned data centers, have re-arranged them into a Cloud scheme, to create their on-premises Cloud. To make it, they simply have to re-arrange their clusters and use a CMP such as OpenNebula[102], OpenStack[103] or VMWare[126] (see Figure 1.4).

Organizing a data center as a private Cloud offers lots of advantages for the owner, as it is possible to use the resources in a more convenient way, and to get economic savings (by both the energy savings and the reduction of the number of equipments). But it also offers advantages to the users, as they will probably gain in quality of service. If the users access directly the physical machines, they will be restricted to a common, limited environment, with a set of libraries installed on the servers, a specific compiler, etc. The user will have to adapt to such environment even if it is not optimal for his applications. If we move on to the Cloud, a specific Virtual Cluster (VC) could be delivered to each user, with the needed libraries installed on it and the optimal environment configured for the applications. Once a VC has been used, it can be disposed of, to make room for other VMs in the Cloud.

The caveat here is that a frequent creation and destruction of VMs in a dynamic context may drive to an inefficient distribution of the running VMs in the servers of an on-premises Cloud, but it may also lead to the fragmentation of the virtualization resources. One consequence is that it may happen that new VMs with specific requirements cannot be hosted, even when the platform as a whole has enough free resources. In this case, the free resources are fragmented among the physical servers and cannot be used together. As an example, under a scenario of two servers with two cores each, in which each of the servers is hosting a VM with one core, a new VM that requests two cores could not be hosted even having two free cores in the infrastructure. If both existing VMs had been hosted in only one of the servers, the new VM request could have been served.

Moreover, we must take into account that in an on-premises Cloud we are also working with conventional servers. Therefore, it would be interesting to apply the aforementioned DPM techniques that consists in powering off the servers that are idle, in order to save energy. An inefficient distribution of VMs would prevent powering off the servers (e.g. few VMs are being hosted by many physical servers). An extreme example is that all physical servers host a small VM. Under such scenario, no server can be powered off, as they are not yet idle. In order to correct such scenario, it would be interesting to include a mechanism to automatically consolidate the VMs into a reduced number of physical servers. Continuing with the previous example, it would probably be possible to host all the small VMs in a few number of physical servers, thus enabling to power off those that will get idle.

1.1 Objectives

The main objective of this thesis is the efficient and elastic management of computing clusters, both physical and virtual. The aim is to obtain a reduction in the cost associated to the idle components of the infrastructure (energetic costs in the case of physical infrastructure or economic costs associated to the VMs in the case of virtual infrastructures).

In order to address this general objective, it is decomposed into several problems that will be tackled in this thesis:

- It has been noted that data centers may be wasting energy and money in case that idle equipment is kept powered on. First of all, we propose the creation of an **energy management system that will automate the power management** of the internal servers of a cluster. The task of this system consists in detecting the servers which are idle, in order to power them off. Conversely, this system will intercept the requests for the execution of jobs, and it will power on some of them, to be able to execute the job(s). This

system should neither modify the way of working of the users, nor interfere with the end-user experience.

- The Cloud has been revealed as an alternative for the economic saving for the users that need computing facilities. Moreover it avoids the up-front investment of money needed to create and to start up a physical data center. It is also an alternative to contain the size of a data center, and eventually increase its capacity to attend to unexpected peaks on the workload. To determine whether it is interesting or not to outsource a cluster, from an economical point of view, we consider the need of **studying the economic cost of using a computing cluster in a commercial Cloud provider**. Once the different constraints are identified, we should be able to decide under which circumstances is better to deploy a cluster in the Cloud or to invest in a physical facility. This analysis should take into account not only the price of the computing clusters, but also the ratio of performance per money.

- Establishing a computing cluster in the Cloud seems to be an immediate task, as it is no more than a set of VMs that are interconnected by a network, with a specific configuration. But as it happens in the case of physical clusters, the VC still has the problem of the idle servers. In the case of physical servers, the waste of energy is translated into waste of money, whereas in the case of virtual server it is directly a waste of money. So, we propose the generalization of the system to automate the power management proposed in the first point, in order **to create Elastic Virtual Clusters**. The underlying idea is to apply the DPM technique that consists of powering off the idle nodes by creating and destroying VMs depending on the workload, instead of dealing with the physical servers. This kind of clusters should be available for both commercial Cloud providers, in order to outsource parts of the infrastructure, and for on-premises Cloud.

- The virtualization techniques and the Cloud have modernized the management of the data centers. These techniques enable to manage the equipments in a more convenient way, and also help to save costs by consolidating servers. But the intrinsic dynamism for a Cloud may end up after several cycles of creation and destruction of VMs with inefficient distributions of the VMs in the physical hosts. Such inefficient distribution may prevent applying the energy saving techniques that we are considering in this work. This is why we propose the creation of **a system that will automatically redistribute the VMs in the physical servers of an on-premises Cloud deployment**, making use of live-migration techniques (in order to avoid downtimes for the users of the VMs). In this case, the objective is to reduce the number of physical servers needed to host all the VMs, with the aim of being able to put the idle ones into a low consuming mode or to power them off.

- As it happens in the case of the physical servers, the VMs in the Cloud may also suffer from the overdimension of the resources that are needed to carry out the calculations. It may also happen that users of the Cloud are forced to use template-based VMs that allocated an amount of memory or number of cores that exceeds the requirements of the applications that are being ran on it. In this case we propose the creation of **a system that automatically and dynamically adapts the resources of the VMs to the actual workload**, by varying the resources that are allocated for it (i.e. memory). The objective is to be able to overcommit the physical resources, in case that they have a low usage. The effect is that the amount of virtual memory requested for the VMs will be greater than the physical amount of memory available in the servers that host the VMs. This will introduce an enhanced VM consolidation per physical node, and also will get more free physical resources. With the usage of live migration techniques, this technique can restore the level of service as the applications running on the VMs demand more memory than what is currently allocated.

By addressing all these problems, we can close the loop that enables to create Elastic Virtual Clusters(EVCs) in an on-premises Cloud, backed by a physical cluster where the memory of the VMs that build the cluster is automatically adapted to the actual workload. Later, the VMs are consolidated into the less possible number of physical server, to automatically power off the servers that are idle. In this way, we achieve the creation of EVCs that run on physical clusters with an elastic behaviour. Moreover, we will have an economic criteria to decide whether it is convenient or not to outsource part of the EVCs requested by the users to an external Cloud provider.

1.2 Summary of the state of the art

The central chapters of this document are academic papers that have been published in different journals. So each of them include a part dedicated to revise the state of the art in the particular problem addressed by the paper. Nevertheless, we are including here a summary of the state of the art in order to help the reader differentiate each of the topics that have been tackled.

1.2.1 Automated Power Management

Many efforts have focused on energy-aware allocation of tasks in clusters, both for HPC clusters and on-premises Clouds. The survey [121] explains some DPM works from other authors that would get idle resources, but any of the reviewed works usually assume that the idle resources are powered on or off automatically and do not consider any scheduling strategy for that. Most of them are basically job schedulers that would substitute the existing schedulers or CMP, and as a

result, they will modify the way that users interact with them. In the revised literature, there are also other approaches, but they also do not integrate with the LRMS. An example is the work presented in [76], that is a method to reduce power in large-scale distributed systems by switching nodes on and off according to the load. This work is actually a booking system for computing nodes but not an energy saver, and therefore it is not suitable for interactive systems. In that case the decssion of whether a cluster can be powered off can be taken because of the reservations.

From the point of view of system administrators, there are not many available tools to implement green policies in clusters. In the case of HPC schedulers, MOAB (which is the Enterprise version of Maui[35]) introduces some features to pack workload and to place idle servers in power-saving modes [111]. The latest versions of SLURM introduced the ability to change CPU frequency and voltage in order to save energy. However, these solutions are tied to each of the particular LRMSs. Since the choice of LRMS is conditioned by many factors, administrators may find that the most suitable LRMS does not take into account energy saving mechanisms. In the case of on-premises Cloud, the most common CMPs in the scientific community (OpenNebula, OpenStack, Eucalyptus[5], etc.) do not offer automatic power management out of the box. In the particular case of OpenQRM[6], it offers power saving features for the enterprise version, which is distributed under a commercial license, but not in the community version.

On the side of the commercial cloud platforms, we can also find several solutions that offer automatic power management features. Probably the most popular example is VMWare vSphere 5.5[7] which is capable of powering on and off physical hosts, but it is restricted to VMWare hypervisor and its cost is very high (in 2015, it starts at USD 2,875.00 for the version which is capable of power management). Huawei's FusionSphere [8] also claims to offer automated power management in recent versions. It builds up on OpenStack, but its solution is also distributed under a commercial license. But the commercial solutions are not of interest for this work because once taken the decision of purchasing a commercial CMS, the possible solutions of the problems that may arise during its lifetime are restricted to the solutions offered by the developer.

[5] `https://www.eucalyptus.com`

[6] `http://www.openqrm-enterprise.com`

[7] `http://www.vmware.com/products/vsphere`

[8] `http://e.huawei.com/en/products/cloud-computing-dc/cloud-computing/fusionsphere/fusionsphere`

1.2.2 Facilitating Power Management

The automated power management can only be made in case that we get idle servers that will be candidates to be powered off. At a scheduler level, there are works that try to reduce the number of physical servers needed to run the jobs in a HPC cluster, or to host the VMs deployed on an on-premises Cloud. In fact, most of the schedulers shipped in the default distributions of the common LRMSs include features for reducing the number of used servers. But the problem arises when the job or the VM are started in a server, because that server cannot be powered off. In the case of HPC, most of BQS include in one way or another, the support for checkpointing the jobs and thus enabling to migrate them from one host to another. But once the physical server has been selected to run a job, it is usually accepted that the job will continue running in that physical server and it will not be migrated unless it is necessary. The job will finish in an estimated time, or it will be terminated if it exceeds the maximum running time[9].

But in the case of the VMs, their lifetime is neither known in advance, nor it can usually be estimated. So the automated power management needs to be facilitated, by obtaining idle servers using other mechanisms. There are several works that try to take advantage from live-migration features to consolidate the VMs in a platform into a few number of physical hosts. The most common techniques are based in artificial intelligence (IA) or model the problem as *the bin packing problem* (BP) and solve it applying some heuristics. In the case of IA, the solutions range from reinforcement learning [109][33][49] to fuzzy logic [87] or nature-based solutions such as the works in [52], which is inspired on the movements of the ants in a colony, [107] and [86], that are inspired on the behaviour of the swarms during migratory flights, or [56], which is based on the movements of a bee colony. In the case of the BP-like problem, there are several proposal of works such as [2] that statically reduces the number of physical hosts, but is not intended to be used in a continuously working platform, [123] that tries to combine VM placement with DVFS, [19] that solves the BP problem and includes a scheduler to take into account energy saving criteria to re-place the VMs, or the works [57] and [115] that also try to reduce the number of used physical hosts. The main limitation of those works is that they do not provide any available implementation of their algorithms that can be used, and it is not easy to reproduce the experiments nor implement the solution in real deployments.

Moreover, users tend to overestimate the amount of memory required by their applications resulting in unused memory that could be dedicated to additional VMs running on the same physical machine [119]. Besides making a low usage of the resources, such overdimension may prevent from consolidating VMs to a reduced number of servers. Some works have tried to adjust the resources of the

[9]The BQS include concepts such as the "wall time", which implies that the job will be killed if it has been running for longer than a predefined amount of time

VMs to the actual workload. As an example, the work shown in [39] tries to adapt the allocation of the CPU in the VMs running on the Xen hypervisor, but it does not study the memory. There are also other works such as [118] and [59] that try to adapt the virtual memory to the actual needs of the applications running in the VMs using various methods, but they make it at a single host level and do not consider the platform as a whole. Therefore, these works are only useful for hand-made distributions of VMs, because the CMP is not able to overcommit the hosts according to the memory that is actually being used. There are also works such as [80] that try to consider the platform level, but do not offer any countermeasure in case that the memory is overcommited an a VM claims the memory that it had requested at first.

1.2.3 Elastic Virtual Clusters

There are different examples of virtual clusters in the literature, such as [31], [44] or [128], that propose the creation of VCs. These works mainly deal with the provision of the VMs and configuration of the cluster topology (e.g. connectivity, shared filesystem, ssh-ability, etc.). Some of them include configuration issues (e.g. installing applications, creating users, etc.). But, while they are plenty of features, most of them lack elasticity. In this sense, once a cluster has been delivered to the user, all the VMs will continue running even if they are idle. Such static behaviour can introduce a waste of money, in case that the VMs are deployed in a commercial platform. In the case that the VMs are deployed in an on-premises Cloud, keeping the VMs always on may prevent from creating other VMs because of the lack of resources.

There are several works that identify the problem, and try to give a elastic behaviour to the clusters. As an example, [15] and [84] that evaluate the possibility of using Amazon EC2 to extend a physical cluster, depending on the workload. The work [99] also explores the dynamic provision of working nodes in the cloud, depending on the size of the jobs in the queues, introducing several policies to limit the amount of working nodes to be powered on. The main limitation of most works in this field is that they seem to be ad-hoc private implementations that have not been released or they remain as theoretical works.

As ready-to-use environments for VC, we can find that the standard distribution of Hadoop [23] includes an easy-to-use mechanism to create VCs in Amazon EC2. The main problem is that it only deals with Hadoop-based clusters, and the number of nodes for the cluster is not dynamically managed (although it is possible to add or destroy working nodes by hand). StarCluster [93] is an open source cluster-computing toolkit for Amazon EC2. But the problem in this case is that it only supports Amazon EC2, and it uses pre-built Virtual Machine Image (VMI) stored with specific software installed. It is based on the Open Grid Scheduler LRMS (formerly known as *Sun Grid Engine*, or *SGE*) and includes common libraries such

as OpenMPI, OpenBLAS, LAPACK, etc. As a good point, it includes the module Elastic Load Balancer that supports shrinking or expanding the cluster based on the statistics of the queues of the LRMS. Recently we can find the work [12], that is a development to create entire VCs running a batch system such as HTCondor that grow and shrink automatically based on the usage. The caveat in this work is that it only is designed to run the VMI distributed by them, and the target is a Cloud exposing an EC2 interface (e.g. Amazon EC2 or OpenStack). While it is a very interesting project, restricting to a precise VMI limits the applicability of the EVC to the embedded software distribution.

1.3 Organization of this Document

The first part of the works made during the research phase is related to the topic of the power management of cluster infrastructures (both physical and virtual), and the economical advantages of applying such techniques. The chapters 2, 3 and 4 correspond to a set of papers related to this topic, that have been already published on different journals in the area. The second part of the work is related to the topic of facilitating the power management in on-premises Clouds. The chapters 5 and 6 are structured as papers, since they are in the process of submission to different journals in the area. All these chapters altogether collect the work made to address the problems proposed in this thesis.

First, chapter 2 explains the *Cluster Energy Management System (CLUES)*, which is an energy management system that automates the power management of a computing cluster for both HPC and on-premises Cloud. It can be considered as a framework for the application of the DPM technique that consists of powering off the idle internal nodes from a cluster. In order to avoid interfering the interaction of the user with the cluster, CLUES tries to give the end-user the illusion of using the cluster as usual, as if all the nodes were available all the time. So, the scheme of CLUES consists of intercepting the requests for executing the jobs (in the case of the Cloud, the requests to host new VMs), and powering on a set of working nodes that meet the requirements of the requests. According to this scheme, we get an elastic behaviour for the physical infrastructure, as it is adapted to the actual workload. The consequence is that it is saved the energy that correspond to the internal nodes that are not being used. This chapter corresponds to the paper [40], that has been published in the journal "Computers & Electrical Engineering", which has an impact factor of 0.992 and it is classified in the third quartile (Q3) of the Journal Citation Report (JCR) for both the topics "Computers Science, Hardware & Architecture" and "Computer Science, Interdiciplinary Applications" in 2013 (when the paper was published). The main contribution for this paper is the creation of the concept of CLUES and its architecture, along with the main functions of the product.

Next, chapter 3 tries to answer the question brought by the Cloud paradigm about when it is convenient to outsource a data center that is mainly dedicated to e-Science, to a commercial Cloud provider. In this chapter it is revisited the Total Cost of Ownership (TCO) of a physical computing cluster (i.e. acquisition of the premises, adaptation of the building, purchase of the servers, maintenance, etc.), also incorporating the cost of energy. The TCO is then compared to the cost of establishing an equivalent cluster in Amazon EC2 (which is used as an example for the commercial Cloud). In both cases it is considered that it is applied the DPM technique of powering off the idle nodes (in order to save energy, and in order to save money). Finally an analysis is made, in order to calculate whether it is more convenient to purchase a physical cluster or to deploy it into the Cloud. This chapter correspond to the paper [41], that has been published in the journal "Future Generation Computer Systems", which has a impact factor of 2.639 and it is classified in the first quartile (Q1) of the Journal Citation Report (JCR) for the topic "Computer Science, Theory & Methods" in 2013 (when the paper was published). The main contribution for this paper is the creation of the whole economic model, along with the comparison of the costs for the physical and the VCs.

Chapter 4 is dedicated to generalize the usage of CLUES to be used for the creation and management of a EVC. The concept of an EVC consists of having a set of VMs deployed in a Cloud (either commercial or on-premises) that follow a cluster arrangement which is equivalent to a physical cluster (i.e. an interconnection network, a shared storage space and a LRMS). An elasticity manager is incorporated into the cluster, to control the number of internal nodes and to carry out the tasks to create or to delete the corresponding VMs. In this case, CLUES is used as the elasticity manager and it behaves as described in chapter 2 but dealing with VMs instead of physical machines. We have called this development *Elastic Cloud Computing Cluster* (EC3) and it profits from the Infrastructure Manager (IM) [27] to create and to configure the VMs in the Cloud. This chapter correspond to the paper [26], that has been published in the "Journal of Computer and System Sciences", which has a impact factor of 1.091 and it is classified in the second quartile (Q2) of the JCR for both the topics "Computer Science, Hardware & Architecture" and "Computer Science, Theory & Methods" in 2013 (when the paper was published). The main contribution for this paper is the concept of the EVC and the usage of CLUES to implement it.

Chapter 5 is focused on the problems related to inefficient distributions of VMs that arises when a data center is arranged as an on-premises Cloud. On one side, an inefficient distribution may prevent the application of the DPM technique of powering off the idle servers. On the other side, it may cause the fragmentation of the virtualization resources, that may prevent from hosting a VM with specific requirements, even when the platform has enough resources. In this chapter, the *Virtual Machine Consolidation Agent (VMCA)* is described. This agent is in

charge of monitoring the Cloud deployment, and to analyze whether it is possible to re-arrange the existing VMs to achieve a more efficient distribution. The underlying objective is to consolidate the VMs into a reduced number of physical servers in order to be able to power off the idle servers. The main contribution for this paper is the concept of VMCA, the algorithms described on it, and the usage of the agent to facilitate the automated power management, together with the design of the use case and tests.

Chapter 6 is dedicated to explore the application of DPM techniques at a VM level, to facilitate automated power management in an on-premises Cloud. In this paper it is described *Cloud Virtual machine Automatic Memory Packer (CloudVAMP)*, a memory overcommitment framework that can be integrated in an on-premises Cloud to automatically monitor the VMs and to dynamically adjust their allocated memory to adapt to the current memory requirements of their running applications. This enables an additional step in VM consolidation per physical host as the amount of requested virtual memory may exceed the physical memory. In case the VMs claim part of the memory that they requested when they were created and the sum of the virtual memory of the VMs in one host exceeds the physical memory, it uses live migration to mitigate the problem. The main contribution for this paper is the collaboration in the design of the architecture and the workflow of CloudVAMP, and the implementation of part of the tool.

The chapters correspond to academic papers, so they are self-contained in the sense that each of them include a part dedicated to revise the state of the art in the particular problem that is addressed by the chapter, a discussion about the contributions and conclusions about the obtained results. The chapter 7 includes a discussion on how the results obtained in the framework of this thesis can be used together to provide elasticity to a data center, at different levels. Then, it is included a summary of the results achieved in this thesis, and collection of the publications generated during the research phase, along with the end-products that have also been generated, where the results of the research have been incorporated. At the end of this chapter, it is discussed about the future directions of this work and the plan for the developments of the generated products.

Finally chapter 8 is dedicated to draw some conclusions about the work that has been carried out during the research period and how the objectives for this thesis have been addressed.

Chapter 2

An Energy Management System for Cluster Infrastructures

Published as

Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, Vicente Hernández, An energy management system for cluster infrastructures, Computers & Electrical Engineering, Volume 39, Issue 8, November 2013, Pages 2579-2590, ISSN 0045-7906, http://dx.doi.org/10.1016/j.compeleceng.2013.05.004.

Abstract

This paper presents a general energy management system for High Performance Computing (HPC) clusters and cloud infrastructures that powers off cluster nodes when they are not being used, and conversely powers them on when they are needed. This system can be integrated with different HPC cluster middleware, such as Batch-Queuing Systems or Cloud Management Systems, and can also use different mechanisms for powering on and off the computing nodes. The presented system makes it possible to implement different energy-saving policies depending on the priorities and particularities of the cluster. It also provides a hook system to extend the functionality, and a sensor system in order to take into account environmental information. The paper describes the successful integration of the system proposed with some popular Batch-Queuing Systems, and also with some Cloud Management middlewares, presenting two real use-cases that show significant energy/costs savings of 27% and 17%.

2.1 Introduction

One of the challenges arising from the use of HPC clusters is reducing their power consumption. This problem is especially important in clusters that are underutilized, either because they form part of large scale distributed systems (grids or clouds) [76], where load can have important variations, or because the clusters have been in production for several years and their usage has decreased in favour of other more modern systems. However, in the last years there have been advances in the energetic efficiency of HPC clusters, which have come as a result of two different approaches: Static Power Management (SPM) techniques that use low-power energy-efficient hardware to reduce energy usage, and Dynamic Power Management (DPM) techniques that are based on the knowledge of resource utilization and application workloads to reduce energy usage [121].

In the case of SPM there are efforts pursuing higher efficiency for power sources [79], [63], which is usually lower than 80%. The hardware designers are also introducing new types of memory to increase the efficiency and reduce consumption. New technologies, such as Solid State Drives (SSD), are also being adopted for disks in order to reduce the energy consumed by mechanical parts, which accounts for up to 65% of the total amount of energy consumed by a computer [58]. Dynamic Voltage and Frequency Scaling (DVFS) is an efficient technology to control the processor power consumption [74].

The DPM approach takes advantage of the fact that many computing nodes that are part of infrastructures such as clusters are usually powered on even when they are not being used (e.g. the workload is low, some computing nodes are not suitable for current calculations, there are reserved nodes for priority users, etc.). These clusters are usually dimensioned for peaks of workload that are not the most common situation. Therefore, energy can be saved by putting the idle nodes into power-saving mode (e.g. turning nodes off). There are different mechanisms that may be used to power on or off the nodes depending on the workload, that go from managing power by hand (e.g. powering off part of the nodes when they are not going to be used for a period of time) to introducing automated mechanisms into the job submission tools (e.g. monitoring a queue of jobs and powering off the computing nodes when the queue is empty).

A further step in automating the power management of nodes is to use energy-aware scheduling/allocation algorithms for assigning resources to jobs. For instance, schedulers may try to use the minimum number of computing nodes, in order to enable energy reduction by powering off the idle nodes. However, implementing an energy-aware allocation method in existing clusters of an organization is a difficult task, since it is necessary to modify the scheduling code of the resource management middleware. Even if the source code is available, modifying it

can be a complex task, and maintaining modification through new releases of the middleware makes it even worse.

In this context, this paper presents CLUES (Cluster Energy Saving System), which is a general power management tool for computer clusters that can work in connection with different resource management middleware by means of easy-to-develop connectors. Thus, the tool is an effective way to implement power management policies in existing clusters, without having to modify the underlying control middleware. It could even be used in multipurpose clusters where different management middlewares coexist, thus enabling cluster-wide energy management policies for those situations. While CLUES was previously introduced in [6], this paper describes the tool in more depth, and provides details about the connectors that are currently available for the interaction with existing cluster management systems. It also presents some other features such as a hook system and a sensor system. A discussion is provided about the algorithm that is considered in CLUES for the power management of the nodes, and how it behaves when it is applied to real computing infrastructures that are currently under production.

The remainder of the paper is structured as follows. First, section 2.2 presents the general power management approach followed, and section 2.3 analyzes related work. Then, section 2.4 presents the architecture of CLUES and describes all its components. Section 2.5 discusses the features of CLUES to support more than one LRMS coexisting in the same cluster. Section 2.6 presents an extended analysis of results to demonstrate the proper interaction of CLUES with the underlying system. Finally, section 2.7 provides conclusions and points to future work.

2.2 Power management approach

Current clusters are usually managed by a Batch-Queuing System (BQS) or, in the case of Cloud Computing, a Cloud Management System (CMS). From now on this middleware will be referred in general as Local Resource Management System (LRMS) or resource manager. Examples of BQS are Torque/PBS[1], SLURM[2], Son of GE[3]. Examples of CMS are OpenNebula[4], OpenStack[5] or CloudStack[6].

There are two alternatives to provide an energy saving mechanism based on powering off idle nodes: (a) modify the LRMS scheduler, or (b) treat the scheduler as a black box (BB) and connect it to some energy saving system that powers nodes on/off as needed.

[1] http://www.clusterresources.com/products/torque
[2] https://computing.llnl.gov/linux/slurm
[3] https://arc.liv.ac.uk/trac/SGE
[4] http://www.opennebula.org
[5] http://www.openstack.org
[6] http://cloudstack.org

Modifying the scheduler may achieve better results, but presents the disadvantage that it requires the creation of a modified version of the original scheduler, and the new versions released by the developers of the LRMS will also need new modifications. Moreover, the power schedule mechanism would be tied to the specific LRMS.

On the other side, a BB approach implies that the LRMS must contact the energy saving system to provision the resources needed by the jobs. It requires some degree of coordination between the job scheduler and the energy saving system, i.e. the energy saving system should not power off a node if that node is useful from the point of view of the scheduler, and conversely, a node that is not useful from the point of view of the scheduler should be powered off to save energy. A BB approach may not provide the best results because the energy saving system does not have the whole information about the workload, and does not control in which nodes the jobs are allocated. However, decoupling the scheduling of jobs and the decision of suspending or restoring nodes eases the incorporation of energy-saving policies in production clusters, since there is no need to modify the resource manager.

This paper considers a BB approach, where the resource manager scheduler is connected to an external energy saving system that powers nodes on/off.

2.3 Related Work

In the last years, many efforts have focused on energy-aware allocation of tasks in clusters, both for virtualized and non-virtualized environments. For instance, [76] presents a method to reduce power in large-scale distributed systems by switching nodes on and off according to the load. The approach considers the possibility of reserving resources in advance, and assumes that the duration of a job (or an estimate of it) is provided by the user when submitting the job. The system interacts with the user that submits a job, suggesting job starting times that are most suitable for energy reduction. This work is actually a booking system for computing nodes but not an energy saver. It decides whether a cluster can be powered off because it is not reserved. It does not integrate with the LRMS and therefore it is not suitable for interactive systems.

[22] presents an approach for virtualized data centres which is based on workload consolidation using virtualization, combined with turning off idle servers. The system uses machine learning in order to predict the consequences of different possible allocations for each job, in terms of performance and energy. It then decides task placing and reallocation in order to concentrate jobs in a reduced number of nodes without degrading performance. The paper deals with task placement but not

with infrastructure management. It does not consider integration with the LRMS, and it also assumes that the user provides information on the job duration.

[116] and [24] also deal with the problem of resource allocation in virtualized clusters, considering workload consolidation in order to be able to switch off idle machines, while at the same time reducing the impact on the system performance. The approach uses heuristics based on multicapacity bin packing over memory and CPU load. [74] considers a power-aware scheduling algorithm for DVFS-enabled clusters, where processor frequencies are scaled down in order to minimize power consumed without substantially increasing execution times. [122] describes an approach to load balance Virtual Machine (VM) provisioning across different servers to save energy and to maintain the performance of the system. The underlying idea of such technique is to try to reduce energy consumption even if nodes cannot be idle. [110] explores the combination of using DVFS and putting idle servers into low-power mode, but a workload profiling phase is needed in order to determine the optimal power configuration.

All of the reviewed results are related to the placement of the jobs and virtual machines (VMs), with the idea of either packing the jobs to get some idle nodes, or altering processor voltage to get less power consumption. However, they do not describe how to manage the idle computing elements. According to [21], one important research topic for getting energy efficiency by applying DPM techniques is to schedule powering on and off computer's components (the whole server in most cases) to adapt to the workload. The survey [121] also explains some DPM works from other authors that would get idle resources, but the reviewed works usually assume that those idle resources are powered on or off automatically and do not consider any scheduling strategy. Most of them are basically job schedulers that would substitute the existing schedulers or cluster management middlewares and would obviously modify the way that users interact with them.

There are many other scientific works exploring this area. However, from the point of view of system administrators, there are not many available tools to implement green policies in clusters. In the case of BQS schedulers, MOAB (which is the Enterprise version of Maui) introduces some features to pack workload and to place idle servers in power-saving modes [111]. The latest versions of SLURM introduced the ability to change CPU frequency and voltage in order to save energy. However, these solutions are of course tied to a particular BQS. Since the choice of BQS is conditioned by many factors, administrators may find that the most suitable BQS does not take into account energy saving mechanisms. In the case of cloud middleware, Convirt 2.0 Enterprise Edition introduces scheduling policies to consolidate VMs to enable the operation of the datacenter in power saving mode, but it does not provide tools to automatically power off idle nodes. VMWare vCenter includes tools to power off hosts when they are not needed. Other cloud middleware do not take into account power consumption.

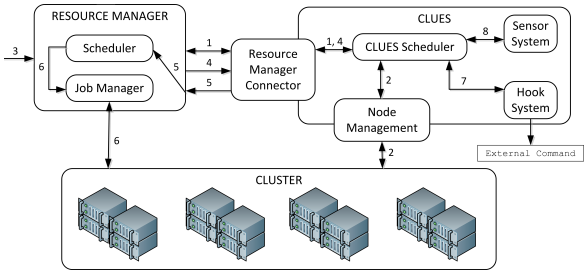


Figure 2.1: Architecture of the CLUES system.

2.4 System description

The purpose of the system proposed in this paper is to provide energy saving mechanisms for a computer cluster, by powering off idle nodes, and conversely powering on nodes when required. The system is able to interoperate with different resource management middleware by means of a plug-in based architecture. Using this approach, energy saving can be easily integrated with LRMS, and also with clusters of generic applications such as Web Servers or emerging Platform as a Service cloud systems. The design goals of the system are:

- It must be unobtrusive. From the point of view of the user, the way job submission or VM launching is done should not be altered by the use of CLUES.
- No changes to the underlying LRMS should be necessary to use the system, unless the developer wants to implement specific features or tighten the coordination between the job scheduling and the power management.
- It should be possible to use different mechanisms for switching on/off the nodes, e.g. mechanisms such as Wake-on-Lan (WOL), Power Device Units (PDU), Intelligent Platform Management Interface (IPMI) or infrastructure-specific mechanisms.
- The system should be easy to extend, e.g. adding the capability to use another LRMS, or adding another mechanism to switch on/off nodes.

As depicted in Figure 2.1, the system consists of a *scheduling component*, a set of one or more *resource manager connectors*, a set of *node management connectors*, and the *hook* and *sensor* subsystems.

The CLUES scheduler uses a connector to periodically ask the resource manager for information on the cluster state (label 1 in the figure). Based on this information, the scheduler determines if new nodes must be switched on, or if there are nodes that can be switched off, and acts consequently (2). When a job is submitted to

the resource manager (3), a request for nodes is made to CLUES by means of the resource manager connector (4). When CLUES finishes processing this request, the job is actually submitted to the resource manager (5), where it will be processed by the scheduler and finally sent to the cluster for execution (6). There are two more components that are called periodically by the CLUES scheduler: the hook system, that enables to perform user defined actions when an event happens (7), and the sensor system, that enables access to a set of environmental values to be stored in the scheduler (8).

The *CLUES scheduler* is the main component, and is described in the following section. The *resource manager connectors* provide a uniform way to interact with different LRMS. This mechanism makes it easy to extend the system so as to consider additional resource managers, by writing the corresponding connectors.

The *node management connectors* are responsible for switching on/off the cluster nodes. The method to switch nodes on and off will be different depending on the particular cluster, e.g. WOL can be used for switching on, and a remote "poweroff" command can be used for ordered switching off, or PDUs can be used for both switching on and off. Additionally, in some cases the underlying middleware must be informed when a node is powered on/off, to activate or deactivate the node in the resource manager. By providing several node management connectors, these different situations can be accommodated. Currently, connectors have been developed for three different mechanisms: WOL, IPMI and a proprietary software to manage PDUs used in IBM clusters.

Note that CLUES intercepts any incoming job and retains it while trying to provide resources for it. Once this has been done, the job is released to the LRMS. Importantly, the jobs are released following a FIFO (First In First Out) strategy, therefore preserving the order in which they are taken into account for its execution. Another possible approach would be not to intercept the jobs at all, but instead make periodic inspections of the LRMS queue in order to detect if there is a need to power on additional nodes.

The approach of intercepting the job enables to prepare the context for the LRMS, instead of modifying it once the job has been scheduled under a state that is going to be changed by CLUES. The idea is that when the jobs arrives to the LRMS all the resources needed are already powered on. It also presents the advantage that it provides a faster response, because the need for resources is detected at the moment the job arrives. A disadvantage is that it can introduce small delays in the start of some jobs, if they are submitted shortly after other less-priority jobs which require nodes to be powered on.

In any of the two approaches, CLUES might try to bootstrap a node for a job that, according to the LRMS policies, does not have the right to execute, e.g. because the user has exceeded the execution quota. This can reduce the effectiveness of the

power saving strategy, since there might be more powered-on nodes than necessary. However, this cannot be avoided with a BB approach because it is unaware of the LRMS policies.

2.4.1 CLUES Scheduler

The CLUES scheduler is the component in charge of: (i) processing requests for available resources and powering-on nodes if necessary; and (ii) powering-off idle nodes. To carry out these tasks it performs the following procedure:

1. When a new request for nodes arrives, the request is evaluated in order to determine if new nodes must be powered on for the request. If this is the case, the appropriate actions are taken. CLUES has a synchronous behavior, blocking the request and appending it to a list of pending requests while the necessary nodes become ready.

2. Periodically, the state of nodes is updated according to the information provided by the resource manager connectors. After each update, the queue of pending requests is examined. Each request is evaluated again and the necessary power-on actions are taken. If the request is at the head of the queue and the corresponding nodes are ready, it is removed from the queue and released so that the associated job can proceed to its execution. Note that a request can be released either because there are enough free nodes, or because there are no more nodes that can be switched on. In both cases, no further action can be done for the request.

   In addition to examine the queue of pending requests, idle nodes are detected and they are powered off if the inactivity time is larger than a predefined value.

Different policies can be used in order to determine if new nodes must be powered on to serve a request, each of them producing a different effect on desirable objectives: minimizing the power consumption, minimizing the impact on the users, minimizing the heat dissipation, etc. The selection of the policy is an important decision to obtain the desired behavior of the cluster. CLUES implements a set of basic policies:

- The most simple one is to switch on all the nodes of the cluster when a job arrives to the system. This is a coarse strategy but it is very simple to implement and can obtain good results with some specific workloads (e.g. large waves of jobs and long inactivity periods) and with clusters where powering off some nodes may affect the network topology and the connectivity of the remaining nodes.

- Switch on the minimum number of nodes to fulfill the request needs. This strategy enables minimum power consumption, but may increase the waiting time of incoming jobs.
- Switch on the nodes using a block size: instead of powering on the exact number of needed nodes, this strategy powers on an extra number of nodes, thus providing extra spare idle nodes that may prevent subsequent requests from waiting.

Obtaining the number of nodes available for a request

In order to apply any of the last two strategies, the scheduler must obtain the number of nodes available for a request, taking into account the following considerations:

- The process of booting up a node takes some time, during which the request will be queued until the nodes are ready. This means that when a new request arrives, there can be previous pending requests, and there can be nodes booting up.
- A node can be shared by more than one job, e.g. a node typically contains several cores, so it is possible to assign some of the cores to a job and other cores to another job. Consequently, a node is considered to contain a number of processing units or "slots" (e.g. cores), and a request asks for "virtual nodes", which are groups of slots in the same physical node.
- Requests can be made for nodes meeting certain conditions, e.g. nodes belonging to a particular batch queue, nodes with a given minimum amount of memory, installed software or configuration.

In order to determine the number of available nodes, the scheduler needs to use information about the current request, such as the number of requested virtual nodes (rv), number of slots per virtual node (sv), and possible conditions on the nodes. It also needs information about previous requests that are waiting for resources to become available, such as the total number of requested slots (trs). Finally, information on the cluster nodes is also necessary. For each node i, the scheduler needs to know its state (e.g. on, off, booting, failed...), number of free slots (fs_i) and total slots (s_i), and other information (e.g. amount of free memory, administrator-defined tags...).

Based on this information, the scheduler performs two steps:

1. Determine the number of virtual nodes that are usable by the current request, without taking into account previous requests. A virtual node is considered usable if it is located in a node that satisfies the conditions of the request,

and its slots are not in use. The process followed can be seen in algorithm 1, which obtains the number of usable virtual nodes in powered-on nodes (uv_{on}) and in booting nodes (uv_{bt}). The algorithm goes through all the nodes that satisfy the conditions of the request, and for each of them the number of virtual nodes provided is obtained and accumulated (e.g. if a node has 5 free slots and the request asks for virtual nodes of 2 slots, 2 virtual nodes are provided).

Algorithm 1 Computing the number of usable virtual nodes.

```
{uv_on: number of usable virtual nodes in powered-on nodes.}
{uv_bt: number of usable virtual nodes in booting nodes.}
{s_i: total number of slots in node i}
{fs_i: number of free slots in node i}
{sv: number of slots per virtual node of current request}
uv_on ← 0; uv_bt ← 0
for all node i that matches current request do
  if state_i = on then
    uv_on ← uv_on + ⌊fs_i/sv⌋
  else if state_i = booting then
    uv_bt ← uv_bt + ⌊s_i/sv⌋
  end if
end for
```

2. Correct these numbers of usable virtual nodes, by taking into account previous requests, that may take some of the slots of the usable virtual nodes. Since the allocation of the requests to particular nodes/slots is decided later at the LRMS level, the corrections are based only on estimations. The resulting numbers are referred to as uv'_{on} for powered-on nodes and uv'_{bt} for booting nodes. Best-case estimates are derived, based on simplifying assumptions. First, the details of previous requests are not taken into account, and only the total number of previously requested slots (*trs*) is used. Second, it is assumed that these slots will be placed preferably in nodes that do not satisfy the conditions of the request. Thus, previous requests will produce minimum disturbance. According to this, uv'_{on} is:

$$uv'_{on} = min(max(0, \lfloor \frac{tfs - trs}{sv} \rfloor), uv_{on}) \tag{2.1}$$

where *tfs* is the total number of free slots.

If $tfs \geq trs$, uv_{bt} is not corrected ($uv'_{bt} = uv_{bt}$). Otherwise, the previous requests may also use booting nodes, and uv_{bt} is corrected accordingly:

$$uv'_{bt} = min(max(0, \lfloor \frac{tfs + tbs - trs}{sv} \rfloor), uv_{bt}) \quad (2.2)$$

where tbs is the total number of slots in booting nodes. Once uv'_{on} and uv'_{bt} have been obtained, the sum of them is computed to get the estimated number of virtual nodes available for the request.

An example In order to illustrate the procedure described above, a cluster is considered with 20 nodes of 4 slots each. There are 2 completely free nodes, a node with one free slot and a node with 3 free slots. 2 other nodes are booting and the rest are switched off. There is a pending request which asked for 7 virtual nodes of 2 slots each, and in this context a new request arrives for 4 virtual nodes of 2 slots each. According to algorithm 1, the number of usable virtual nodes are computed, obtaining $uv_{on} = 5$ and $uv_{bt} = 4$. Then, pending requests are taken into account as explained in step 2. Taking into account that $tfs = 12$, $trs = 14$ and $tbs = 8$, equations (2.1) and (2.2) yield:

$$uv'_{on} = min(max(0, (12 - 14)/2), 5) = 0$$
$$uv'_{bt} = min(max(0, (12 + 8 - 14)/2), 4) = 3$$

This shows that the current request can get only 3 virtual nodes from currently booting nodes. Since it needs 4 virtual nodes, more nodes have to be powered on.

Selecting the nodes to be powered on

The next step is to select which nodes, of the list of nodes that match the request, will be switched on. There are also different strategies:

- Homogeneous Clusters: In this case, basic strategies such as selecting the nodes using a fixed order or a random algorithm are good solutions, as all the nodes provide the same features to all the jobs.
- Heterogeneous Clusters: In this case, more advanced strategies can provide advantages by selecting the nodes according to different node features: performance, power consumption, heat dissipation, etc. It is also possible to use a combination of some factors, e.g. selecting the nodes with the best ratio of performance / power consumption. In order to realize these strategies, CLUES must obtain additional information about the nodes (e.g. performance or power consumption). Currently, the information must be provided by the system administrator using a set of static files, but CLUES is prepared to use in the future some sensors or systems such as IPMI that can provide the information automatically.

Powering off idle nodes

Another task to be done after each state update is to detect idle nodes that can be switched off. The time of inactivity used to power off the nodes must be specified by the system administrator. It is important to correctly select this time to obtain good results with CLUES. Using a short time may reduce the power consumption, but it can also increase the number of jobs having to wait, and the number of power on/off operations. On the other hand, using a long time will produce opposite results.

Some other factors are also considered when powering off nodes. In some cases, because of the particularities of the hardware or the network topology, it is required that some of the nodes (or all of them) remain powered in order for the cluster to work properly (such as in the first cluster shown in the results evaluation section).

Re-evaluation of Jobs

A re-evaluation mechanism has been implemented, by means of which the queue of the LRMS is periodically inspected, identifying jobs that have remained queued for a specified amount of time. For each of those jobs, a request for nodes is sent again to be re-evaluated by CLUES. This mechanism is introduced to correct some possible undesirable conditions that arise when following a black-box approach (e.g. a node may be powered off while a job that has not enough nodes is in the queue). CLUES processes re-evaluation requests just like ordinary requests, checking the resources needed by the job and switching on nodes if necessary.

2.4.2 Resource Manager Connectors

The *resource manager connectors* provide a uniform way to interact with different LRMS. Each connector consists of two parts.

The first one is a monitoring system, that obtains information about the nodes of the LRMS and presents it in a uniform way. The monitoring system connectors are implemented as external executable files, which can be created using any programming language. The connectors get the information directly from the LRMS and publish it as a list of key-value pairs separated by semicolons, with one line for each node, e.g.

```
host=node1;state=down;total_slots=2;free_slots=2;keywords=ok;
     queues=sci
host=node2;state=free;total_slots=2;free_slots=1;keywords=ok;
     queues=sec,sci
...
```

There are only four mandatory fields: `host`, `state`, `total_slots` and `free_slots`. The rest of fields depend on the type of LRMS used. For example the `queues` field is used in the batch systems but not in the cloud ones.

The second part is a job interceptor, that comes into action whenever a new job is to be submitted to a LRMS. Before the job is actually submitted, the connector requests the necessary resources to the CLUES scheduler. When a response to the request is received, the job is submitted to the LRMS.

Batch system connectors

Torque/PBS and SGE, two of the most popular queue systems, have been considered here and a connector has been implemented for each of them.

As mentioned above, one of the functions of a connector is to provide information about the node states. In the case of PBS, this information is extracted by using the command `pbsnodes`, and in the case of SGE with the `qhost` command. In both cases, an option of the command is used in order to get the output in XML format, which can be parsed more easily.

The other function of the connector is to catch incoming job submissions, in order to request the corresponding resources to the CLUES scheduler. PBS provides a feature known as "job submission filter" (or "qsub wrapper") which is useful to intercept the submission of jobs. A similar feature called "job submission verifier" has been used in the case of SGE. By means of these features, a script can be specified to be run before the effective submission of a job into the queue system. In this case, the script must first determine relevant information of the job being submitted (such as the number of required virtual nodes, the number of slots per virtual node or the queue name), then send a request for nodes to CLUES, and wait for a response. When a response is obtained, job submission can proceed.

Cloud system connectors

In the case of CMS, OpenNebula and OpenStack connectors have been developed. The OpenNebula connector intercepts the creation of VMs by a mechanism provided by the middleware called "hooks". Such mechanism enables the execution of an application whenever a VM is created, and it is used to ask CLUES for working nodes. The OpenStack connector does not provide any similar feature, and it was necessary to modify one file of the middleware API to connect to CLUES.

The result is that each newly created VM is held while CLUES decides whether extra nodes should be powered on, and in such case, while the nodes are booted. If the VM were released before the node being ready to accept VMs, the scheduler

Hook Name	*Description*
POWEREDON, POWEREDOFF	Before powering on/off a node
POWEREDON UNEXPECTED, POWEREDOFF UNEXPECTED	When CLUES detects that a node has been powered on/off unexpectedly
MONITORING, MONITORED	Before or after the monitoring procedure
ENABLED, DISABLED	Once a node has been enabled or disabled
SENSOROVER, SENSORBELOW	The value of a sensor is over or below a threshold

Table 2.1: Hook types

might try to assign it to a working node that does not have enough resources. While the VM is retained, CLUES tries to make its best for provisioning resources.

The information about the hosts is extracted in both cases using the corresponding internal API for direct access. The main issue in the OpenNebula case is that the information provided about the hosts is not enough for the CLUES scheduler, regarding both the memory and the virtual CPUs booked: the internal information system tracks the number of VMs that are running in a particular host (but not the virtual CPUs), and the remaining free memory that is reported by the operating system (that considers swap memory as real memory). The workaround has been to extract the information from the description of the VMs.

2.4.3 Hook system

The hook system enables the extension of the functionality of CLUES without the need to modify the source code. It specifies user defined actions (e.g. custom scripts) to be executed before or after some event happens (eg. a node is being powered on). The user must provide an executable file that receives as a parameter some value related with the hook event. The events considered in the hook system are shown in table 2.1.

This system also covers the CLUES monitoring system, to enable tasks to be performed each time the CLUES scheduler monitors the system, or some measures to be taken when a particular state is detected, e.g a message can be sent to the system administrator when a node does not power on correctly or when it powers off unexpectedly.

2.4.4 Sensor System

Nowadays it is quite common for the clusters to be monitored using some kind of sensors to know some environmental parameters. Typical examples are the temperature or the humidity. In some cases these sensors are managed by a piece of software (e.g. Nagios) that can send notifications to the administrators to take corrective measures.

A sensor system has been included in CLUES, enabling access to environmental information, which can be used by the hook system to take automatic corrective measures. CLUES can call periodically a set of sensor plugins (typically scripts) that return a set of key-value pairs with the name of the parameter and the value measured by the sensor. These values are stored and it is possible to configure the system so that actions are taken whenever the value of any parameter is over or below a given threshold.

In particular, the hook system can be used in order to take an action when the parameter values are out of the specified limits. To implement this feature a new type of hooks has been added to the scheduler where the user must define an upper and/or lower limit for a measured value in the sensor system, and a command that must be executed when the "exception" happens. The executed command will receive as a parameter the string with the key-value pair obtained by the sensor system. Corrective measures could be e.g. powering off the idle nodes, or even powering off all the nodes, or, if a software provides the functionality, sending a signal to switch on or off the air cooling system.

2.5 Mixed cluster

It is not very frequent to have more than one LRMS coexisting in the same cluster. However, with the advent of cloud management systems, this option is not unreasonable. Additionally, in some clusters used for testing purposes, it makes sense to have two LRMSs installed, such as PBS and SGE. CLUES has been designed to support this kind of mixed clusters, making it possible to manage nodes shared by two or more LRMSs.

Although one node can be shared by different LRMSs, the number of slots must be divided among them. For example, if a node has 6 slots, one LRMS could be using 2 of them and another one could use 4. It is a task of the system administrator to configure the LRMSs properly.

The CLUES scheduler can manage a list of nodes included in each of the configured LRMSs, storing the state and the features provided by the different connectors. When processing an incoming request, the scheduler checks for available nodes only within the list of nodes of the LRMS corresponding to the request. When selecting

the nodes to switch off, the scheduler must check the combined information about all the nodes to select only the nodes that are considered idle in all the LRMSs.

2.6 Results Evaluation

In a previous work [40] of the authors, an analysis was made of the jobs launched to the Torque/PBS LRMS of a HPC cluster, in order to have an estimate of the benefits of applying green computing techniques to that cluster. Now the first version of CLUES has been developed and it has been installed and working during seven months in two different clusters. During this time period an evaluation, using the current real workload of the clusters, has been made of the software behaviour, and of the real impact on the power consumption and on the cluster users. This evaluation was also useful to detect some aspects that were not initially considered but are important in a production version.

2.6.1 Cluster 1

The tests have been performed in a cluster composed of 51 bi-processor nodes with Intel Xeon CPUs at 2.80GHz, interconnected by a SCI network in a 10x5 2D torus topology. Each node has 2 GB of RAM memory. The front-end node is the access point to the cluster, and the other (50) are used as the working nodes. This cluster is configured with a NFS system that is exported by the front-end node and accessed by the computing nodes.

This cluster is used as a development and private testbed platform for parallel and sequential high performance applications . The cluster is typically used to execute both sequential and parallel CPU-intensive applications. During the seven months considered for the evaluation, the system was used normally, with a total of 20,497 jobs submitted. 41% of the jobs were parallel and used an average of 14 nodes. The average time per job was 14 hours, 41 minutes and 14 seconds.

A clamp meter was used to get the power consumption of each component of the whole rack, and the corresponding data are shown in the rightmost column of table 2.2. In particular, power consumption has been obtained for three different states of a cluster node: switched off ("N. off"), switched on but idle ("N. idle"), and fully used ("N. used"). Finally, the entry "Other" refers to the power consumption of the essential components of the cluster (front-end node, switches, KVMs) that are always on.

Based on the analysis made in [40], a period of inactivity of 2 hours was considered in order to switch off idle nodes. This period of time is the appropriate for the

[7] Cost: 0.091 €/kWh. Data obtained from the Ministerio de Industria, Turismo y Comercio del Gobierno de España

	Using CLUES			Not using CLUES (est.)			Consumption per node
	PCT	**kWh**	**€**[7]	**PCT**	**kWh**	**€**	**W**
N. Off	45.6%	350	32	0.0%	0	0	3
N. Idle	4.9%	1,624	148	50.4%	16,234	1,477	130.9
N. Used	49.6%	26,051	2,371	49.6%	26,051	2,371	205.4
Other	100%	10,296	973	100%	10,296	973	2,012
TOTAL		38,321	3,487		52,581	4,785	

Table 2.2: Cluster 1 power consumption and cost

deployment of the use-case, but it should be adjusted according to the features of the actual deployment in which CLUES is used (the usage pattern of the cluster, the power consumption for the nodes, etc.). The fact that the SCI network has a 2D torus topology implies that a message from a node A to another node B can be routed through other intermediate nodes. Thus, these intermediate nodes should not be powered off even if they are idle. In order to tackle this problem, the whole cluster is kept switched on whenever a parallel job (using more than one node) is running.

Figure 2.2 shows an evolution over the time of the number of requested slots in the LRMS, and of the number of used and idle slots in the cluster. In the figure, a slot is marked as used not only when a job is using it, but also when the slot is part of a node that has at least one used slot. In this case, "used" means the slot cannot be switched off. The vertical axis has been truncated to 100 to remove peaks of requested slots that would make it difficult to see the figure. The period of time considered in the figure has been reduced to the first two months, also for the sake of clarity. In this first case there is a clear correlation between the number of requested slots in the system and the number of nodes switched on by CLUES. The figure shows that one node is always powered on, because this node had some problems with the WOL configuration. Near the end of the two-month period (about days 47 - 50) there is a peak where the whole cluster is switched on with a reduced number of requested slots. This is produced by some parallel jobs requiring all the cluster to be switched on due to the commented network topology restrictions.

Table 2.2 shows the results of energy and money spent during the considered period, in both the cases of using CLUES and not using it. The left part of the table contains the data for the case of using CLUES. The first column (titled "PCT") represents the percentage of time a node spent on average in each state. The second column (titled "kWh") represents the total amount of energy consumed by the specified components of the cluster, expressed in kilowatt hours. Last column (titled "€") contains the amount of money dedicated to those components. The center part of the table corresponds to the estimation of the energy and money

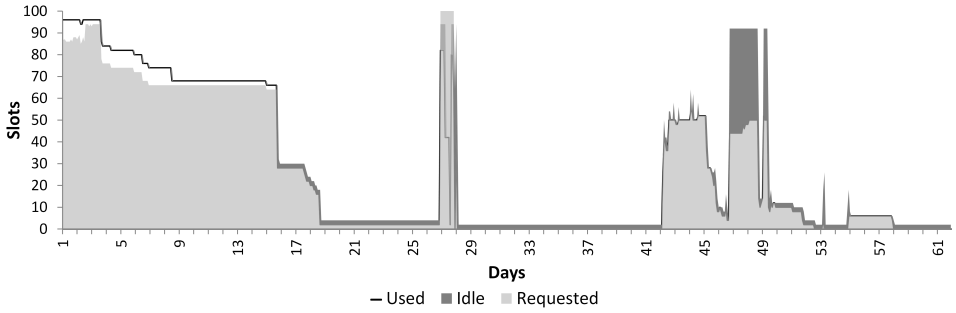


Figure 2.2: Evolution of the number of used, idle and requested slots in cluster 1

spent if the CLUES system had not been used, presenting the same columns as the left half. The energy consumption without CLUES has been estimated by changing all the accumulated time of nodes in "Off" state to the "Idle" state, because without CLUES these nodes would have never been switched off.

Table 2.2 shows that the total amount of energy saved is 14,260 kWh, which represents 27.1% of the total amount of energy, but also means saving 1,297 €.

On the user impact side, an analysis has been made of the number of jobs that needed to wait to access the resources. During all the period, 268 jobs had to wait for some node to be switched on (1.31% of total jobs). The average waiting time for these jobs was 1 minute and 40 seconds. This is a short enough waiting time, considering that the average time per job exceeded 14 hours.

Another important issue is related with the number of switch-on/off cycles performed in the cluster nodes. These operations can damage the hardware (mainly the disk drives) and may cause consumption peaks that could increase the total power consumption. During the evaluated period, an average of 38 switch-on/off cycles were performed for each node, with a maximum of 54 cycles. This means that a node completes a switch-on/off cycle once every 6 days on average, with a maximum of once every 4 days.

2.6.2 Cluster 2

The CLUES system has also been tested in a cluster composed of an M1000e blade server chassis with 6 Dell M610 and 3 Dell M910 nodes. Each M610 node has two quad-core Intel Xeon E5620 processors, making a total of 8 cores and 16 GB of RAM per node. The M910 node has four quad-core Intel Xeon E7520 processors, with a total of 16 cores and 64 GB of RAM per node. The cluster uses Torque/PBS and a NFS system is exported by the front-end node and accessed by the computing nodes.

	Using CLUES			Not using CLUES (est.)			Consumption per node
	PCT	**kWh**	**€**	**PCT**	**kWh**	**€**	**W**
N. Off	39.3%	181	16	0.0%	0	0	9
N. Idle	7.8%	363	33	47.1%	2,853	260	65
N. Used	52.9%	7,407	674	52.9%	7,407	674	187
Other	100%	3,070	279	100%	3,070	279	600
TOTAL		11,021	1,003		13,331	1,213	

Table 2.3: Cluster 2 power consumption and cost

This cluster is used in a production grid environment, as one of the computing nodes of the Spanish National Grid Initiative[8] in the European Grid Infrastructure[9]. The cluster is typically used to execute high throughput applications launching sequential jobs. There is a wide range of different applications with different behavior and requirements in terms of CPU, memory and I/O access patterns. In particular, the workload of the system during the evaluation period was composed of a total of 107,197 jobs, 13% of which were parallel and used an average of 2.28 nodes. The average time per job was 2 hours, 39 minutes and 20 seconds. In contrast to the previous case, this cluster has no network restriction and the nodes can be switched on individually. A time of 30 minutes has been selected as the time of inactivity to power off the nodes.

Figure 2.3 shows an evolution over the time of the number of requested slots and the number of used and idle slots in the cluster. As in the previous case, the "used" state represents the slots that cannot be switched off. The vertical axis has been truncated to 200 to remove peaks of requested slots that would make it difficult to see the figure. As in the previous case, the period of time for the figure is two months. The correlation between the number of requested slots and the number of switched-on nodes is not as clear as in the previous case. The main reason is that there are heterogeneous multicore nodes (with 16 or 32 slots per node), and the job distribution among all the nodes depends on many factors: the LRMS scheduler, the finalization of the jobs, the arrival of new jobs, etc. The LRMS can be configured to pack the jobs in the minimum number of nodes, but other factors cannot be controlled. It is possible, for instance, that only 9 jobs requesting 1 slot each, end up keeping all the nodes switched on.

There are also some restrictions in the LRMS such as a maximum of 40 running slots for each user group. This issue explains the behavior of the system about the days 3 - 5 and 55 - 59, where the number of requested slots is bigger than the number of used slots and no new nodes are switched on.

[8] http://www.es-ngi.es
[9] http://www.egi.eu

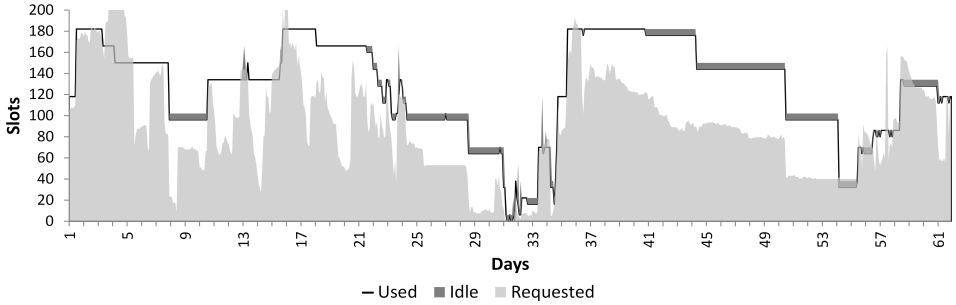


Figure 2.3: Evolution of the number of used, idle and requested slots in cluster 2

Table 2.3 shows the economic and energetic saving obtained by using the green computing software. The left part of the table shows the power consumption using CLUES, and the center part shows an estimation of the power consumption without CLUES. The M1000e chassis has a complete set of energy management tools to monitor the power consumption of the whole system and the individual blades. These tools have been used to obtain the power consumption to perform this study. The rightmost column of the table shows the power consumption of one blade system in different states: switched off, switched on but idle, and fully used, with the maximum number of jobs running. In this case the "Other" row corresponds to the chassis.

The estimated economic saving is 210€, which means a reduction of 17.3% of the total expenses. Unsurprisingly, the impact of the application of green measures in this cluster is lower than in the previous case. The main reason is that, as a production node of the EGI infrastructure, the cluster is periodically receiving jobs in order to monitor the status of the system. These monitoring jobs cause that at least 2 of the 9 nodes are always on. Other reasons are the important power consumption of the chassis compared to that of the 9 nodes, and the fact that the number of cores per node in this case is larger, which makes it easier for the nodes to be only partially used.

On the user impact side, 2.9% of the jobs had to wait for some node to be switched on, with an average waiting time of 1'54". These are short enough values, considering that the average time per job exceeded 2 hours.

The average number of switch-on/off cycles for a node was 38, and 62 for the node with the maximum number of these operations. It means that a node completes a cycle once every 6 days, with a maximum of once every 4 days, that are very low ratios.

2.7 Conclusion and Future Jobs

The proposed CLUES tool is an energy manager for both HPC clusters and cloud infrastructures, that is able to power off the nodes when they are not being used, and power them on when they are needed. CLUES considers the underlying LRMS as a BB. The advantage of this approach is that it can be integrated with different resource management middleware, without needing any modification of that middleware. Because of this flexibility, it can be used both for HPC clusters and for cloud infrastructures. It can also be used with multipurpose clusters where different management middleware coexist, thus enabling cluster-wide energy management policies. Additionally, it considers different mechanisms for powering on and off the cluster nodes. The performance of CLUES is shown with two real use-cases that show significant energy and cost savings of 27% and 17%.

Future directions of work include the introduction of modifications to the CLUES scheduler, the use of alternative energy saving mechanisms such as DVFS, or the use of other heuristic methods which may take into account prediction of performance and energy consumption. At the same time, the integration with other middlewares such as Eucalyptus or CloudStack is an ongoing work.

Another important issue to be considered in the future is the impact of CLUES in systems using some kind of parallel file system like Lustre, GFS, GlusterFS, etc. This kind of systems supports data replication, making it possible to switch off some nodes of the infrastructure without losing access to the data. Configuration issues imposed by this kind of systems must be analyzed, as well as the impact of switching off nodes on the data access performance.

Finally, CLUES also opens possibilities for research in the field of scheduling policies for powering on and off the working nodes in multi-purpose clusters governed by several coexisting middleware, with the aim of reducing energy consumption.

Chapter 3

An Economic and Energy-Aware Analysis of the Viability of Outsourcing Cluster Computing to the Cloud

Published as

Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, Germán Moltó, An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud, Future Generation Computer Systems, Volume 29, Issue 3, March 2013, Pages 704-712, ISSN 0167-739X,
http://dx.doi.org/10.1016/j.future.2012.08.014.

Abstract

This paper compares the total cost of ownership of a physical cluster with the cost of a virtual Cloud-based cluster. For that purpose, cost models for both the physical cluster and the cluster on the Cloud have been developed. The model for the physical cluster takes into account previous works and incorporates a more detailed study of the costs related to energy consumption and the usage of energy saving strategies. The model for the cluster on the Cloud considers pricing options offered by Amazon EC2, such as reserving instances on a long-term basis, and also considers using tools for powering nodes on and off on demand, in order to avoid the costs associated to keeping idle nodes running. Using these cost models, a comparison is made of physical clusters with Cloud clusters of a similar size and performance. The results show that Cloud clusters are an interesting option for start-ups and other organizations with a high degree of uncertainty with respect to the

computational requirements, while physical clusters are still more economically viable for organizations with a high usage rate.

3.1 Introduction

One of the main problems faced when deploying a cluster of PCs relates to the high Total Cost of Ownership (TCO). This cost involves not only the purchase and installation of the equipment (computational nodes, network components, cables, hard disks, etc.), but also the operating costs. The latter includes the salaries of the personnel in charge of the installation and maintenance, the electricity consumed, and the costs related to rent appropriate housing and its associated cooling systems. The problem is that the usage patterns of these machines are highly dynamic, where peak loads are often restricted to the context of specific experiments or deadlines. In addition to this, the prices of clusters of PCs rapidly decrease (due to the technology obsolescence), thus reducing the value of the initial investment in hardware.

As an alternative, researchers might access the resources at the Computing Centers of national or international institutions. This is the case of the *Spanish Supercomputing Network*, which aggregates several supercomputing centers in Spain. The access to this equipment is supervised by an access committee that grants limited resource access according to the scientific merit of the proposals. Another example is the not-for-profit organization *Partnership for Advanced Computing in Europe* (PRACE), which provides access to a world class computing and data management infrastructure [108].

Another alternative is to use *Cloud computing*, a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [88]. This is the case of Infrastructure as a Service (IaaS), which performs on demand resource provision of computational resources, storage capacity, network access, etc. This is achieved by means of virtual machines that run on the Cloud provider's computing center. In the case of public Cloud providers, a pay-per-use pricing model is typically employed, where users are only charged for the resources that they have consumed.

Virtualization was not considered a feasible approach for High Performance Computing (HPC), due to the penalties involved mainly in the I/O. However, many applications running on PC clusters are CPU-bound, thus their performance is hardly affected by virtualization [100]. In addition, certain public Cloud providers, such as Amazon, offer low latency links among instances, thus leveraging the idea of using virtual clusters on the Cloud. Having a virtual cluster infrastructure on a public Cloud provider has a large number of advantages for the end user, since

no hardware costs are involved. However, the sustainability of this infrastructure in the long term might represent a high cost, since the pay-as-you-go model offered by the Cloud providers implies that a running virtual machine costs money regardless of it being used for computations or not.

Due to the increase of the use of virtualisation and Cloud technologies, some initiatives to create HPC clusters over Cloud infrastructures are emerging. One of the first approaches, described in [23], enabled to launch a fully functional Hadoop cluster over Amazon EC2 using a set of simple scripts[1]. Other tools can create HPC clusters in the Cloud using some kind of Local Resource Management System (LRMS) to manage the jobs. StarCluster[2] uses this approach to create a cluster in the Amazon EC2 infrastructure, with a set of predefined installed applications (Sun Grid Engine, OpenMPI, NFS, etc.) to enable launching parallel jobs to the queue system.

Recently Cycle Computing used Cloud infrastructures to create a 30,000-core HPC cluster using Amazon EC2 standard instances[3]. The cluster ran for about seven hours, with 3,809 compute instances and a total of 26.7TB of RAM and 2PB (petabytes) of disk space, with a 10 Gigabit Ethernet network. Amazon itself actually built a supercomputer on its own Cloud that made it onto the list of the world's Top 500 supercomputers. With 7,000 cores, that specific Amazon cluster ranked number 232 in the world in November 2010 with speeds of 41.82 teraflops, falling to number 451 in June of this year[4]. It is estimated that the whole Amazon EC2 infrastructure can be ranked number 42 among the world's Top 500 supercomputers[5].

These tools and services turn the Cloud into a technologically feasible option for the deployment of clusters of PCs. However, it is important to assess the economic viability of outsourcing the deployment of a cluster on the Cloud, compared to the purchase of a physical cluster. For that, this article analyses and compares the cost of having a physical HPC cluster with that of a similar infrastructure on a public Cloud provider.

The remainder of the paper is structured as follows. First, section 3.2 describes the related work in the literature comparing physical clusters with virtual clusters deployed on the Cloud, either economically or in terms of performance. Then, section 3.3 dissects the TCO of an HPC cluster, introducing an energy-aware cost model for physical clusters, and a cost model for virtual clusters on the Cloud. Later, section 3.4 introduces some simplifying assumptions and considerations in

[1] http://wiki.apache.org/hadoop/AmazonEC2

[2] http://web.mit.edu/stardev/cluster/

[3] http://arstechnica.com/business/news/2011/09/30000-core-cluster-built-on-amazon-ec2-cloud

[4] http://www.top500.org/system/details/10661

[5] http://www.readwriteweb.com/cloud/2011/11/amazon-ec2-now-42-supercompute.php

the cost models developed in the previous section, also presenting data related to energy consumption, prices of hardware components and Cloud instance prices. The resulting models are then used in subsection 3.4.2 in order to compare physical clusters with Cloud clusters. Finally, a discussion of the results is presented in section 3.5, followed by concluding remarks in section 3.6.

3.2 Related work

There is recent work in the literature comparing large scale public Cloud infrastructures with PC clusters, especially for HPC. In [84], the authors include a comprehensive evaluation of performance comparing physical HPC clusters to virtual HPC clusters on Amazon EC2, where the larger network latency in the latter introduces a severe performance penalty for parallel applications. In [65], a similar performance comparison is made with workflow applications composed of loosely-coupled parallel applications consisting of computational tasks linked via data and control dependencies. Different EC2 instance types were employed to assess the performance of the applications and a virtualisation overhead below 8% was computed.

Other works have studied the cost or benefit of using Cloud technologies from different points of view: in [42], the authors study the cost of executing the Montage astronomy application in public Cloud environments. In [15], the authors evaluate the cost of expanding a local virtual cluster using a Cloud technology provider, in order to reduce the response time of the user requests. In [69] the authors compare the performance and monetary cost-benefits of Clouds versus desktop Grids (or Volunteer Computing) infrastructures, ranging in size and storage. In [77], the TCO and Utilisation Cost of a Cloud infrastructure are analysed from the point of view of the IaaS service provider. They also developed a web tool where the users can introduce the parameters of their Clouds and obtain the total cost analysis. Finally, other works such as [14], [10] or [131] have tried to compare the cost of owning a datacenter infrastructure versus the pay-per-use costs of Cloud deployments.

Those contributions show that it is crucial to evaluate the economic impact of outsourcing an organisation's HPC computing infrastructure to an external Cloud provider. As opposed to previous works, this paper performs a detailed comparison between physical and virtual HPC clusters from the point of view of the TCO, considering energetic, management and infrastructural issues, using concepts and estimations from related work, but considering a more detailed analysis. In previous works, the cost of the energy is only estimated, while in this work a detailed model of the energy consumption cost has been defined, where green-aware technologies are a key task to minimize the energetic consumption and the costs in the Cloud deployments. The electrical power required to run a cluster and the price

of the energy have made owners to take such cost into account when operating the computing infrastructures and to create heuristics to try to reduce its impact [130][19]. This paper also analyzes how the Amazon EC2 reserved instances can be used to reduce the cost when the users can estimate the average usage of the cluster.

3.3 The Total Cost of Ownership (TCO) of an HPC Cluster

TCO is generally used as a means of addressing the real costs attributed to owning and managing an IT infrastructure in a business. Therefore, the TCO of owning a HPC cluster not only includes the capital cost, but also the cost of operating the IT infrastructure, and other factors [14].

The cost of owning an HPC cluster can be modelled according to the expression (3.1), where C_F stands for the fixed costs, which only occur once, as opposed to variable costs (C_V), required during the operation of the equipment. C_F can be detailed as in (3.2).

$$C = C_F + C_V \tag{3.1}$$

$$C_F = C_P + C_S + C_{CP} + C_A \tag{3.2}$$

Concerning the costs related to the purchase and configuration of the equipment (C_P), we have considered the computing nodes, additional components, such as switches, Power Device Units, etc., and auxiliary physical elements (racks, cables, etc.). Besides the costs of purchasing the cluster itself, it is important to consider the costs related to buying or renting the physical space where the cluster will be located, together with appropriate refurbishment (C_S). The expression includes the costs related to the purchase of the cooling system (C_{CP}) and the administrative costs (C_A) involved in the purchase (mortgages, loans, infrastructure documentation, etc.). These costs have already been studied in the literature (see e.g. [71]). Another concept traditionally considered when calculating the TCO is the equipment disposal. This topic has not been included in the equation since due to initiatives such as the European Recycling Platform[6], most computer vendors like Dell[7] or HP[8], etc. offer a free recycling program to their clients. In addition,

[6] http://www.erp-recycling.org/

[7] http://www1.euro.dell.com/content/topics/topic.aspx/emea/topics/services/recycle?c=es&cs=esbsdt1&l=es&s=bsd

[8] http://www8.hp.com/es/es/hp-information/environment/hardware-recycling.html

other tasks such as backing up the data or removing data from hard drive to ensure data privacy, etc, are included as part of the maintenance costs.

In addition to the fixed costs, it is possible to break down the variable costs C_V, which account for periodic costs during the lifetime of the hardware:

$$C_V = C_L + C_M + C_O + C_E \tag{3.3}$$

Among the variable costs included in (3.3), one should consider the purchase and update of the software licenses employed (C_L), together with the costs of preventive and corrective maintenance (C_M) to repair the machine and to update certain parts. The operation costs of the cluster (C_O) broadly include the costs of the personnel in charge of deploying, updating and securing the cluster.

Finally, we have to consider the energy cost (C_E), which is one of the most complex and highly variable aspects to evaluate. This cost is expressed in (3.4), where four principal components are included, which correspond to different aspects of the cluster energy consumption. These components are aggregated and multiplied by the cost of the energy unit (C_U), in order to obtain the final cost.

$$C_E = (E_0 + E_I + E_J + E_{CO}) \cdot C_U \tag{3.4}$$

$$E_0 = t \cdot P_0 \tag{3.5}$$

$$E_I = P_I \cdot \sum_{i=1}^{n} t_I(i) \tag{3.6}$$

$$E_J = \sum_{j=1}^{m} (P_P \cdot t_P(j) \cdot n_j + P_U(j) \cdot t_U(j) \cdot n_j) \tag{3.7}$$

We have considered four energy consumption patterns, related to the different main states in which a cluster can operate:

- **Energy consumption of the essential components** (E_0) for the normal functioning (switches, front-end, network cards, etc.), represented in (3.5). It is related to the power consumed by those components (P_0) and to the considered time (t).

- **Energy consumption of idle nodes** (E_I), represented in (3.6). It depends on P_I (power consumed by an idle node), $t_I(i)$ (amount of time node i is idle), and n (number of nodes in the cluster).
- **Energy consumption dedicated to workload computation** (E_J), as expressed in (3.7), which can be split in two. On the one hand, the energy consumed while preparing the nodes for the job (data staging, file transfers, environment setup, etc.), which depends on P_P (power consumed by a node while it is under preparation), $t_P(j)$ (time invested in that preparation for job j) and n_j (number of nodes used by the job). On the other hand, the energy consumed while the job uses the resources, which depends on the function $P_U(j)$ (power consumed), $t_U(j)$ (job duration) and n_j (number of nodes used by the job). The total energy is the sum of these two components for all the m jobs.
- **Energy consumption of the cooling system**, denoted by E_{CO}.

These states have been considered to be different because $P_I < P_P << P_U(j)$. In our case we have represented the power consumed by a node while it is computing as a function $P_U(j)$, since that consumption largely depends on the number of cores and processors being simultaneously employed.

$$C = C_P + C_S + C_{CP} + C_A + C_L + C_M + C_O + \left(t \cdot P_0 + P_I \cdot \sum_{i=1}^{n} t_I(i) + \sum_{j=1}^{m} (P_P \cdot t_P(j) \cdot n_j + P_U(j) \cdot t_U(j) \cdot n_j) + E_{CO} \right) \cdot C_U \tag{3.8}$$

Expression (3.8) includes the TCO of an HPC cluster during a certain amount of time. This expression collects the principles and concepts employed in other studies as covered in the related work section. In addition, it introduces the energy consumption breakdown. The previous studies focus on aspects related to the purchase and maintenance of hardware, while our approach puts more emphasis on analysing the costs that arise when operating the cluster.

Some of the aforementioned costs are constant, while others are typically covered by the economic resources of the organisations. In research centers or universities a space reorganisation can be performed in order to reduce C_S to the bare minimum. This is precisely the case with C_A since this cost might be reduced if the existing administrative personnel take responsibility for the administrative management of the cluster. In some cases, costs such as C_L can also be neglected since licences could be included in the purchase contract of the equipment.

Finally, the energy required for the cooling system tends to be estimated as proportional to the energy consumed by the cluster components, using the Power Usage Efficiency (PUE) ratio. This energy consumption increment can involve from 30% to 200% of the energy consumed by the cluster components alone [18].

Taking into account the aforementioned considerations, the simplified total cost of the cluster mainly depends on the purchase of the hardware, the maintenance and operation of the cluster and its energy consumption. We are not considering C_S and C_{CP} costs, which have been studied earlier in some papers detailed in the related work section. For the discussion, we assume that the owner of the cluster hosts the hardware in an available data center and, thus, the analysis focuses on the costs related to the purchase and operation of the cluster. Buying or building the appropriate infrastructure to host the cluster represents a larger cost than the cluster itself. In this last case, other more exhaustive studies should be carried out, as in [71], in order to decide the features of the installation, the amount of hardware to host, the facilities included, together with the related cost.

Expression (3.8) presents a term related to the energy consumption of idle nodes. In some cases, the amount of time in which the nodes remain idle can represent a high percentage of time of the hardware lifetime. This is due to over-provisioning of resources in order to better cope with peak workloads. Therefore, it is not uncommon to see usage rates of clusters of PCs in the order of 20% [14]. Under these conditions, we could power off the idle nodes in order to save energy and reduce the total cost, as pointed out in [40]. Powering off the idle nodes tries to reduce $\sum_{i=1}^{n} t_I(i)$ from (3.6) and (3.8) to zero, thus eliminating the corresponding term. In this case, the time involved in starting up the nodes would be included in the preparation phase, $t_P(j)$. Considering all these assumptions, the expression reads as follows:

$$C = C_P + C_M + C_O + \left(t \cdot P_0 + \sum_{j=1}^{m} \left(P_P \cdot t_P(j) \cdot n_j + P_U(j) \cdot t_U(j) \cdot n_j\right) + E_{CO}\right) \cdot PUE \cdot C_U \tag{3.9}$$

3.3.1 The Cost of an HPC Cluster on the Cloud

For the case of clusters on the Cloud, the economic analysis is based on the pay-per-use model that Cloud platforms introduce. Moving the cluster to the Cloud discards the fixed costs, such as the purchase of the equipment and the supporting infrastructure, the cooling system, etc. In addition, the administrative costs are substantially reduced since there is no longer need to perform such an upfront investment to purchase the hardware. In any case, the energy consumption related to the cluster and the cooling system is entirely covered by the Cloud provider. However, in this case there are usage costs, represented by the pay-as-you-go model of Cloud computing.

For the discussion, we consider the pricing model proposed by Amazon EC2, since it is one of the pioneer Cloud providers and it has the biggest market share in the provision of computational resources in Cloud. In addition, many providers are adopting the pricing model proposed by Amazon. In this case, an "instance", which corresponds to a virtual machine running with specific virtual hardware features (processors, disk, memory, etc.), is charged per hours running, regardless of its utilization. The different instances offered by Amazon EC2, with the corresponding features and cost, are shown in [7].

There are also additional costs that should be considered, such as the data storage cost and the network transfer cost. Both Amazon and other IaaS-supplier companies charge for using storage space apart from the one required for the proper functioning of the virtual machine, i.e., databases, persistent volumes, etc. The network bandwidth usage for inbound and outbound connectivity of virtual machines is also considered in the pricing model.

With the aforementioned considerations, it is possible to propose a cost model of a cluster of PCs based on the pricing model of Amazon EC2, as shown in expression (3.10).

$$C = C_A + C_L + C_O + t \cdot (C_H + n \cdot C_W) + C_{ST} + C_N \tag{3.10}$$

This model includes the administrative, license and operation costs, together with the usage costs of the virtual machines, where we differentiate the cost of the front-end instance (a.k.a the head node, C_H) and the cost of the working node instance (C_W). The usage cost also depends on the number n of computational nodes of the cluster and the amount of time that these nodes are running. Following the same considerations as in the case of a physical cluster, we will assume that administrative and license costs are negligible. In this case the prices of virtual machine images can also include the license costs of preinstalled applications.

Finally, the model also considers the costs related to data storage (C_{ST}) and network bandwidth usage (C_N). For the former, a virtual machine that does not require additional storage space, other than the one provided by the virtual machine image, will not incur in further charges. For the latter, if a virtual machine does not perform or receive an outbound connection (external to the Cloud provider's network) it will not produce any economic charges. In the case of an HPC cluster, it is important to notice that, since the computational nodes are deployed inside the infrastructure of a Cloud provider, the network communications among the nodes will not cause any charge.

Therefore, the network cost would be caused by the data transfers from the user network to the cluster (and to retrieve the results of the computations), as well as the downloads that the working nodes could eventually perform (if they are

allowed). In the case of Amazon EC2, uploading data to the Cloud incurs in no additional charges.

Note that working nodes are charged regardless of whether they are busy or idle. Considering the cost model for a physical cluster (3.9) and the ability to power off nodes in order to reduce energy consumption, it is easy to realize that an approach for energy reduction would also be valid for reducing the economic cost in the Cloud. In order to reduce this cost there is software such as StarCluster, Hadoop, Globus Provision[9] and elasticwulf[10], which enable to deploy a cluster with a given number of nodes on a Cloud infrastructure, such as Amazon EC2. In these cases, the resources are released when the computations finish, in order to save money. Due to the growth of this kind of tools, different computer services providers are offering the HPC cluster creation as a service to their clients. As an example SGI[11], Sabalcore[12] or Penguin Computing[13] provide access to HPC clusters that can scale on demand. Also some research centres such as SARA[14] offer services to create "Virtual Private HPC Clusters".

With these considerations, it is possible to break down the economic cost of (3.10), applying the criteria from (3.7) to the pay-per-use model, which results in expression (3.11). In this case, the startup time of the virtual nodes is included in the preparation phase of the instances $t_P(j)$.

$$C = C_O + t \cdot C_H + C_W \cdot \sum_{j=1}^{m} (t_P(j) + t_U(j)) \cdot n_j + C_{ST} + C_N \tag{3.11}$$

3.4 Cost Analysis of Moving HPC to the Cloud

This section thoroughly analyses the tradeoffs of moving HPC to the Cloud, in the shape of a virtual cluster deployed on Amazon EC2, compared to a traditional in-house physical cluster, from an economic perspective. The expressions in this section no longer include the C_O cost, because we assume this cost is of the same order of magnitude for both the physical and the virtual cluster, and thus can be omitted.

The cost of a cluster throughout a period of time largely depends on its usage rate and the workload distribution among the nodes, determined by its LRMS. In our case, using the expression (3.9), we compute the lower and upper bounds to the

[9] http://globus.org/provision
[10] http://code.google.com/p/elasticwulf
[11] http://www.sgi.com/products/hpc_cloud/cyclone
[12] http://www.sabalcore.com
[13] http://www.penguincomputing.com/POD
[14] http://sara.nl/services/cloud-computing

TCO of a physical cluster. For that, according to [10], we estimate the maintenance cost of the machines (C_M), including out of warranty repairs, to be an annual 10% of the initial cost of the hardware. Concerning the PUE, choosing the right value depends on the specific installation but, according to [18], it typically varies from 1.3 to 3.0. For the analysis we choose the average value of that scale, i.e. 2.15, although other cases might involve adjusting that ratio, such as in [131, 9].

The lower bound for the TCO of a physical cluster is shown in expression (3.12) and it corresponds to the situation where a cluster is purchased but no computations are ever performed. Therefore, all the internal nodes are powered off and only the front-end and the essential components remain powered on. The upper bound expression is shown in (3.13), and it corresponds to the theoretical state in which the cluster is working at full workload, where all the computing nodes are running and using 100% of all the computing cores (c). In (3.13), the amount of power consumed by the n nodes is estimated using the power of an idle node (P_I) and the additional power consumed by each of the c cores in the cluster (P_C). Y corresponds to the duration of one year.

$$C = C_P + 0.1 \cdot C_P \cdot \frac{t}{Y} + t \cdot P_0 \cdot 2.15 \cdot C_U \tag{3.12}$$

$$C = C_P + 0.1 \cdot C_P \cdot \frac{t}{Y} + [t \cdot P_0 + t \cdot (n \cdot P_I + c \cdot P_C)] \cdot 2.15 \cdot C_U \tag{3.13}$$

Just like the physical cluster, the cost of a cluster in the Amazon EC2 Cloud depends on its usage, since this determines the number of computational nodes to provision. In our case, instead of creating a synthetic benchmark to obtain the average cluster activity, we use a percentage of usage (u) of the equipment through the time of the study:

$$C = C_H \cdot t + C_W \cdot n \cdot u \cdot t + C_{ST} + C_N \tag{3.14}$$

However, to increase the diversity of cases in the study, we have included the reserved instances model offered by EC2. This allows users to make a one-time payment to reserve an instance for a three-year period, and in turn receive a significant discount on the hourly charge for that instance. Considering our approach, a number of p reserved instances would be kept up and running for the reservation period t_R, and the other instances could be fired up and down on demand up to the maximum number of n nodes. The expression now reads as in (3.15).

$$C = (C_{HR} + C_{WR} \cdot p) \cdot \lceil \frac{t}{t_R} \rceil + C_W \cdot (n - p) \cdot u' \cdot t + (C_{HU} + C_{WU} \cdot p) \cdot t + C_{ST} + C_N \tag{3.15}$$

This expression introduces the head node instance reservation price (C_{HR}) and the hourly price for using the reserved instance (C_{HU}). In a similar way, it also considers the cost of the reserved working nodes (C_{WR} and C_{WU}). It also considers u', which corresponds to the usage rate of nonreserved instances, which we will refer to as *nonreserved usage*.

From now on, we will assume that the costs related to storage (C_{ST}) are zero, since our study focuses on the operating costs of the cluster. This assumption is reasonable considering that the front-end instance has enough free storage space to operate the cluster. For example, an *m1.small* instance in Amazon EC2 has a 160 GBytes disk size. The front-end will probably be shared with the internal computing nodes via the network (using NFS, for example).

Concerning the costs of the network bandwidth C_N, this will depend on the applications to be executed on the cluster, the amount of data required to start the computations, and the generated output data of the executions. In the case of Amazon EC2, internal transfers do not incur in additional charges. However, outer transfers, i.e., the data movement between the Cloud provider's network and the client, are billed. Considering a reference value of 100 Gb of data transfer per month (more than 3 Gb of daily results by the users), this would represent a monthly cost of 12$. Since this study handles values in the order of thousands of euros, the cost of the data transfers is negligible.

It is important to point out that expression (3.15) is an estimation that includes simplifications and, therefore, the actual results will depend on the usage patterns of the cluster and the LRMS. In addition, the expression represents an upper bound to the cost of the cluster in the Cloud, since we have assumed that the reserved instances will remain active all the time. This is because we have assumed that the usage rate of the reserved nodes is very high, in order to match the typical workload of the system. Notice that additional savings could be made if those reserved nodes were powered off if they remain idle for a certain amount of time.

An important issue to be considered is that the variable u' included in expression (3.15) is not the usage rate of the cluster, but the usage rate of the extra $(n - p)$ unreserved nodes. If we assume that the reserved nodes are always used, the global usage rate of the cluster would be $p/n + ((n - p)/n) \cdot u'$.

Figure 3.1 shows a conceptual view of the costs of the cluster related to its workload, according to the proposed model. In the left hand side of the figure, an example of the workload of a physical cluster is shown, which is characterized by the number of nodes being used during the time study. Considering the proposed model, this workload results in an economic cost due to the energy consumption. In the right hand side of the figure, the same workload is depicted for the case of a cluster in the Cloud. Assuming the initial purchase of a set of reserved instances in EC2, the final cost would consist of the cost of these reserved instances (lower part

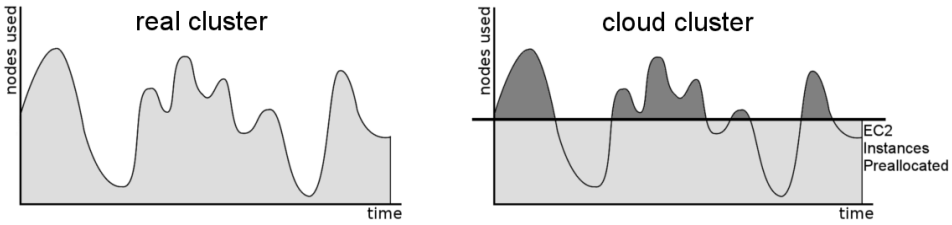


Figure 3.1: Usage rate beyond the amount of preallocated nodes (reserved instances)

Parameter	Value
C_U ($ per kWh)	0.0988
P_0 (kW)	0.7659
P_I (kW)	0.0966
P_C (kW)	0.01075

Table 3.1: Energy consumption a cluster node (Intel Xeon E5520 2.3 GHz)

of the figure, in light gray), together with the cost of the dynamically provisioned instances, which cope with the excess of workload that cannot be executed by the reserved instances (upper part of the figure, in dark gray). In the expression (3.15), the dark gray part corresponds to $(n - p) \cdot u' \cdot t$.

As an example, in the case of a 64-node cluster where 40 nodes are reserved instances, there are 24 nodes that should be powered on and off depending on the workload of the cluster. Therefore, the nonreserved usage u' refers to those 24 nodes. Thus, a nonreserved usage of 0.5 would represent that an average of 12 nodes (out of the 24) are powered on during all the time of study, or equivalently, the 24 nodes are powered on during half of the time of study.

Introducing the nonreserved usage and the previous assumptions, we will be able to identify a broad spectrum of cases when using a cluster infrastructure, thus performing a thorough study. The following section analyses the cost for two clusters, a small-sized one with 64 cores and a larger one with 1024 cores.

3.4.1 Supporting Data for the Case Study

The cost of a physical cluster related to energy consumption has been computed considering the values in Table 3.1. The energy price C_U comes from the average value of the energy in the USA [1]. In some European countries (like Spain), the energy price can reach 0.20$ per kWh [46].

In order to obtain the actual power consumed by the essential components of the cluster (P_0) and by an idle node (P_I), we have relied on the monitoring systems of

Instances	1	2	3	4	5	6	7	8
Increment (W)	34	44	57	68	71	73	78	86
Increment / core (W)	34.0	22.0	19.0	17.0	14.20	12.17	10.71	10.75

Table 3.2: Power consumption of a modern cluster node

Component	Unit Price
Computational nodes with two Quad-core Intel Xeon E5620 2.4Ghz processors, 16 GB of RAM and 146GB of hard disk	3600 $
Front-end node with similar features than the computational node	5000 $
Network by means of Gigabit Ethernet switches	2500 $
Rack	4000 $

Table 3.3: Configuration of a physical cluster

a physical cluster of PCs. To estimate the power consumption of an individual core we have used the well-known Linpack [45] benchmark, which has been executed with a different number of instances ranging from 1 to the maximum number of cores of a node whose features are described in Table 3.3. The average power consumption has been analysed and the increment of power with respect to the idle state (P_I) has been computed, taking into account the number of instances employed. Since the benchmark only stresses the CPU, this power consumption increment can only be attributed to the CPU. The resulting data is shown on Table 3.2.

As shown in the table, the power consumption per core is not linearly related to the number of cores. If we want to compute an upper bound of the power consumption, we can use the increment of power per core corresponding to the case when all the cores are being employed.

Concerning the price of purchasing a cluster, we have assumed a standard rack-based configuration, with the components detailed on Table 3, for which prices have been obtained through different vendors in early 2012.

To compute the cost of the cluster in the Cloud, the values in Table 3.4 have been employed, which reflect the pricing policies of Amazon EC2 as of early 2012 [7]. To make the comparison, the "Cluster Compute Eight Extra Large" instance (named cc2.8xlarge in Amazon EC2) has been used. It has been selected because it gets the best price/ performance ratio. This instance type, with 16 cores, provides 88 EC2 Compute Units (ECU) and it has also been considered for HPC usage in previous works [47]. ECU has been used to compare the performance of the different instances, as it has become the "de facto" standard unit to measure the performance of Cloud systems. Other type of instance could be used depending on the concrete cluster needs and the specific required resources for each virtual node. Concerning the reservation of instances, Amazon considers different prices

Item	Value
Front-end instance price ($/3 years)	300 $
Front-end instance price ($/hour)	0.013 $
Internal reserved instance price ($/3 years)	10490 $
Internal instance price ($/hour)	2.4 $
Internal reserved instance price ($/hour)	0.494 $

Table 3.4: Considered values to compute the cost of a cluster on Amazon EC2

	MFlops (Linpack)	Normalized
Amazon EC2 (m1.small)	522.26	1
Real Node Xeon E5620 2.4Ghz	1516.52	2.903758
Amazon EC2 (cc2.8xlarge)	1614.90	3.092139

Table 3.5: Performance value of the studied nodes (per core)

for low, medium and high use instances. With respect to low usage instances, high usage instances have a higher one-time reservation fee and lower hourly price. For simplicity, in this study we consider only high use instances, although results for low or medium use instances would not be substantially different.

Two different cases have been considered. On the one hand, a small-size cluster composed of 1 rack, 1 Gigabit Ethernet switch, 1 front-end, 8 nodes and 64 cores, with an approximate cost of 40,300 $. On the other hand, a large cluster composed of 4 racks, 6 Gigabit Ethernet switches, 1 front-end, 128 nodes and a total 1024 cores, with a total approximate cost of 496,800 $.

3.4.2 Comparing clusters

Using the previous data, a cost comparison analysis has been performed between a physical cluster (considering the lower and upper bounds in (3.12) and (3.13), respectively) and an EC2 virtual cluster of the same size, using the model in (3.15), from which lower and upper bounds can be derived using the minimum and maximum usage of the cluster. We consider not only the cluster size for the comparison, but also the performance. For that purpose we have run the Linpack benchmark both in an EC2 instance and in a physical cluster node equivalent to the computation node described in the previous section. The results and their normalized values are shown in Table 3.5, where we see that the performance of the physical cluster node and the cc2.8xlarge EC2 instance are quite similar, both obtaining a result about three times greater than the m1.small instance.

For the comparison, a period of 4 years has been considered, following the recommendations of [83]. The evolution of the cost with respect to the time is shown on

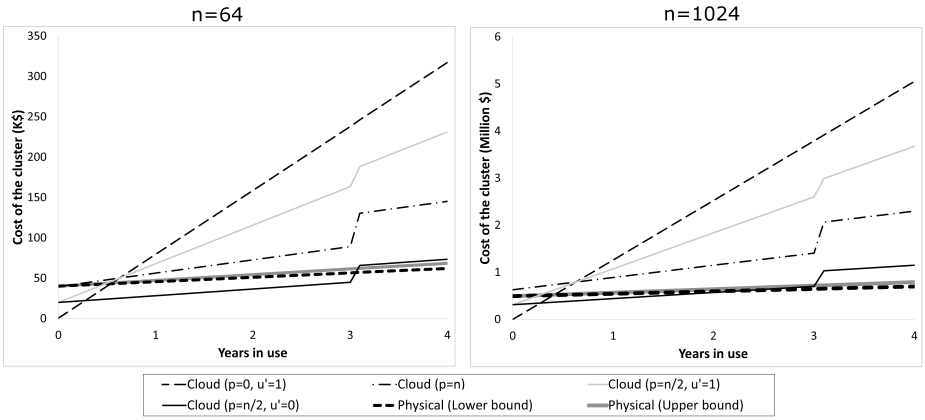


Figure 3.2: Cost comparison between the physical cluster and the cluster in the Cloud for a 64-core cluster (on the left), and for a 1024-core cluster (on the right) for 4 years

Figure 3.2. The left hand graph corresponds to the 64-core cluster, while the righ hand graph shows the data for the 1024-core cluster.

For each case, four different cases are shown, which correspond to different combinations of reserved nodes (p) and nonreserved usage (u') in the virtual cluster. The combinations illustrate the upper limit and lower limit of the virtual cluster cost for different cases of reserved instances ($p = 0$, $p = n/2$ and $p = n$). There are two special cases: a cluster that has not reserved instances and it is not used at all ($p = 0$ and $u' = 0$), and a cluster that has all its nodes reserved ($p = n$). In the first case, the cost is exactly 0 and it is not shown in the graph because it coincides with the horizontal axis. In the second case, both the upper and lower bounds of the costs are the same since there is no variable part in the cost.

In both graphs, the thick lines correspond to the lower bound (dashed black line) and the upper bound (solid gray line) of the physical cluster cost. The increment in cost with respect to time corresponds to the energy cost and the hardware maintenance. This cost can represent more than 70% of the initial cluster price after 4 years.

We can see that the virtual cluster cost on the left graph is approximately proportional to the same cost on the right graph, as is to be expected according to the expression (3.15). The differences between the graphs correspond to the scale of the prices and to the price of the physical cluster.

The graphs point out the influence of the saving plans of the Cloud provider (i.e. reserved instances) in the total cost of the cluster. In particular, if no reserved instances are pre-purchased, the price of a cluster in the Cloud shoots up to more

than 350% (with respect to the physical cluster) in the case of the small cluster ($n = 64$) and more than 500% in the case of the large cluster ($n = 1024$) for a maximum usage rate. However, using no reserved nodes could be better if the cluster usage rate is sufficiently low. This option is certainly interesting in the case of internet-based start-ups in which estimating the workload of the computing infrastructure is difficult, since this might depend on the success of the product.

Using reserved instances, it is possible to gain an economic advantage in the long term with respect to provisioning all the computing resources on demand. However, sometimes this means no real advantage with respect to the physical cluster. It is true that the hourly rates of reserved resources are lower than those of non-reserved ones. However, the initial investment in the reserved instances makes the total cost of the virtual cluster to easily surpass the cost of the physical cluster. In addition, the slope related to the hourly prices of the instances is steeper than the slope corresponding to the electricity and maintenance costs for the physical cluster. In this case we face the additional problem of having to decide the number of instances to purchase in advance This requires a careful planning and estimation of the computational resources to be needed in the near future. Otherwise, the user would incur in a penalty cost for having an overdimensioned cluster.

It is worth noting that the lines corresponding to the upper bound of the virtual cluster cost intersect in one single point that corresponds to about 7.6 months. In fact, using expression (3.15) and considering the costs corresponding to any two different values of p, we obtain that both costs are equal for time t_E given in (3.16), which depends only on u' and on the prices of reserved and nonreserved instances for the cluster working nodes.

$$t_E = \frac{C_{WR}}{C_W \cdot u' - C_{WU}} \tag{3.16}$$

In particular, the intersection point mentioned above corresponds to the case where $u' = 1$, which substituting in (3.16) yields $t = 5503.67$ hours, or approximately 7.64 months.

In general, expression (3.16) could help us decide if the option of reserving nodes is preferrable for a particular case, assuming that we have enough information about the cluster usage rate.

3.5 Discussion

Deciding whether it is still convenient to purchase hardware to enable scientific computations in the shape of HPC (High Performance Computing), HTC (High Throughput Computing) or MTC (Many Task Computing) instead of outsourcing computations to an IaaS Cloud provider depends on several factors. Obviously, a cluster in the Cloud provides the user with the inherent benefits of the Cloud technology, such as avoiding the upfront investment in hardware, maintenance, cooling, etc. In addition, it also frees the user from setting up the space dedicated to host the cluster together with its refurbishing costs. This represents both a considerable investment, studied in publications such as [106], and an entry barrier that might delay the start of the operations of the cluster. In many cases, these delays are not admissible. Therefore, the Cloud could be employed as the final computing infrastructure or as a transition solution until the physical infrastructure is set up.

We found that, in some cases, a cluster in the Cloud can deliver an amount of computational power comparable to a physical cluster with a similar cost. If the usage of the cluster is going to be very high during all its life-cycle, the physical cluster is the best option. But if the average usage of the cluster is going to be moderate or low, only having some periods with peak workloads, the Cloud can be a very good option. Note that it is important to correctly estimate the usage of the cluster in order to purchase the correct number of reserved instances to take advantage from commercial IaaS.

Another important aspect to be considered is that the hardware depreciates with time, due to the rapid obsolescence of the equipment and its delivered performance. Therefore, a physical cluster cannot be considered a middle or long term investment unless a plan of Return of Investment (ROI) exists. In this sense, one should expect public Cloud providers to periodically upgrade the underlying hardware (or reduce the price) so that users can benefit from the performance improvements. In the case of Amazon EC2 the hardware upgrade produced in the last years enabled an important reduction of the price per ECU (more than 75%) in the last years [125]. So it is logical to think that the price fall could continue in the next years. However, this fact has not been included in the proposed model due to the related uncertainty concerning a future price fall. Nevertheless, this should be considered by a customer when deciding among different Cloud providers.

It is also important to point out that estimating the cluster size is far from being a trivial task. If the cluster is underdimensioned, we face the risk of being unable to fulfill the computational requirements of the users. However, if the cluster is overdimensioned, we face an unnecessary cost for unused resources. In the case of a physical cluster, this decision is critical, since the dimension of the cluster relates to space for housing, energy supply, cooling systems, etc. In the case of a cluster

in the Cloud, an inappropriate dimensioning of the cluster is less problematic since computational resources can be provisioned and released on demand, in order to satisfy unexpected peak workloads.

Another aspect to be considered is that one might think that a usage rate of 20% or 40% (73 or 146 days in a year) in a cluster is significantly low. Even though there are scientific applications that require sustained computing power for weeks or months, not many applications used in clusters have usage patterns over that usage ratio. This represents a total of 224,256 and 439,296 CPU hours for a 128-node cluster. For example, a node that our research group dedicates as part of the Spanish NGI (National Grid Initiative) has a usage rate of 33%.

One of the target users of a cluster in the Cloud would be a start-up (an enterprise with a high degree of uncertainty with respect to the computing requirements) or an organisation whose workloads match a high level rate during medium or small time periods (e.g., without exceeding a 20-40% sustained usage rate across the year). In these cases, the organizations avoid the upfront investment, reduce the Time to Market (TTM) and can postpone the decision of investing in a physical infrastructure, possibly depending on the ROI of the business activity. Such organisations could objectively map the Cloud costs to the price charged to the customers, and adjust the use of reserved instances according to the changing needs.

3.6 Conclusions

Nowadays, there are several IaaS public Cloud providers, which represent an alternative to the traditional purchase of computing infrastructure. However, the users of computer clusters have traditionally relied on physical clusters. This article has focused on the convenience of outsourcing cluster-based computations (HPC, HTC, MTC, etc.) to the Cloud.

For that purpose, a cost model of a physical cluster has been developed, which considers the different aspects studied in the literature, but also includes the cost related to energy consumption and the usage of energy saving strategies. We have used the pricing options of Amazon EC2 to create a cost model of a cluster in the Cloud. This model considers the reserved instances approach offered by the provider and it also considers the usage of tools to power on and off instances on demand, in order to avoid idle nodes on the Cloud and its associated cost.

A comparison has been made of the cost of physical clusters and their virtual counterparts, with the same number of nodes and similar performance. The result is that from the point of view of a data center, which expects a high usage rate for their clusters, it is still economically preferable to purchase a physical cluster hosted on its facilities. The fact is that these centers expect a ROI related to

the usage and renting of their equipment and, therefore, can benefit from the economies of scale to turn themselves into infrastructure providers. But if the goal is obtaining high performance computing, and the sustained usage rate is moderate or low, the Cloud can provide similar equipment at a competitive price. Having a good estimation of the workload is necessary in order to select the correct number of reserved instances. We have also obtained an expression to help us decide whether the option of reserving nodes is economically preferrable, depending on the estimated usage rate of the cluster.

Our cost model does not include additional features such as spot instances, which enable the user to bid for unused Amazon EC2 capacity, since their cost is not deterministic. However, it could be an interesting option when performing HTC computations. A scenario could be envisaged in which the cluster grows and shrinks opportunistically according to the instance prices.

Chapter 4

EC3: Elastic Cloud Computing Cluster

Published as

Miguel Caballer, Carlos de Alfonso, Fernando Alvarruiz, Germán Moltó, EC3: Elastic Cloud Computing Cluster, Journal of Computer and System Sciences, Volume 79, Issue 8, December 2013, Pages 1341-1351, ISSN 0022-0000, http://dx.doi.org/10.1016/j.jcss.2013.06.005.

Abstract

This paper introduces Elastic Cloud Computing Cluster (EC3), a tool that creates elastic virtual clusters on top of Infrastructure as a Service (IaaS) Clouds. The clusters are self-managed entities that scale out to a larger number of nodes on demand, up to a maximum size specified by the user. Whenever idle resources are detected, the clusters automatically scale in, according to some predefined policies, in order to cut down the costs in the case of using a public Cloud provider. This creates the illusion of a real cluster without requiring an investment beyond the actual usage. Two different case studies are presented to assess the effectiveness of an elastic virtual cluster. The results show that the usage of self-managed elastic clusters represents an important economic saving when compared both to physical clusters and to static virtual clusters deployed on an IaaS Cloud, with a reduced penalty in the elasticity management.

4.1 Introduction

The usage of clusters of PCs as a computing facility is currently widespread in the scientific community. In the last years, the success of this computing platform, either for High Performance Computing (HPC) or for High Throughput Computing (HTC) has been unparalleled. However, one of the main drawbacks of these computing platforms is the relatively large upfront investment together with the maintenance cost. For small and medium-sized research groups or organizations the purchase of such an equipment might represent an important cost.

Traditionally, virtualization was not considered as a viable option for HPC, mainly due to the overhead costs in I/O and network devices. However, the major improvements in hypervisor technologies have paved the way for Cloud computing to rise as a paradigm where resources (in the shape of virtual machines (VM), network, storage capacity, etc.) can be dynamically provisioned and released on a pay-as-you-go basis [88]. This is the case of public Infrastructure as a Service (IaaS) Cloud providers such as Amazon Elastic Compute Cloud (EC2) or Rackspace.

In a previous work [4] we concluded that, in some cases, it is interesting to deploy a virtual cluster instead of a physical one. A virtual cluster in an IaaS provider is able to get a competitive performance per price rate, but also gets important benefits from the Cloud provider such as reducing the administration costs (both personnel costs and maintenance of equipments), avoiding hiring or buying the physical building to host the infrastructure, avoiding the upfront investments in hardware, cooling systems, etc. Therefore, deploying a cluster in the Cloud can also inherit these advantages.

However, clusters are generally not used at 100% of its capacity during their lifetime [14]. The total lifetime of a cluster can be divided into two parts: the time while the cluster is calculating (T_c) and the time when the system is idle (T_i). In a Cloud environment it would be adequate to stop the VMs while they are not being used and pay only for T_c. The idea is similar to what is currently done with energy saving techniques in a datacenter, where physical nodes are dynamically powered on and off in order to reduce energy consumption while maintaining the required level of service. However, whereas for Green computing the aim is to save energy, for Cloud computing the main aim is to save money (in the case of public Cloud providers). In this case, the Green computing techniques seem to be suitable for creating elastic clusters in an IaaS cloud deployment.

This article proposes the combination of Green computing, Cloud Computing and HPC techniques to create Elastic Cloud Computing Cluster (EC3), a tool that creates elastic virtual clusters on top of IaaS Clouds. EC3 creates elastic cluster-like infrastructures that automatically scale out to a larger number of nodes on demand up to a maximum size specified by the user. Whenever idle resources are detected, the cluster dynamically and automatically scales in, according to some

predefined policies, in order to cut down the costs in the case of using a public Cloud provider. This creates the illusion of a real cluster without requiring an investment beyond the actual usage. Therefore, this approach aims at delivering cost-effective elastic Cluster as a Service on top of an IaaS Cloud.

The remainder of the paper is structured as follows. First, section 4.2 describes the related work and the main contributions of EC3 to the state-of-the-art. Later, section 4.3 details the architecture and implementation details of EC3. Then, section 4.4 describes two case studies to demonstrate the functionality of EC3 both in the case of a stable cluster and in the case of an ad-hoc cluster for a specific application. Finally, section 4.5 concludes the paper and points to future work.

4.2 Related Work

This work encompasses the dynamic deployment of virtual elastic clusters on the Cloud. Regarding the creation of clusters in the Cloud, some works such as [97], [128] and [81] have analyzed architectures, algorithms and frameworks to deploy HTC clusters over private, public and hybrid Cloud infrastructures. In [97] the authors analyze the performance of virtual clusters deployed on top of hybrid Clouds obtaining good results that demonstrate the feasibility of these kind of deployments. The work by Wei et al [128] is focused on algorithms to deploy VMs over a set of physical resources in the most efficient way trying to obtain the best performance on the virtual cluster. In [81], the authors try obtain the best resource allocation solution for a set of virtual clusters from different users with different job features. However, none of them deals with the elastic adaptation of the cluster size to the workload submitted by their users.

In [84, 85], the Nimbus toolkit is employed to implement and evaluate an elastic site manager, which dynamically extends existing physical clusters based on Torque with computational resources provisioned from Amazon EC2 according to different policies. A similar approach is employed in [15], where the benefits of using Cloud computing to augment the computing capacity of a local infrastructure are investigated, but no details about the underlying technologies are given.

The standard distribution of Hadoop [23] includes a utility to create a virtual cluster in the Amazon EC2 infrastructure, managed by the Hadoop middleware. The utility powers on the master virtual machine, using a pre-defined Amazon Machine Image (AMI) and creates the computing nodes using another AMI, performing the required network configuration. It is possible to add new computing nodes to the running cluster. However there is no additional support to remove the nodes from the Hadoop cluster. Finally, when the cluster is no longer needed, it is immediately terminated in order to free the allocated resources. ViteraaS [44] allows

creating virtual clusters on hybrid Clouds. This software enables the submission of jobs that need to be run in a cluster to the system. The middleware creates a cluster to fit the job and manages the execution of the job. The main problem is that ViteraaS does not allow the user to remotely access the cluster. Instead, ViteraaS is devoted to execute jobs as done in classic Grid approaches without providing access to a cluster. StarCluster [93] enables the creation of a Sun Grid Engine based cluster in the Amazon EC2 infrastructure, following a predefined configuration of applications (Sun Grid Engine, OpenMPI, NFS, etc.). It includes a plugin system that enables the user to add new elements to be installed on the cluster nodes, but the VMs are based on a pre-defined Amazon AMI. As it happens with Hadoop, the creation of the cluster is made from a User Interface (UI) that connects to EC2 and starts the instances, prepares the network, configures the middleware, etc. Along with StarCluster a plugin called Elastic Load Balancer [94] has been developed that is able to grow and shrink the cluster according to the length of the cluster's job queue. The caveat of this plugin is that it requires the StarCluster UI to be connected to the Cloud infrastructure, in order to create and destroy the VMs. Thus, any failure on the UI results in the loss of control of the elasticity capabilities of the cluster.

In the case of commercial solutions we can find different approaches to create virtual clusters. One example is IBM Platform Dynamic Cluster [64] that aims at partitioning the owned resources to deliver each user a custom cluster with specific features by using virtual machines. This is different from common clusters where any application or library is installed on every node and is available for every user, and it is sometimes hard for administrators to install different applications or libraries at the same time. The virtual machines ease this situation and enable user activity isolation. The drawback in this case is that this product is oriented to manage on premise infrastructures and cannot be connected to commercial Cloud providers. Another example of commercial solutions is CycleCloud [38] that is a service provided by CycleComputing that deploys virtual clusters, but it is restricted to Amazon EC2. This service provides the user with a virtual cluster based on SGE, Torque or Condor where a subset of popular scientific applications, offered by CycleCloud, are installed. The user is able to manage the virtual nodes using a web interface, and it is possible to configure the cluster so that it is automatically sized based upon pending jobs.

The summary is that several solutions have appeared for different environments, but there is no general framework that enables the creation and management of elastic clusters in general IaaS deployments. Moreover (i) most of them provide a virtual cluster that comprises a fixed number of nodes; (ii) each of the solutions is oriented to a specific LRMS since they take advantage of the individual features of such implementation; and (iii) most of them are oriented to Amazon EC2 and do not consider other public IaaS deployments, or even on-premise Cloud deployments (e.g. based on OpenNebula, OpenStack, Eucalyptus, etc.).

The main advantage of our approach consists in the development of a generic framework to create and manage elastic clusters that dynamically adapt to the current workload. The deployed clusters turn into self-managed entities where elasticity is handled from the cluster itself, without requiring external entities to monitor and act upon the cluster. In this way, a cluster on a Cloud dynamically shrinks and grows according to predefined policies without human intervention. In addition, the tool seamlessly harnesses resources from public and on premise Clouds.

4.3 EC3: Elastic Cloud Computing Cluster

It is important to point out that the user experience should be maintained regardless of the cluster being physical or virtual, i.e., the user should be unaware that the virtual cluster is actually formed on top of a virtual infrastructure that dynamically adapts (by increasing and decreasing the number of nodes) to the cluster workload. The user is provided with a remote secure shell as the entry point to the cluster.

We have previously stated that it could be an important investment to move a cluster to a Cloud. Nevertheless, the usage of Green computing techniques in a Cloud can alleviate the cost involved in deploying and maintaining the cluster. This is achieved by introducing the concept of virtual elastic cluster, which automatically fires up and down virtual machines depending on the workload. This provides the user with the illusion of a real infrastructure at a fraction of its cost in the case of using a public IaaS Cloud provider. Virtual elastic clusters are also interesting for on-premise Cloud infrastructures as they may be used to partition a real cluster into smaller ones with specific features (Operating System, applications, libraries, etc.) that are delivered to specialized users such as scientists. In this case the elasticity is applied to multiplex the real infrastructure, and being able to show an aggregated number of nodes greater than it actually is. Such case assumes that the clusters are not being simultaneously executed using the total amount of nodes. Otherwise the risk of overcommiting will arise, but its impact is reduced if we consider that the servers are not working at 100% all the time.

Anyway it is crucial to introduce trade-off strategies that consider the orthogonal criteria of both reducing the idle nodes and reducing the waiting time of the users to the resources. The aim of this work is to combine transparency for the end users and ease of administration for the sysadmins. The sysadmin should only specify the maximum size of the cluster and delegate on automatic mechanisms to scale in and scale out the cluster depending on the current workload. As an advanced functionality, the sysadmin might indicate the precise software configuration to be available on each node of the cluster. The underlying system is responsible for deploying the infrastructure and the installation and configuration of the libraries,

applications and required middlewares. The resulting system becomes autonomous and self-managed, increasing and decreasing the number of computational nodes according to the usage rates.

The following subsection describes the components employed to create the architecture required to deploy virtual elastic clusters on a Cloud.

4.3.1 Virtual Infrastructure Deployment

The deployment of a virtual elastic cluster broadly requires two tasks. Firstly, the virtual infrastructure must be deployed, which results in a fully configured cluster running on top of virtualized resources. Secondly, an elasticity manager must be included in the cluster in order to self-manage the elasticity rules to provision and release computational nodes on demand.

In order to deploy the virtual cluster (first task), we rely on previously developed components that help to deploy a virtual infrastructure on top of a Cloud. Even though these component are detailed in [5], we provide a concise description for each of them here for the sake of completeness.

- VMRC (Virtual Machine image Repository and Catalog). This system is used to find a suitable Virtual Machine Image (VMI) that accomplishes the requirements of the user (in terms of Operating System, CPU architecture, applications installed, etc.), and is compatible with the hypervisor available in the Cloud system. This component stores and indexes VMIs in order to be reused in multiple contexts. It also implements matchmaking algorithms to obtain a ranked list of VMIs that satisfy the aforementioned given set of requirements.

- RADL (Resource and Application Description Language). It is a declarative language for users to describe the computational infrastructure needed to run their applications. The purpose of the RADL is to describe the features that a given virtual infrastructure should have, by declaring the capabilities or requisites of the VMs to be deployed.

- Contextualizer. Ansible [11] has been used to enable the unattended execution of commands specified in an YAML document in order to perform the automated installation of software dependences. Therefore, this tool performs the installation of the software so that the VM is configured to successfully execute the application.

- Infrastructure Manager (IM): It orchestrates the different components, enabling the effective deployment of an initial computing infrastructure, and the operations required to modify it on demand, by adding or removing virtual nodes.

To provide elasticity to the virtual cluster, it is possible to modify the source code of an LRMS that considers energy management to support shutting down the nodes, in order to power on or off virtual machines instead of real nodes. An alternative is to take advantage of existing energy management software to create ad-hoc solutions for specific combinations of LRMS and Cloud deployments. Some examples of such LRMS are MOAB (the Enterprise version of Maui) [36] or SLURM [75]. In this work, the Cluster Energy Saving system (CLUES) [40], [6] has been used. CLUES is an energy management system for High Performance Computing (HPC) Clusters and Cloud infrastructures. The main function of the system is to power off internal cluster nodes when they are not being used, and conversely to power them on when they are needed. CLUES is integrated with the cluster management middleware, such as a LRMS or a Cloud infrastructure management system, by means of different connectors. CLUES is also integrated with the physical infrastructure by means of different plug-ins, so that nodes can be powered on/off using the techniques which best suit each particular infrastructure. It also provides a hooking system enabling actions to be executed before or after an action is done by the CLUES scheduler.

Section 4.3.3 will describe the whole process of creating the infrastructure by orchestrating all these components.

4.3.2 Elasticity Rules

Depending on the usage patterns, each cluster might rely on different elasticity rules (e.g. having available as many nodes as required to match the number of jobs to be executed, having a specified queue size, etc.). Therefore, it is important to provide different policies in order to match the requirements of the sysadmins.

These policies are related to the elasticity manager which, in our case is handled by CLUES. This software implements different policies that aim at balancing the tradeoff that arises when trying to minimize the waiting time for the jobs (which involves a larger number of available nodes) and the minimization of a Cloud infrastructure cost (which involves a reduced number of nodes, which generate a cost).

In our case, we have modified the CLUES scheduler to adapt it to the purpose of the virtual elastic cluster and to introduce new policies. The policies can be divided in two groups: the policies used to decide when to increase the capacity of the cluster and those used to decide when to decrease the number of nodes.

In the first case CLUES can interact with the LRMS at two levels. On the one hand, it intercepts the submitted jobs before they reach the LRMS. In this way, CLUES can decide if it is required to increase the capacity of the cluster to make room for the job. When the submitted job reaches the LRMS, the cluster will have

the appropriate size to accommodate the workload (depending on the elasticity rule, the job might end up on the queue waiting for other jobs to finish). On the other hand, CLUES also monitors the queued jobs in the LRMS to check if these jobs need new nodes to be added to the cluster.

The next paragraphs describe the different elasticity rules for the intercepted jobs and for the jobs queued up at the LRMS. It also addresses some considerations in order to shut down the virtual machines.

Starting nodes for intercepted jobs

For the intercepted jobs, these policies provide them with the illusion of a cluster with the required capacity, since nodes will be automatically started (i.e., virtual machines will be automatically deployed and included in the cluster as internal nodes) before the job is scheduled to run on these nodes via the LRMS. By intercepting jobs, it is possible to give an immediate response to the user, before the job arrives to the queue system. In contrast with other solutions that periodically polls the information about the LRMS queues that have some kind of delay on detecting the new jobs.

These are the policies that have been implemented in EC3 by means of integrating the CLUES software, for intercepted jobs:

- Group-based start. Every time a new internal node is required, a group of them are started. This policy assumes a workload model in which as soon as a job reaches the LRMS, there is a high probability that other subsequent jobs will be submitted in a short period of time. By overprovisioning a larger number of nodes, the waiting time of the subsequent jobs will be reduced. This represents an appropriate policy when a sudden burst of relatively large jobs are submitted to the cluster. By properly adjusting the size of the group, the waiting time of the jobs, together with the related cost, might be reduced.

- Starting the whole cluster. When a job is submitted, the whole cluster is started. This is useful in the cases where the configuration of the cluster requires all the nodes to be active in order to properly operate This policy also pays off when workload peaks occur, since the whole cluster might cope with the workload if it has been properly dimensioned. It is important to point out that idle nodes will be later shut down according to the inactivity rules of the elasticity manager.

Notice that it is difficult to choose a one-size-fits-all policy for a general cluster. However, virtual clusters on a Cloud can be deployed to accommodate specific workloads that arise in scientific applications, specially for High Throughput Computing (HTC) schemes such as parameter sweep or Bag of Tasks (BoT), and High Performance Computing (HPC).

Starting nodes for jobs in the LRMS

For the jobs that are already queued up at the LRMS, the following policies to start nodes are oriented to maximize the energy saving (in the case of a physical cluster) or the cost (in the case of a virtual cluster in a Cloud):

- When a job has remained in the queue for a certain amount of time (X). This policy tries to prevent jobs from exceeding a threshold waiting time, increasing the size of the cluster when this rule is triggered. In this case, the requirements of the job are reevaluated (as if it was submitted again) to decide if new nodes should be provisioned in order to satisfy them.

- When the queue size is larger than N jobs during X time units. This policy aims at maximizing the energy saving (or the cost), since new nodes are only provisioned when the cluster workload is relatively large for a sustained amount of time. This represents a tradeoff between the quality of service in the cluster (in terms of reduced waiting time for the jobs) and the economic savings. Therefore, there is a limit in the number of jobs waiting to be executed, but a large number of internal computing nodes is avoided.

Shutting down nodes that are not in use

Deciding when to shut down a node is a difficult task because it has an impact on the end user, since starting the node again forces to wait an extra time for subsequent jobs. Concerning the policies used to decide when to decrease the number of nodes, the strategy is to remove a node from the cluster when it has been idle for a specified amount of time. Increasing this time results in an increased cost of the cluster, although the waiting time of the jobs will probably decrease. The selection of this time depends on the workload of the cluster and it is important to achieve a good tradeoff between the cost and the waiting time of the jobs. In any case it is important to consider the following aspects:

- Time blocks. In some cases, such as when using public Clouds, the user pays a (typically) hourly-rate for using VMs. Therefore, once this block of time has been payed it is a good option to maintain the VM active until the whole block of time has expired (i.e. hour or fraction). Considering time blocks enables the system to keep the maximum number of nodes active in order to try to minimize the waiting time of jobs without an additional cost.

- Queued jobs. The queued jobs at the LRMS must be considered before removing any node from the cluster. If a number of idle nodes are going to be necessary for the execution of queued jobs, those nodes are not shut down. This strategy avoids an increased waiting time for queued jobs when the scheduler decides to run them. However, it may increase the cost due to the number of idle nodes not removed.

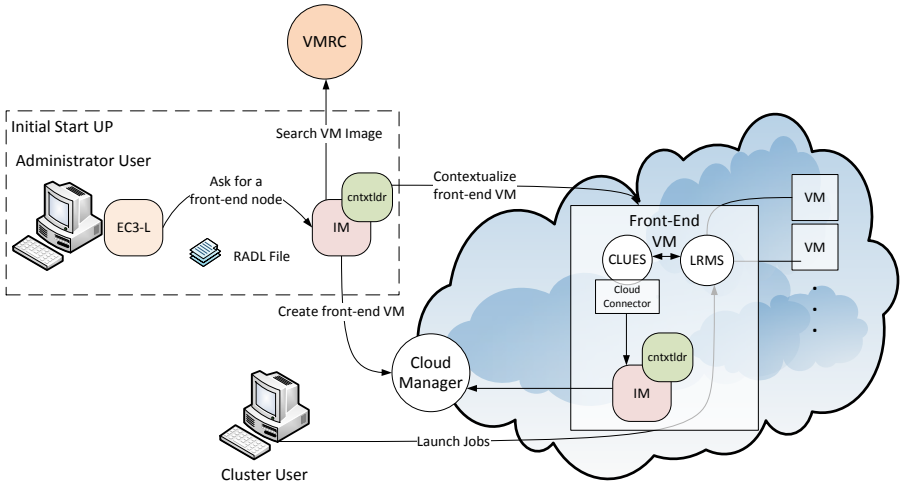


Figure 4.1: EC3 architecture.

- Keeping some nodes always active. This strategy enables the cluster to have at least n idle nodes waiting for jobs. This way, we try to prevent incoming jobs from waiting while internal computing nodes are started. This is a good policy if the cluster workload consists of series of relatively small jobs. The value of n may also depend on the time needed to boot up new nodes. If the time needed is small, a smaller value of n could be used. In this way, there is a tradeoff between the cost of the initially idle nodes and the ability to accommodate jobs with minimum waiting time.

4.3.3 Overall Architecture

Figure 4.1 summarizes the main architecture of EC3. As stated earlier, the deployment of the virtual elastic cluster consists of two phases. The first one involves starting a VM in the Cloud to act as the cluster front-end while the second one involves the automatic management of the cluster size, depending on the workload and the specified policies.

For the first step, a launcher (Elastic Cloud Computing Cluster Launcher or EC3-L) has been developed that deploys the front-end on the Cloud using the infrastructure deployment services described in section 4.3.1. The sysadmin will run this tool, providing it with the following information:

- Maximum cluster size. This serves to establish a cost limit in case of a workload peak. The maximum cluster size can be modified at any time once the virtual cluster is operating. Thus, the sysadmins can adapt the maximum cluster size to the dynamic requirements of their users. In this case the LRMS must be reconfigured to add the new set of virtual nodes and in some cases it may imply a LRMS service restart.

- RADL document specifying the desired features of the cluster front-end, regarding both hardware and software (OS, LRMS, additional libraries, etc.). These requirements are taken by the launcher and extended to include additional ones (such as installing CLUES and its requirements together with the libraries employed to interact with the IaaS Cloud provider, etc.) in order to manage elasticity.

The launcher starts an IM that becomes responsible of deploying the cluster front-end. This is done by means of the following steps: (i) selecting the VMI for the front-end. The IM can take a particular user-specified VMI, or it can contact the VMRC to choose the most appropriate VMI available, considering the requirements specified in the RADL; (ii) choosing the Cloud deployment according to the specification of the user (if there are different providers); (iii) submitting an instance of the corresponding VMI and, once it is available, installing and configuring all the required software that is not already preinstalled in the VM. One of the main LRMS configuration steps is to set up the names of the cluster nodes. This is done using a sysadmin-specified name pattern (e.g. vnode-*) so that the LRMS considers a set of nodes such as vnode-1, vnode-2... vnode-n, where n is the maximum cluster size. This procedure results in a fully operational elastic cluster.

Once the front-end and the elasticity manager (CLUES) have been deployed, the virtual cluster becomes totally autonomous and every user will be able to submit jobs to the LRMS, either from the cluster front-end or from an external node that provides job submission capabilities. The user will have the perception of a cluster with the number of nodes specified as maximum size. CLUES will monitor the working nodes and intercept the job submissions before they arrive to the LRMS, enabling the system to dynamically manage the cluster size transparently to the LRMS and the user, scaling in and out on demand. The scale functionality is provided by CLUES with a developed connector that interacts with different IaaS Clouds, as will be explained in section 4.3.4.

Just like in the deployment of the front-end, CLUES internally uses an IM to submit the VMs that will be used as working nodes for the cluster. For that, it uses a RADL document defined by the sysadmin, where the features of the working nodes are specified. Once these nodes are available, they are automatically integrated in the cluster as new available nodes for the LRMS. Thus, the process to deploy the working nodes is similar to the one employed to deploy the front-end.

Note that the EC3-L tool can be executed on any machine that has a connection with the Cloud system and it is only employed to bootstrap the cluster. Once deployed, the cluster becomes autonomous and self-managed, and the machine from which the EC3-L tool was used (the dashed rectangle in Figure 4.1) is no longer required. The expansion of the cluster while it is operating is carried out by the front-end node, by means of CLUES, as explained above.

On the other hand, even though the front-end duties could be assumed by a machine outside of the Cloud, deploying the front-end together with the working nodes in the Cloud has several advantages: (i) there is direct connectivity between the front-end and the working nodes, without requiring Virtual Private Network (VPN) tunnels. In addition, the connectivity among the nodes does not generate a cost, since traffic does not crosses the boundaries of the public Cloud provider (this is the case of a region in Amazon EC2, for example) and it increases communication speed between the front-end and the working nodes; (ii) it avoids managing the physical machine together with its drawbacks (power outages, network problems, hardware problems, etc.), since these issues are delegated (never avoided) to the Cloud provider; (iii) this will conform a self-managed cluster on the Cloud, independently from other external machines.

4.3.4 Connecting to the IaaS

In order to manage the elastic cluster on top of a Cloud infrastructure, we created the appropriate CLUES Cloud connector which enables the interaction with virtual machines instead of physical ones. This connector interacts directly with the IM for the deployment of VMs in the Cloud and, therefore, it provides a gateway to operate with the different Cloud middlewares supported by the IM (currently supporting Amazon EC2, OpenStack and OpenNebula) and standards (OCCI), contacting the corresponding Cloud Manager. The user must provide the appropriate credentials to access the different Cloud deployments. The connector will pass these credentials to the IM, enabling access to the Cloud providers.

The Cloud connector is in charge of starting and stopping the machines requested by CLUES. In this case, instead of dealing with physical machines, it deploys or terminates instances of virtual machines in the Cloud infrastructure. Besides, an additional step is required to integrate new VMs into the LRMS. In particular, the LRMS software must be installed and configured in the newly created VM, and the front-end must be reconfigured to include the new node. This last step includes mapping the IP of the new Cloud instance to one of the configured virtual nodes that are initialized with an unreachable IP, to enable the front-end node to access it. All these steps are defined via a recipe that is processed by the contextualizer software, and they are configurable so that the users can adapt it to their own configuration requirements. In case of terminating an instance, no reconfiguration

is needed since the LRMS will detect that the node is down, and it will be discarded to receive new jobs.

In addition to the aforementioned tools to start and terminate nodes, the Cloud plugin includes a monitoring system that checks the state of the submitted VMs, together with the monitoring of the LRMS available in CLUES. This is employed to prevent having active VMs (generating cost) that are not integrated in the LRMS (which have been disconnected from the system). In that case, these VMs can be terminated if the LRMS reports that they are in a wrong state for a certain amount of time, defined by the sysadmin.

4.4 Case studies

In order to assess the effectiveness of an elastic virtual cluster on a Cloud infrastructure we have used two different use cases. The first one tries to analyze the usage of the EC3 solution in the case of two clusters during a long usage period. It uses the analysis made in [6] with two physical clusters and applies the same analysis to a "Cloud version" of the clusters. The second one uses EC3 to create an ad-hoc cluster to execute an HTC-based scientific application with dynamic computing requirements, which can greatly benefit from an elastic computing infrastructure that adapts to the workload changes as the execution progresses

4.4.1 Clusters with long usage period

In [6], CLUES was tested with two real use cases, involving two physical HPC clusters of 51 and 7 nodes. The first one (Cluster 1) is composed of 51 bi-processor nodes with Intel Xeon CPUs at 2.80GHz, interconnected by a SCI network in a 10x5 2D torus topology. Each node has 2 GB of RAM memory. The second one (Cluster 2) is composed of an M1000e blade server chassis with 7 Dell M610. Each M610 node has two quad-core Intel Xeon E5620 processors, making a total of 8 cores and 16 GB of RAM per node. It was shown that significant energy/cost savings could be obtained by using green computing techniques (38% saving in the first cluster and 16% in the second one). In this section the idea is to explore what is the cost of running the same clusters on Amazon EC2 and, especially, what is the saving obtained by using the elasticity features provided by EC3/CLUES. The instance types *c1.medium* and *m3.2xlarge* respectively have been selected since they are the most similar ones to the original physical nodes. For simplicity, the use case does not consider using reserved instances to decrease the cost of the allocated resources in the Cloud. However, as shown in [4], a proper selection of the number of reserved instances, considering the cluster workload, can deliver significant cost savings.

	Elastic cluster (EC3)		Static cluster		Cost per node
	PCT	**$**	**PCT**	**$**	**$**
N. Off	52.0%	0	0.0%	0	0
N. Idle	12.7%	8,066	64.7%	41,091	1,270
N. Used	35.3%	22,419	35.3%	22,419	1,270
Other	100.0%	1,270	100.0%	1,270	1,270
TOTAL		31,755		64,780	

Table 4.1: Cluster 1 cost

In [6], the power consumption of the clusters was measured to get the energy consumed and its associated cost. In particular, three different states for a cluster node were considered: switched off ("N. off"), switched on but idle ("N. idle"), and fully used ("N. used"). The clusters worked using CLUES for a period of eight months, and the percentage of time a node had stayed on average in each state was obtained. Then, the total cost of the energy consumed was derived.

In order to obtain the cost of the same clusters if they are deployed on EC2, we consider the cost of a virtual node instance running for a period of one year. This cost is shown in the rightmost column of Tables 4.1 and 4.2. Additionally, since the cluster uses the elasticity features of EC3, we need to know the percentage of time a node stays in each state. These percentages can be expected to be the same as in [6], if we assume that the computational power and boot time of a virtual node are approximately the same as those of a physical node. Some studies such as [105] show that the mean boot time of an EC2 instance is about 60-90 seconds, which is consistent with the boot time of the clusters considered in [6].

In accordance to the previous discussion, tables 4.1 and 4.2 show the cost of the two clusters if they are deployed on EC2 for a period of one year. The left part of the table contains the data for the case of an elastic cluster (EC3) and the center part corresponds to the case of a static cluster. In each of these two cases, the column "PCT" represents the percentage of time a node spent on average in each state (taken from [6], as discussed above), and the column "$" contains the amount of money dedicated to keep the node in that state. Note that in the case of a static cluster a node is never shut down, so the percentage of time in that state is added to that of the idle state. The row "Other" refers to the cost of the essential components of the cluster that are always on, in this case the front-end instance. The results show that the total amount of money saved is 33,025 $ in cluster 1 and 29,223 $ in cluster 2, which represents 49% and 52% of the total amount of money, respectively.

On the user impact side, the results shown in [6] are also applicable here, which means that only 1.31% and 2.9% of total jobs of clusters 1 and 2, respectively,

	Elastic cluster (EC3)		Static cluster		Cost per node
	PCT	**$**	**PCT**	**$**	**$**
N. Off	55.6%	0	0.0%	0	0
N. Idle	11.6%	6,097	67.2%	35,320	8,760
N. Used	32.8%	17,240	32.8%	17,240	8,760
Other	100.0%	8,760	100.0%	8,760	8,760
TOTAL		32,097		61,320	

Table 4.2: Cluster 2 cost

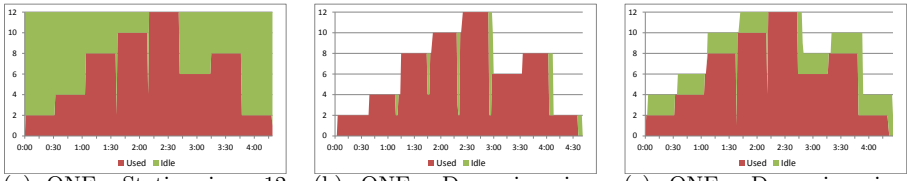
needed to wait to access the resources. The average waiting time for these jobs was 1'40' and 1'54' respectively.

4.4.2 Ad-Hoc Cluster

For this particular case study, we have used a scientific application that performs the optimization of photonic crystal fibres using automated procedures based on Genetic Algorithms [95]. The large amount of computations involved during the optimization process demand the usage of efficient technologies that are able to cope with these computational requirements. Genetic Algorithms (GA) are employed to optimize the features of the fiber and the corresponding injected pulse. It is well known that increasing the number of individuals increases the search capabilities of the GA, although this increases the computational cost in order to evaluate the fitness of all individuals [13]. Therefore, a scheme where the population size of the GA shrinks and grows according to the evolution of the fitness function is of importance in order to tradeoff computational cost and enhanced search capabilities. For this case study, this generates several jobs to be executed where each job corresponds to the evaluation of a single individual.

Two different cloud deployments have been used to perform this second test. The first one is a private Cloud managed by OpenNebula 3.4.1 using the KVM hypervisor. Amazon EC2 has been selected as the second cloud deployment. The test includes a total of 52 job executions in 8 series of 2, 4, 8, 10, 12, 6, 8 and 2. This execution pattern simulates the changes in a dynamic population of a GA of up to 12 individuals in which the population increases or decreases in each generation. The case study has been slimmed down in order to focus on the execution patterns rather than addressing a large problem. With a larger amount of computational resources, a larger population size can be employed. In this particular application, a population size of 30-50 is typically employed.

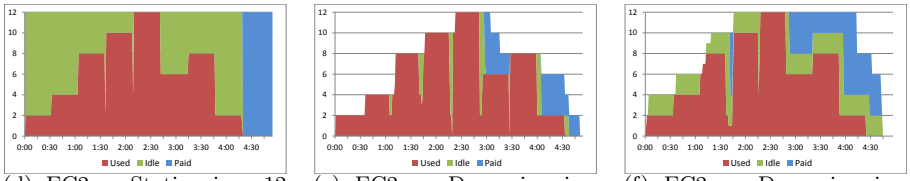
Six different tests have been carried out. In the first two (cases a and d in Figure 4.2), the virtual cluster is kept fully operational during the whole duration of



(a) ONE. Static size, 12 nodes

(b) ONE. Dynamic size, group-based start policy (size 2)

(c) ONE. Dynamic size, keeping at least 2 nodes alive



(d) EC2. Static size, 12 nodes

(e) EC2. Dynamic size, group-based start policy (size 2)

(f) EC2. Dynamic size, keeping at least 2 nodes alive

Figure 4.2: Node usage with different configurations of the virtual clusters with up to N nodes.

the test. In the other four (cases (b), (c), (e) and (f) in Figure 4.2), CLUES is employed to dynamically manage the cluster size depending on the workload, using two different policies to increase the size of the cluster. In particular, cases (b) and (e) use a group-based start policy of size 2, while cases (c) and (f) use a policy of keeping 2 extra nodes alive. Concerning the termination policy of these four tests, a reduced waiting time was employed (5 min.), in order to minimize the number of VMs up during all the execution time, reducing the usage of the Cloud platform. In the case of the EC2 tests, as the price is charged per instance-hour, the EC3 platform does not terminate the instances until the whole hour has passed.

In the OpenNebula (ONE) tests, Figure 4.2 shows the number of existing VMs in Used and Idle states during the execution time. In the EC2 tests there is also a third state named "Paid". This state corresponds to idle VMs that are not terminated because their paid hour is not over yet (with a margin of 5 min.).

In the case (a), the global execution time reaches a total of 4h19', with an average job time of 31'27". Since a total of 12 nodes remain active during the whole execution process, this represents a total 51h48' of CPU time, where 46.31% of the time is effective and the remainder 53.69% corresponds to idle time for the VMs. In the EC2 case (d), the total CPU time consumed increases to 59h due to the instance-hour price. 40.68% of the total time is effective, 47.16% corresponds

to idle VMs, and 12.16% is for VMs that are kept idle until the end of the paid hour.

In the case (b) (using OpenNebula and CLUES with a group-based start policy of size 2), the total execution time rises up to 4h36', due to the waiting time of some jobs while the VMs are started and get ready to accept the execution of jobs. This represents 6.5% extra time. In the tests performed, a total of 14 jobs have to wait for a node to start, with an average waiting time of 2'47". During the execution, there are six moments that require nodes to be started, which provokes that extra time. In this case, the effective usage rate of the cluster is 50.9% but the main difference with respect to the previous case is that the nodes are idle for only 3.4% of the time, since CLUES is responsible to terminate the nodes for the remainder 45.7%, thus without generating an excessive usage. Therefore, a total 29h51' of CPU time is required from the 12 virtual nodes, which represents a saving of 42.3% of CPU time.

In the case (e) (the same as (b) but EC2 instead of ONE), the total execution time is almost the same (4h35'), but in this case only 12 jobs have to wait for a VM to start, with an average waiting time of 2'56". This is due to the fact that some VMs that are kept idle until the end of the paid hour are reused for other jobs (this happens in the transition from 6 to 8 nodes at approximately 3h30'). As in the previous EC2 test, the total CPU time used increases to 34h27', compared with the OpenNebula test, due to the instance-hour price. There is a saving of 47.37% of money with respect to the static EC2 test (d).

In the case (c) (using OpenNebula and CLUES with 2 extra nodes alive), the total execution time is 4h26', that represents 2.7% extra time with respect to the case (a). This is clearly lower than the case (b), due to the use of the policy to keep 2 extra nodes alive, reducing to two the number of moments that require nodes to be started. In this case, the main difference with respect to the case (b) is that 16.1% of the time the nodes are idle, and 34.9% are off. A total of 34h36' of CPU time is required from the 12 virtual nodes, which represents a saving of a 33.2% with respect to having the whole cluster on.

Finally in the case (f) (the same as (c) but EC2 instead of ONE), the total execution time is very similar to (c) with 4h28'. Like in the other EC2 tests, the total CPU time increases, in this case to 44h23', which still represents a saving of 28.2% of money compared to the static EC2 test (d).

As expected, using the elastic cluster considerably reduces the usage of the private Cloud infrastructure and the total cost of the execution on a public IaaS Cloud provider, although a small increase in the execution time is introduced. It depends on the selected policy to obtain a lower extra time or a lower cost. The first policy (cases (b) and (e)) obtains a greater cost reduction but it introduces some extra execution time, because this strategy only adds the required nodes on demand.

In the second one (cases (c) and (f)) the cost reduction is less important but the extra execution time is reduced, because keeping 2 extra nodes alive (with this particular workload) enables a reduction in the number of times the jobs must wait. The impact in the execution time between the two different policies is very low (below 4% of total time).

Another important issue to choose the right policy is to forecast the time needed to add a node into the cluster. In the previous tests, the time needed to have a node running and configured into the LRMS is 2'47" on average in the OpenNebula case and 2'56" in the EC2 tests. But it is important to notice that this time may have important variations. In case of using a large infrastructure such as Amazon EC2, some studies [105] have demonstrated that launching simultaneous VMs has little impact in the time required to have the VMs up and running. However, in private infrastructures some factors may affect this time such as the size of the Cloud deployment, the number of running VMs, the usage of the network, the policy used to balance the workload, etc. For example, launching a set of VMs to start simultaneously has an important impact in this time. Therefore, if the block size selected to add nodes to the cluster is large, and the Cloud infrastructure is relatively small, then the time may have an important increase.

4.5 Conclusion and Future work

This article has introduced Elastic Cloud Computing Cluster (EC3), a tool to create HPC clusters on top of Cloud infrastructures that, using Green computing concepts, creates a self-managed system that dynamically scales to adapt to the workload of users.

The tool builds on top of CLUES, an energy manager system for cluster-like infrastructures, enabling the cluster to gain elastic capabilities on a Cloud deployment. For that, a Cloud connector has been developed to manage the interaction with the infrastructure manager in charge of starting and terminating the virtual machines that correspond to the internal nodes. Moreover the CLUES scheduler has been modified to adapt its policies to be able to manage elasticity to create virtual elastic clusters.

EC3 enables end users to deploy elastic clusters of a given maximum size in a matter of minutes with just a command line tool. Besides, the policies to start and shutdown nodes have been contributed back in order to enhance CLUES enabling more possibilities in order to consider the tradeoff between minimizing the impact in the waiting time for the user (which implies having a larger number of nodes started) and the reduction of the Cloud costs (which implies having a reduced number of running virtual machines).

Future works include several research lines. In the case of commercial Clouds it is crucial to incorporate cost-aware schedulers that consider not only the number of nodes but also the cost of the instances in order to better manage the budget without surpassing the specified limits (daily, monthly, etc.) specified by the user. Concerning the costs, it is important to consider deploying multi-core virtual machines that can share the execution of several jobs. This would enable reduced prices per execution unit. Finally, it is important to consider the case for heterogeneous cluster computing, where each node could exhibit different capabilities (more storage space, different CPU type, support for GPUs, etc.).

Chapter 5

Automatic Consolidation of Virtual Machines in On-Premises Cloud Computing Platforms

Submitted as

Carlos de Alfonso, Ignacio Blanquer, Germán Moltó, Miguel Caballer, "Automatic Consolidation of Virtual Machines in On-Premises Cloud Computing Platforms".

Abstract

After a sequence of creation and destruction of virtual machines (VMs) in an on-premises Cloud computing platform, the scheduling decisions to host the VMs are far from being optimal and the fragmentation of the physical resources may impede the platform to host some VMs despite the free available virtualization resources. This paper describes a Virtual Machine Consolidation Agent that addresses this problem by analyzing the distribution of the VMs in the virtualization platform to migrate some of them among hosts, in order to defragment the physical resources and to enhance the efficiency on their usage. The agent has been validated in a production platform, where it is capable of minimizing the number of servers needed to host the VMs. The algorithms achieve near-optimal values at a very reduced computational cost, thus making it suitable for production platforms.

5.1 Introduction

Cloud Computing resources are typically provisioned from datacenters by means of virtualization to share computational resources. In particular, a user is provided with Virtual Machines (VMs) in the well-known Infrastructure as a Service (IaaS) Cloud model [88], where VMs represent the partition of the physical computers. A Cloud datacenter is dimensioned to ensure that the expected workload is satisfied, but the peak of the demand is rarely achieved. In fact the mean usage ratio of a datacenter was estimated to be between 10% and 50% [110]. Moreover the mean efficiency of datacenters is around 50% [55]. That leads to a waste of energy which gains importance when it is known that the power dedicated to feed the datacenters of the world represents the 0.5% of the total energy consumed, and it is estimated to achieve 2% of the total by 2020 [53].

In order to reduce the wasted energy it is possible to apply Static Power Management (SPM) or Dynamic Power Management (DPM) techniques [121]. SPM consists of using more efficient components to build the computing nodes (low power CPU, enhanced memory, hard drives without mechanical parts, etc.). DPM techniques consist of adapting the computing infrastructure to the actual workload, whether modifying behaviour the individual components (e.g. modifying the frequency of the processor using Dynamic Voltage and Frequency Scaling) or enhancing the distribution of the jobs to use the most efficient working nodes and powering off those that are not needed. Obviously, both techniques may be applied together to obtain better results.

In the Cloud Computing field, some previous works [19] [22] have introduced techniques to enhance the VM scheduling into the hosting nodes as these VMs are being created, to get a better distribution of the usage of the resources. The scheduling decision for a VM to be hosted into a specific host may be correct when the VM was created. However, during the lifecycle of the Cloud platform (i.e. sequences of creation and destruction of VMs) the distribution of the VMs gets worse and the distribution of the resources may be degraded. It is possible to reach very inefficient situations: (1) there are physical nodes that are not hosting any VM, or (2) the running VMs could be hosted in a fewer number of physical nodes or in a more energy efficient subset of nodes. In the former case, some of the idle hosts can just be powered off. But in the latter case the virtualization resources are fragmented and the VM distribution should be rearranged, if possible, to reduce the number of nodes hosting the very same VMs, without affecting the Quality of Service delivered to the VMs.

Moreover, the creation and destruction of VMs may drive to a fragmentation of the physical resources, in case that the lifecycle of the VMs make that the physical resources do not get efficiently used. Such fragmentation of resources may make that the platform cannot host some VMs while there are enough free resources (e.g.

trying to host VM_1 that requests 16 Gb. RAM on a platform in which hosts H_1 and H_2 have 10 Gb. RAM free each). If a continuous creation of VMs takes place, the new scheduling decisions would probably correct these scenarios. But for long lasting VMs and smaller scale on-premise Cloud platforms where there are usually periods of time where the activity is very reduced (e.g. night, lunchtime, weekend, holidays, etc.), it would be cost-efficient to enhance the distribution of the VMs in the physical hosts to reduce fragmentation. Moreover, green computing techniques can be employed to power off the unused nodes and save energy.

This paper describes the Virtual Machine Consolidation Agent (VMCA), that analyzes the distribution of the VMs of the platform and migrates a set of them from the hosts in which they are hosted to other hosts, to achieve a more efficient usage of the resources of the Cloud platform. Finding the best placement for a set of VMs in an empty platform is usually modelled as the well-known multi-dimensional bin packing problem, that is considered to be NP-hard. Its asymptotical computational cost is exponential and it is not guaranteed that the resulting VM distribution can be achieved from the initial distribution of VMs.The task of VMCA is harder in the sense that it starts from a given distribution of VMs and tries to migrate them to other hosts in order to reduce the power consumption or to enhance the usage of the resources of the physical hosts. It is important to notice that VMCA is not an scheduler, as it does not try to select the host in which a VM should be hosted. Instead, VMCA will select a set of VMs that are candidate to be migrated and will delegate to the scheduler of the platform the decission of selecting the host to which the VMs can be migrated, then VMCA will evaluate the different VM movements and will select which of them is the best to get to a enhanced distribution of the VMs. In most of the cases it is crucial delegate the responsibility of scheduling the VMS to the existing scheduler, as it may take into account co-allocation of VMs (e.g. a set of VMs that must be deployed by the same physical host, or in the same virtual network) or other constraints or policies for hosting the VMs in the platform. In the end, VMCA can be considered as a DPM technique that actuates "a posteriori".

After the introduction, the remainder of the paper is organized as follows. First, section 5.2 introduces the related works in the area. Next, Section 5.3 introduces the concept of fragmentation of the virtualization resources and the problems associated to it. Then, section 5.4 explains the architecture of VMCA and the different heuristics that have been considered for this work. Section 5.5 explains the problems that can arise when trying to integrate a system such as VMCA in a production environment. Later, section 5.6 describes the experimental study that has been carried out to validate the behaviour of VMCA and to analyze which heuristics are more suitable to obtain a better usage of the resources. Finally, section 5.7 summarizes the obtained results and explains the future works for VMCA.

5.2 Related works for the problem of redistributing the VMs

According to [124] the techniques for server consolidation can be classified in (a) static techniques, where VMs are scheduled according to the current resource distribution; (b) semi-static techniques, where VMs are re-placed after long periods of time; and (c) dynamic techniques, where VMs are re-placed in real time according to the workload. In the group of *static techniques* we can include the schedulers that are shipped in most of the Cloud Management Platforms (CMPs), since they select the hosting node according to the actual workload. These schedulers follow different strategies such as packing the VMs in the minimum number of hosts, scattering the VMs across the hosts, random hosting, distributing the workload, etc. After the VM has been placed on a particular host, it will be kept in that host until it is migrated by the administrator or it is terminated. The problem is that after a sequence of creation and destruction of VMs, the former scheduling decisions are far from being optimal.

At this point it is possible to correct the situation by re-distributing the VMs in the physical hosts. The most obvious approach is to undeploy VMs and deploy them back using the scheduler of the CMP or an algorithm that will calculate the most efficient VM placement. However, this might introduce a significant downtime of the VMs that may last for a long time, depending on the features of the VMs and the characteristics of the Cloud platform. Using an algorithm to calculate the optimal VM placement by exploring any VM placement combination (such as brute force or branch and bounding) will only be feasible for small Cloud platforms because the cost is known to be exponential ($O(N^H)$ in the case of brute force, where N is the number of VMs in the platform and H the number of physical hosts).

A common technique to address these problems is to apply *reinforced learning* algorithms. This approach is described in [22], where the existing applications in a datacenter are consolidated by shipping them into VMs and placing them on demand. But the most noticeable work in this field was made in the project "Green Active Management of Energy in IT Service Centers" (GAMES) [109], although Cloud platforms are not specifically considered in this project. The datacenter is considered from a global point of view and utilizes sensors and context-modelling techniques for the datacenter and its activity. Once analyzed the situation using a reinforcement learning algorithm, the system uses actuators for very different purposes (e.g. powering on or off the cooling machines, or migrating the VMs across the servers). The main problem is that it is difficult to create heuristics for such different kind of sensors and while the authors suggest that it would be feasible to apply their system to a Cloud platform [33], the simulations shown in [112] need a considerable execution time for the analysis, even for small Cloud deployments. Other example of using reinforced learning algorithms for VM consolidation is de-

scribed in [49]. In this work, it is tried to optimize the number of active hosts, according to the resources needed by the VMs that are deployed in the infrastructure. The work shown in [87] introduces fuzzy learning as an improvement for the reinforced learning strategies.

One different approach to solve the problem is shown in [52], where the authors describe an algorithm inspired on the movements of the ants in a colony. Other example of VM consolidation is shown in [107] where a methodology for server consolidation inspired on the behaviour of the swarms during migratory flights is described. The main drawback in these cases is the long running time required even for small problems. The ant-inspired algorithm has been improved using a de-centralized approach to benefit from parallel calculations in [51]. Other similar distributed solution has been proposed in [86]. The work of Ghafari et al [56] describes a method based on a bee colony algorithm that tries to detect over utilized hosts and selects VMs to be migrated to reduce their utilization.

A widely adopted approach to solve the VM distribution is by modelling the problem as a *multidimensional bin packing* (mBP) problem where the physical nodes are modeled as multi-dimensional containers, and each dimension corresponds to a type of resource (typically CPU, memory or hard disk). The VMs consume these resources and must be placed using the minimum number of bins. Then the problem is solved by applying different approaches and heuristic-based optimization techniques that include *First Fit (FF)* or *Best Fit (BF)* techniques. These techniques consist of sorting the items that have not been placed in the bins yet, and allocate them according "the first item that fits the bin" or "the item that best fits the bin" criteria, respectively. Both types of algorithms have the variant "Decreasing" that sort the remaining items in the reverse order, to create the *FF Decreasing* (FFd) and the *BF Decreasing* (BFd) algorithms.

These techniques are widely applied to the server consolidation field. In [2] a BFd algorithm is applied to a virtualized datacenter. In this work, the VMs are placed into real servers but there is no dynamic re-placement. The decision is treated as static and there will be a new BFd placement decision when a new VM is created. The work shown in [123] considers Dynamic Voltage and Frequency Scaling (DVFS) adjustment, powering off servers and dynamic placement of VMs. It applies a one dimensional FFd algorithm that only takes into consideration the CPU resource to enhance performance of VMs.

According to the topology introduced in [133], placing a set of VMs into several hosts can be categorized as the Variable Sized Bin Packing Problem [32], which is an extension of the Classic One-Dimensional Bin Packing Problem in which several bin types are introduced. But the scenario proposed in the current paper starts from a given distribution, which makes it is slightly more complex.

An interesting work in this sense is shown in [19] and [20], where a system is described that: (i) selects the VMs that are candidates to be moved, according to the CPU utilization of the host node and, (ii) places them according to a BFd algorithm whose criteria is based on minimizing the energy consumption. While this is a very interesting approach, it is focused on selecting the VMs that should be migrated, instead of enhancing the usage of the physical resources. Moreover, they schedule the new placement of the VMs (according to power consumption criteria), thus bypassing the criteria of the current scheduler of the CMP. It also lacks a multi-dimensional approach, as it only consideres the physical CPU utilization to decide the host whose VMs are considered to be moved. Moreover, it does not take into account the cost of the migration or other restrictions about VM placement. The work also does not consider or discusses about heterogeneous hosts (i.e. hosts with different capacities).

The work shown in [115] describes an algorithm that addresses some of the problems of the previous one (such as using multiple resources), but the work is focused on scheduling the VMs in the other physical hosts, and it does not integrate the criteria of the actual scheduler of the CMP. This work also makes a survey of how different heuristics affect to the vector bin packing algorithms for server consolidation.

Other approaches to solve this problems include optimization techniques such as simulated annealing (SA) or genetic algorithms (GA). On the one hand, the work [132] demonstrates that while SA can get better solutions, it is more time consuming that the classic FF-like approaches. On the other hand, works such as [48] and [78] have applied GA techniques to try to solve the bin BP problem using different dimensions, concluding that while GA can get better solutions, these algorithms have a high computational cost compared to the classical FF-like approaches.

While there are several approaches to re-locate the VMs, we have not found any work that addresses the multi-dimensional approach in BF algorithms for VM consolidation when starting from a given VM distribution, and also integrates with the existing VM scheduler. Most of the works try to schedule the VMs according to different criteria (e.g. energy saving, physical host stress, etc.), but integrating with the existing scheduler may be crucial, as it also takes into account co-allocation of VMs, the policies for VM scheduling and usage of other resources, or restrictions on the placement of the VMs. Introducing a “green” scheduler will probably reduce energy consumption, but it can also interfer in the QoS of the platform, may violate the restrictions for the placement of the VMs, or may even make that the VMs cannot take profit from some of the features offered by a scheduler that is deployed specifically in the platform (e.g. reservation of hosts). The mechanism to consolidate the VMs should integrate with the existing scheduler even when the VMs are scheduled in a way that is not the best for the objective pursued when consolidating VMs. On the other side, most works usually consider a one dimensional approach when consolidating VMs, as they only

deal with the CPU resource. As shown in [28], the multi-dimensional approach is considerably harder than the single dimensional.

5.3 VMs distribution among physical hosts

Distributing the VMs across the physical hosts using green computing techniques represents a trade-off between energy saving and Quality of Service (QoS) for the VMs. On the one hand, using the least number of physical hosts leads to reduce energy consumption since idle nodes can be powered off. On the other hand, increasing the density of VMs per physical host leads to resource contention thus reducing the QoS and performance of the VMs. Moreover some VMs will have special requirements that are only met by some physical hosts (e.g. the amount of RAM memory, the number of CPUs or even access to specific devices such as GPUs). In the end this is a problem that must be addressed depending on the objectives of the virtualization platform, and balancing the different available criteria.

There are different works addressing the problem of identifying the best VM placement, as seen in section 5.2. However, during the lifecycle of the virtualization platform the creation and destruction of VMs will surely end up with a distribution in which the VMs are using only one part of the real resources in the physical hosts and the rest of resources are idle. That is what is commonly known as *the fragmentation of resources*.

The problem of the fragmented resources can affect the capacity of a platform to host VMs. An example to illustrate the problem of the fragmented resources is shown in Figure 5.1. For the sake of simplicity, in this figure the VM slots represent *virtual CPUs*, but a similar discussion can be made for RAM memory or a combination of resources. Even if the criteria of the scheduler of the VMs in the platform is to try to reduce the number of physical servers hosting the VMs, the lifecycle may lead to a distribution of VMs such as the one shown in the upper part of figure (i.e. there were other VMs hosted in the hosts, but they have been destroyed). Obviously, in this context, we know that the VMs could be hosted using fewer physical hosts (e.g. VM_2 hosted on $host1$). But the problem goes further, because if a user asks for a VM such as VM_4 that needs 6 virtual CPUs, the platform would not be ready to meet the requirements. If the resources were defragmented as in the lower part of the figure, the requested VM (VM_4) could be hosted in $host2$.

The problem of re-arranging the distribution of the VMs on a platform is complex to tackle. The most simple solution will be to suspend the current VMs and to schedule them again according to the order in which they were requested. Other immediate solution is to calculate the optimal distribution and to try to place the

VMs according to it. Both approaches require free hardware where VMs will be temporarily moved to keep them running. If the VM movement is done offline, the impact on the user experience will be noticeable, because storing the state of every VM in secondary storage and restoring it later, requires a considerable amount of time.

Instead of using a temporary infrastructure, we should identify the sequence of VM movements needed to find the optimal placement from the current state without violating any constraint [62]. But it is not guaranteed that such sequence could be computed, as Figure 5.2 shows. This reinforces the thesis proposed in [2], that a convenient approach is to find a near-optimal solution in a reasonable time.

5.4 The Virtual Machine Consolidation Agent

The Virtual Machine Consolidation Agent (VMCA) is an agent that monitors a virtualization platform managed by a CMP (such as OpenNebula or OpenStack), analyzes the distribution of the deployed VMs and the physical hardware, and schedules a set of VM migrations between hosts that will drive to a *better distribution of the VMs in the hosts*. The sequence of phases follows the "Monitoring, Analyzing, Planning and Executing" (MAPE) model proposed in [112]. The concept of a *better distribution of VMs* depends on the level of service that is being provided by the platform (e.g. allocating one real core per virtual CPU, allocating 1 GB. of real RAM per 1 GB. of virtual RAM, limit the number of VMs per physical host, etc.) and the policies of the service provider (e.g. save energy, scatter

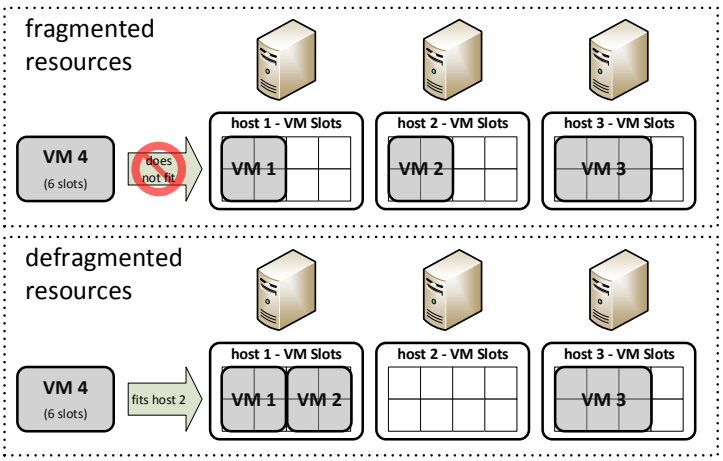


Figure 5.1: Fragmented resources vs defragmented resources.

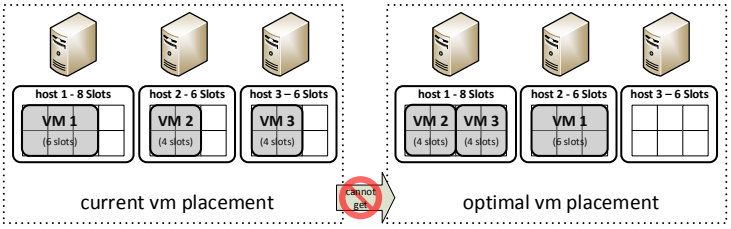


Figure 5.2: There is no feasible sequence of movements that starting from the distribution in the left and without using additional resources, can end up with the distribution in the right.

VMs across physical hosts, defragment resources, have spare resources per physical host, etc.). In this paper we are considering that we get *a better distribution of VMs* when we reduce the number of physical hosts needed to host the VMs or when we reduce the global fragmentation of the non-allocated resources, while maintaining the level of service (e.g. allocating 1 core per virtual CPU and 1 GB. of real RAM per 1 GB. of virtual RAM). This level of service will not choke up the real resources as the hosts are dedicated to virtualization tasks. That level of service is the one that is accepted in the common CMPs such as OpenNebula (ONE), OpenStack or even commercial ones such as VMWare vCenter.

The architecture of VMCA is shown in Figure 5.3. There are four basic decoupled components: (1) the connector to the platform, (2) the monitoring system, (3) the analyzer, and (4) the migration plan manager. The following subsections describe these components.

5.4.1 Connector to the platform

VMCA makes an abstraction of the virtualization platform, with the purpose of not being tightened to any particular platform. By decoupling VMCA from the platform it is possible to use the same algorithms for different CMPs.

The abstraction consists of two parts: an output *data model* that obtains the information model of the platform, and an *actuator* that can be used to order the migration of a VM from one host to another. Both parts need to be implemented for the target platform, using the specific platform API.

The information about the platform is represented by (i) the *physical hosts*, which are abstracted by using the resources that are dedicated by that host to virtualization tasks (i.e. real memory, number of cores, shared disk, etc.), and (ii) the *VMs*, which are abstracted by using the virtual resources that are requested by

the VMs (i.e. virtual memory, virtual CPUs, virtual disk, etc.), its state (running, stopped, unknown, etc.), and the server that is hosting the VM.

Depending on the support to VM migration of the CMP, the actuator may even be a simple function that makes the effective migration of a VM from the host in which it is hosted to another host.

5.4.2 Monitoring system

The information obtained by the connector to the platform represents a snapshot of the virtualization platform. The purpose of the monitoring system is two-fold: on the one hand, to collect the evolution of the state of the platform; on the other hand, to be used as a simulator of the platform to record the changes in the allocation of resources due to the migration of the VMs.

The evolution of the state of the platform is modeled by storing the changes in the state of the VMs, obtained by monitoring snapshots, along with the timestamp of the instant when such change occurred. The tuple $(identifier, state, host, timestamp)$ defines the state of a VM. Using this state, we can infer the *stability* of a VM in the context of VMCA, which is related to how long the VM remains in a state, in one host. And by aggregating the state of the VMs hosted in a host, we can infer the *stability* of a host in the context of VMCA, that it represents how long the VMs of the host remain in the same state.

A VM is considered to be stable when its state has not changed for a period of time. A VM that is not stable will not be considered for migration by VMCA. This is useful to avoid problems related to very frequent migrations in dynamical Cloud deployments. The most immediate example is the "ping-pong" effect, that consists of continuously moving a VM between two hosts because the analysis of the platform results in doing it in subsequent decissions of VMCA. This is a concept which is very similar to the "cool down period" concept used in Amazon Web Services to control auto scaling groups [8]. The value for the stability depend on the features of the platform and the QoS constraints, so it must be tuned for the specific deployment. In the case of VMCA this value defaults to 30 minutes.

The evolution of the state of the platform is guessed from the discrete values obtained from the monitoring snapshots, as it is not possible to know what happens in-between. Getting valid information about the state (when a VM changed from a state to another) is also impossible for a CMP, because it relies on periodically checking the state of the VMs in the hosts. But the underlying idea is that the VMCA monitoring system will rely on the middleware that manages the platform (by using the proper connector to the platform).

This monitoring system is also conceived to act as a simulator for the platform, and enable VMCA to test different sequences of movements instead of effectively

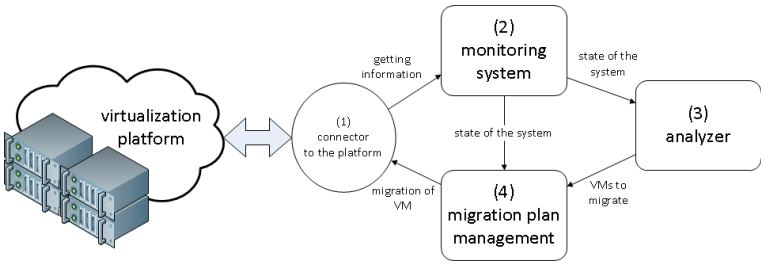


Figure 5.3: Architecture of VMCA.

ordering them on the real platform. The monitor is able to simulate how the migration of one VM from one host to another would change the allocation of the resources of the physical hosts. In the end, this monitor provides the VMCA re-scheduling algorithms with a view of the platform. The simulation makes it possible for the algorithms to try multiple sequences of movements, and to evaluate which of them will provide a better distribution of the VMs.

5.4.3 Analysis of the platform and planning the migrations

The main task for this component is to detect whether it is possible to obtain a better distribution of the VMs in the physical nodes, and to provide the sequence of movements to get it. In this paper, the objective of VMCA is to obtain as many empty nodes as possible. So the key objective for the analysis of VMCA is to move every VM from a physical host to the other servers that are already hosting a VM to get some physical nodes idle. The final consequence is that the idle nodes may be suspended or powered off by using other software such as CLUES [4].

Other works have focused on selecting a set of VMs to migrate, depending on the stress of the physical hosts, or based on thresholds of usage of physical hosts. And most of other works, schedule the VMs according to a specific scheduler. These strategies introduce two problems: on the one hand, the mechanism to select the VMs do not take into account the features of the VMs. Moreover it does not guarantee that the physical host will be free of VMs as they only move enough VMs to get the usage of the resources under a threshold. On the other hand, using a specific schedulers modify the behaviour of the platform, as it does not integrate with the policies used by the CMPs used to schedule the VMs when they are first created.

In order to surpass those problems, VMCA incorporates an Iterative BF-FFd (IBFFFd) algorithm that follows the principles introduced in [66] to re-allocate the VMs from a set of host to the rest. In that work it is introduced a BF mechanism to select which of the distributions fits best for the final objective (in

our case, which physical host is the best one for moving its VMs to other hosts). The underlying idea is (i) to have a distribution of VMs in physical nodes, (ii) for each server that is hosting VMs invidually, compute the state of the platform if only its VMs were moved to other physical nodes using a FFd algorithm (e.g. how balanced are the resources, how much power is consuming the platform or how many nodes can be powered off), (iii) move the VMs from the host that gets the platform in the best state to other hosts, and (iv) start over until we cannot re-arrange the distribution. The algorithm is shown in Algorithm 2.

In the case of VMCA, it starts from a specific VM distribution and then detects which nodes are in a *stable state*. For each stable node (lines 7-21), the algorithm tries to re-place its VMs following a FFd scheme (lines 10-16), where the VMs are selected starting from the one that needs less resources. The destination for the VM is obtained by using the *scheduler from the platform* (line 11), considering all the nodes but the one that is analyzed to be powered off. This way VMCA integrates with the existing scheduler, instead of trying to create a new one. The criteria for scheduling the VMs is fully delegated to the scheduler of the platform, to adjust to the policies of the organization. Using this approach, integrating with other schedulers such as those shipped with the CMPs or to green schedulers such as the one used in [19] simply consists in establishing a mechanism to communicate VMCA with the scheduler or integrating part of the code of the scheduler in VMCA.

In this case we introduce the concept of *reward* and *cost* for the migration (line 18), to evaluate the movements of the VMs. The *reward* is a value that represents, in a heuristic-defined scale, the value that adds the migration to the platform (i.e. the "goodness of the migration"). An example of the evaluation is the difference between the variances of the resources in the platform, which could be used to balance the usage of the resources in the physical nodes. The *cost* corresponds to the time needed to make the migrations of the VMs from the node to other nodes. The estimation of the cost of the migration mainly depends on the platform and the features of the VM that is being migrated. An example of model to estimate the cost of live migration is shown in [117]. In our case, we are simplifying the calculation of the cost of migrating one VM by approximating it to the amount of memory of the VM, multiplied by a factor of magnitude *epsilon*.

If not all the VMs in a host can be re-placed into another hosts, the sequence of migrations will be discarded, because the aim of this algorithm is to get nodes idle, in order to power them off. If a successful sequence is found, it is evaluated and recorded to be compared with the other possible sequences. At the end, the most valuable distribution is selected (line 22). Then the host analyzed is not further considered to host VMs (as it is likely to be powered off) and the hosts that are the destinations for the migrations will not be considered as stable anymore (lines 23-28). Once the new situation is decided, VMCA continues trying to move the VMs from the next host, until every stable host has been analyzed.

The asymptotic cost of Algorithm 2 is $O(h^2 * max(n_i) * cost(scheduler))$, where h is the number of stable nodes (those whose VMs have not changed for a period of time) and n_i is the number of VMs in the stable node i. The h^2 term is included because there is a "for" loop that schedules the VMs inside a "repeat" loop that explores the stable hosts. At the end, the upper bound of the cost of this algorithm is $O(H^2 * N * cost(scheduler))$, which is a quadratic cost instead of the exponential cost of an exhaustive search solution (where N is the number of VMs in the platform and H the number of physical hosts). It must be noticed that the cost of the schedulers shipped in the default distributions of common CMPs typically have an upper bound of their computational cost which is linear to the number of hosts H (i.e. checking the number of VMs of each host in order to pack or scatter the VMs in the hosts).

The quadratic cost of this algorithm is a problem when dealing with big platforms (hundreds of hosts and thousands of running VMs). In that case, we want to use VMCA as part of a DPM technique to reduce power consumption, and the analysis will be triggered frequently, and the total running time should be reduced.

VMCA incorporates a modification that we have called Iterative FFd-FFd (IFFdFFd) that tries to reduce the total cost of the computation of the analysis of the platform, while trying to maintain the quality of the solution. In this case, the underlying idea is (i) to have a distribution of VMs in physical nodes, (ii) select which node is the most appropriate one to move its VMs to other hosts, (iii) make the migrations, and (iv) start over until we cannot re-arrange the distribution. This second algorithm is shown in Algorithm 3.

The underlying concepts of this algorithm are pretty much the same included in Algorithm 2, but in this case it is tried to determine the node that is more interesting to migrate its VMs away to other nodes a priori, by applying a FFd algorithm in function *select_node_to_empty* (line 6) to select which of the nodes is going to be analyzed to move its VMs away from it. This function introduces heuristics to evaluate the function used to sort the items for the FFd algorithm. In this case we have implemented the "select the node with more VMs hosted" and "select the node with fewer VMs hosted" functions to validate VMCA, but it is possible to create some more complex criteria to select the order in which the nodes are going to move its VMs away. We can introduce heuristics that consider the consumption of the nodes (e.g. select the node that is consuming more in first place), heuristics that take care of the cost of migrating the VMs from one host (e.g. select the node that would cost less to migrate its VMs in first place), or even heuristics that only take into account VM specific features (e.g. select the node that has the biggest VMs in first place). Evaluating these criteria falls out of the scope of this paper, as we are not trying to compare heuristics.

The rest of the algorithm is almost the same, adapted to this alternate scheme: lines 9 to 15 apply a FFd algorithm to schedule the VMs from the selected host

Algorithm 2 The Iterative BF-FFd analysis algorithm to redistribute VMs

```
1: nodes = detect_stable_nodes(monitor)
2: possible_destinations = nodes with VMs
3: simulator = clone(monitor)
4: repeat
5:    migration_plan = []
6:    possible_migrations = []
7:    for all node in nodes do
8:       migrations = []
9:       local_simulator = clone(simulator)
10:      for all vm in node.vms do
11:         destination = platform_scheduler.schedule(
            local_simulator.hosts_information(), vm, possible_destinations - node)
12:         if destination is not None then
13:            migrations.append(vm, destination)
14:            local_simulator.make_migrations([(vm, destination)])
15:         end if
16:      end for
17:      if len(migrations) == len(node.vms) then
18:         reward, cost = evaluate(migrations)
19:         possible_migrations.append((reward, cost, node, migrations))
20:      end if
21:   end for
22:   (node, migrations_selected) = select_migration(possible_migrations)
23:   nodes.remove(node)
24:   for all (vm, destination) in migrations_selected do
25:      possible_destinations.remove(destination)
26:   end for
27:   simulatior.make_migrations(migrations_selected)
28:   migration_plan.append(migrations)
29: until no node is empty or there are no stable nodes pending of analysis
```

into other hosts and lines 16 to 24 apply the migrations (in case that every VM from the selected host can be moved away to other hosts) and adjust the hosts that are considered as stable and those that can host VMs.

The asymptotic cost of Algorithm 3 is $O(h * max(n_i) * cost(scheduler))$, where h is the number of stable nodes (those whose VMs have not changed for a period of time) and n_i is the number of VMs in the stable node i. In this case, the upper bound of the cost of this algorithm is $O(H * N * cost(scheduler))$, which is a linear cost with respect to the number of nodes instead of the quadratic cost of Algorithm 2 (where N is the number of VMs in the platform and H the number of physical hosts).

Algorithm 3 The Iterative FFd-FFd analysis algorithm to redistribute VMs

```
1: nodes = detect_stable_nodes(monitor)
2: possible_destinations = nodes with VMs
3: simulator = clone(monitor)
4: repeat
5:    migration_plan = []
6:    node = select_node_to_empty(nodes)
7:    migrations = []
8:    local_simulator = clone(simulator)
9:    for all vm in node.vms do
10:      destination = platform_scheduler.schedule(
         local_simulator.hosts_information(), vm, possible_destinations - node)
11:      if destination is not None then
12:         migrations.append(vm, destination)
13:         local_simulator.make_migrations([(vm, destination)])
14:      end if
15:   end for
16:   if len(migrations) == len(node.vms) then
17:      possible_destinations.remove(node)
18:      for all (vm, destination) in migrations do
19:         possible_destinations.remove(destination)
20:      end for
21:      nodes.remove(node)
22:      simulatior.make_migrations(migrations)
23:      migration_plan.append(migrations)
24:   end if
25: until no node is empty or there are no stable nodes pending of analysis
```

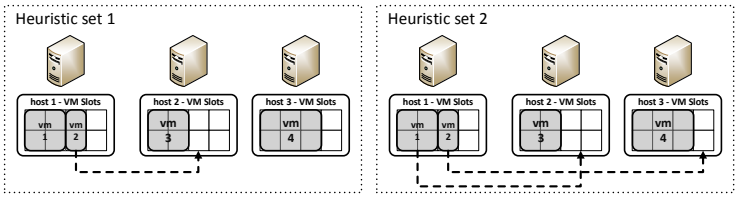


Figure 5.4: Different criteria provide different results.

5.4.4 Execution of the migration plan

Once the migration plan has been calculated, VMCA will schedule and make the migrations to try to achieve the new VM distribution in the hosts. In the case of the algorithms considered in VMCA, it is not possible to find any dependency between the migrations. Once a VM is moved, it will not be moved until a period of time has passed, and the migration plan has been fully carried out. Moreover, two migrations can be carried out in parallel, if the source and destination hosts of one migration are not the source or destination host of the other migration. So the migrations in the migration plan can be re-arranged to try to make migrations in parallel to reduce the total time for executing the migration plan.

Once the migration plan has been started, the analysis of the platform will not be performed again until the migration plan is fully executed, or a disruptive event occurs. This kind of events are those that make that the platform enters into an unexpected state. Some examples of the *disruptive events* are: a new VM is executed in the platform, an execution of a migration fails, a node is powered on or off, etc.

5.5 Integrating VMCA with the policies of the platform

Integrating with the virtualization platform while trying to reduce energy consumption is a complex task. So when deploying VMCA in production some issues should be addressed. On one side, it is important to notice that the configuration of the different criteria and techniques for re-placing the VMs will obtain different results, as it is shown in Figure 5.4. In the left side of the figure, VM_2 is moved first to $host_2$, so there is no place for VM_1 anywhere. In the right side of the figure, VM_1 is moved first to $host_2$ and then VM_2 can be moved to $host_3$ to get $host_1$ empty. In this case, the important part to adapt is the criteria to apply the FFd order for scheduling the VMs, but a similar situation can happen for the case of the mechanism to select the node whose VMs are moved away to other hosts in first place (the function to evaluate the cost and reward in Algorithm 2 or the function to select the host in Algorithm 3).

There are also other problems that should be addressed when integrating with the scheduler of the CMP, such as reducing energy consumption and the local scheduler may pursue opposite objectives. In the case of ONE, the scheduler included in the bundle package has four different policies to schedule the VMs (apart from the customized policies) [101]: (1) Stripping the VMs, to maximize resources available for the VMs by spreading the VMs in the hosts, (2) Packing the VMs, to minimize the number of hosts in use by packing the VMs in the hosts to reduce VM fragmentation, (3) Load-aware, to maximize resources available for the VMs by using those nodes with less load and (4) Fixed, where hosts will be ranked according to a prioritized order for the hosts.

In case that we are using the Stripping criteria for ONE, when re-locating the VMs, these will be tried to be distributed among the different hosts. The consequence will be that these hosts that are receiving VMs will not be considered stable for a while. Consider the scenario of using the *stripping scheduler* and having three equal hosts (H_1, H_2 and H_3). At one moment, we have 2 VMs on each host, but all the VMs can be hosted on one host. If we try to move the VMs of H_1 to other hosts, the first one will be allocated on H_2 and the second one on H_3. That will prevent from moving the VMs of H_2 to H_3 because H_2 is not considered *stable*, as it has just received one VM. Under that scenario, getting the minimum physical hosts to host the VMs will imply the payload of an extra period of *stability* and an extra analysis of VMCA. A change from the stripping policy to the packing policy would make the deal.

5.6 Experiments with VMCA

In order to validate VMCA, we have deployed it in an on-premises cloud platform managed by ONE. The platform consists of 8 dual processor with 14 core nodes (28 cores per node), with 64 Gb. of RAM and a shared storage system of 10 Tb., backed by a Storage Area Network (SAN) where the hard disks are stored as volumes. This on-premises cloud is used in production by our research group both for providing computational resources to the users. The VM types that are allowed in the platform are shown in Table 5.1, and they resemble the default types in the well known OpenStack CMP.

There are different possible configurations to deploy VMCA. So instead of choosing any combination of parameters for the deployment of VMCA in production platform, we have made a study to select a suitable configuration for the platform.

Type	Memory	CPU (Cores)	disk (GB.)
Tiny	512 MB.	1	10
Small	2 GB.	1	20
Medium	4 GB.	2	20
Large	8 GB.	4	50
Extra Large	16 GB.	8	50

Table 5.1: Types of Virtual Machines.

5.6.1 Selecting a configuration of parameters

To select a criteria for the deployment of VMCA, we have tried to identify a configuration of the available possibilities that would provide good results for the platform. For each of the available algorithms we have implemented several criteria for the customizable functions and, at the end we have constructed nine configurations that are shown in Table 5.2.

These configurations should be considered as a sample of common configurations that are used in the context of this paper to validate VMCA, and do not try to be an exhaustive survey of the posibilities of server consolidation heuristics. There are other works such as [67] or [115] that try identify which kind of heuristics provide better results in FFd-like algorithms for server consolidation.

In these algorithms we are using the expression $|R_h^u|$ to refer to the normalized amount of used resources in host h. It is calculated using the expression (5.1), which is the weighted euclidean distance normalized. In this expression m_h^u, c_h^u and d_h^u represent the quantity of memory, cpu and disk used by VMs in host h, $\max m$, $\max c$ and $\max d$ are the maximum amount of memory, cpu and disk available in any host in the platform, and α, β and κ are the weights that represent the importance of memory, cpu and disk, respectively in the platform. We are also using the normalized amount of the free resources ($|R_h^f|$, calculated applying the same expression to the free resources of host h). In our tests, we have set $\alpha = 1$, $\beta = 1$, $\kappa = 0$ (κ is set to zero because we have a common backend for the disk, so the space consumed is not relevant to differentiate the VMs).

$$|R_h^u| = \frac{\sqrt{(\alpha \cdot \frac{m_h^u}{\max m})^2 + (\beta \cdot \frac{c_h^u}{\max c})^2 + (\kappa \cdot \frac{d_h^u}{\max d})^2}}{\sqrt{\alpha^2 + \beta^2 + \kappa^2}} \tag{5.1}$$

ID	Name	Description
1	FF-fewer VMs	FF algorithm that moves the VMs from the host with fewer VMs in first place.
2	FF-More VMs	FF algorithm that moves the VMs from the host with more VMs in first place.
3	FF-Less Used resources per VM	FF algorithm that moves the VMs from the host that has less normalized amount of used resources per VM in the host (calculated as $\|R_h^u\|/count(vm, vminh)$).
4	FF-More Used resources per VM	FF algorithm that moves the VMs from the host that has more normalized amount of used resources per VM in the host (this is the same algorithm than the previous one, but with the reversed criteria).
5	BFd-Quicker sequence	BFd algorithm that estimates the time needed for migrating each VM and selects the sequence of movements that gets the VMs away from one host to others quicker.
6	BFd-Less time per VM	BFd algorithm that estimates the time needed for migrating each VM and selects the sequence of movements that gets the VMs away from one host to others quicker per VM (as a mean). This is the same algorithm than the previous one, but dividing the time by the number of VMs.
7	BFd-Balance free resources	BFd algorithm that tries to balance the distribution of the free resources of the servers that are hosting VMs. To compute out how the free resources are balanced in the platform, we calculate the variance of the normalized amount of the free resources in the hosts, using expression (5.2) (less variance means more equality in the distribution).
8	BF-Unbalance free resources	BF algorithm that tries to unbalance the distribution of the free resources of the servers that are hosting VMs. It is the same algorithm than the previous one, but with the reversed sort.

Table 5.2: Configurations of criteria for VMCA.

$$S^2 = \frac{1}{hosts} \sum_{h=1}^{hosts} (|R_h^f| - \overline{R^f})^2 \tag{5.2}$$

To compare the behaviour of these configurations, we have generated thousands of random synthetic platform configurations with random workloads of VM deployed on them, and we have analyzed the use cases using VMCA, applying each one of the nine configurations of the implemented criteria, and the two VM scheduling policies available in ONE that do not depend on the features of the VMs while they are running: *stripping* and *packing*. For each of the analysis, we have recorded the number of free nodes at the end of the analysis and the number of migrations that would be needed to achieve the final distribution, assuming that the migration plan can be carried out and nothing happens in-between. Then we have calculated the number of use cases in which each configuration got the best final result among the different analysis (getting the maximum number of empty hosts).

In the case of platforms that were built up from homogeneous hosts (i.e. platforms in which the hosts had the same amount of physical resources), we got that almost everytime (99% of times), all the configurations got the same number of empty hosts. In the case of heterogeneous platforms the results were very different. Figure 5.5 shows the percentage of times that each configuration achieved the

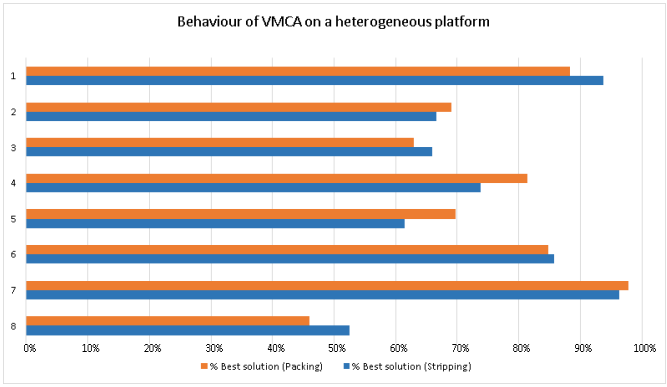


Figure 5.5: Percentage in which a configuration of criteria gets the maximum free servers when hosting all the VMs deployed in the platform, on a sample of thousands of heterogeneous random platforms (higher is best).

maximum number of empty hosts, for the case of heterogeneous platforms, for both the *stripping* and the *packing* VM scheduling policies. The figure reveals that the best configurations are "FF-fewer VMs" (1), "BFd-Less time per VM" (6) and "BFd-Balance free resources" (7). From the results, it seems that the configurations of criteria that get the best results tend to move the VMs away from the hosts that have fewer VMs: try to get a node empty using the minimum number of migrations.

From the synthetic results, in the case of our homogeneous platform, it seems that getting the maximum number of empty nodes is not a matter of criteria. Any of the implemented configurations is likely to get a good result if it is allowed to make enough migrations. Then, the main difference is the number of migrations needed to reach the stable state. A migration is a costly task that can drive to a downtime of a VM or to a QoS penalty in many cases. Therefore, the lower the number of migrations is, the better the migration plan obtained is. Figure 5.6 shows the percentage of use cases in which each of the configurations of criteria that provided best results got the minimum number of possible migrations (when getting the maximum number of free nodes), in the case of the stripping scheduler, which is the one used in the target CMP.

In this figure we can see that the configuration that provides better results (less migrations) is "FF-fewer VMs" (1), in both homogeneous and heterogeneous platforms, but it has even better results for homogeneous platforms. Therefore, this is the configuration that we selected for the production tests.

It is also noticeable that when we focus on the different sheduling options, the "packing scheduler" always needed to make less migrations than the "stripping

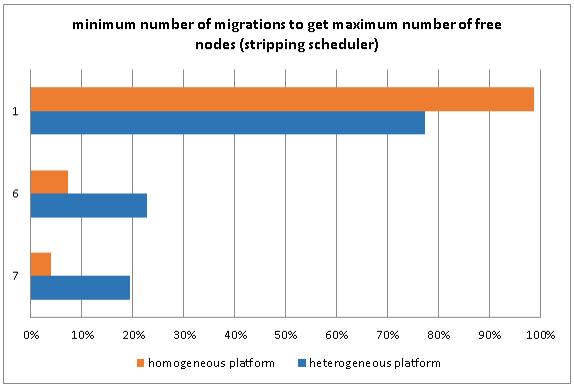


Figure 5.6: Percentage in which a configuration of criteria gets the minimum needed migrations to get the maximum free servers when hosting all the VMs deployed in the platform, on random platforms when using the stripping VM scheduler in the CMP (higher is better).

scheduler", under the same other parameters. This is because the packing scheduler tries to schedule the VMs in the physical hosts that have more hosted VMs and that helps to keep the VMs concentrated into a lower number of physical hosts. That scheduler facilitates clearing the other physical hosts. But we have to take into account that the underlying scheduler is a constraint of the virtualization platform with which VMCA has to integrate. In the case of our tests, the policy for the ONE platform is "stripping" and it cannot be changed since it is used in production.

5.6.2 Tests into the production platform

As a result from the synthetic analysis, VMCA was configured to use algorithm 3, applying the criteria of selecting in first place the node that had the less number of VMs ("FF-fewer VMs"). The VM scheduler in ONE is "stripping the VMs" (which is the default criteria for a fresh ONE installation). We have executed VMCA in 10 different periods of time, that we will consider the use cases from C_1 to C_{10}. These use cases correspond to moments in which VMCA was not run for several days and the plaform was in a state driven by the lifecycle of the VMs and the interaction of the users with the platform. The state of the platform at the start of each execution is summarized in Table 5.3. In this table, each row C_j represents one use case named according to the first column. The second column represents the total number of VMs that were hosted by the platform, and each of the columns named as H_i from H_1 to H_8 represent the different physical hosts in the platform. The value contained in each cell in the table is the tuple formed by

Case	VMs	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
C_1	45	1,1,1,0,0	1,2,2,1,2	2,0,2,0,0	2,4,2,0,0	0,1,1,1,1	1,1,2,0,1	0,2,0,0,1	0,4,5,1,0
C_2	53	3,3,2,0,0	2,1,1,0,0	0,1,5,1,1	2,2,4,0,0	1,2,3,0,2	1,2,4,0,0	0,2,1,0,1	1,2,1,1,1
C_3	39	0,0,2,0,0	1,4,0,0,0	1,2,1,1,0	1,0,3,1,0	0,2,1,2,0	0,1,4,2,0	0,1,1,2,1	0,0,3,1,1
C_4	41	0,2,3,0,0	0,3,3,0,0	2,2,3,0,0	2,2,2,0,0	1,1,1,0,2	1,3,0,0,0	0,3,0,1,0	0,1,2,0,1
C_5	62	2,3,6,1,0	1,4,4,2,0	1,4,2,1,0	1,6,1,1,0	1,3,3,0,0	0,2,0,1,0	1,1,3,0,0	2,2,1,1,1
C_6	45	2,0,2,1,0	1,2,3,0,1	0,1,1,1,0	0,0,2,1,0	2,4,3,1,0	2,0,2,0,0	1,4,1,2,0	2,2,0,1,0
C_7	61	2,2,5,2,0	3,0,4,0,2	2,1,5,1,0	3,2,1,1,1	0,4,2,0,2	1,0,3,1,1	0,1,2,0,1	1,2,0,2,1
C_8	58	0,4,4,1,0	1,2,4,2,1	2,2,1,0,1	1,1,1,0,0	1,6,3,1,1	1,1,5,2,0	1,1,3,0,0	0,2,1,1,0
C_9	73	3,2,2,2,0	1,3,2,1,1	0,5,4,0,1	1,3,4,2,1	0,4,3,2,1	0,2,2,1,1	0,9,1,0,0	1,2,3,3,0
C_{10}	53	1,1,1,1,0	2,3,5,0,0	1,1,1,0,2	1,1,2,0,0	0,2,0,2,0	2,3,1,0,0	1,6,5,1,0	1,1,2,2,1

Table 5.3: States of the platform before running VMCA.

Case	VMs	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
S_1	45	2,4,3,0,2	-	2,3,5,1,1	2,5,2,1,1	-	1,3,5,1,1	-	-
S_2	53	3,4,4,0,0	-	0,2,7,1,1	-	2,2,4,0,2	-	2,4,3,0,1	3,3,3,1,1
S_3	39	0,1,5,2,1	1,5,4,0,0	2,2,3,2,1	-	0,2,3,5,0	-	-	-
S_4	41	0,2,5,0,2	-	4,6,5,0,1	-	-	-	2,9,4,1,0	-
S_5	62	2,4,9,1,0	-	1,7,6,2,0	-	-	-	2,8,4,2,0	4,6,1,2,1
S_6	45	4,1,4,3,0	-	-	1,4,4,2,0	-	4,2,4,0,1	1,6,2,2,0	-
S_7	61	2,2,5,2,0	3,1,4,0,2	2,3,5,1,1	3,3,1,1,2	-	1,0,7,1,1	-	1,3,0,2,2
S_8	58	0,5,6,1,0	2,2,4,2,1	-	-	1,6,3,1,1	-	2,2,6,0,1	2,4,3,3,0
S_9	73	3,3,3,2,1	1,5,3,2,1	-	1,3,4,2,1	0,5,3,2,1	-	0,10,3,0,1	1,4,5,3,0
S_{10}	53	3,5,6,2,0	-	2,2,2,1,2	1,8,6,1,0	-	-	-	3,3,3,2,1

Table 5.4: States of the platform after running VMCA.

the number of (tiny, small, medium, large and extra large) number of instances, hosted in host i for use case j.

The resulting distribution of the VMs after each analysis is summarized in Table 5.4. For each use case C_i we have obtained a new distribution of VMs S_i that is the result of moving VMs from one hosts to others using live migration techniques. As in Table 5.3, each row of this table contains the number of tiny, small, medium, large and extra large instances, respectively, hosted in host i when VMCA has made the migrations and has achieved a stable state. In case that a node is not hosting any VM, it has been marked using a "-" to ease its visual identification.

According to the objectives of this work, *a better distribution* will be that with the maximum number of empty nodes. The optimal number of physical hosts needed to host an infrastructure of VMs can be theoretically obtained from the number of physical hosts used by applying a *FFd* algorithm to place the VMs in an empty platform using the expression (5.3) derived from the study made in [134]. In this expression, L represents the set of VMs in the platform, $FFD(L)$ the number of

Case	Total time	Total migrations	Iterations	Hosts used	FFd_i	OPT(L)
S_1	1975.46	25	1	4	4	3 (2,45)
S_2	1574.70	19	1	5	5	4 (3,27)
S_3	1966.45	24	2	4	4	3 (2,45)
S_4	2480.83	33	3	3	3	2 (1,64)
S_5	2477.83	31	2	4	4	3 (2,45)
S_6	2033.08	25	1	4	4	3 (2,45)
S_7	926.91	12	1	6	6	5 (4,09)
S_8	1545.54	18	1	5	5	4 (3,27)
S_9	1292.61	16	1	6	6	5 (4,09)
S_{10}	2747.54	33	1	4	4	3 (2,45)

Table 5.5: Evaluation of the results obtained by VMCA.

physical hosts needed to host the VM set L obtained as a result of applying the FFd algorithm, and $OPT(L)$ the number of physical hosts needed to host the VM set L in the optimal solution.

$$(FFD(L) - 1) \cdot \frac{9}{11} \leq OPT(L) \tag{5.3}$$

The evaluation of the results obtained by VMCA is summarized in Table 5.5. The column titled "Total migration" represents the time in seconds since VMCA was started to the instant when the platform was considered as "stable" by VMCA (that means that VMCA could not make any extra migration). This time includes the VMCA analysis and all the migrations. The total execution time for the VMCA analysis algorithm (3) was less than 0.02 seconds in any of the cases. The column "Total migrations" is the total number of migrations to get to the distribution of VMs S_i from the starting situation C_i. Column "Iterations" refers to the number of times that the analysis of VMCA took place. The problem of needing multiple iterations is explained in section 5.5, but it occurs because VMCA needs to move a VM several times and it has to wait for the VMs to become stable again. In our case, this was due to the "stripping scheduler" used by ONE in our platform. Column "Hosts used" refers to the number of hosts needed to host all the VMs when achieving to the distribution obtained in S_i. Column FFd_i is the number of hosts needed to host the VMs if applied the aforementioned FFd algorithm to host all the VMs if the platform was empty. The last column named "OPT(L)" is the optimal number of hosts, according to expression (5.3) using the value FFd_i (for the purpose of VMCA, a fraction of the host is considered as a whole host, although the obtained value is included between parentheses).

In this table we can see that, in all of the cases, we have achieved a solution in which we obtained the same number of hosts needed to host all the VMs in the

platform, that a offline FFd algorithm would obtain if it distributed the VMs in a platform starting from the beginnning (i.e. when the platform was empty, but knowning which VMs were to be created). In this case, VMCA had the initial constraint of starting from a given distribution, and the solution has been obtained in a reduced number of steps. That means that VMCA has always obtained a very good solution, as it is always only one host greater than the theoretical optimal value.

Although we have always obtained the same value than using the FFd algorithm we cannot infer that VMCA will always obtain that value, facing any distribution of VMs. In the use cases shown in this paper, we have obtained that nice solution probably because the platform is not heavily occupied and there was enough empty space in the platform to make the needed migrations. Notice that in constrained scenarios with less empty space, VMCA might require an extra number of migrations or may get solutions that are not so near to the theoretical optimal value.

Regarding the total time to reach from use case C_i to S_i we can see that it represents a large amount of time that ranges from 15,45 minutes in case S_7 to 41,35 minutes in case S_4. That is due to the time needed to effectively migrate the VMs (which was 80,59 seconds as a mean in our tests) and the fact that VMCA had not been active for a large period of time. If VMCA was run more frequently, the subsquent analysis would not imply too many VM movements. In a platform where there are several empty nodes (they would ideally be powered off by automatic mechanisms such as CLUES [40]) and there is little space to host VMs, the task of VMCA is very limited and it will probably not try to move any VM if it could not get a node empty.

5.7 Conclusions and future work

In this paper we have presented VMCA, which is a Virtual Machine Consolidation Agent that tries to re-arrange the distribution of the VMs in an on-premises cloud computing deployment to try to use the physical resources in a more efficient manner. We have shown an iterative BFd-FFd algorithm to re-arrange the distribution which is based on multidimensional Bin Packing. This algorithm is a contribution to the state-of-the-art algorithms to re-place the VMs. We have also created an iterative FFd-FFd algorithm that reduces the computational cost and enables to introduce VMCA in a production ready platform to get physical hosts idle. The result is that VMCA can be considered as a DPM technique since it has a reduced computational cost, and it can be integrated with green computing techniques so that hosts can be powered off to save energy. Furthermore VMCA is not only a theoretical exercice, and we have integrated the results of this paper in a product that can be downloaded from [3] in source-code.

Since VMCA makes use of FF-like algorithms, it is based in heuristics. The configurations of criteria implemented in the framework of this paper are very straightforward, but we consider that they are valid for common on-premises deployments. In this paper we have demonstrated that these configurations obtain near-optimal values at a reduced computational cost. We have not tried to make a survey of different criteria for packing VMs on to hosts. Instead the objective of this paper is to create light versatile algorithms that achieve good results and can be used in medium-high used production platforms. Moreover, the paper focuses on the consolidation of the VMs instead of creating a new scheduler that affects the criteria of the existing scheduler. The algorithms shown in this paper enable to create more sophisticated criteria that can take benefit from runtime features to enhance the migration plan at a reduced computational cost (e.g. taking into account the dirty memory pages to select which VMs are likely to live-migrate in first place). Moreover both algorithms try to interfer as less as possible with the scheduling policies of the platform. So they do not implement any VM scheduler and fully integrate with the VM scheduler used in the platform.

The use cases shown in this paper show the real usage of a scientific cloud platform in a production state. VMCA got profit from the fact that the platform is not under an extreme ocupation, so the migrations were feasible and VMCA got idle resources that could be powered off.

As a immediate work for VMCA, we have to integrate it with other on-premises platforms such as OpenStack. Previous releases of this middleware made hard to migrate VMs from one host to other, but current version provide an enhanced support for making it.

As a future work, VMCA will focus in resource defragmentation and integration with elastic infrastructure managers such as CLUES to consider the possibility of powering on hosts that have been powered off because they were empty, to move VMs to them, in order to enhance the distribution of VMs. The work described in this paper focuses on server consolidation instead of a strict resource defragmentation. The main difference for these two cases is the pursued objective. Whereas a “server consolidation” approach tries to reduce the number of physical hosts that are hosting the VMs, a “resource defragmentation” approach would focus on balancing the usage of the resources. That would mean that, as an example, in a ”server consolidation” approach the VMs will not be moved if such movements do not result in a reduced number of physical hosts, but a “resource defragmentation” approach will move the VMs if they enhance the resource distribution even if no hosts become idle.

Chapter 6

Automatic Memory-based Vertical Elasticity and Overcommitment on Cloud Platforms

Submitted as

Germán Moltó, Miguel Caballer, Carlos de Alfonso, "Automatic Memory-based Vertical Elasticity and Oversubscription on Cloud Platforms".

Abstract

Hypervisors and Operating Systems support vertical elasticity techniques such as memory ballooning to dynamically assign the memory of Virtual Machines (VMs). However, current Cloud Management Platforms (CMPs), such as OpenNebula or OpenStack, do not currently support dynamic vertical elasticity. This paper describes a system that integrates with the CMP to provide automatic vertical elasticity to adapt the memory size of the VMs to their current memory consumption, featuring live migration to solve overcommitment scenarios, without downtime for the VMs. This enables an enhanced VM per host consolidation ratio while maintaining the Quality of Service for VMs, since their memory is dynamically increased as necessary. The feasibility of the development is assessed via two case studies based on OpenNebula featuring i) horizontal and vertical elastic virtual clusters on a production Grid infrastructure and ii) elastic multi-tenant VMs that run Docker containers coupled with live migration techniques to alleviate overcommitment. The results show that memory oversubscription can be integrated on CMPs to deliver automatic memory management without severely impacting the performance of the VMs. This results in a memory oversubscription framework combined with live migration techniques to safely enable transient overcommitment of physical resources in a CMP.

6.1 Introduction

Elasticity [54], or the ability to rapidly provision and release resources, is one of the integral characteristics of Cloud Computing. Horizontal elasticity is commonly employed to provision additional computational nodes in order to sustain the quality of service delivered by an architecture deployed on a Cloud platform, specially after an increase in the number of users or workload. Horizontal elasticity has been extensively studied in the past, with services already available for public Clouds, such as Auto Scaling[1] for Amazon Web Services (AWS), and Heat[2] for OpenStack.

Instead, vertical elasticity enables to increase and decrease the number of resources allocated to a single Virtual Machine (VM). The increased support to techniques such as *memory ballooning* [127] and *CPU hot plugging* by popular hypervisors such as KVM, Xen or VMware paves the way for vertical elasticity to be adopted by Cloud platforms. However, popular open source CMPs such as OpenNebula and OpenStack do not currently support vertical elasticity without downtime. As an example, the KVM hypervisor fully supports memory ballooning in order to dynamically modify the allocated memory to a given VM without any downtime, and the main Operating Systems (OSs) support this feature. However, CMPs require to stop the VM in order to change its allocated memory.

In our previous work [96] we demonstrated the benefits of introducing vertical elasticity to dynamically adjust the allocated memory of VMs to their current memory consumption, specially for applications with dynamic memory requirements during their execution. In fact, the number of VMs that one physical machine can support is typically limited by its memory size. Besides, users tend to overestimate the amount of memory required by their applications resulting in unused memory that could be dedicated to additional VMs running on the same physical machine [119]. In addition, CMPs typically provide templates, such as the *flavors* in OpenStack, which enforce a certain amount of memory size regardless of the actual memory requirements of the application. Just as airlines sell more tickets than available seats (i.e. oversubscribe the plane) in the hope that some passengers do not show up, Cloud providers can oversubscribe their resources by deploying additional VMs in a host, in the hope that VMs will actually use less memory than initially requested.

However, this situation might incur in memory overcommitment for a host, where the sum of memory of its VMs exceeds the physical memory of the host. This situation, at the Cloud infrastructure level, is called oversubscription [60], which is a technique that can lead to an increase in the number of VMs per physical host though it can have an impact on the Quality of Service and probably violate the

[1] Auto Scaling: `http://aws.amazon.com/autoscaling`
[2] Heat: `https://wiki.openstack.org/wiki/Heat`

Service Level Agreement established by the Cloud provider. However, oversubscription can enable Cloud providers to better use the available memory in their physical systems if the appropriate countermeasures are introduced. As Williams et al. [129] state, in well-provisioned datacenters, overload is unpredictable, relatively rare, uncorrelated, and transient, indicating that an opportunity exists for memory oversubscription in those facilities.

In this paper we introduce CloudVAMP (*Cloud Virtual machine Automatic Memory Procurement*) a memory oversubscription framework that can be integrated in an on-premises CMP to automatically monitor the VMs and to dynamically adjust their allocated memory to adapt to the current memory requirements of their running applications. Without any user intervention, the system automatically manages the memory of the VMs (or a subset of VMs) irrespective of the memory initially allocated by the user. This introduces enhanced VM consolidation per physical node while live migration is employed in case of overcommitment.

The remainder of the paper is structured as follows. First, section 6.2, describes the related works in the area of vertical elasticity and memory oversubscription. Next, section 6.3 briefly describes the problem addressed and the underlying technologies employed. Later, section 6.4 describes the architecture of CloudVAMP, in order to manage vertical elasticity in an on-premises Cloud. Then, section 6.5 describes two case studies carried out to assess the behaviour and benefits of the developed platform. Finally, section 6.6 summarises the paper and points to future work.

6.2 Related work

There can be found other works in the literature that have focused on vertical elasticity and memory oversubscription, though most of them are just focused on virtualisation platforms and, thus, not covering the intricacies of CMPs. In [39], the authors propose an Elastic VM architecture that scales the number of cores, CPU capacity and memory using the Xen hypervisor. They study the adaptation of the VM capacities to the requirements of a web application. However, their case study does not address memory scaling but only increasing the virtual CPU allocation.

In [118], a system to provide proactive dynamic memory allocation based on the Bayesian predictions is introduced to increase server consolidation. In [59], the Ginkgo memory overcommitting framework is introduced, which dynamically estimates VM memory requirements for applications and automates the distribution of memory across VMs through ballooning techniques. It uses performance profiles of the applications to characterise incoming load. The case study focuses on VMs running on a single physical host. These two works focus on a set of virtual machines running in a single hypervisor, while our work focuses at the

whole infrastructure provided by the CMP, involving memory management across multiple physical hosts. In [113] an extension of ballooning techniques is applied to applications (using as example a database engine and the Java runtime) to reallocate memory between memory managers of different applications. However, these requires modifications of the Xen Balloon Driver and does not address the overcommitment problems that arise in CMPs.

Overdriver [129] is a system to mitigate the problems that arise in oversubscribed virtualised hosts, by automatically deciding when to use network memory, using a cooperative swap approach, or live migration depending on whether the workload is considered to be transient or sustained, respectively. However, they do not consider memory ballooning as a mitigation strategy for oversubscription. This is the case of the work by Hwang et al. [61] where a system to opportunistically use memory during periods of light loads is introduced. For that, they allow the hypervisor to dynamically allocate memory at fine granularity, focusing on disk and application level caches. The work by Baset et al [17] describes the different techniques employed to alleviate oversubscription and mitigate the overload. They designed an event-driven simulator to develop an understanding of oversubscription. However, they focus exclusively on offline and live migration but ballooning techniques are discarded.

Regarding memory ballooning, the KVM hypervisor has a project called Automatic Ballooning [72] where the management of the balloon is automatic. When the host is under pressure, it asks guests to relinquish memory. When a guest detects memory pressure, it gets some memory back from the host. This requires Linux kernel 3.10+ and a specific version of QEMU. However, this approach focuses exclusively on the VMs running on a single physical machine and, thus, it does not solve the problems that arise when the host is overcommitted, specially within an on-premises Cloud, where VMs could be live migrated across other physical hosts to restore the level of service.

The most similar work to our proposal is the one carried out by Litke [80], where the Memory Overcommitment Manager (MOM) is introduced. This system requires a daemon to be installed in the VMs to gather information regarding the memory usage from the VMs and a policy actuator that runs on the host's OS to decide when to increase or decrease memory though memory ballooning techniques. While this approach is of interest for a virtualisation platform where VMs have dynamic memory requirements, it does not introduce countermeasures for overcommitted hosts.

As the authors of [60] state, much of the research conducted thus far has focused on managing oversubscription of a single physical machine though this narrow focus is rather limiting. While other projects successfully manage memory overcommitment at a host level, we have not found any previous work that automatically manages oversubscription in an on-premises Cloud. Therefore, building on pre-

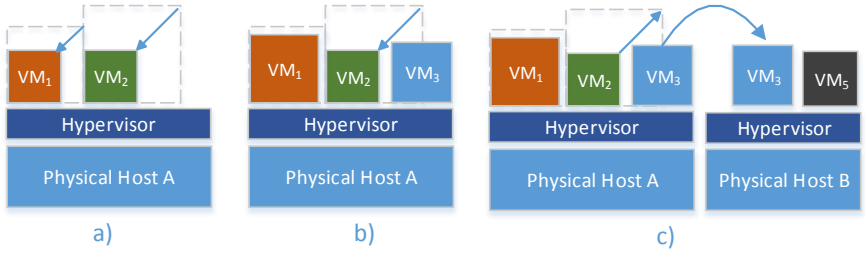


Figure 6.1: Depiction of an on-premises Cloud with support for dynamic memory management. a) the allocated memory of the VMs has been reduced because there is enough free memory, b) a third VM is deployed on the same physical host and c) live migration is employed to solver overcommitment of memory in the physical host.

vious works in the area we introduce CloudVAMP, a memory oversubscription framework combined with live migration techniques to safely enable automatic transient overcommitment of physical resources within a CMP.

As opposed to previous work, our approach considers memory management not at a single physical host but at the the whole infrastructure level in an on-premises Cloud. In addition, CloudVAMP is responsible to safely reduce the allocated memory to the VMs in order to enable transient oversubscription. The fact that CloudVAMP is integrated with a CMP enables the latter to deploy additional VMs to the same physical host according the stablished scheduling policies within Cloud infrastructure. Therefore, CloudVAMP does not only manage, but also enables, oversubscription at the Cloud infrastructure level, which is a feature not included in previous aforementioned related works.

In addition, we introduce a proof-of-concept open source implementation based on OpenNebula, which can be easily adapted to other CMPs (such as OpenStack). Therefore, this introduces unattended efficient memory management for on-premises Clouds.

6.3 Problem, Methods & Materials

This paper is based on the following underlying technologies. First, KVM [68], a popular open source hypervisor that fully supports memory ballooning. Second, OpenNebula [98], an open-source Cloud Management Platform that manages the life cycle of VMs on a physical infrastructure. However, the system described in this paper can also be adapted to work with a different hypervisor and or CMP.

According to [17], there are different mechanisms to mitigate the problems that arise with oversubscription: i) stealing, which allows a hypervisor to steal (actually borrow) resources from underloaded VMs running on the same physical host; ii) quiescing VMs, so that a VM is shut down and migrated offline to an underloaded physical machine; iii) live migration, to hot migrate VMs from an overloaded physical machine to an underloaded one; iv) streaming disks, to transfer the minimum portion of a VM's local disk to allow the VM to be started on another physical machine, and v) network memory, to use memory of another machine as a swap space over the network.

In this paper we focus both on memory ballooning and live migration techniques and its integration in a CMP. We rely on these techniques because they are fully supported on most hypervisors and by the main OSs (including Linux and Windows). Therefore, this enables to create a system that can be easily integrated in today's on-premises Cloud deployments to seamlessly leverage these techniques.

Figure 6.1 summarises the main problem that aims to be addressed. In *a*), two VMs (VM_1 and VM_2) have been deployed by a CMP on the same physical host (A). Depending on the scheduling configuration of the CMP this situation can be very frequent. For example, OpenNebula can be configured to use a packing scheduler and so, the VMs tend to be allocated to the same physical machine if there is enough memory available. In KVM, a deployed VM has both a *memorysize* and a *maxmemorysize* attribute. A VM cannot grow beyond the *maxmemorysize*, which corresponds to the memory initially allocated when the VM was created. However, its *memorysize* (the memory currently allocated to the VM) can range from the minimum amount of memory to support the OS, typically in the order of 200-300 Mbytes for a Linux VM [96], to its *maxmemorysize*. Notice that in *a*) the memory size of both VMs has been shrunk due to the usage of memory ballooning, because the applications running on the VM were not using that amount of memory. Then, in *b*) since there is enough available memory to host an additional VM (because the actual memory needed by VM_1 and VM_2 is less than the original amount requested), the CMP's scheduler has decided to allocate a new VM to that physical host.

Later, in *c*), VM_2 requires more memory because the application (or applications) running inside has requested so and, thus, the physical host incurs in overcommitment. Therefore, one or more VMs (in this case, only VM_3) have to be relocated to another physical host to restore as soon as possible the quality of service across the infrastructure managed by the CMP. In our case, this involves live migration, according to a certain policy, so that no downtime is introduced for the migrated VM.

6.4 Architecture

The architecture of CloudVAMP consists of three components:

- Cloud Vertical Elasticity Manager (CVEM). An agent that analyses the amount of memory actually needed by the VMs and dynamically updates the memory allocated to each of them, according to a set of customisable rules. It is an agent that queries the monitoring system of the CMP, and has access to the hypervisors (e.g. ssh access to the physical nodes of the on-premises Cloud). It can decide to live migrate VMs in order to restore the level of service under memory overcommitment situations.
- Memory Reporter (MR). An agent that runs in the VMs and reports to a monitoring system the free and used memory by the applications in the VM. This information must be available for CVEM, so it should be integrated within the CMP's monitoring system (as it has been currently implemented) or by relying on a third-party monitoring system (e.g. Ganglia).
- Memory Overcommitment Granter (MOG). A system that informs the CMP about the amount of memory that can be overprovisioned on the hosts, to be taken into account by the scheduler of the CMP.

Figure 6.2 depicts the architecture of the proposed system and how it fits in an on-premises Cloud. The proof-of-concept implementation is based on OpenNebula (ONE) and it is seamlessly integrated using the components that it offers. OpenNebula requires a cluster-based installation in which the main services are installed in the front-end node (ONE Front-end in Figure 2.*a*) whereas the VMs are deployed on the internal working nodes (ONE Host in Figure 2.*b*), where the KVM hypervisor (other hypervisors are supported as well) has to be installed.

The architecture of CloudVAMP has been implemented via lightweight Python-based agents. For example, CVEM runs alongside ONE to obtain the monitoring information regarding the actual memory usage of all the VMs in the infrastructure. For that, we rely on the MR, which runs in the VM. The MR agent periodically (by default every five seconds although it can be configured on a per-VM basis) reports the memory usage to OneGate[3] by properly querying $/proc/meminfo$ to obtain both the total and free memory in the VM as well as the usage of the swap space. We rely on the contextualisation mechanisms provided by OpenNebula to dynamically stage in the running VM the agent that periodically monitors the memory consumption and the memory available reporting back to OneGate. This enables CVEM to access centralised monitoring information about the memory usage of all the VMs deployed in the on-premises Cloud (by default every

[3]OneGate: `http://docs.opennebula.org/4.12/advanced_administration/application_insight/onegate_overview.html`

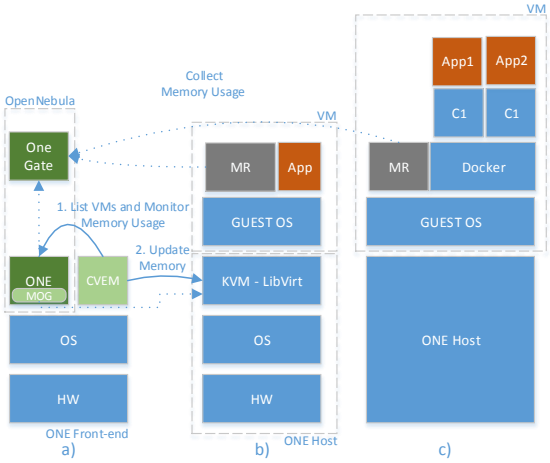


Figure 6.2: Architecture of an on-premises Cloud with support for the CloudVAMP. a) the OpenNebula (ONE) frontend host, b) a sample ONE host that executes VMs and c) a sample configuration employed for the case study.

five seconds). Notice that MR and CVEM are decoupled systems which can work at different frequencies. In addition, the MR only reports significative memory changes so it can run very frequently.

The usage of contextualisation avoids the need to have pre-packaged Virtual Machine Images (VMIs) with the MR agent pre-installed. Instead, by using the contextualisation offered by OpenNebula, our solution is independent of the VMI chosen by the user (though our proof-of-concept is based on GNU/Linux-based VMIs), since the agent is installed on-the-fly when the VM is deployed. Notice that, for other CMPs, DevOps tools such as Puppet or Ansible could also be used to dynamically deploy the MR agent right after the VM has booted.

Finally, to inform the ONE scheduler about the amount of memory from the hosts that can be oversubscripted, we have created a modified version of the KVM Virtual Machine Manager (VMM) monitoring driver component that is shipped with OpenNebula. This version calculates the amount of *stolen memory* from each host and instructs the ONE scheduler to use part of it to allocated additional VMs in that host. We define the *stolen memory* for a given physical host as the total amount of memory that CVEM has been able to freed from the different VMs running on that physical host, in our case, through the use of memory ballooning via KVM. CVEM decides to enlarge or shrink the VM's allocated memory depending on the actual memory usage reported by the MR to OneGate.

Notice that the current Allocated Memory (AM) to a VM is divided between the current Used Memory (UM) by the applications running inside and the Free Memory (FM) and, therefore, $AM = UM + FM$. The vertical elasticity rules implemented in CloudVAMP build on our previous work [96] to maintain a Memory Overprovisioning Percentage (MOP) of an additional 20% of the current UM. The goal is to keep that extra amount of free memory in case the application running in the VM starts requesting more memory. In our previous work we assessed the behaviour of different values of MOP, in particular 10% and 30% to understand the tradeoff between reducing the free memory in a VM at the expense of increasing the chances of an application to start thrashing due to lack of free memory in case the application requires a memory increase [96].

However, the vertical elasticity rules are only triggered if the percentage of free memory of the VM is smaller than 80% or greater than 120% of the MOP. This enables the system to only react when substantial changes in the used memory of the VM occur, thus removing unnecessary oscillatory memory changes. In these circumstances, CloudVAMP dynamically adapts the VM memory size using (6.1),

$$AM = UM \times (1 + MOP) \tag{6.1}$$

where AM is the newly allocated memory to the VM by the hypervisor and UM is the current used memory by the applications in the VM. As an example, a MOP of 20% means that the elasticity rule will only be triggered when the free memory of the VM is lower than 16% (80% of 20%) or greater than 24% (120% of 20%) of the used memory of the VM. If a VM has 1000 MB of AM and the application starts using 900 MB, then the new AM will be 1080 MB (900×1.2).

For the sake of clarity, Figure 6.3 shows the memory thresholds that trigger the vertical elasticity rules in an example VM (left part of the figure) that was initially deployed with 2500 MB and was subsequently downsized to 1200 MB (AM), of which 1000 MB are being used by the application (UM) and 200 MB are the free memory (FM) provided by the MOP (20% of the UM). Whenever the UM passes the Increase Memory Threshold (IMT) or the Decrease Memory Threshold (DMT) the vertical elasticity rule (6.1) is applied in order to maintain that extra 20% free memory. Any memory consumption changes between those thresholds will not trigger the elasticity rules to avoid unnecessary oscillations. In case the application starts demanding additional memory, the system allocates extra memory resulting in the case shown in the right part of the Figure 6.3, which corresponds to the VM with, for example, 2000 MB of UM.

The elasticity rule has been complemented with a fail-safe mechanism when thrashing has already occurred within a VM. In that case, the memory size increase should be much larger to rapidly counteract the devastating effects that thrashing has in application performance [43]. For that we use a mechanism that greatly

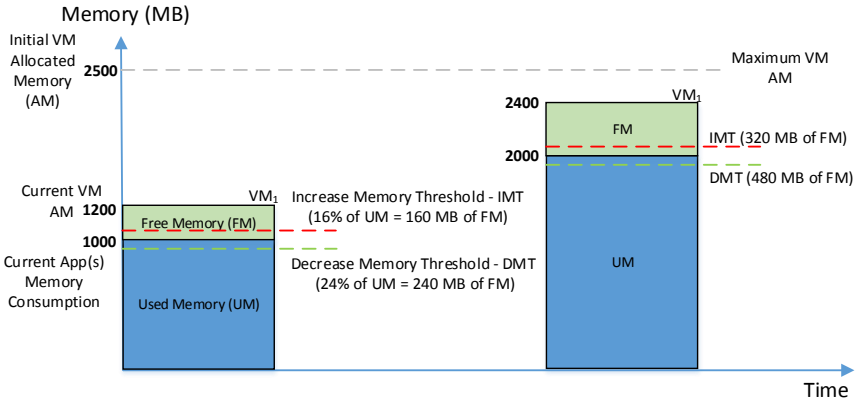


Figure 6.3: Memory thresholds that trigger the vertical elasticity rules in two example VM configurations. In the left, a VM with 1000 MB of Used Memory (UM) and, in the left, a VM with 2000 MB of UM. MOP=20% of UM.

inspires in exponential backoff [91]. If there is no available free memory in the VM, an additional 50% of the difference between the maximum memory and the current allocated memory is assigned. This enables to rapidly increase the allocated memory to the VM, attempting to scape from thrashing as fast as possible. If there is still shortage of memory, the same additional allocation of memory is performed. Finally, if the third monitoring interval still reports a shortage of memory in the VM (probably because the application running in the VM is requesting memory faster than the rate at which CloudVAMP is increasing the allocated memory to the VM), the VM is allocated its maximum memory size. Notice that any excess of allocated memory will be corrected in subsequent steps by CloudVAMP by reducing the allocated memory according to the rule in (6.1), leading to a self-regulatory system.

6.4.1 Oversubscription via Stolen Memory

The KVM VMM monitor shipped with ONE has been modified in order to instruct the ONE scheduler to overcommit the memory of the physical hosts. The actual amount of memory available in the host that is reported to the ONE monitoring system is the amount of physical memory obtained by the actual monitoring system plus a percentage O from the amount of memory that could be *stolen* from the free memory available in the VMs. The scheduler shipped in ONE is unaware of the memory reduction of the VMs, and calculates the amount of memory available

for virtual machines in one host as the memory available in the host minus the memory requested by the VMs when they were deployed, as shown in (6.2).

$$HostVMs_{mem} = Host_{mem} - \sum_{VM\ in\ host} VM_{mem} \tag{6.2}$$

Using this approach, the ONE scheduler will act as if the hosts had more memory available for the VMs and will try to deploy new VMs in the physical host even if the total amount of memory requested by the VMs is greater than the physical memory available at the destination host.

The value of O can be configured for the on-premises Cloud in order to increase the degree of memory oversubscription. It is a percentage so a value of 0% means that no memory oversubscription will be introduced by CloudVAMP. This means that the sum of allocated memory of all the VMs of a host in the on-premises Cloud will never exceed the available memory of that host. A value of 100% for O means that CloudVAMP will try perform as much oversubscription as possible. This means to reclaim all the free memory from the VMs to enable maximum oversubscription, since the CMP scheduler will allocate additional VMs to the underlying hosts. Notice that this may require to migrate VMs more frequently if applications start demanding additional memory. Notice that under no circumstances CloudVAMP will reclaim used memory from the VMs since that would have a dramatic impact on the performance of applications. In the end, this parameter should be properly fine-tuned depending on the requirements of the on-premises Cloud.

Live Migration in On-premises Clouds

KVM fully supports live migration among physical hosts without any downtime provided that i) the Virtual Machine Image is located on a shared storage among the source and destination physical machines, and ii) both physical machines reside in the same subnet. These assumptions are commonly (and easily) met in an on-premises Cloud deployment.

Migration involves copying the memory pages from source to destination machines. The time involved in the live migration depends on the memory size of the VM but it is much more dependent on the rate at which dirty pages are created, which depends on the application usage of memory. As Clark et al. [82] noted, if the VM continuously dirty pages faster than the rate of copying, then the copy of pages work will be in vain. In particular, we have detected stalled live migrations for VMs executing memory-intensive applications, in which the memory is being frequently modified, thus creating new dirty pages at a faster rate than the ability of KVM to transfer those pages to destination.

This behaviour of live migration affects the policy employed to select the VM that should be live migrated under memory overcommitment scenarios. Notice that the VM whose allocated memory is being increased, as happens in Figure 6.1.c, is expected to later use that memory, thus being a candidate to produce more dirty pages. Therefore, CloudVAMP will try to avoid choosing that VM when considering which VM should be migrated. In particular, CloudVAMP uses the following approach: First, it selects the VM with the least amount of allocated memory, running on the same machine that hosts the VM whose memory is growing. This policy tries to minimise the migration time. Then it selects the destination host, selecting the one with the largest amount of free memory. In case that none of the available hosts has enough free memory to receive the VM, the migration is not performed. Notice that enhanced live migration strategies can be addressed although they lie out of the scope of this paper.

6.5 Assessment via Case Studies

This section assesses the usefulness of the developed system in a standard production environment based on OpenNebula 4.8 that consists of three dual 4 core Xeon E5620@2.40GHz with 16 GB RAM and three quad 4 core Xeon E7520@1.86GHz with 32 GB RAM, for a total of 72 cores and 144 GB RAM. The operating system for the platform is Ubuntu 12.04.5 LTS, and the version of KVM is 1.0. Any piece of software is installed from the official repositories of Ubuntu and OpenNebula, except for the implementations made in this paper. In our tests, the value of O is set to 100% to gain the maximum amount of memory for other VMs, thus fostering maximum oversubscription.

For that, two case studies are executed. The first one integrates this technique in a production elastic virtual cluster of the es-NGI[4] infrastructure, the Spanish National Grid Initiative, to seamlessly accommodate workload of different sizes within a virtual cluster that features both horizontal and vertical elasticity. The second one focuses on the deployment of Docker containers running on a multi-tenant vertical elastic VM to adapt its memory size to the varying workload.

6.5.1 Fully Elastic Virtual Clusters for Grid Infrastructures

The es-NGI infrastructure is the spanish national Grid initiative that contributes computing and storage resources to the European Grid Initiative (EGI[5]) which is a global Grid infrastructure (also supporting federated Clouds) that supports scientific activities. More than 320 organisations across 43 countries offer a com-

[4]es-NGI: `http://www.es-ngi.es`
[5]EGI: European Grid Initiative

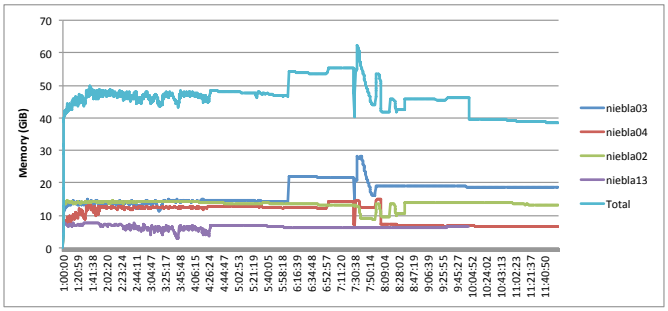


Figure 6.4: Evolution of the stolen memory of the hosts that execute the VMs that support the virtual elastic cluster that executes the jobs from the es-NGI.

puting capacity that exceeds 480.000 cores where more than 1.4M jobs per day are executed[6].

Our research group contributes with computing capacity in the shape of elastic virtual clusters in which VMs are dynamically provisioned to support the execution of incoming jobs in a sandboxed environment created by EC3 (Elastic Cloud Computing Cluster)[7] [26] an open-source tool to create elastic virtual clusters on hybrid Cloud infrastructures. This virtual cluster is based on a front-end node managed by CLUES [4] that monitors the LRMS (Local Resource Management System) and decides when to scale out (provision additional working nodes) and scale in (terminate working nodes) according to a set of configurable rules. However, the nodes of the cluster, which are VMs, are deployed with a fixed amount of memory, regardless of the amount of memory actually consumed by the applications being executed in them. The Workload Management System (WMS) of the Grid infrastructure is responsible for allocating the applications to resources with at least as much free memory as the application requests. However, the running applications typically use less memory than the one actually available in the VMs. That memory could be employed to allocate new VMs for the execution of other jobs thus increasing both the job throughput and the usage of our on-premises Cloud platform.

In this case study we wanted to assess the effectivity of introducing vertical elasticity in the shape of dynamic memory management within this production platform. We introduced CloudVAMP into the platform, and recorded data from a representative workload that arose from different real jobs in a period of 12 hours. The case study involves three physical hosts with 16 GB of RAM (*niebla02*, *niebla03* and *niebla04*) and one physical host with 64 GB of RAM (*niebla13*).

[6] `http://www.egi.eu/infrastructure/operations/egi_in_numbers/`
[7] EC3: `http://www.grycap.upv.es/ec3`

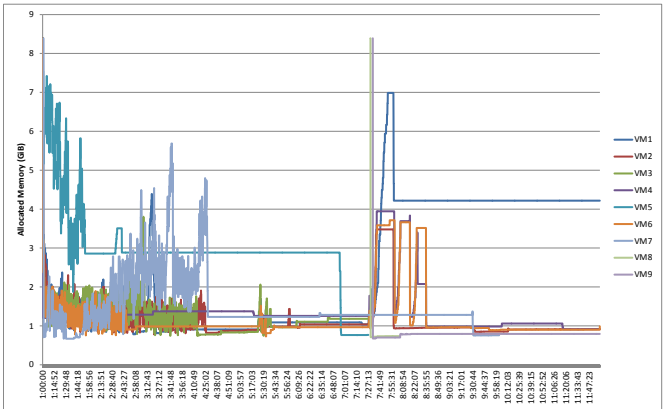


Figure 6.5: Evolution of the allocated memory of some of the VMs that compose the virtual elastic cluster.

Figure 6.4 represents a summary of the evolution of the stolen memory for the physical hosts that hosted the VMs that where part of the elastic virtual cluster deployed to support the execution of jobs coming from the es-NGI. Remember that the stolen memory is the memory that has been freed in a physical host as a result of the usage of CloudVAMP. The figure also depicts the sum of stolen memory across the physical hosts. The larger the amount of stolen memory per host, the higher the chances are that the CMP deploys additional VMs in that host. The figure shows that under real workload scenarios, CloudVAMP is able to free memory from the VMs by adjusting their allocated memory to the real memory requirements of the VM.

Within the same time frame, Figure 6.5 describes the evolution of a representative subset of nine VMs of that virtual cluster. Take into account that the number of nodes of the virtual cluster dynamically changes depending on the number of jobs currently received by our Grid site. Notice that all the VMs are initially deployed with 8 GB and they are almost instantly downsized depending on the actual memory consumption of the application. A VM can host the execution of different simultaneous Grid applications, depending on the number of virtual CPUs, which for this study is set to 4 vCPUs. Subsequent executions of different applications in the same VM is also possible.

The oscillatory memory allocation patterns that can be seen at the beginning of the execution of some VMs, as is the case of VM1 and VM7, can be both due to the highly dynamic memory consumption patterns of a single application or the concurrent execution of different applications and, henceforth, with different memory consumption.

Host	niebla13	niebla02	niebla04	niebla03
Phys. Mem. (GiB)	64	16	16	16
Avg. Stolen Mem. (GiB)	4.97	12.82	9.91	16.38

Table 6.1: Comparison of the physical memory of the hosts vs the stolen memory (the freed memory per node when using CloudVAMP) .

Notice that VM8 and VM9 are deployed at around 7:30 minutes since the study was started. These represents the horizontal elasticity of the virtual cluster in action, where two additional nodes are deployed because new incoming jobs are requested to be executed in the virtual cluster. Then, VM8 is terminated approximately 20 minutes after its deployment. These depends on the horizontal elasticity rules provided by CLUES, where new VMs are dynamically deployed to host the execution of incoming jobs and they are terminated when no longer required.

Table 6.1 compares the physical memory of the hosts with the average stolen memory obtained as a result of the application of CloudVAMP during the 12-hour case study. Notice that in the case of the host *niebla03* the average stolen memory exceeds its physical memory. To understand this, consider a scenario in which two 8 GB VMs are deployed on a physical host with 17 GB of RAM in which, with the help of CloudVAMP are reduced to 1 GB per VM to fit the memory consumption of the applications running in the VMs. An additional 8 GB VM is deployed on the same host and later shrank to 1 GB. These VMs fit in the physical host and you are saving 7 GB per VM, which represents a total save of 21 GB, an amount larger than the physical host's memory. Therefore, CloudVAMP enables memory oversubscription to take place, by allowing the CMP to schedule the deployment of additional VMs in the physical hosts. However, when applications running in the VMs start using more memory, and CloudVAMP increases their allocated memory, the physical host might incur in memory overcommitment. This is why live migration techniques can be used to restore the quality of service delivered by the on-premises Cloud. This is the topic addressed in the next case study.

6.5.2 Addressing Memory Overcommitment via Live Migration

This section introduces an approach to solve memory overcommitment by using live migration techniques available in the KVM hypervisor. For that, we are going to introduce a multi-tenant scenario based on Docker containers.

Docker [89] introduces the ability to package applications and their dependences into lightweight containers tailored for specific distributions which, as opposed to VMs, can be spun up very fast. This technology is of special interest for multi-tenant scenarios in which a set of physical resources has to be shared among different users, by leveraging process isolation and without the overhead introduced

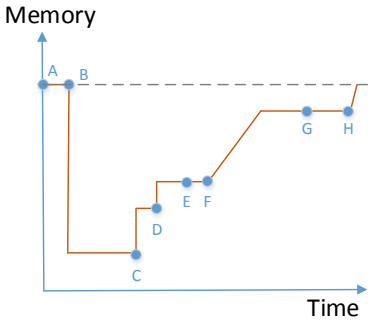


Figure 6.6: Sequence of events that introduce memory overcommitment in a multi-tenant scenario based on Docker containers.

by a hypervisor layer. In fact containerization is one of the underlying technologies among popular open-source PaaS tools such as CloudFoundry[8] and OpenShift[9].

This case study features the deployment of a VM with Docker that supports the deployment of containers to host different applications within the same VM. This approach separates infrastructure provision (from the Cloud) and application deployment (using Docker containers) which introduces significant benefits to deploy applications on multiple back-ends. In multi-tenant scenarios, where a single VM can be used to deploy multiple containers from multiple users, it is expected a larger variation in the memory consumption patterns, when compared to a single application running on a single VM. This is why we believe that CloudVAMP can be beneficial by automatically managing the allocated memory to the VM (or a set of VMs) according to the memory used by its active containers.

Figure 6.6 describes the scenario employed, along with the following events:

A A VM (VM_1) is provisioned with a certain amount of RAM on a specified physical host of the on-premises Cloud.

B CloudVAMP reduces the allocated RAM of VM_1 since the memory consumption of the VM after its boot is very low (no application is being ran yet).

C Docker is installed and the first container is deployed based on an image with the Apache Tomcat application server. This will result in an increase of the allocation of memory to VM_1, as requested by CloudVAMP.

[8]CloudFoundry:`http://www.cloudfoundry.org`
[9]OpenShift:`http://www.openshift.com`

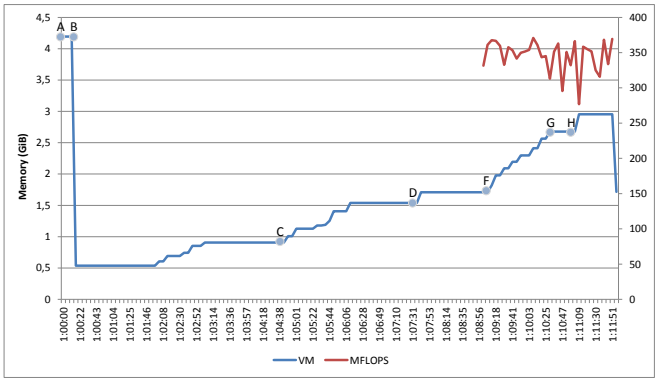


Figure 6.7: Memory consumption of a VM

D A second container is deployed based on the same image. We expect a memory increase, although slightly lower due to the sharing of some pages between the two containers.

E Since VM_1 memory was reduced, there is enough free available memory in the physical machine to host another VM (VM_2), as decided by the OpenNebula scheduler, which will be running in the same physical host.

F A memory-intensive application is executed on a third container which introduces memory pressure for VM_1. We will use a synthetic benchmark application that enables us to control the memory allocation pattern and to obtain the performance of the application (in MFLOPS) as described in [96].

G CloudVAMP will try to increase the memory of VM_1 but since this would result in memory overcommitment, it will use a live migration strategy as a contingency plan. This involves migrating a VM from the physical host to restore the capability of VM memory without overcommitment.

H When enough memory has been freed from the physical host, VM_1 can be allocated more memory. Remember that the VM memory size will not be able to grow beyond the amount of memory specified when initially created the VM.

The aforementioned sequence of events has been carried out in the same on-premises Cloud. Figure 6.7 shows the real memory allocation of the VM that hosted the different Docker containers.

First of all, the VM is deployed with 4 GB of RAM at time instant 1:00:00 (which corresponds to event A in Figure 6.6). A few seconds later, CloudVAMP detects

that the VM has enough free memory and decides to reduce its allocated memory to slightly over than 500 MB (event B). At 1:01:57 we perform the installation of required packages to use Docker, what demands additional memory and results in a periodic increase in the allocated memory to the VM.

At 1:04:44, the first Docker container is deployed (C) what increments the memory requirements for the VM thus resulting in an increase of its allocated memory. The plateau of allocated memory to the VM that can be noticed from 1:06 until 1:07 is due to the steady state of the VM, since once the Docker containers are started, no activity is really performed with those containers, for the sake of clarity in this case study.

At 1:07:36, the second container is deployed (D) what introduces memory pressure in the VM resulting in a periodic increase in the allocated memory, according to the increasing memory requirements for the containers, which host Tomcat, a memory-intensive Java application server. Within this period we have purposely deployed other VMs within the same physical host to introduce memory pressure in the host when the analysed VM starts demanding more memory.

At instant 1:09:01 the memory-intensive application is deployed in the VM to force a steady memory consumption from 0 to 1000 MB in 2 minutes and maintain that memory consumption for other 60 seconds (F). This results in CloudVAMP to periodically increase the allocated memory of the VM at relatively similar memory chunks according to the periodic memory consumption increase of the VM. At instant 1:10:36, there is so much memory pressure in the physical host that no additional memory can be allocated to the VM. Although the application is constantly demanding more memory, CloudVAMP cannot allocate additional memory to the VM because the host is already overcommitted. In this situation, the application might incur in thrashing because it has to rely on swap memory. Since this situation dramatically affects the performance of applications, it is important to alleviate the memory overcommitment in the physical host. This requires live migration techniques to move a VM away from the physical host so that the available free memory can later be allocated to additional VMs running on the physical host.

In this case CloudVAMP was configured to live migrate the VM with the least amount of allocated memory to alleviate as fast as possible the memory overcommitment situation. Remember that the time invested in live migration is typically related to the memory size, though it is much dependent on the applications running inside, in particular the rate at which dirty pages are created.

Therefore, at around instant 1:10:36 (G) a VM other than the one considered in this case study is migrated away from the physical host in a process that lasted less than a minute. This way, at instant 1:11:09 (H) the VM can now be allocated more

memory to comply with the increasing memory requirements of the application. Shortly after, the application is stopped and the case study is finished.

It is important to point out that the usage of CloudVAMP in an on-premises Cloud has enabled to dynamically manage the memory allocated to the VMs and to alleviate the memory pressure that arises due to overcommitment via live migration techniques without any VM downtime.

Notice that Figure 6.7 also shows the MFLOPS that delivers the application, to evaluate the impact of the memory overcommitment scenario and the live migration of the VM on the performance of the application being executed. You can notice a reduction of up to 15% in the MFLOPS delivered by the application which can be attributed mainly to thrashing and secondarily to live migration. However, this reduction is very transient and, for long running applications, might be negligible. In addition, CloudVAMP can be fine tuned in order to try to prevent the applications from thrashing at the expense of wasting additional memory by increasing, for example, the value of MOP or reducing the value of O at the infrastructure level.

As a final remark, notice that certain type of applications that require low latency responses may prefer not to be live migrated to other hosts, which might have an impact (although relatively small, as shown on the case study) on its performance and the level of service expected by the client. If a Cloud provider needs to run applications that are very sensitive to performance, this can be supported in our system by allocating the VMs that run those applications to a subset of hosts that will not be monitored by CloudVAMP. This way, the allocated memory to those VMs will not be reduced and applications will run on the requested resources without being migrated to other hosts.

Also, notice that the goal of CloudVAMP is not to allocate more resources to increase the performance of an application but to reclaim the unused resources (in particular we focus on the memory because hypervisors support their dynamic management) without affecting the performance of the application.It is possible to reduce the allocated memory of a VM that is currently not being used by an application for other VMs to use it. Of course, depending on the memory consumption patterns the application might require the extra memory back and this might introduce a performance penalty. In the end, these techniques can be further customised for a specific on-premises Cloud depending on the workload and application characteristics.

6.6 Conclusion and Future Works

This paper has introduced CloudVAMP, a customisable system to safely enable transient memory overcommitment in on-premise Clouds via vertical elasticity without VM downtime and featuring live migration to solve oversubscription scenarios. By leveraging the memory ballooning techniques and live migration capabilities available in the KVM hypervisor, CloudVAMP integrates with Cloud Management Platforms to dynamically reduce and increase the allocated memory to the VMs so that they fit the memory requirements of the applications running in the VMs.

We have introduced a generic architecture that can be deployed for different CMPs, and we have implemented a fully functional proof-of-concept based on OpenNebula which is currently being used in production at our research center. We have also released CloudVAMP as open-source to the scientific community[10]. The benefits of CloudVAMP have been assessed via a case study that uses horizontal and vertical elastic virtual clusters that run jobs from a production Grid infrastructure and a multi-tenant scenario based on Docker containers. The ability of CloudVAMP to reclaim unused memory from the VMs to enable temporary overcommitment for the CMPs has resulted in increased VM-per-host consolidation ratio with a reduced impact for the running applications. The usage of live migration has been beneficial to restore the level of service in overprovisioning scenarios.

Future works includes adjusting the O percentage on a per-VM level considering the stability of each VM. For example, CloudVAMP could reclaim different percentages of free memory depending on the amount of time in which a VM's memory consumption has remained among a certain range. For VMs with long periods of stable memory consumption it might be safe to assume that the unused memory will not be used, and a greater percentage can be reclaimed by CloudVAMP to be used for additional VMs to be hosted on the same physical node.

We plan to explore memory bursting, where a VM could temporarily allocate more memory than the one initially requested, much in the same way as CPU bursting is available for certain instance types (e.g. $t2.micro$) in Amazon EC2. This can be easily implemented by increasing the VM deployment memory request by a certain percentage, which would depend on the policies of the on-premises Cloud, and letting CloudVAMP to dynamically manage the memory consumption, which could temporarily exceed the amount of memory initially requested.

Finally, we plan to generalise our development to other CMPs (e.g. OpenStack). For that, one can use Ganglia as the memory reporting system and modify the monitoring system of OpenStack to integrate CloudVAMP.

[10] CloudVAMP, available at `https://github.com/grycap/cloudvamp`

Chapter 7

Discussion of the Results

The results of the works of this thesis have been detailed in the papers that have been published in different journals which address the related topics. These papers have been included as the central chapters of this document, as each one of them addresses one part of the key problems in which the main objective has been decomposed. We did not want to restrict our research to a set of theoretical results, since they can be applied to real deployments. This is why we have created open-source developments that can be used by others.

In the next sections we include a description of how the different components can be used in conjunction to manage a data center, a summary of the papers that have been written under the framework of this research, and the products that have been also created. Finally we outline the future directions for this work.

7.1 Putting Things Together: the Multi-Elastic Data Center

The products that we have developed are the building blocks of a *multi-elastic data center*, which is a data center that is elastically managed at different levels. The scenario for the multi-elastic datacenter is that in which an organization (such as a research center) owns a physical cluster that is managed by an on-premises Cloud, using a CMP such as OpenNebula or OpenStack. The end-users are granted EVCs whose VMs meet the requirements of their applications. The physical servers that are idle, are automatically powered off, in order to save energy, and activated again when needed.

Figure 7.1 shows the schema of the multi-elastic data center. In the figure, we can see that two sides are distinguished: (1) the physical infrastructure and (2) the Cloud. On the one side, the physical infrastructure is intended to be managed by the sysadmins, who will install and manage the physical servers. On the other side, the end-users will only interact with the Cloud side. The components described in this document are the green shaded boxes:

- The physical side (lower part of the figure)
  - An instance of CLUES ($CLUES_p$) that interacts with the physical servers. It is connected to the CMP, and intercepts the queries for the creation of new VMs to power on new physical servers, in case that it is needed. If the physical servers are detected to be idle for a while, they are powered off, in order to save energy.
  - An instance of VMCA, that interacts with the CMP in order to detect inefficient distributions of VMs in the physical servers.
  - An instance of CloudVAMP, that interacts with the CMP in order to detect if the applications in the VMs are using all the memory that has been assigned, or it is possible to reduce it. It interacts with $CLUES_p$ to power on servers if the overcommitment makes the servers to thrash.
- The Cloud side (upper part of the figure)
  - Multiple instances of EC3 ($EC3_i$), that are used to create the EVCs. And one instance of the IM, that is needed by EC3 in order to deploy and to configure the VMs.
  - One instance of CLUES ($CLUES_e^i$) for each instance i of EC3, that will be the responsible of managing the elasticity of the EVCs.

The interaction between all the components in the figure is explained in the form of an use-case that starts with part of the physical servers powered off (as a result of the action of $CLUES_p$), because the Cloud platform is partially idle:

1. On the Cloud side (the upper part of the figure), a user needs a EVC and deploys it by using EC3 to run an High Throughput Computing (HTC) application which is composed by multiple loosely-coupled jobs. The user describes the VMs that are needed, along with the maximum number of working nodes for the EVC. EC3 interacts with the IM to deploy the front-end of the EVC in the CMP, and installs and configures CLUES as the elasticity manager ($CLUES_e$).

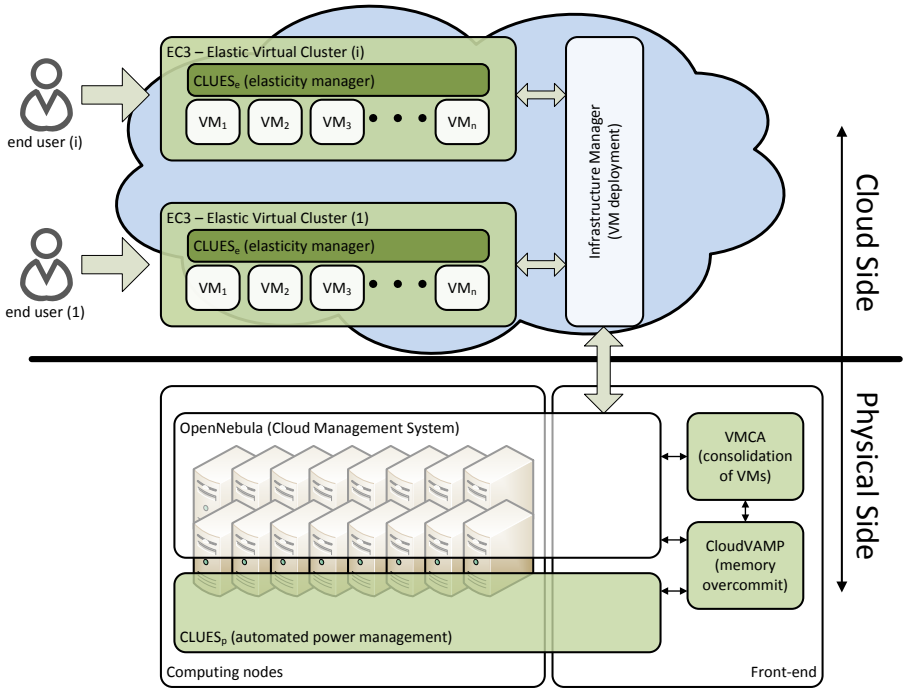


Figure 7.1: The multi-elastic data center.

2. When the user of the EVC queries for the execution of jobs, $CLUES_e$ will create new VMs that will be deployed by the CMP on the physical infrastructure.

3. The instance of CLUES that is installed in the physical side ($CLUES_p$) intercepts the queries for the creation of new VMs and detects that there is not enough room for the new VMs, and then it powers on some physical nodes.

4. Once the physical nodes are on, the new VMs are deployed in the CMP.

5. When the VMs are running and available, $CLUES_e$ detects that the working nodes are on, and they integrated to the EVC. The jobs can be finally started in the working nodes.

6. The user interacts with the EVC, running jobs and creating new working nodes (if needed), as happened in previous steps.

7. After a while, the execution goes on and as jobs finish, the corresponding working nodes become idle. $CLUES_e$ detects these idle nodes and deletes the corresponding VMs by interacting with the IM that, in fact, interacts with the underlying CMP. The other working nodes from the EVC continue with the calculations.

8. VMCA detects that the current distribution of the VMs in the physical servers prevent from powering them off. It calculates a new distribution of VMs that will reduce the number of physical servers that are hosting VMs, and starts moving some VMs to obtain the new distribution of VMs in the servers. The VMs are moved to the other hosts using live-migration, in order to avoid downtimes.

9. $CLUES_p$ detects that the physical servers are getting idle, as their VMs are being live-migrated to other physical servers. After a period of inactivity, $CLUES_p$ powers the idle physical servers off.

10. The calculations that are being made in the cluster do not use all the memory that is included in the VMs that EC3 is creating. $CloudVAMP$ detects that the VMs can be shrinked and dinamically varies the memory using the *memory ballooning* technique available in the KVM hypervisor.

11. VMCA detects that it is possible to pack the VMs into a lower number of physical hosts, and starts migrating VMs to obtain the new distribution of VMs.

12. $CLUES_p$ detects that there are more physical servers idle, as their VMs are being migrated to other physical servers (now the physical memory is being

overcommited, as a result of the action of $CloudVAMP$). After a period of inactivity, $CLUES_p$ powers the idle physical servers off.

13. The calculations start to reclaim the memory that it was assigned to them and the hosts start thrashing. CloudVAMP detects the situation and requests $CLUES_p$ to power on some hosts to be able to migrate the VMs.

14. When the hosts have finally been powered on, CloudVAMP live-migrates some VMs to the empty nodes, in order to make room to grant the amount of memory that the applications need.

The use case would continue, and once the EVC is not needed anymore, EC3 will delete the VMs (via the IM), and $CLUES_p$ will power off the nodes that got idle.

As this use case has shown, elasticity can be introduced in the management of a data center at different levels. On the one side, the more natural way is the EVC, taking profit from the elasticity of the Cloud, that enables the on-demand provision of VMs. In an on-premises Cloud, the elasticity enables to manage the infrastructure in a more efficient way. In this example, other EVCs could have been deployed and the underlying infrastructure had been shared, while each of the user would have it own environment.

On the other side, the physical elasticity enables to save the energy that would have been dedicated to power some idle physical servers. But under usual circumstances the platform would probably drive to distributions of VMs without idle servers. Therefore, such "idleness" can be stimulated, by redistributing the VMs in the platform.

7.2 Summary of the Achievements

The results that have been obtained from the works that have been carried out during the research phase of this thesis are summarized below:

- Enabling the reduction of the energy consumption for physical computing cluster infrastructures. This is achieved by powering off the servers that are in idle state. In the tests shown in the papers, we have reduced the consumption between 17% and 27%, but in newer tests we have achieved near 50% of savings in real deployments. We achieve an elastic behaviour for the physical computing cluster infrastructures by adapting the performance of the system to the actual workload. Moreover, we enable the reduction of the energy consumption and the elastic behaviour not only for classic batch LRMS systems, but for other services such as on-premises Clouds where there are VMs hosted.

- Calculating the TCO of deploying a cluster in a commercial Cloud provider, and comparing it to the TCO of owning a physical cluster. Besides other criteria, we introduced an economical variable to be able to decide if it is better to create the physical facility, or to deploy a virtual cluster in the Cloud.

- Easing the creation of VC in the Cloud and giving them an elastic behaviour, to create EVCs. Such elastic behaviour enables adapting the economic expenses to the actual workload, and the consequence is that it reduces the cost of virtual clusters in the Cloud, as VMs that are not executing jobs are powered off.

- Facilitating the automated power management for the physical infrastructure that supports an on-premises Cloud, by exploiting live-migration technology to consolidate the active VMs into a reduced number of physical servers. In our tests, we reduced the number of physical servers to near the theoretical minimum number in most of the cases: we achieved between the 75% and the 85% of the free hosts in the theoretical optimal solution, and the 100% of the free hosts in the off-line algorithm used to estimate the theoretical optimal value.

- Introducing the ability of overcommiting the physical memory of the servers in an on-premises Cloud, at a CMP level, by dynamically adapting the virtual memory that is allocated by the VMs to the actual needs of the applications. To our knowledge, none of the open-source CMPs manages the memory of the VMs in such a dynamic way, to enable overcommission or aggresive server consolidation. In our tests, we got hosts hosting a set of VMs that had requested more than the 200% of the physical memory.

7.3 Publications

In the framework of this thesis, we have written several papers, with the collaboration of other researchers. These papers describe the solutions proposed to the different problems that have been exposed during the research phase.

- De Alfonso, C., Caballer, M., Hernández, V. (2010) "Efficient power management in high performance computer clusters". In 1st International Multi-Conference on Innovative Developments in ICT (pp. 39-44). `http://dx.doi.org/10.5220/0003036500390044`.

- Alvarruiz, F.; de Alfonso, C.; Caballer, M.; Hernández, V., "An Energy Manager for High Performance Computer Clusters", Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Sympo-

sium on , vol., no., pp.231,238, 10-13 July 2012. http://dx.doi.org/10.1109/ISPA.2012.38

- Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, Vicente Hernández, "An energy management system for cluster infrastructures", Computers & Electrical Engineering, Volume 39, Issue 8, November 2013, Pages 2579-2590, ISSN 0045-7906, http://dx.doi.org/10.1016/j.compeleceng.2013.05.004.

- Carlos de Alfonso, Miguel Caballer, Fernando Alvarruiz, Germán Moltó, "An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud", Future Generation Computer Systems, Volume 29, Issue 3, March 2013, Pages 704-712, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2012.08.014.

- Miguel Caballer, Carlos de Alfonso, Fernando Alvarruiz, Germán Moltó, "EC3: Elastic Cloud Computing Cluster". Journal of Computer and System Sciences, Volume 78, Issue 8, December 2013, Pages 1341-1351, ISSN 0022-0000, http://dx.doi.org/10.1016/j.jcss.2013.06.005.

7.4 Products

Apart from the scientific papers, we have created a set of products that gather the ideas and the implementations used for the validation of the research works. Apart from the implementations that have been made to validate the ideas and algorithms described in the papers, an extra effort has been invested for the creation of final products that can be exported to other deployments. The next sections describe these products, and also their level of maturity.

7.4.1 CLUES

CLUES is an energy management system for HPC Clusters and Cloud infrastructures. The main function of the system is to power internal cluster nodes off when they are not being used, and conversely, to power them on when they are needed. The CLUES system integrates with cluster management middlewares, such as a BQS system or a CMP, by means of different connectors.

CLUES also integrates with the physical infrastructure by means of different plugins, so that nodes can be powered on/off using the techniques that best suit each particular infrastructure (e.g. using wake-on-LAN, IPMI or PDUs).

CLUES is an implementation oriented to the final product, of the work described in chapter 2. We have created a main application that acts as a server, and a set of

command line utilities that enable the interaction with the server. Other features included for CLUES are summarized below:

- Web interface for the administration of the cluster. It is possible to obtain an snapshot of the state of the physical servers, and also to issue power management operations on them.
- A tool for the creation of energy management reports, that makes it possible to obtain a summary about the cluster operation over a period of time. In this report it is possible to identify the usage of the internal nodes, which of them have been more (or less) used, etc., to assist in the decisions with respect to the equipment administration (e.g. purchase of new servers, enhancement of existing server, etc.).
- Establishment of adaptable energy saving policies, that enable to keep powered on an excess of nodes (to handle an eventual peak of workload), powering on sets of nodes in order to better execute a set of jobs, etc.

CLUES has a web page (`http://www.grycap.upv.es/clues`) where it is explained in depth. The web page also includes the product image, some real use cases, etc., and also it enables to download it in the form of source code which is ready to be used. A snapshot of the front web page is shown in figure 7.2. CLUES is licensed under the GPL 3.0 license. At the current point of writing this document, the first version[1] of the CLUES runtime has been downloaded more than 300 times, for a total of more than 1200 downloaded files including the user's manual, plug-ins, etc.

We have recently finalized the development of CLUES 2, which is a major enhancement with respect to the version which is described in chapter 2. It has been fully rewritten, in order to enhance the legibility of the source code, and to adapt it to a object oriented approach. The architecture of the system has also been refurbished, in order to enable new features. This new version has been released in a public repository in GitHub (`https://github.com/grycap/clues`), to match the strategy of the GRyCAP.

From its first release, some contacts from users have been received[2]. We have helped these users deploy CLUES, according to a best-effort policy, and their suggestions have been considered to be included in the new versions of CLUES. The most noticeable contacts are:

[1] CLUES was first released in 2012, as a initial version that includes the preliminary results of the research work. Since then, it has been enhanced according to the feedback of the users and the evolution of the research.

[2] CLUES has a support mail list: clues@upv.es

Figure 7.2: The CLUES web page.

- The center "Fundación de Supercomputación de Castilla y León" (FSCL) that uses CLUES for the day-to-day function of the supercomputer *Caléndula*. This is one of the use-cases that are included in the CLUES web page.

- The Reference Network in Theoretical and Computational Chemistry from Catalonia (XRQTC, from the catalan *Xarxa de Referència de Química Teòrica i Computacional* agreed a seminar about CLUES under the framework of the event "Grid Engine 2012" from the *HPC Knowledge Portal*[3].

CLUES is registered in the CARTA system from the Universitat Politècnica de València (UPV), as part of the technological offer from the UPV. In the framework of the marketing plan of the UPV, some press releases have been published in the mass media. These press releases include some highlights about the features of the product and the use-cases. Moreover, the developers of CLUES appeared on TV (in the UPV-TV channel), to describe the product. The following urls link to the different publications in the mass media:

- "Un sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "El Confidencial / Cotizalia", available online at `http://www.elconfidencial.com/ultima-hora/economia/2013/02/sistema-ahorra-hasta-energia-grandes-conjuntos-20130203-545127.html`

- "Un nuevo sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "Canarias7.es", available online at `http://www.canarias7.es/articulo.cfm?id=291345`

- "Un sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "Las Provincias.es", available online at `http://www.lasprovincias.es/agencias/20130203/economia/sistema-ahorra-hasta-energia-grandes_201302031158.html`

[3] http://www.hpckp.org/index.php/training/grid-engine-training-12

- "Un sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "Invertia", available online at `http://www.invertia.com/noticias/sistema-ahorra-energia-grandes-conjuntos-ordenadores-2813719.htm`
- "Un sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "Finanzas.com", available online at `http://www.finanzas.com/noticias/empresas/20130203/sistema-ahorra-hasta-energia-1705791.html`
- "Un sistema ahorra hasta un 50% de energía en grandes conjuntos de ordenadores" appeared in "La Verdad.es", available online at `http://www.laverdad.es/agencias/20130203/comunidad-valenciana/sistema-ahorra-hasta-energia-grandes_201302031144.html`
- "Noticias Destacadas: CLUES, sistemas de ahorro de energía [2013-02-11]", available at `https://youtu.be/41RlxxrUA3U`
- "Politécnica Tal Cual: Sistema CLUES para el ahorro energético. [2013-02-13]", available at `https://youtu.be/pdMBwrutiPo`

7.4.2 EC3

EC3 is an acronym for Elastic Cloud Computing Cluster, and it is a tool to create EVCs in both commercial Clouds (such as Amazon Web Services) and on-premises (based on OpenNebula or OpenStack). The first release of EC3 was an implementation oriented to the final product of the ideas and the developments from chapter 4. The current version of EC3 has been greatly enhanced due to the effort of other researchers in the GRyCAP[4].

EC3 consists of the combination of the ability of the *Infrastructure Manager* (IM)[5] [27] for the deployment and contextualization of VMs in different Cloud providers, and the elastic management features of CLUES. In order to exploit this combination, EC3 includes a set of *recipes*[6] for the IM, and a plug-in that makes it possible that CLUES uses the IM to implement the elasticity for the cluster (i.e. the creation and destruction of the VMs). According to this approach, EC3 benefits both from CLUES and from the IM. Thanks to the effort of other researchers of the GRyCAP, EC3 has been enhanced to include additional features such as exploitation of spot instances on Amazon EC2, enhancements in the contextualization, etc.

[4] http://www.grycap.upv.es

[5] The IM is a development made by the GRyCAP research group.

[6] The IM is based on "recipes" that describe the desired features for the VMs (e.g. the installed software, the users that should be available, etc.)

EC3 has a webpage (`http://www.grycap.upv.es/ec3`) where its features, the architecture, etc. are explained in depth. But the core of the development and distribution of EC3 has been moved to a public repository in GitHub (`https://github.com/grycap/ec3`). That repository includes both the current developments of EC3, and the documentation about them. EC3 is licensed under the GPL 3.0 license.

7.4.3 VMCA

The Virtual Machine Consolidation Agent (VMCA) is a system that facilitates the automatic power management of servers in an on-premises Cloudbased on OpenNebula and OpenStack. The agent monitorizes the Cloud deployment and analyzes it with the aim of defragmenting the available resources by applying live-migration techniques. The objective is to reduce the number of physical servers needed to host the VMs that are hosted in the platform. The result is an increase in the density of VMs per physical server, thus getting a consolidation of the existing resources. The consequence is that some physical servers get rid from the VMs that they were hosting, and so they are candidates to be powered off by using an automated power management system, such as CLUES.

VMCA is a version oriented to the final product of the ideas and developments of the algorithms described in chapter 5. The implementation consists of an autonomous agent that inspects and analyzes the platform and, if possible, arranges and supervises the live-migrations needed to achieve a distribution of VMs in which some physical servers are free from VMs.

This product has a web page (`http://www.grycap.upv.es/clues/vmca`) that explains VMCA in depth. It is also possible to download the source code of VMCA in this web page, and it is licensed under the GPL 3.0 license. Since its first release, it has been downloaded more than 50 times.

Recently an enhanced version has been created, where the architecture of VMCA and the algorithms included on it have been improved. This new version has been released through a public repository in GitHub (`https://github.com/grycap/vmca`) in order to match the strategy of the GRyCAP. This repository includes information about the product, and also offers VMCA as open source under the GPL 3.0 license.

7.4.4 CloudVAMP

The Cloud Virtual Machine Automatic Memory Packing (CloudVAMP) is an automatic system that enables and manages memory overcommiting in an on-premises Cloudplatform. The current version of CloudVAMP is integrated with OpenNebula, and it makes use of monitoring system and other products that are shipped in the default distribution of that CMP to borrow the memory that is not used in the running VMs, and to make it available for other VMs. The activity of CloudVAMP is motivated by idea that the users usually have VMs with more resources than needed whether because they are able to request a free amount of resources and they do not know what are the actual requirements for the application that they are intended to run, or because there are fixed templates for the VMs and they exceed the requirements of the applications.

CloudVAMP is a version oriented to the final product of the ideas and developments described in chapter 6. The implementation consists of three parts: (1) an agent that is injected in the VMs via the contextualization features of OpenNebula, to monitor the memory usage, (2) a subsystem that re-integrates the borrowed memory to the platform, in order to enable OpenNebula to consider it for scheduling the VMs, and (3) and agent that dynamically adapts the memory granted to the VMs to the actual needs of the applications, and schedules migrations in case that the physical hosts are likely to incur on memory trashing.

This product has been released through a public repository in GitHub (`https://github.com/grycap/cloudvamp`). The repository includes information about the product, and also offers the source code of the different components under the Apache 2.0 license.

7.5 Future Directions

With respect to the future lines for the work made under the framework of this thesis, we have distinguished two areas. On the one side, we have the future works related to the research, and on the other side, we have the future improvements for the products that have been released. The next sections explain the plan for each of these blocks.

7.5.1 Future Research Lines

The current research has a roadmap which is mainly focused on two of the topics: on the one side, the works related to the elastic management of the infrastructures and the automation of the power management of the servers, and on the other side, facilitating the automated power management in on-premises Cloud.

The current work with respect to the scheduling of powering on and off the server are in an early state. The viability of such research has been demonstrated by using basic schedulers that may be enough for most of common use-cases. However, we are planning the exploration of new scheduling policies such as those described below:

- Managing the power of the servers according to energetic criteria, with the aim of powering on those nodes with lesser energetic consumption, and also prioritizing powering off those servers that can help save more energy.
- Managing the power of servers according to physical criteria and heat dissipation. This feature is a result of the feedback of the users, who have asked for the possibility of powering on/off nodes according to where they are physically located in the rack, with the aim of physically distributing the heating surfaces. It is foreseen that such criteria would help manage the cooling machines that counteract the heat of the servers in a more efficient manner.
- In the case of receiving multiple jobs that cannot be executed using the servers that are already powered on, it is possible to introduce new criteria based on the similarity of the features of the servers, in order to concentrate as many jobs as possible in the lesser number of servers. The current policy consists in using a prioritized list, and selecting the first server that will meet the job requests. Other policies are likely to be introduced. As an example, if we get two jobs that request two cores each, we could have different alternatives: (1) powering one single node with four cores to serve both jobs, (2) powering on one single node with more cores than needed, or (3) powering on one node with two cores for each job.
- The current policies for powering on/off nodes are reactive in the sense that they are evaluated when new requests for the execution of jobs (hosting VMs) are received. It would be interesting to introduce proactive policies that try to guess patterns of workload (e.g. time slots, bunch of jobs, user preferences, etc.).

On the side of facilitating power management on on-premises Cloud, we have also opened different working lines. We are confident in the approach introduced in VMCA, from a end-product point of view. But from the point of view of the strict research, it would be interesting to explore other optimization algorithms

(e.g. simulated annealing or genetic algorithms), and try to get optimized versions that are comparable with the current solution. Moreover it would be interesting to bring some techniques that come from the OSs and from the *caché* memory management to the selection of the VMs that are candidates to be migrated from one server to other (e.g. considering different generations of VMs, based on the age of the VM and the times that they have been previously migrated).

Moreover, we have focused the action of VMCA on facilitating the automatic power management of the physical servers, with the aim of reducing the energetic consumption. But it is important to explore additional approaches where all the servers are considered for receiving VMs, even when they are off. Such approach is likely to provide additional energy savings, if some servers are powered on and they host some of the VMs in a more efficient manner. As an example, under an scenario in which four physical servers with one core each are powered on, and each server is hosting one VM that requests one core, it would probably be more efficient to power on a modern server with four cores that will host all the VMs, and power off the other four.

But these techniques should be considered in conjunction with other criteria that take into account the possible fragmentation of the resources. It should be possible to decide under which condition is more interesting to power a server on, even when the new distribution of the VMs would prevent hosting new VMs with special requirements.

Finally, for the case of CloudVAMP, we have focused the activity on oversubscribing the memory of the servers. This is because CloudVAMP relies on the features of the hypervisors and the capacity of the OS of adapting to the variations of the hardware. We are exploring the possibility translating the techniques that we are currently using for the memory to dynamically vary the number of virtual CPUs that the VMs are granted. Currently, it is not well supported. Desipite the hypervisors support memory ballooning and the OSs is able to adapt to the variations on the amount of memory, and adding CPUs at runtime is supported by some hypervisors and OSs (using the CPU-hotplug feature), removing CPUs (i.e. CPU hot-unplug) is not supported by popular hypervisors such as KVM. So we have to wait for the new versions of the hypervisors[7], or to explore different alternatives such as varying other parameters such as the fraction of physical CPU dedicated to the VMs, and try to develop new techniques.

[7] According to http://wiki.qemu.org/Features/CPUHotplug (visited on may, 2015) qemu is working in implementing hot-unplugging interfaces

7.5.2 Future Improvements for the Products

The future lines for the products that are derived from this research have two parts. On the one side, we plan to incorporate the results from the research works to the final products. On the other side, we are planning new features to produce end-products of high quality. For this second case, we have already planned some new enhancements that include the following (but are not limited to them):

- CLUES has to be integrated with some of the most common LRMS such as SLURM or Condor, but also with OpenStack. The current release is integrated with Torque and OpenNebula.
- CLUES will focus on the automation of power management for on-premises Cloud. This feature already existed, but it was prioritized the aspect of energy management on HPC clusters.
- We are going to create a simplified version of EC3 that will enable to deploy elastic virtual clusters in Amazon EC2 with a reduced set of options, as other products such as StarCluster do. The underlying idea is to have a simple version that does not need the IM in order to work.
- We have planned the creation of a new product so-called Cluster as a Service (ClusterAAS) that will enable to create on-demand elastic virtual clusters, that will incorporate CLUES as the elasticity manager. The initial release of this product will be oriented to Amazon EC2, and it is planned that it will provide a limited set of options on the basis of the simplified version of EC3.
- The migration manager of VMCA is going to be enhanced, with the objective of making several migrations in parallel. The main aim is to reduce the time needed to reach to the distribution of VMs that has been calculated.
- VMCA is going to be connected to CLUES, in order to be able to work in a coordinated manner for (1) avoiding a double monitorization of the platform, and (2) for enabling that VMCA requests that a server is powered on, to be able to host some of the VMs on it. VMCA.
- CloudVAMP has to be integrated with other popular open source CMPs such as OpenStack.
- CloudVAMP is going to be connected to CLUES and to VMCA. On one side, the current version of CloudVAMP has not been fully integrated with CLUES, as it is simply able to request to power on some physical hosts but it does not control whether they have been powered on or not. On the other side, CloudVAMP is currently ordering simple oportunistic live-migrations by itself, but it needs to be integrated with VMCA in order to get better

distributions of VMs and to integrate with the existing CMP to take into account the specific requirements of the VMs.

- The parameter O in CloudVAMP controls the fraction of memory that is not being used that can be granted to other running VMs. It is currently a static value, but it can be transformed into a dynamic value that can be calculated depending on the usage of the memory by each of the VMs. This way the VMs that vary the usage of memory frequently will keep the allocation of memory for a longer time. This will make CloudVAMP borrow more memory from those VMs that have a stable low usage of memory.

Chapter 8

Conclusions

The objective proposed for this thesis is the *efficient and elastic management of computational infrastructures* such as computing clusters and on-premises Clouds. During the early stages of the research, we focused on the strictly physical side of the infrastructure: the data centers and the servers that are hosted in them. This is why the main research line starts with CLUES and the work described in chapter 2. But the enhancement of the virtualization techniques and the generalization of the use of the Cloud in the scientific community have made evolve the management of the data centers towards on-premises Clouds. Such evolution has also brought new requirements for the cluster server arrangements both for the physical servers and for the VMs.

So, during in the course of the research, we have broadened the scope of the initial work to address the concepts for the *elastic* and *efficient* management for both physical and virtual computing infrastructures.

The concept of *efficiency* applied to the physical machines has been addressed from a energetic point of view. In this sense, the work that has been carried out during this thesis has tried to achieve energy saving for the physical clusters. Energy saving in large computing facilities is a problem of main concern, and a proof of that is the existence of the list "Green500" [37] whose purpose is to classify the supercomputers in the "Top500"[1][120], according to their energetic efficiency. Our thought is that a server that is powered on is a server that is consuming energy (even if it is a little amount, because it uses energy-efficient components). In our research we have tried to exploit the fact that the minimum energy ever needed for a server that is available to be used, is the residual amount of energy that is needed when the server is powered off but can be remotely powered on.

[1]The Top500 list is a ranking of the most powerful supercomputers in the world.

Therefore, we have focused on a concept that, as seen in the state of the art in chapters 1 and 2 and its updates in the other chapters, only few systems implement. Most of the scheduling mechanisms for jobs (or VMs in the case of the Cloud) that appear in the literature that we have read assume that the servers that are idle will be powered off "automatically" and will be powered on again "when they are needed". Moreover, most the commercial products that we have reviewed do not implement such automation, and only a few of them implement the automatic power management for the "enterprise" versions (e.g. MOAB, vmWare's vSphere 5.5 and Huawei's FusionSphere).

But the servers cannot be powered off as soon as they get to a idle state, because powering on/off them has a high cost in time, and repercussions for the system that manages the cluster. Some other problems arise in the case of deciding which servers have to be powered on, because we have to take into account the requirements of the jobs (or the VMs), the need of powering on servers, and the consequences of keeping off some other servers. Implementing a bad automation of the power management may have a devastating impact on the end-user experience, on the efficiency of the applications and on the quality of service. Such automation is not a trivial task, and it is worth to research on it. In the end, it is a problem of scheduling the power on/off of the servers, working together with the LRMS in the cluster.

In chapter 2 we have described CLUES, that is a system that automates power management in physical cluster infrastructures and on-premises Clouds. By using CLUES, the users have the illusion of having the whole physical infrastructure available, all the time, while it is partially powered off to avoid the waste of energy dedicated to maintain idle servers on. Its architecture is very versatile and can adapt to virtually any cluster-like infrastructure that is managed using a LRMS and whose power can be remotely managed (i.e. using IPMI, Wake-on-LAN, or others). It includes different policies for scheduling the power of the servers, but it is indeed an extensible power scheduling framework that enables to develop particular scheduling policies that adapt to the specific requirements of a computing infrastructure.

The key concept of *elastic management* in the case of physical infrastructures has been resembled to the fact that powering on or off the servers gives the infrastructure a elastic-like behaviour with respect to the workload. In this way, the physical infrastructure is elastic as it adapts its potential performance to the actual workload, while the rest of the infrastructure is powered off. This happens in contrast to the traditionally static behaviour of this kind of facilities, where all the computing capacity is usually available, just for the case that it occasionally arrives a peak of workload. Moreover, as the framework proposed in this thesis is generic, new servers could be aggregated to the infrastructure, and they will be considered to be powered on/off as if they have been always there. Conversely,

the obsolete servers could be removed from the infrastructure, and they will not be considered any more.

In chapters 5 and 6 we have introduced VMCA and CloudVAMP, respectively, that implement two different techniques that facilitate the application of automated power management in on-premises Clouds. In these virtualization platforms, getting idle servers needs to be stimulated, because the lifetime of the running VMs cannot usually be foreseen, and their distribution in the physical hosts may prevent from applying automated power management techniques because the physical servers do not get idle. The underlying ideas described in these chapters consist in reducing the size of the running VMs, and consolidating the running VMs into a few number of physical servers. Using these techniques in conjunction with CLUES enables to create an on-premises Cloud, that is backed by an elastic data center.

In the virtual side, the concept *efficient management* has been addressed from the economical point of view. It is common to find users that see in the Cloud an opportunity to save money, since no upfront investments for the equipment or building are required. But it is important to take into account that using a commercial Cloud for a long term, makes us spend an amount of money that is by no means negligible. In the case of an on-premises Cloud, the upfront investment is needed unless the data center already exists. During the course of this thesis (see chapter 3), we have considered that it is important to calculate the real cost of a cluster in the Cloud, as it is a popular arrangement in the context of the e-Science. Once the cost is known, we can compare it to the cost of having a physical cluster, and to be able to decide whether is more interesting to invest in the physical infrastructure or to deploy a virtual one in the Cloud.

In the end, we can see that using a commercial Cloud is expensive if no cost-control mechanisms are implemented. So, in the framework of this thesis we have focused on containing the economic costs of virtual clusters. In our work, we have started from the concept of the automated power management, and we have extended it to the virtual infrastructures. So, in order to schedule the VMs the build the virtual clusters we have adapted the systems that connect to the physical infrastructure to a virtual context. For the specific case described in chapter 4, we have connected CLUES to the IM, in order to use it as a bridge to the different Cloud providers, and to get customized VMs that are suitable for the VC.

Finally, the objective of an *elastic management* of the virtual clusters has been oriented according to the concept of elasticity in the Cloud. In this way, the virtual clusters are elastic because its size is adapted to the actual workload on them, instead of using a traditional approach by which all the VMs that build the cluster will be wasting money in the commercial Cloud even if they are idle. An important feature for the EVC is that the elasticity is automatic and is self-managed by the cluster. Moreover, the owner of the VC should be able to adapt

the policies for the elasticity to his particular use-case. Our proposal prevents the user to have to take any action to grow or to shrink the virtual infrastructure, and such behaviour is automatically carried out in the runtime, by the elasticity manager.

As a final conclusion, we consider that we have fully covered the objectives for this thesis. But we have also opened new lines to continue the work and to progress in our research. The most immediate work plan has been described in chapter 7. Moreover, as the works carried out during this thesis address problems from the real world, that are faced by system administrators in the data centers and the scientists as individuals, we have accepted the chance of transferring the results to the scientific community, and thus we have created different products that are likely to be used by those users. In this sense, the research described in this document has gone one step forward from the theoretical results, since the products that we have developed have been transferred to the community and used in production environments. The thought of helping to other users using the products developed in the framework of this thesis for their daily work, greatly encourages us to continue working on them, and also to continue the research on this topic.

Bibliography

[1] U.S. Energy Information Administration. *Electricity Explained: Factors Affecting Electricity Prices*. URL: http://www.eia.gov/energyexplained/index.cfm?page=electricity_factors_affecting_prices.

[2] Yasuhiro Ajiro and Atsuhiro Tanaka. "Improving Packing Algorithms for Server Consolidation." In: *Int. CMG Conference*. Computer Measurement Group, Feb. 1, 2008, pp. 399–406.

[3] C. de Alfonso. *Virtual Machine Consolidation Agent (VMCA)*. 2015. URL: https://github.com/grycap/vmca.

[4] Carlos de Alfonso et al. "An energy management system for cluster infrastructures". In: *Computers & Electrical Engineering* 0 (2013), pp. –. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2013.05.004.

[5] Carlos de Alfonso et al. "Infrastructure Deployment Over the Cloud". In: *3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. 2011, pp. 517–521.

[6] F. Alvarruiz et al. "An Energy Manager for High Performance Computer Clusters". In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. 2012, pp. 231–238. DOI: 10.1109/ISPA.2012.38.

[7] Amazon. *Amazon EC2 Pricing*. URL: http://aws.amazon.com/ec2/pricing/.

[8] Amazon. *Auto Scaling Concepts*. 2011. URL: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AS_Concepts.html.

[9] Amazon. *The Economics of the AWS Cloud vs. Owned IT Infrastructure*. Tech. rep. URL: http://aws.amazon.com/economics.

[10] Amazon. *User Guide: Amazon EC2 Cost Comparison Calculator*. URL: http://awsmedia.s3.amazonaws.com/User_Guide_Amazon_EC2_Cost_Comparison_Calculator.pdf (visited on 11–2011).

[11] AnsibleWorks. *Ansible*. URL: http://ansible.cc.

[12] CERN CernVM Software appliance. *Creating Elastic Virtual Clusters*. Mar. 2015. URL: http://cernvm.cern.ch/portal/elasticclusters.

[13] J. Arabas, Z. Michalewicz, and J. Mulawka. "GAVaPS-a genetic algorithm with varying population size". In: *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*. IEEE, pp. 73–78. ISBN: 0-7803-1899-4. DOI: 10.1109/ICEC.1994.350039.

[14] Michael Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UC Berkeley Reliable Adaptive Distributed Systems Laboratory, 2009. URL: https://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

[15] Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. "Evaluating the Cost-benefit of Using Cloud Computing to Extend the Capacity of Clusters". In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*. HPDC '09. Garching, Germany: ACM, 2009, pp. 141–150. ISBN: 978-1-60558-587-1. DOI: 10.1145/1551609.1551635.

[16] Steve Ballmer. *Worldwide Partner Conference 2013 Keynote*. Microsoft. 2013. URL: http://news.microsoft.com/2013/07/08/steve-ballmer-worldwide-partner-conference-2013-keynote/ (visited on 2015).

[17] Salman A. Baset, Long Wang, and Chunqiang Tang. "Towards an understanding of oversubscription in cloud". In: (Apr. 2012), p. 7.

[18] C Belady and A Rawson. *Green grid data center power efficiency metrics: PUE and DCiE*. Tech. rep. The Green Grid, 2008.

[19] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for

Cloud computing". In: *Future Gener. Comput. Syst.* 28.5 (May 2012), pp. 755–768. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.04.017.

[20] Anton Beloglazov and Rajkumar Buyya. "Energy Efficient Resource Management in Virtualized Cloud Data Centers". In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 826–831. ISBN: 978-0-7695-4039-9. DOI: 10.1109/CCGRID.2010.46.

[21] Anton Beloglazov et al. "A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems". In: *Advances in Computers*. Academic Press 82 (2011). Ed. by Marvin V. Zelkowitz, pp. 47–111. ISSN: 0065-2458. DOI: 10.1016/B978-0-12-385512-1.00003-7.

[22] Josep Ll. Berral et al. "Towards energy-aware scheduling in data centers using machine learning". In: *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*. e-Energy '10. Passau, Germany: ACM, 2010, pp. 215–224. ISBN: 978-1-4503-0042-1. DOI: 10.1145/1791314.1791349.

[23] A Bialecki et al. *Hadoop: a framework for running applications on large clusters built of commodity hardware*. Tech. rep. 2005. URL: http://hadoop.apache.org.

[24] D. Borgetto et al. "Energy-aware resource allocation". In: *Grid Computing, 2009 10th IEEE/ACM International Conference on*. 2009, pp. 183–188. DOI: 10.1109/GRID.2009.5353063.

[25] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011. ISBN: 9780470887998.

[26] Miguel Caballer et al. "EC3: Elastic Cloud Computing Cluster". In: *Journal of Computer and System Sciences* 0 (2013), pp. –. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2013.06.005.

[27] Miguel Caballer Fernández. "Gestión de infraestructuras virtuales configuradas dinámicamente". PhD thesis. Universitat Politècnica de València, 2014.

[28] Alberto Caprara and Paolo Toth. "Lower bounds and algorithms for the 2-dimensional vector packing problem". In: *Discrete Applied Mathematics*

111.3 (2001), pp. 231 –262. ISSN: 0166-218X. DOI: http://dx.doi.org/10.1016/S0166-218X(00)00267-5.

[29] Barcelona Supercomputing Center. *MareNostrum III (2013) System Architecture*. Barcelona Supercomputing Center. 2013. URL: http://www.bsc.es/marenostrum-support-services/mn3 (visited on 2015).

[30] CESGA. *Centro de Supercomputación de Galicia - Finisterrae*. Centro de Supercomputación de Galicia (CESGA). 2015. URL: https://www.cesga.es/es/infraestructuras/computacion/finisterrae (visited on 2015).

[31] Jeffrey S. Chase et al. "Dynamic Virtual Clusters in a Grid Site Manager". In: *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. HPDC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 90–. ISBN: 0-7695-1965-2.

[32] C. Chu and R. La. "Variable-Sized Bin Packing: Tight Absolute Worst-Case Performance Ratios for Four Approximation Algorithms". In: *SIAM Journal on Computing* 30.6 (2001), pp. 2069–2083. DOI: 10.1137/S009753979834669X.

[33] T. Cioara et al. "Energy Aware Dynamic Resource Consolidation Algorithm for Virtualized Service Centers Based on Reinforcement Learning". In: *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on*. 2011, pp. 163–169. DOI: 10.1109/ISPDC.2011.32.

[34] Citrix Systems, Inc. *Xen*. URL: http://www.xen.org.

[35] Cluster Resources Inc. *MAUI Cluster Scheduler*. URL: http://www.clusterresources.com/products/maui-cluster-scheduler.php.

[36] *Cluster Resources Inc. Green Computing Powered by Moab*. URL: http://www.clusterresources.com/solutions/green-computing.php.

[37] LLC CompuGreen. *The Green 500*. Web Page. 2015. URL: http://www.green500.org (visited on 2015).

[38] CycleComputing. *CycleCloud*. URL: http://www.cyclecomputing.com/cyclecloud.

[39] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. "Elastic VM for Cloud Resources Provisioning Optimization". In: *Advances in Computing and Communications. First International Conference, ACC 2011, Kochi,*

India, July 22-24, 2011. Proceedings, Part I. Vol. 190. 2011, pp. 431–445. URL: http://www.springerlink.com/index/K75M2705443R2402.pdf.

[40] Carlos De Alfonso, Miguel Caballer, and Vicente Hernández. "Efficient Power Management in High Performance Computer Clusters". In: *Proceedings of the 1st International Multi-Conference on Innovative Developments in ICT*. Proceedings of the International Conference on Green Computing 2010 (ICGreen 2010). Athens, Greece, 2010, pp. 39–44. ISBN: 978-989-8425-15-7.

[41] Carlos De Alfonso et al. "An economic and energy-aware analysis of the viability of outsourcing cluster computing to a cloud". In: *Future Gener. Comput. Syst.* 29.3 (Mar. 2013), pp. 704–712. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.08.014.

[42] Ewa Deelman et al. "The cost of doing science on the cloud: the montage example". In: *2008 ACM/IEEE conference on Supercomputing*. Austin, Texas, 2008.

[43] Peter J. Denning. "Thrashing: Its causes and prevention". In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*. New York, New York, USA: ACM Press, Dec. 1968, p. 915. DOI: 10.1145/1476589.1476705.

[44] F. Doelitzscher et al. "ViteraaS: Virtual Cluster as a Service". In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. 2011, pp. 652–657. DOI: 10.1109/CloudCom.2011.101.

[45] J Dongarra. *LINPACK: users' guide*. SIAM, 1979.

[46] *Energy Price in Spain*. URL: http://www.boe.es/boe/dias/2011/12/31/pdfs/BOE-A-2011-20650.pdf.

[47] Roberto R. Expósito et al. "Performance analysis of HPC applications in the cloud". In: *Future Generation Computer Systems* 29.1 (2013), pp. 218–229. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.06.009.

[48] E. Falkenauer and A. Delchambre. "A genetic algorithm for bin packing and line balancing". In: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. 1992, 1186–1192 vol.2. DOI: 10.1109/ROBOT.1992.220088.

[49] F. Farahnakian, P. Liljeberg, and J. Plosila. "Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers Using Reinforcement Learning". In: *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. 2014, pp. 500–507. DOI: 10.1109/PDP.2014.109.

[50] FCSCL. *Fundación Centro Supercomputación Castilla y León*. Fundación Centro Supercomputación Castilla y León. 2015. URL: http://www.fcsc.es/ (visited on 2015).

[51] 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*. 2012, pp. 26–33. DOI: 10.1109/CloudCom.2012.6427585.

[52] E. Feller, L. Rilling, and C. Morin. "Energy-Aware Ant Colony Based Workload Placement in Clouds". In: *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on*. 2011, pp. 26–33. DOI: 10.1109/Grid.2011.13.

[53] William Forrest, James Kaplan, and Noah Kindler. *Datacenters: How to cut data centre carbon emissions?* McKinsey & Company, 2008, pp. 4–13.

[54] Guilherme Galante and Luis Carlos E. de Bona. "A Survey on Cloud Computing Elasticity". In: *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE, Nov. 2012, pp. 263–270. ISBN: 978-1-4673-4432-6. DOI: 10.1109/UCC.2012.30.

[55] Lakshmi Ganesh. "Data Center Energy Management". PhD thesis. Faculty of the Graduate School of Cornell University, 2012.

[56] S.M. Ghafari et al. "Bee-MMT: A load balancing method for power consumption management in cloud computing". In: *Contemporary Computing (IC3), 2013 Sixth International Conference on*. 2013, pp. 76–80. DOI: 10.1109/IC3.2013.6612165.

[57] Marco Guazzone, Cosimo Anglano, and Massimo Canonico. "Exploiting VM Migration for the Automated Power and Performance Management of Green Cloud Computing Systems". In: *Proceedings of the First International Conference on Energy Efficient Data Centers*. E2DC'12. Madrid, Spain: Springer-Verlag, 2012, pp. 81–92. ISBN: 978-3-642-33644-7. DOI: 10.1007/978-3-642-33645-4_8.

[58] Taliver Heath et al. "Energy conservation in heterogeneous server clusters". In: *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. PPoPP '05. Chicago, IL, USA: ACM, 2005, pp. 186–195. ISBN: 1-59593-080-9.

[59] Michael R. Hines et al. "Applications Know Best: Performance-Driven Memory Overcommit with Ginkgo". In: *2011 IEEE Third International Conference on Cloud Computing Technology and Science*. IEEE, Nov. 2011, pp. 130–137. ISBN: 978-1-4673-0090-2. DOI: 10.1109/CloudCom.2011.27.

[60] Rachel Householder, Scott Arnold, and Robert Green. "On Cloud-based Oversubscription". In: *International Journal of Engineering Trends and Technology* 8.8 (2014), pp. 425–431.

[61] Jinho Hwang et al. "Mortar: Filling the Gaps in Data Center Memory". In: *ACM SIGPLAN Notices* 49.7 (Sept. 2014), pp. 53–64. ISSN: 03621340. DOI: 10.1145/2674025.2576203.

[62] Chris Hyser et al. *Autonomic Virtual Machine Placement in the Data Center*. Tech. rep. HPL-2007-189. HP Laboratories, 2008.

[63] U. Hölzle and B. Weihl. *High-efficiency power supplies for home computers and servers*. Tech. rep. Presented at the Intel Developer Forum, September, 2006. Google, 2006. URL: http://services.google.com/blog_resources/PSU_white_paper.pdf.

[64] IBM. *IBM Platform Dynamic Cluster*. URL: http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/lsf/dynamiccluster.html.

[65] Gideon Juve, Ewa Deelman, and Karan Vahi. "Scientific workflow applications on Amazon EC2". In: *Workshop on Cloud-based Services and Applications (5th IEEE International Conference on e-Science (e-Science 2009)* (Dec. 2010), pp. 59–66. DOI: 10.1109/ESCIW.2009.5408002.

[66] Jangha Kang and Sungsoo Park. "Algorithms for the variable sized bin packing problem". In: *European Journal of Operational Research* 147.2 (2003). Fuzzy Sets in Scheduling and Planning, pp. 365 –372. ISSN: 0377-2217. DOI: 10.1016/S0377-2217(02)00247-3.

[67] G. Keller et al. "An analysis of first fit heuristics for the virtual machine relocation problem". In: *Network and service management (cnsm), 2012*

8th international conference and 2012 workshop on systems virtualiztion management (svm). 2012, pp. 406–413.

[68] A Kivity, Y Kamay, and D Laor. "KVM: the Linux virtual machine monitor". In: *Proceedings of the Linux Symposium* (2007), pp. 225–230. URL: http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf.

[69] Derrick Kondo et al. "Cost-benefit analysis of cloud computing versus desktop grids". In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. 2009.

[70] Jonathan Koomey. *Growth in Data center electricity use 2005 to 2010*. Oakland, CA: Analytic Press, 2011.

[71] Jonathan Koomey et al. *A simple model for determining true total cost of ownership for data centers*. Tech. rep. 2008. URL: http://www.uptimeinstitute.org/wp_pdf/(TUI3011C)SimpleModelDeterminingTrueTCO.pdf.

[72] KVM. *Automatic Ballooning*. URL: http://www.linux-kvm.org/page/Projects/auto-ballooning.

[73] linux KVM.org. *Kernel Based Virtual Machine*. URL: http://www.linux-kvm.org/.

[74] G. von Laszewski et al. "Power-aware scheduling of virtual machines in DVFS-enabled clusters". In: *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*. 2009, pp. 1–10. DOI: 10.1109/CLUSTR.2009.5289182.

[75] Lawrence Livermore National Laboratory. *Simple Linux Utility for Resource Management. Power Saving Guide*. URL: https://computing.llnl.gov/linux/slurm/power_save.html.

[76] Laurent Lefèvre and Anne-Cecile Orgerie. "TOWARDS ENERGY AWARE RESERVATION INFRASTRUCTURE FOR LARGE-SCALE EXPERIMENTAL DISTRIBUTED SYSTEMS". In: *Parallel Processing Letters* 19.03 (2009), pp. 419–433.

[77] Xinhui Li et al. "The method and tool of cost analysis for cloud computing". In: *IEEE International Conference on Cloud Computing, 2009*. 2009, pp. 93–100. DOI: 10.1109/CLOUD.2009.84.

[78] Xueping Li, Zhao Zhaoxia, and Zhang Kaike. "A Genetic Algorithm for the Three-Dimensional Bin Packing Problem with Heterogeneous Bins". In: *Proceedings of the 2014 Industrial and Systems Engineering Research Conference*. Ed. by Y. Guan and H. Liao. 2014.

[79] S.-A. Liang. "Low cost and high efficiency PC power supply design to meet 80 plus requirement". In: *Industrial Technology, 2008. ICIT 2008. IEEE International Conference on*. 2008, pp. 1 –6.

[80] Adam Litke. *Manage resources on overcommitted KVM hosts*. Tech. rep. 2011. URL: http://www.ibm.com/developerworks/library/l-overcommit-kvm-resources/.

[81] Feifei Liu and Xiaoshe Dong. "A Novel Elastic Resource Allocation Strategy of Virtual Cluster". In: *2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*. IEEE, Dec. 2011, pp. 168–173. ISBN: 978-1-4577-1808-3. DOI: 10.1109/PAAP.2011.28.

[82] "Live migration of virtual machines". In: USENIX Association, May 2005, pp. 273–286.

[83] John Mahvi and Avi Zarfaty. *Using TCO to Determine PC Upgrade Cycles*. Tech. rep. Intel, 2009. URL: http://communities.intel.com/docs/DOC-3172/version/1.

[84] P. Marshall, K. Keahey, and T. Freeman. "Elastic Site: Using Clouds to Elastically Extend Site Resources". In: *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. 2010, pp. 43–52. DOI: 10.1109/CCGRID.2010.80.

[85] Paul Marshall et al. "Architecting a Large-Scale Elastic Environment: Recontextualization and Adaptive Cloud Services for Scientific Computing". In: *7th International Conference on Software Paradigm Trends (ICSOFT)*. Rome, Italy, 2012.

[86] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. "Server consolidation in Clouds through gossiping". In: *Proceedings of the 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. WOWMOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–6. ISBN: 978-1-4577-0352-2. DOI: 10.1109/WoWMoM.2011.5986483.

[87] S.S. Masoumzadeh and H. Hlavacs. "Integrating VM selection criteria in distributed dynamic VM consolidation using Fuzzy Q-Learning". In: *Network and Service Management (CNSM), 2013 9th International Conference on*. 2013, pp. 332–338. DOI: 10.1109/CNSM.2013.6727854.

[88] Peter Mell and Tim Grance. *The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Final)*. Tech. rep. 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[89] Dirk Merkel. "Docker: lightweight Linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (Mar. 2014), p. 2. ISSN: 1075-3583.

[90] Microsoft. *Hyper-V*. URL: https://technet.microsoft.com/library/cc753637.

[91] L.E. Miller. "Performance analysis of exponential backoff". In: *IEEE/ACM Transactions on Networking* 13.2 (Apr. 2005), pp. 343–355. ISSN: 1063-6692. DOI: 10.1109/TNET.2005.845533.

[92] Rich Miller. *Who Has the Most Web Servers?* 2009-2013. URL: http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/ (visited on 2015).

[93] MIT. *StarCluster*. URL: http://web.mit.edu/stardev/cluster/.

[94] MIT. *StarCluster Elastic Load Balancer*. URL: http://web.mit.edu/stardev/cluster/docs/0.92rc2/manual/load_balancer.html.

[95] G Moltó et al. "Optimization of Supercontinuum Spectrum Using Genetic Algorithms on Service-Oriented Grids". In: *Proceedings of the 3rd Iberian Grid Infrastructure Conference (IberGrid 2009)*. 2009, pp. 137–147.

[96] Germán Moltó et al. "Elastic Memory Management of Virtualized Infrastructures for Applications with Dynamic Memory Requirements". In: *Proceedings of the International Conference on Computational Science (ICCS 2013)*. Elsevier, 2013, pp. 159–168. DOI: 10.1016/j.procs.2013.05.179.

[97] Ruben S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. "An elasticity model for High Throughput Computing clusters". In: *Journal of Parallel and Distributed Computing* 71.6 (2011). Special Issue on Cloud Computing, pp. 750 –757. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2010.05.005.

[98] Rafael Moreno-Vozmediano and Ignacio M. Llorente. "IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures". English. In: *Computer* 45.12 (Dec. 2012), pp. 65–72. ISSN: 0018-9162. DOI: 10.1109/MC.2012.76. URL: http://www.computer.org/csdl/mags/co/2012/12/mco2012120065.html.

[99] Shuangcheng Niu et al. "Cost-effective Cloud HPC Resource Provisioning by Building Semi-elastic Virtual Clusters". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC '13. Denver, Colorado: ACM, 2013, 56:1–56:12. ISBN: 978-1-4503-2378-9. DOI: 10.1145/2503210.2503236.

[100] L Nussbaum et al. "Linux-based virtualization for HPC clusters". In: *Proceedings of the Linux Symposium*. 2009.

[101] OpenNebula. *Scheduler*. 2015. URL: http://docs.opennebula.org/4.10/administration/references/schg.html.

[102] OpenNebula Project Leads. *OpenNebula*. URL: http://www.opennebula.org.

[103] OpenStack.org. *OpenStack Cloud Software*. URL: http://www.openstack.org.

[104] ORACLE. *Virtual Box*. URL: https://www.virtualbox.org.

[105] Simon Ostermann, Alexandru Iosup, and Nezih Yigitbasi. "A performance analysis of EC2 cloud computing services for scientific computing". In: *Cloud Computing*. Ed. by Geoffrey Avresky, Dimiter R. and Diaz, Michel and Bode, Arndt and Ciciani, Bruno and Dekel, Eliezer and Akan, Ozgur and Bellavista, Paolo and Cao, Jiannong and Dressler, Falko and Ferrari, Domenico and Gerla, Mario and Kobayashi, Hisashi and Palazzo, Sergio and Sa. Springer Berlin Heidelberg, 2010, pp. 115–131. URL: http://www.springerlink.com/index/T640753R2597524U.pdf.

[106] W. Pitt and K. Brill. *Cost Model: Dollars per kW plus Dollars per Square Foot of Computer Floor*. Tech. rep. Uptime Institute, 2008, Whitepaper N° TUI3028A.

[107] Cristina Bianca Pop et al. "A swarm-inspired data center consolidation methodology". In: *Proceedings of the 2nd Int. Conference on Web Intelligence, Mining and Semantics*. WIMS '12. Craiova, Romania: ACM, 2012, 41:1–41:7. ISBN: 978-1-4503-0915-8. DOI: 10.1145/2254129.2254180.

[108] *PRACE Research Infrastructure*. URL: http://www.prace-ri.eu.

[109] The GAMES research project. *The GAMES research project*. 2013. URL: http://www.green-datacenters.eu.

[110] Mustafa M. Rafique et al. "Power management for heterogeneous clusters: An experimental study". In: *Green Computing Conference and Workshops (IGCC), 2011 International*. Los Alamitos, CA, USA: IEEE Computer Society, 2011, pp. 1–8. ISBN: 978-1-4577-1222-7. DOI: 10.1109/IGCC.2011.6008549.

[111] Cluster Resources. *MOAB Adaptative HPC Suite*. URL: http://www.clusterresources.com/products/adaptive-hpc-suite.php.

[112] Ioan Salomie et al. "An energy aware context model for green IT service centers". In: *Proceedings of the 2010 international conference on Service-oriented computing*. ICSOC'10. San Francisco, CA: Springer-Verlag, 2011, pp. 169–180. ISBN: 978-3-642-19393-4.

[113] Tudor-Ioan Salomie et al. "Application level ballooning for efficient server consolidation". In: *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*. New York, New York, USA: ACM Press, Apr. 2013, p. 337. ISBN: 9781450319942. DOI: 10.1145/2465351.2465384.

[114] Parliament Office of Science and Technology. *ICT and CO2 Emissions*. 319. UK: The Parliament Office of Science and Technology, 2008.

[115] Lei Shi, J. Furlong, and Runxin Wang. "Empirical evaluation of vector bin packing algorithms for energy efficient data centers". In: *Computers and Communications (ISCC), 2013 IEEE Symposium on*. 2013, pp. 000009–000015. DOI: 10.1109/ISCC.2013.6754915.

[116] Mark Stillwell et al. "Resource Allocation Using Virtual Clusters". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 260–267. ISBN: 978-0-7695-3622-4. DOI: 10.1109/CCGRID.2009.23.

[117] A. Strunk. "Costs of Virtual Machine Live Migration: A Survey". In: *Services (SERVICES), 2012 IEEE Eighth World Congress on*. 2012, pp. 323–329. DOI: 10.1109/SERVICES.2012.23.

[118] Evangelos Tasoulas, Hå rek Haugerund, and Kyrre Begnum. "Bayllocator: a proactive system to predict server utilization and dynamically allocate memory resources using Bayesian networks and ballooning". In: *Proceedings of the 26th international conference on Large Installation System Administration: strategies, tools, and techniques*. USENIX Association, Dec. 2012, pp. 111–122.

[119] Luis Tomás and Johan Tordsson. "Improving cloud infrastructure utilization through overbooking". In: *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference on - CAC '13*. New York, New York, USA: ACM Press, Aug. 2013, p. 1. ISBN: 9781450321723. DOI: 10.1145/2494621.2494627.

[120] Top500.org. *Top 500, the list*. Web Page. 2015. URL: http://www.top500.org (visited on 2015).

[121] GiorgioLuigi Valentini et al. "An overview of energy efficiency techniques in cluster computing systems". In: *Cluster Computing* 16.1 (2013), pp. 3–15. ISSN: 1386-7857. DOI: 10.1007/s10586-011-0171-x.

[122] Hien Nguyen Van, F.D. Tran, and J.-M. Menaud. "Performance and Power Management for Cloud Infrastructures". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010, pp. 329–336. DOI: 10.1109/CLOUD.2010.25.

[123] Akshat Verma, Puneet Ahuja, and Anindya Neogi. "pMapper: power and migration cost aware application placement in virtualized systems". In: *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '08. Leuven, Belgium: Springer-Verlag New York, Inc., 2008, pp. 243–264. ISBN: 3-540-89855-7.

[124] Akshat Verma et al. "Server workload analysis for power minimization using consolidation". In: *Proceedings of the 2009 conference on USENIX Annual technical conference*. USENIX'09. San Diego, California: USENIX Association, 2009, pp. 28–28.

[125] K Vermeersch. *A Broker for Cost-efficient QoS aware Resource Allocation in EC2*. Tech. rep. University of Antwerp, 2011.

[126] VMware Inc. *VMware*. URL: http://www.vmware.com.

[127] Carl A. Waldspurger. "Memory resource management in VMware ESX server". In: *ACM SIGOPS Operating Systems Review* 36.SI (Dec. 2002), p. 181. ISSN: 01635980. DOI: 10.1145/844128.844146.

[128] Xiaohui Wei et al. "Dynamic Deployment and Management of Elastic Virtual Clusters". In: *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*. IEEE, 2011, pp. 35–41. ISBN: 978-1-4577-0885-5. DOI: 10.1109/ChinaGrid.2011.31.

[129] Dan Williams et al. "Overdriver: handling memory overload in an oversubscribed cloud". In: *ACM SIGPLAN Notices* 46.7 (July 2011), p. 205. ISSN: 03621340. DOI: 10.1145/2007477.1952709.

[130] M. Witkowski et al. "Practical power consumption estimation for real life HPC applications". In: *Future Generation Computer Systems* 29.1 (2013), pp. 208 –217. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.06.003.

[131] M. Woitaszek and H.M. Tufo. "Developing a cloud computing charging model for high-performance computing resources". In: *IEEE 10th International Conference on Computer and Information Technology (CIT)*. 2010, pp. 210–217. DOI: 10.1109/CIT.2010.72.

[132] Yongqiang Wu, Maolin Tang, and W. Fraser. "A simulated annealing algorithm for energy efficient virtual machine placement". In: *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*. 2012, pp. 1245–1250. DOI: 10.1109/ICSMC.2012.6377903.

[133] Gerhard Wäscher, Heike Haußner, and Holger Schumann. "An improved typology of cutting and packing problems". In: *European Journal of Operational Research* 183.3 (2007), pp. 1109 –1130. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2005.12.047.

[134] Minyi Yue. "A simple proof of the inequality FFD (L) less or equal than 11/9 OPT (L) + 1, forany L for the FFD bin-packing algorithm". In: *Acta Mathematicae Applicatae Sinica* 7.4 (1991), pp. 321–331. ISSN: 0168-9673. DOI: 10.1007/BF02009683.

Acronyms

BQS Batch Queueing System. 4, 12, 131

CMP Cloud Management Platform. 4, 7, 13, 125, 126, 128, 130, 131, 136, 139, 140

DPM Dynamic Power Management. 3, 4, 8–10, 14–16

DVFS Dynamic Voltage and Frequency Scaling. 3, 4, 12

EVC Elastic Virtual Cluster. 10, 14, 15, 125, 126, 128–130, 134, 143

HPC High Performance Computing. 4, 10–12, 14, 131, 139

HTC High Throughput Computing. 126

IM Infrastructure Manager. 15, 126, 128, 129, 134, 139

IPMI Intelligent Platform Management Interface. 6

LRMS Local Resource Management System. 4, 11–15, 129, 142

OS Operating System. 3, 138

PG Power Gating. 3, 4

SPM Static Power Management. 3

TCO Total Cost of Ownership. 15, 130

UPS Uninterruptible Power Supply. 3

VC Virtual Cluster. 7, 9, 13–15, 130, 143

VM Virtual Machine. 4–10, 12–16, 125, 126, 128–130, 134–143

VMI Virtual Machine Image. 13, 14