UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

High Performance Scientific Computing over Hybrid Cloud Platforms

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

October 2016

Author: Amanda Calatrava Arroyo

Advisor: Dr. Germán Moltó Martínez

"Croyez ceux qui cherchent la vérité, doutez de ceux qui la trouvent."
"Cree a aquellos que buscan la verdad. Duda de los que la han encontrado"
André Gide (1952).

Abstract

Scientific applications generally require large computational requirements, memory and data management for their execution. Such applications have traditionally used high-performance resources, such as shared memory supercomputers, clusters of PCs with distributed memory, or resources from Grid infrastructures on which the application needs to be adapted to run successfully. In recent years, the advent of virtualization techniques, together with the emergence of Cloud Computing, has caused a major shift in the way these applications are executed. However, the execution management of scientific applications on high performance elastic platforms is not a trivial task.

In this doctoral thesis, Elastic Cloud Computing Cluster (EC3) has been developed. EC3 is an open-source tool able to execute high performance scientific applications by creating self-managed cost-efficient virtual hybrid elastic clusters on top of Infrastructure as a Service (IaaS) Clouds. These self-managed clusters have the capability to adapt the size of the cluster, i.e. the number of nodes, to the workload, thus creating the illusion of a real cluster without requiring an investment beyond the actual usage. They can be fully customized and migrated from one provider to another, in an automatically and transparent process for the users and jobs running in the cluster.

EC3 can also deploy hybrid clusters across on-premises and public Cloud resources, where on-premises resources are supplemented with public Cloud resources to accelerate the execution process. Different instance types and the use of spot instances combined with on-demand resources are also cluster configurations supported by EC3. Moreover, using spot instances, together with checkpointing techniques, the tool can significantly reduce the total cost of executions while introducing automatic fault tolerance. EC3 is conceived to facilitate the use of virtual clusters to users, that might not have an extensive knowledge about these technologies, but they can benefit from them. Thus, the tool offers two different interfaces for its users, a web interface where EC3 is exposed as a service for non-experienced users and a powerful command line interface.

Moreover, this thesis explores the field of light-weight virtualization using containers as an alternative to the traditional virtualization solution based on virtual machines. This study analyzes the suitable scenario for the use of containers and proposes an architecture for the deployment of elastic virtual clusters based on this technology. Finally, to demonstrate the functionality and advantages of the tools developed during this thesis, this document includes several use cases covering different scenarios and fields of knowledge, such as structural analysis of buildings, astrophysics or biodiversity.

Resumen

Las aplicaciones científicas generalmente precisan grandes requisitos de cómputo, memoria y gestión de datos para su ejecución. Este tipo de aplicaciones tradicionalmente ha empleado recursos de altas prestaciones, como supercomputadores de memoria compartida, clústers de PCs de memoria distribuida, o recursos provenientes de infraestructuras Grid, sobre los que se adaptaba la aplicación para que se ejecutara satisfactoriamente. El auge que han tenido las técnicas de virtualización en los últimos años, propiciando la aparición de la computación en la nube (Cloud Computing), ha provocado un importante cambio en la forma de ejecutar este tipo de aplicaciones. Sin embargo, la gestión de la ejecución de aplicaciones científicas sobre plataformas de computación elásticas de altas prestaciones no es una tarea trivial.

En esta tesis doctoral se ha desarrollado Elastic Cloud Computing Cluster (EC3), una herramienta de código abierto capaz de llevar a cabo la ejecución de aplicaciones científicas de altas prestaciones creando para ello clústers virtuales, híbridos y elásticos, autogestionados y eficientes en cuanto a costes, sobre plataformas Cloud de tipo Infraestructura como Servicio (IaaS). Estos clústers autogestionados tienen la capacidad de adaptar su tamaño, es decir, el número de nodos, a la carga de trabajo, creando así la ilusión de un clúster real sin requerir una inversión por encima del uso actual. Además, son completamente configurables y pueden ser migrados de un proveedor a otro de manera automática y transparente a los usuarios y trabajos en ejecución en el cluster.

EC3 también permite desplegar clústers híbridos sobre recursos Cloud públicos y privados, donde los recursos privados son complementados con recursos Cloud públicos para acelerar el proceso de ejecución. Otras configuraciones híbridas, como el empleo de diferentes tipos de instancias y el uso de instancias puntuales combinado con instancias bajo demanda son también soportadas por EC3. Además, el uso de instancias puntuales junto con técnicas de checkpointing permite a EC3 reducir significantemente el coste total de las ejecuciones a la vez que proporciona tolerancia a fallos. EC3 está concebido para facilitar el uso de clústers virtuales a los usuarios, que, aunque no tengan un conocimiento extenso sobre

este tipo de tecnologías, pueden beneficiarse fácilmente de ellas. Por ello, la herramienta ofrece dos interfaces diferentes a sus usuarios, una interfaz web donde se expone EC3 como servicio para usuarios no experimentados y una potente interfaz de línea de comandos.

Además, esta tesis doctoral se adentra en el campo de la virtualización ligera, mediante el uso de contenedores como alternativa a la solución tradicional de virtualización basada en máquinas virtuales. Este estudio analiza el escenario propicio para el uso de contenedores y propone una arquitectura para el despliegue de clusters virtuales elásticos basados en esta tecnología. Finalmente, para demostrar la funcionalidad y ventajas de las herramientas desarrolladas durante esta tesis, esta memoria recoge varios casos de uso que abarcan diferentes escenarios y campos de conocimiento, como estudios estructurales de edificios, astrofísica o biodiversidad.

Resum

Les aplicacions científiques generalment precisen grans requisits de còmput, de memòria i de gestió de dades per a la seua execució. Este tipus d'aplicacions tradicionalment hi ha empleat recursos d'altes prestacions, com supercomputadors de memòria compartida, clústers de PCs de memòria distribuïda, o recursos provinents d'infraestructures Grid, sobre els quals s'adaptava l'aplicació perquè s'executara satisfactòriament. L'auge que han tingut les tècniques de virtualitzaciò en els últims anys, propiciant l'aparició de la computació en el núvol (Cloud Computing), ha provocat un important canvi en la forma d'executar este tipus d'aplicacions. No obstant això, la gestió de l'execució d'aplicacions científiques sobre plataformes de computació elàstiques d'altes prestacions no és una tasca trivial.

En esta tesi doctoral s'ha desenvolupat Elastic Cloud Computing Cluster (EC3), una ferramenta de codi lliure capaç de dur a terme l'execució d'aplicacions científiques d'altes prestacions creant per a això clústers virtuals, híbrids i elàstics, autogestionats i eficients quant a costos, sobre plataformes Cloud de tipus Infraestructura com a Servici (IaaS). Estos clústers autogestionats tenen la capacitat d'adaptar la seua grandària, es dir, el nombre de nodes, a la càrrega de treball, creant així la il·lusió d'un cluster real sense requerir una inversió per damunt de l'ús actual. A més, són completament configurables i poden ser migrats d'un proveïdor a un altre de forma automàtica i transparent als usuaris i treballs en execució en el cluster.

EC3 també permet desplegar clústers híbrids sobre recursos Cloud públics i privats, on els recursos privats són complementats amb recursos Cloud públics per a accelerar el procés d'execució. Altres configuracions híbrides, com l'us de diferents tipus d'instàncies i l'ús d'instàncies puntuals combinat amb instàncies baix demanda són també suportades per EC3. A més, l'ús d'instàncies puntuals junt amb tècniques de checkpointing permet a EC3 reduir significantment el cost total de les execucions al mateix temps que proporciona tolerància a fallades. EC3 està concebut per a facilitar l'ús de clústers virtuals als usuaris, que, encara que no tinguen un coneixement extensiu sobre este tipus de tecnologies, poden beneficiar-

se fàcilment d'elles. Per això, la ferramenta oferix dos interfícies diferents dels seus usuaris, una interfície web on s'exposa EC3 com a servici per a usuaris no experimentats i una potent interfície de línia d'ordres.

A més, esta tesi doctoral s'endinsa en el camp de la virtualització lleugera, per mitjà de l'ús de contenidors com a alternativa a la solució tradicional de virtualització basada en màquines virtuals. Este estudi analitza l'escenari propici per a l'ús de contenidors i proposa una arquitectura per al desplegament de clusters virtuals elàstics basats en esta tecnologia. Finalment, per a demostrar la funcionalitat i avantatges de les ferramentes desenrotllades durant esta tesi, esta memòria arreplega diversos casos d'ús que comprenen diferents escenaris i camps de coneixement, com a estudis estructurals d'edificis, astrofísica o biodiversitat.

Contents

Chapter 1

Introduction

"You don't generate your own electricity. Why generate your own computing?"
Jeff Bezos, CEO, Amazon (2008)

Scientific applications generally require large computational requirements, memory and data management for their execution. Such applications have traditionally used high-performance resources, such as big supercomputers of shared memory or clusters of PCs with distributed memory, on which the application needed to be adapted to run successfully, by using parallel computing techniques. For that, scientists use libraries and parallel computing specifications, like Message Parsing Interface (MPI) [49] or OpenMP [180], to obtain the best performance possible in multiprocessing. However, physical clusters suffer from several drawbacks. One of the main disadvantages of these 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 [58]. In addition, physical clusters cannot be easily enlarged or shrinked according to the dynamic requirements of the organization and they lack the ability to provide a tailored execution environment customized for each application to be executed, specially when incompatible libraries are required by different applications running on the same infrastructure.

Over ten years ago, the concept of Grid computing emerged [76] as a computing mechanism geographically distributed that brought together computing resources and storage from various organizations, by creating Virtual Organizations (VOs). International projects such as Enabling Grids for E-Science (EGEE) and Large Hadron Collider Computing Grid (LCG), helped to establish large computing infrastructures provided by the major universities and international research centers. However, the adaptation of scientific applications to this type of infrastructures

is often a difficult, time consuming and complicated process, more remarkable for scientific personnel without computer skills. Furthermore, the use of Grid infrastructures requires that the application meets the requirements imposed by the providers of the computing resources. In this sense, if an application requires specific external libraries or a particular version of an operating system, it cannot be executed on a Grid infrastructure if none of its resources meet the requirements of the application.

In recent years, the advent in virtualization techniques, together with the major improvements in hypervisor technologies such as Xen [23] and Kernel-based Virtual Machine (KVM) [108], have paved the way for Cloud computing. According to the National Institute of Standards and Technology (NIST) definition of Cloud Computing, *"Cloud computing 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."* [127]. Thus, this paradigm can solve the disadvantages of traditional technologies with fully customizable Virtual Machines (VMs) that decouple the application execution from the underlying hardware and are dynamically provisioned and released on a pay-as-you-go basis (in the case of using public Cloud providers).

Cloud computing opens a new range of possibilities to access computing, storage and network resources. One of its main features is that it allows the dynamic size adaptation of virtual infrastructures. A virtual cluster is a cluster of computational nodes that have been provisioned from a virtualized infrastructure (e.g. a Cloud infrastructure) and which support certain hardware, software and configuration requirements for the execution of one or many applications. Therefore, virtual clusters can automatically be enlarged and shrinked to cope with increases and decreases in the workload, thus adapting the size of the cluster to the workload. Moreover, this paradigm allows each individual Virtual Machine (VM) to adapt its characteristics to the application requirements, especially in terms of Central Processing Unit (CPU) and memory, through vertical elasticity. This ability allows a more efficient use of computing infrastructures, avoiding overprovisioning. Therefore, Cloud technology gives users the capability of creating virtual infrastructures with the required resources and software, facilitating the efficient execution of their applications. For the administrator point of view, it is also a way to consolidate resources, allowing a more efficient management of the datacenter.

To meet the challenges posed by traditional clusters, this thesis aims to develop a tool to automatically manage all the aspects involved in the execution of scientific applications on virtual hybrid elastic clusters deployed over Cloud resources, abstracting the details of cluster deployment, configuration and management. Specifically, the tool will be in charge of the infrastructure management, including contacting with the cloud providers to request the deployment of virtual resources,

as well as the automatic configuration and the elastic management of the virtual infrastructure deployed, data management for jobs that require working with big datasets and the job execution itself, by terms of the Local Resource Management System (LRMS) properly configured with a Network File System (NFS) that facilitates the user obtaining of the job output.

1.1 Motivation

Traditionally, virtualization was not considered a viable option for High Performance Computing (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 VMs, network, storage capacity, etc.) can be dynamically provisioned and released on a pay-as-you-go basis. This is the case of public IaaS Cloud providers such as Amazon Web Services (AWS) [12] or on-premises Cloud infrastructures deployed by Cloud Management Frameworks (CMFs) such as OpenNebula [175], OpenStack [150], Apache CloudStack [17] or Eucalyptus [147].

In the last years, the adoption of on-premises Cloud solutions for datacenters has increased [78], in part due to the maturity of the CMFs, which are becoming reliable enough for production-ready workloads. Moreover, in recent years the trend towards light-weight virtualization paved the way for container technology to considerably evolve, emerging as a new technology that can be a light-weight alternative to traditional VMs. Therefore, the current computational landscape includes, among others, public Clouds and on-premises Clouds that can support containers and VMs. This fact opens a new way of interaction with computing resources for the scientific community that needs to be deeply investigated.

Indeed, there is a great challenge in providing cost-effective and flexible computing capabilities both for datacenters and end users. Therefore, this thesis tries to address these issues with the introduction of virtual hybrid elastic clusters. A virtual cluster deployed in a public Cloud is able to achieve a competitive price-performance ratio, but also brings important benefits for an organization such as reducing the administration costs (both personnel costs and maintenance of equipments), avoiding hiring or buying the physical building to host the infrastructure together with the upfront investments in hardware, cooling systems, etc. A virtual cluster deployed in an on-premises Cloud enables to logically partition the physical cluster on which the on-premises Cloud is deployed. This enables both to deploy multiple virtual clusters fully customized to each application requirements and to maintain data within the boundaries of the organization, when special privacy measures are required. Another advantage is the ability to multiplex the hardware resources by having a larger number of virtual nodes than actual physical nodes.

The hybrid clusters combine the benefits of both approaches. A fully virtual cluster can simultaneously and dynamically harness computational resources provisioned from both the on-premises Cloud and the public Cloud, or from different public Cloud providers, thus being able to shift High Throughput Computing (HTC) based workload from on-premises to a public Cloud and vice-versa. This enables seamless Cloud bursting, where workload peaks are automatically handled by deploying new working nodes on the public Cloud. The hybrid clusters will be elastic which means the ability to dynamically decrease (scale in) and increase (scale out) the computational capacity of the cluster (in terms of working nodes) to the workload without any downtime. The deployed hybrid clusters will be cost-aware. This refers to the economic cost of the provisioned resources from the public Cloud provider, on a pay-as-you-go basis. This cost will be minimized by integrating the usage of spot instances together with checkpointing techniques to minimize the cost while delivering the expected performance for those computational nodes.

Another important aim of this thesis is migration. The developed platform will address migration at different levels. Firstly, it is considered migration as the ability to move a running application from one virtual cluster to another. This enables to decouple the application from the underlying physical infrastructure, which introduces mobility for the application across different infrastructures (even different Cloud providers). Secondly, the migration of virtual clusters across different infrastructures. This introduces the ability to move (part of) the cluster considering the underlying state of the on-premises infrastructures. This will have an impact to cope with the changing workload conditions in a datacenter.

Finally, in recent years, the container technology has emerged as a new solution that can encapsulate the execution environment for applications from the underlying infrastructure in a more light-weight manner than traditional virtualization. A container includes a complete filesystem that contains everything needed to run an application: code, runtime, system tools and system libraries. Containers running on a host share the same operating system kernel and, thus, it allows running multiple isolated processes in a host. The usage of container technologies in the virtual clustering scenario would provide several advantages comparing with VMs, such as better performance for the applications, due to the light-weight virtualization, and low deploying times for new working nodes, since deploying a container only takes a few seconds, like spawning a process. This alternative to traditional virtualization deserves a deep study to better analyze its advantages and drawbacks, and examine possible architecture configurations under virtual clustering scenarios.

Considering that in the future the hybrid deployments will play a fundamental role [78], this thesis will produce technology that will have an impact on the communities that require cluster-based computing, like datacenters and scientific communities, which will be able to easily outsource part of their computing power to Cloud platforms and to take advantage of the light-weight virtualization tech-

nologies to obtain isolation with near native performance. This implies the ability to deploy and access on demand clusters, composed by VMs or containers, specially configured for the applications to be executed. In consequence, this thesis focuses on harnessing resources from hybrid scenarios that include disparate resources from the aforementioned infrastructures in order to create ready-to-use customized virtual elastic clusters that satisfy the dynamic computing requirements of an individual or organization and which minimize the energy consumption of the underlying infrastructure (in the case of on-premises resources) and the economic cost (in the case of public Cloud resources). These virtual elastic clusters will feature automatic elasticity and will be deployed and managed by a tool exposed as a Software as a Service (SaaS) application, online accessible for end users to access cluster computing on demand.

1.2 Summary of the state of art

The central chapters of this document are academic papers that have been published or have been submitted to different conferences and journals. Each of them include a subsection dedicated to revise the state of the art in the particular problem addressed by the paper. Nevertheless, the following paragraphs include a summary of the state of the art in the field of the thesis for the sake of clarity.

Regarding the creation of clusters in the Cloud, some works such as [138], [187] and [113] have analyzed architectures, algorithms and frameworks to deploy HTC clusters over private, public and hybrid Cloud infrastructures. In [138] 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 [187] 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 [113], the authors try to obtain the best resource allocation solution for a set of virtual clusters from different users with different job features. However, none of the previous work deals with the elastic adaptation of the cluster size to the workload submitted by their users.

Several tools have emerged in the literature in recent years, trying to facilitate the deployment of virtual clusters in the Cloud. For example, in [120], [121], 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 Elastic Compute Cloud (EC2) according to different policies. A similar approach is employed in [22], 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. ViteraaS [62] allows creating virtual clusters to manage the execution of user-defined jobs, but the user cannot remotely access the cluster. Another

example is StarCluster [134], a tool that enables the creation of a Sun Grid Engine (SGE) based cluster in the Amazon EC2 service, following a predefined configuration of applications (SGE, OpenMPI, NFS, etc.). It includes a plugin system that enables the user to add new software packages to be installed on the cluster nodes, but the VMs are based on a a set of pre-defined Amazon Machine Image (AMI)s. Moreover, the elasticity in this tool is controlled by a plugin external to the cluster deployed. Finally, Elasticluster [203] can be employed to launch virtual cluster over several public and on-premises cloud platforms, but the elasticity is managed by the user. Instead our proposal is focused on self-managed hybrid clusters, where the elasticity rules are embedded in the cluster and no external entity is required.

All of the above examples only consider the usage of one Cloud infrastructure, where no hybrid scenarios are supported. Concerning the creation of clusters over hybrid Cloud infrastructures, the authors of works such as [138], [139] and [140] have analyzed architectures, algorithms and frameworks to deploy HTC clusters over this type of infrastructures. They analyze the performance of virtual clusters deployed on top of hybrid Clouds obtaining good results that demonstrate the feasibility of these kind of deployments. In [120], [121], [66], 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 [22], 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. An important point in a hybrid environment is the interconnection between nodes that are deployed in different Clouds. The authors of [106] and [182] study the inter-Cloud communication problem across independent Clouds in detail.

Concerning cost-efficiency of virtual clusters, we can find cost-effective mechanisms to provision transient computing capacity (commonly known as Spot Instances in AWS [13]). This type of instances cost a fraction of on-demand instances in exchange for reduced reliability, because these instances can be terminated abruptly due to price and demand fluctuations. We need to benefit from cost-effective advantages offered by the cloud providers in order to reduce the total cost of the executions, but trying to minimize the looses in execution progress. For that, checkpointing enables to periodically save the job progress before the spot instance is terminated by the provider, thus being able to resume from the latest checkpoint. There are works in the literature that have elaborated this idea. For example, the goal of Khatua et al. [107] is to reduce the total cost in virtual resources using spot instances. They need to migrate the applications to other spot instances before the actual instance is terminated. For that, they propose an scheme based on the usage of Amazon Elastic Block Store (EBS) to save the tasks. In [198] the authors propose a similar strategy based on checkpointing and migra-

tion in order to save money and get good results of task completion time using spot instances. But in this case, they use Amazon Simple Storage Service (S3) to store the application state. Also, the migration strategy consists on changing the instance type (if the bid of the user is high enough) in order to reduce the waiting time until the spot price for the previously selected instance type is achieved.

Regarding migration, we can differentiate three levels that are applicable to a virtual cluster. First, application migration, that usually involves checkpointing, a mechanism by which an application stores its state periodically to the disk. By checkpointing a program's state at regular intervals, the amount of lost computation is limited to the interval from the last checkpoint to the time of crash. There are several functional tools and libraries of checkpointing in the literature, such as CryoPID [55], Distributed MultiThreaded CheckPointing (DMTCP) [16], ComPiler for Portable Checkpointing (CPPC) [163] or OpenVZ [153]. Berkeley Lab Checkpoint/Restart (BLCR) [65], a hybrid kernel/user implementation that provides a robust, production quality implementation that checkpoints applications without requiring changes in the their code, is currently one of the most used tools of checkpointing. Second, there is live migration of VMs [48], which consists on moving a running VM from one physical host to another. Two models can be found at this level, i) intra-cluster migration, exemplified by works such as [185], [117], and [194], that assume shared storage for migration operations and contemplate consolidation operations at both the OS level and application level inside a single cluster; and ii) inter-cluster migration, where works like [181], [189] and [162] discuss live migration of VMs across networks. Third, there is virtual cluster migration, where works like the framework Virtual Cluster (VC)-Migration [196] try to control the live migration of virtual clusters, applying well-known solutions to migrate a single VM. The results of the experiments performed with this framework show the main limitation of migrating a virtual cluster, i.e., the large amount of data that needs to be transmitted together over a limited network bandwidth.

The recent trend towards lightweight virtualization has paved the way for container technology to considerably evolve. Containers are not a new idea, but their usage is gaining prominence in the last years and it seems that they will change the way applications are built, shipped, deployed, and instantiated [126]. Linux containers enable to run multiple isolated processes in one host without the overhead caused by the hypervisor layer introduced by VMs in CPU, memory and storage. Tools such as Docker [60] introduce an approach based on containers to pack, ship and run an application either on VMs, bare metal or Clouds. There are works in the literature that experiment with containers. In [73], the authors explore the performance of traditional VM deployments and contrast them with the use of Linux containers (using Docker). Several benchmarks are used to demonstrate that containers result in equal or better performance than VMs in terms of CPU, memory and storage. Other works in the literature have also analyzed the performance of

containers to execute scientific applications and workflows, such as [27] and [202]. Skyport [80] utilizes Docker Linux containers to execute scientific workflows instead of the usage of VMs, reducing the overhead caused by VM virtualization. Also, analysis of the requirements of the applications to be executed in containers have been performed [170]. However, authors of [192] point out that containers are not good enough for isolation, the only resource that could be successfully isolated in their experiments was CPU, promoting container-based virtualization as a powerful technology for HPC environments.

In conclusion, as far as the author is concerned, there is no work in the literature that describes a tool that integrates all the developments needed to deploy virtual hybrid elastic clusters combined with the use of spot instances and checkpointing techniques, and featuring migration capabilities, offered thought a user-friendly interface. Moreover there are no tools that to deploy elastic clusters based on containers. The development of such tools aggregates the main objectives of this thesis, discussed in the next subsection.

1.3 Objectives of the thesis

The main hypothesis of the thesis is that the usage of virtual hybrid elastic clusters, that can span and be migrated across infrastructures (on-premises Clouds and public Clouds), introduces significant benefits for the cost-effective and flexible management of datacenters, and they are a solution for end users that require customized clusters with dynamic computing requirements for the efficient execution of applications. Therefore, the main goal of the thesis is the execution management of scientific applications over self-managed virtual hybrid elastic clusters on multi-cloud environments. To reach this objective, the general sub-objectives of this thesis, represented in section 1.1, are the automatic elastic management of virtual infrastructures, the management of resources provided by several Cloud providers to deploy and configure virtual hybrid elastic clusters, the migration of those virtual clusters and its workloads, achieving cost efficiency and, finally, the definition and execution of application use cases to showcase the benefits of the approach. In the next paragraphs, a deep description of each one of the objectives is presented.

O1. Elastic management of virtual infrastructures. This objective aims at creating elastic virtual clusters out of computational resources provisioned from IaaS Clouds. These clusters will be 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 dynamically and automatically scale in, according to some simple 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. To cope with this objective, the

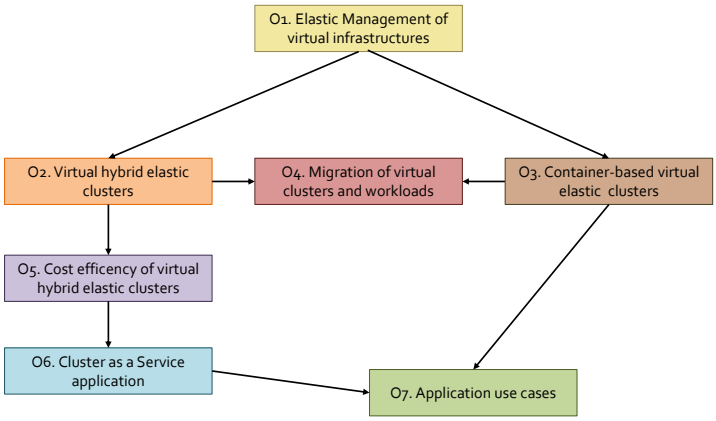


Figure 1.1: Goals of the thesis and their relation.

usage of CLUster Energy management System (CLUES) [9], an energy management system for HPC Clusters and Cloud infrastructures, is proposed.

O2. Virtual hybrid elastic clusters. This objective aims at developing a software platform to deploy and manage customized hybrid elastic virtual clusters, with resources provisioned simultaneously from an on-premises Cloud and from a public Cloud, thus introducing Cloud bursting capabilities. Notice that two extreme cases are also considered, which are deploying a virtual cluster fully on a public Cloud or fully on an on-premises Cloud, to include the requirements for research communities that might require instant access to a cluster of PCs and organizations that require that data or computation does not leave the boundaries of the organization. The users will be offered a software tool approach in order to specify the hardware, software and configuration requirements of the cluster as well as the initial allocation of nodes for each different Cloud infrastructure.

O3. Container-based virtual elastic clusters. The third objective of this thesis aims at developing software tools to deploy virtual elastic clusters based on containers. Like the first objective, these clusters will be self-managed entities that automatically scale out to a larger number of nodes on demand when the workload of the cluster requires it, and scale in when resources are idle for a period of time. However, in this objective, we will consider the usage of containers instead of VMs as the key technology to support the deployment of clusters. As we mention previously, containers are a light-weight solution to isolate the execution of tasks inside a customised environment, where the underlying infrastructure is shared by several tasks with different environment configurations. Their usage introduce several advantages in contrast with traditional VMs, such as less overhead in CPU

and Random Access Memory (RAM) memory and reduced deployment times. These benefits will be evaluated in this thesis.

O4. Migration of virtual clusters and workloads. This objective aims at developing the methodology and software tools required to migrate (partly or fully) virtual clusters and their workloads across different infrastructures. Migration introduces an unprecedented degree of flexibility for a datacenter administrator, with the ability to decouple the execution from the underlying hardware. For example, this enables a system administrator to temporarily outsource cluster-based workloads to a public Cloud (or to another datacenter running an on-premises Cloud) to deal with a planned outage. As another example, a scheduled downtime might affect a long-running scientific application. By migrating the application or even the complete virtual cluster of VMs where the application is being executed to another Cloud infrastructure, the application can resume its execution. This objective includes assessing different migration approaches that range from i) application migration, which requires application checkpointing techniques either through third-party libraries, ii) VM migration, where (a set of) VMs are live migrated to another hardware nodes (either inter-cluster or intra-cluster) with the help of the underlying hypervisor; and iii) cluster migration, which involves transferring the computing nodes (and possibly the front-end) from one source Cloud infrastructure to a destination Cloud infrastructure, even across different Cloud providers.

O5. Cost efficiency of virtual hybrid elastic clusters. One of the objectives of this thesis is to reduce the cost of deployment of such virtual hybrid elastic clusters. For that purpose, we propose the usage of spot instances, a cloud pricing scheme available in Amazon EC2, where the users decide the maximum price they are willing to pay for the instance, thus offering up to 86% savings with respect to on-demand instances [13]. The user bids on spare Amazon EC2 instances and run them whenever the bid exceeds the current spot price, which varies in real-time based on supply and demand. This variation causes the instance to terminate if the spot price is higher than the user bid, thus causing the interruption of the job executions. Therefore, this type of instances offers lower cost at the expense of a reduced reliability. To cope with this problem, we have used checkpointing techniques. Checkpointing enables to periodically save the job progress before the spot instance is terminated by the provider, thus being able to resume from the latest checkpoint. Creating a tool that combines the usage of both spot instances together with checkpointing techniques will produce cost-efficient virtual clusters that offer a fault-tolerant environment for the execution of scientific applications.

O6. Cluster as a Service application. To ease the adoption of these virtual clusters deployed over Cloud Computing platforms, the results obtained in this thesis will be exploited by building a SaaS application that provides Cluster as a Service for a community of users. The developments will be encapsulated as a web application available for users to deploy and manage hybrid clusters with a

specific configuration and user requirements on top of public IaaS Cloud providers, such as AWS, on-premises cloud providers such as OpenNebula or OpenStack and community cloud platforms like European Grid Infrastructure (EGI) FedCloud. For that, the clusters will be deployed on behalf of the users with their specified credentials. The users will not be required to introduce payment information in our platform. Instead, they can use their own credentials to access Cloud providers. In case of using a public cloud provider, the cost of the running virtual clusters will be assumed by the users, which will pay for the resources consumed by the running virtual clusters. Users can also deploy their clusters over on-premises cloud providers that can be available inside their organization. Our platform will offer the services to automate and simplify the deployment, configuration and management of the clusters.

O7. Application use cases. A key aim of the technological advances from the previous objectives is to improve scientific research on disciplines that deal with variable workloads of computing. Therefore, a main objective of the thesis is to validate and demonstrate such technological advances on real scientific cases, to evaluate the benefits and impact of the implemented solution. Three complementary use cases have been defined with different requirements, application areas and computational loads. The first use case involves a parallel computationally intensive gyro kinetic plasma turbulence scientific application. The second use case is devoted to the dynamic analysis of building structures, which requires dealing with a variable load of multiparametric executions. The third use case focuses on biodiversity, where the Galaxy platform [81] will be configured in a virtual elastic cluster to facilitate the execution of jobs in charge of investigate for patterns of biodiversity, from molecular data sets of freshwater diatoms.

This PhD thesis focuses on providing advanced computational tools for the deployment and execution of scientific applications over hybrid Clouds. Moreover, these tools will be open to the community as applications that will abstract away the details of cluster deployment, configuration and management over hybrid Clouds.

1.4 Structure of the document

In order to cover all the objectives proposed, this document has been structured as follows. The chapters 2, 3 and 4 correspond to a group of papers related to the deployment of virtual hybrid elastic clusters, that have been already published on different journals and conferences in the area. The chapters 5, 6 and 7 are structured as papers, since they are in the process of submission or under review in different journals in the area. Because of the structure of the above chapters, all of them include introduction and state of art sections. Thus, the chapters can be read independently, helping the readability of the document. Therefore, it is to be expected some overlap in the related work sections. The remaining chapters

correspond mostly with the dissemination of the results obtained during the thesis. All these chapters collect the work made to address all the objectives proposed in this thesis. The following paragraphs describe each chapter in detail.

Chapter 2, "Towards Migratable Elastic Virtual Clusters on Hybrid Clouds", deeply analyzes the context of the PhD thesis, aligned with the research project "Migrable Elastic Virtual Clusters on Hybrid Cloud Infrastructures (CLUVIEM)", financed by the Spanish Ministry of Economy and Competitiveness. This chapter corresponds to the paper [36], that has been published in the conference "IEEE 8th International Conference on Cloud Computing", which is classified into the CORE B category of the CORE ranking [52].

Chapter 3, "Virtual Hybrid Elastic Clusters in the Cloud", refers to the paper [37], published in the "8th Iberian Grid Infrastructure Conference (IberGrid 2014)". This work introduces the early developments to produce virtual hybrid elastic clusters. Those clusters can simultaneously harness on-premises and public Cloud resources. They can be fully customized, dynamically and automatically enlarged and shrinked (in terms of the number of nodes) and they offer migration capabilities to outsource workload from one datacenter to another (or to a public Cloud) in different situations. This is exemplified with case studies that involve a parallel computationally intensive gyro kinetic plasma turbulence code running on such hybrid clusters with resources provisioned from an on-premises OpenNebula Cloud and the AWS public Cloud.

Chapter 4 refers to the paper "Self-managed Cost-efficient Virtual Elastic Clusters on Hybrid Cloud Infrastructures", published in the journal "Future Generation Computer Systems", which has an impact factor of 2.786 and it is classified in the first quartile (Q1) of the Journal Citation Report (JCR) for the topic "Computer Science, Theory & Methods" in 2016 (when the paper was published). This paper describes the developments to produce EC3, a tool that creates self-managed cost-efficient virtual hybrid elastic clusters on top of IaaS Clouds. It can be considered as an evolution of the ideas presented in the previous chapter, together with new work in the area of cost-efficiency. Using spot instances, together with checkpointing techniques, EC3 can significantly reduce the total cost of executions while introducing automatic fault tolerance. A case study is presented to assess the effectiveness of the tool featuring the structural dynamic analysis of buildings. In addition, checkpointing algorithms are evaluated in a real Cloud environment with existing workloads to study their effectiveness.

Chapter 5 presents a case study of the EC3 web interface. This chapter includes the contents of the paper "eScience with a Galaxy web-service connected to an Elastic Cluster in the Cloud for Computational Biodiversity" submitted to the "Journal of Grid Computing", which has an impact factor of 1.507 and it is classified in the first quartile (Q1) of the JCR for the topic "Computer Science, Theory & Methods" and in the second quartile (Q2) for the topic "Computer Science, Information

Systems" in 2016 (when the paper was submitted). In this paper, a case study of the deployment of the Galaxy web-service inside a virtual elastic cluster launched with EC3 is described.

Chapter 6 addresses migration, developing the methodology and software tools required to fully migrate virtual clusters and their workloads across different Cloud infrastructures. It presents a deep analysis of the state of art in this field, and proposes an architecture to migrate virtual clusters, based on EC3. A proof-of-concept is presented at the end of the chapter.

Chapter 7 introduces the concept of containers and presents EC4Docker, a tool to deploy container-based virtual elastic clusters, based on the usage of CLUES to manage the elasticity, and Docker Swarm as the resource scheduler. A case study evaluates the performance of container-based virtual elastic clusters and compare them to traditional virtual clusters deployed over Cloud platforms with EC3. It is structured as a paper, "Container-based Virtual Elastic Clusters", that was submitted to the "Journal of Systems and Software". This journal has a impact factor of 1.424 and it is classified in the second quartile (Q2) of the JCR for the topic "Computer Science, Theory & Methods" (when the paper was submitted). The work is currently under review.

Chapter 8 presents EC3aaS, a SaaS application offered to the community to facilitate the usage of EC3 to non-experienced users. A brief tutorial of how to deploy and then destroy a virtual elastic cluster by using the web service is shown. Also, other available materials such as videotutorials or the source code, are pointed out in this chapter.

Chapter 9 presents the discussion of the results to analyze the relevance of the works presented in the previous chapters. It also includes dissemination and exploitation of the results, listing all the contributions that the work presented in this document has produced. Moreover, research projects that are using the development of this PhD thesis are cited.

Finally, chapter 10 exposes conclusions and future work of this PhD thesis. Also the fundings that have permitted the development of this thesis are enumerated.

Chapter 2

Towards Migratable Virtual Elastic Clusters on Hybrid Clouds

Published as

A. Calatrava, G. Moltó, E. Romero, M. Caballer, and C. De Alfonso, "Towards Migratable Elastic Virtual Clusters on Hybrid Clouds," in IEEE 8th International Conference on Cloud Computing (2015), pp.1013-1016, June 27 2015-July 2 2015. ISSN: 2159-6182 `http://dx.doi.org/10.1109/CLOUD.2015.139`

Abstract

This paper describes the research work in the context of the CLUVIEM project towards achieving migratable, self-managed virtual elastic clusters on hybrid Cloud infrastructures. These virtual clusters can span across on-premises and public Cloud infrastructures thus leveraging hybrid Cloud platforms. They are elastic since working nodes are automatically provisioned and relinquished to dynamically adapt the capacity of the virtual cluster (in terms of number of nodes) according to the current workload. They are self-managed since the elasticity rules are managed via the head node without requiring any external software entity for monitoring and deciding when to scale in and out. Finally, they are migratable since they consider both application migration, via application checkpointing, and infrastructure migration, by cloning infrastructures across multi-Clouds. These features introduce unprecedented flexibility for cost-effective cluster-based computing with minimal impact for cluster users. The paper summarises the current state of developments and future roads to achieve this vision.

2.1 Introduction

Clusters are one of the most widely used computing facilities across the world. They can be used for HPC, where tightly-coupled tasks require intensive communication, and for HTC, where loosely-coupled tasks are typically executed as a Bag of Tasks (BoT) or a parameter sweep application. However, physical clusters suffer from several drawbacks which include, but are not limited to, an initial large capital investment, electricity costs for operation and refrigeration and the inability to cost-effectively enlarge and decrease the number of nodes according to the workload.

With the introduction of virtualization and the advent of Cloud Computing, the idea of deploying virtual clusters on computational resources provisioned from Cloud infrastructures took shape in the form of tools such as StarCluster [134] or Elasticluster [203]. StarCluster enables to provision a virtual cluster on top of AWS. It also supports to automatically scale out the cluster (and scale in) considering the number of jobs queued up at the LRMS. However, since this tool can only provision clusters from AWS, no virtual clusters can be deployed on on-premises Cloud platforms created with Cloud Management Platforms (CMPs) such as OpenNebula or OpenStack. In addition, the scaling capabilities of the virtual cluster require a client-side monitoring application that is always running and periodically polls the cluster. Therefore, the cluster is not self-managed and requires the StarCluster application running on the client side. In contrast, Elasticluster can be employed to create virtual clusters on several Cloud providers (Amazon EC2 and Google Compute Engine) as well as on-premises Cloud platforms (OpenStack supported). The clusters support elasticity but, unfortunately, the user decides when to scale the cluster by using the appropriate command. Therefore no automated elasticity is supported.

Other tools to deploy virtual clusters can be found in the literature, such as Wrangler [104] or the work by Niu et al. [146]. The former does not support elasticity while the latter, although it does include elasticity rules to scale the clusters, it does not consider support for *spot instances*, which is a cost-effective mechanism to provision computational resources for interruptible tasks, supported by Amazon EC2. In addition, none of the aforementioned tools support hybrid virtual clusters, where resources can span several Clouds (either on-premises or public).

In this paper we build on the state of the art and describe the goals, the road map and the milestones achieved so far in the CLUVIEM project. The project, funded by the Spanish government, aims at developing software (accessible via SaaS and Command-Line Interface (CLI)) to create migratable self-managed cost-effective virtual elastic clusters on hybrid Cloud infrastructures which, for the sake of brevity, will be named enhanced virtual clusters. After the introduction, the remainder of the paper is structured as follows. First, section 2.2 introduces the

main architecture of the platform to be developed featuring capabilities such as automated elasticity, hybrid scenarios and migration. Next, section 2.3 addresses different scenarios in which these virtual clusters introduce significant benefits. Finally, section 2.4 summarizes the paper and points to future work.

2.2 Enhanced Virtual Clusters

Virtual clusters on the Cloud are composed of virtual machines which are provisioned from different Cloud providers. In the case of public Cloud providers such as AWS, a pay-as-you-go cost model is employed with no upfront investments. In the case of on-premises Clouds the cost of the provisioned resources is typically measured in terms of energy consumption. The greatest advantage of these virtual clusters is that they can naturally leverage the underlying elasticity of the Cloud platforms. You can start with a single head node (a.k.a. front-end node) that provides the users with the illusion of a fully active cluster and when they start submitting jobs to the LRMS, these are transparently intercepted to provision the required working nodes, using different customizable provisioning approaches, and are configured and automatically integrated in the LRMS before releasing the jobs to be executed. Therefore, users just notice a small delay until the jobs actually start their execution. The worker nodes are automatically relinquished whenever they are no longer used (or expected to be used according to a set of policies). This introduces a cost-effective approach for cluster-based computing where computational resources are provisioned and released as required.

Figure 2.1 shows the underlying software components[1] employed to create the platform to deploy these enhanced clusters: Virtual Machine image Repository & Catalog (VMRC), a catalog of Virtual Machine Image (VMI)s available in different Cloud platforms (e.g. AMIs in AWS and images in an OpenNebula repository), supporting matchmaking capabilities according to metadata; CLUES, an elasticity management system for clusters and Cloud infrastructures; Virtual Machine Consolidation Agent (VMCA), a tool to consolidate VMs featuring migration across physical nodes; EC3, a platform to create elastic virtual clusters on multi-Clouds; Infrastructure Manager (IM), a platform to provision and configure virtual infrastructure from different Cloud providers; and, finally, CloudVAMP, a memory overcommitment manager for on-premises Clouds, used to control vertical elasticity.

The following subsections outline the features of these enhanced virtual clusters and provide some additional details regarding the technologies and tools employed to achieve them.

[1] These open source components are available at `https://github.com/grycap`

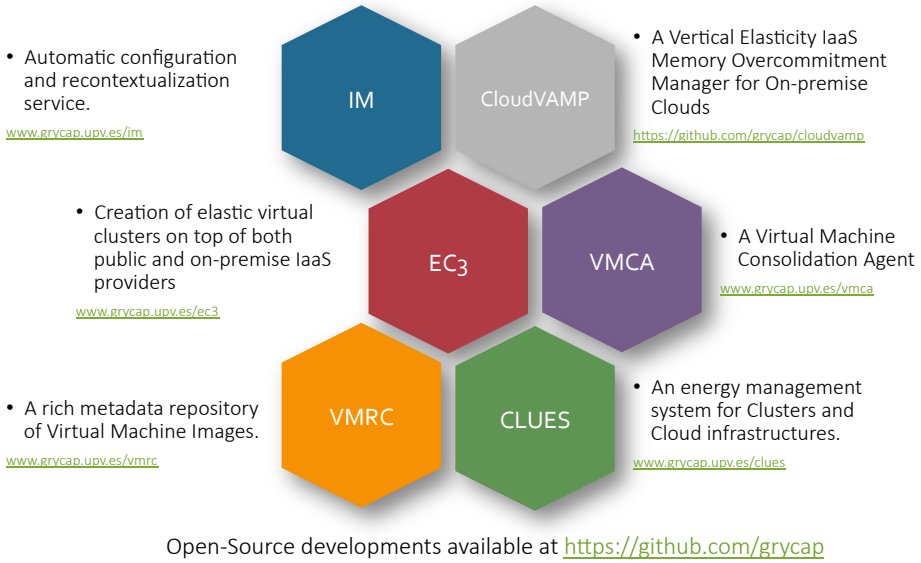


Figure 2.1: Open source software supporting the CLUVIEM project. See www.grycap.upv.es/{im,ec3,vmca,vmrc,clues}.

2.2.1 Elasticity

Virtual clusters must feature elasticity in order to cope with the dynamic computation requirements of the applications being executed. Figure 2.2 includes the different elasticity schemes addressed by CLUVIEM. First of all, vertical elasticity (Figure 2.2.a) enables to modify the capacities of the virtual machine at runtime without any downtime. Depending on the hypervisor support, these features include dynamic memory resizing, through memory ballooning, an dynamically adding virtual CPUs. Vertical elasticity allows, for example, to adapt the memory of the virtual machines that are executing a scientific application with dynamic memory requirements during its execution. A demonstration of that can be found in [137], in which it is monitored the memory consumption of an application within a VM and dynamically adapted the memory size of the VM to fit that memory consumption. In this way, the application does not incur in *thrashing* thus affecting its performance.

On the one hand, downsizing the memory of the VM when no longer required provides additional available free memory for other VMs that are currently being executed on the same physical host, a common situation on multi-tenant on-premises Cloud platforms. On the other hand, increasing the amount of memory of a VM might exceed the capacities of the underlying physical host. For that

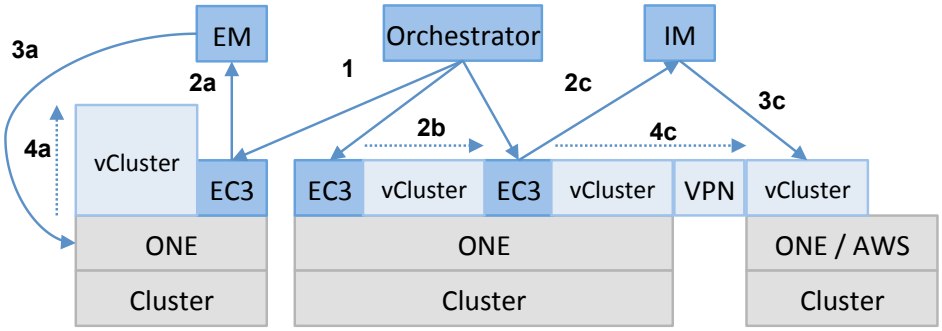


Figure 2.2: Elasticity schemes for virtual clusters: a) scaling up an already deployed working node (wn); b) deploying a new working node on the on-premises Cloud; c) deploying a new working node on another Cloud.

reason, live migration (without any downtime) of VMs to restore the Quality of Service is imperative. This enables to rebalance the workload of VMs across the datacenter (or across a physical cluster of nodes) so that the VMs have access to the required computational resources. This situation requires an appropriate migration plan that considers the whole state of the underlying physical infrastructure and decides which VMs should be migrated to which nodes. For that, we plan to use VMCA, an add-on to CLUES that defragments the available resources by migrating VMs among physical hosts. This results in an increased density of VMs per real host.

Second, horizontal elasticity enables to shrink and grow the cluster size, in number of nodes, according to the values of some metrics such as the number of jobs queued up at the LRMS. Figure 2.2.b represents the scale out of a virtual cluster deployed on an on-premises Cloud managed by OpenNebula (ONE). The ability to resize a virtual cluster enables to cope with increased workloads at the expense of an increased cost (either in terms of energy, when running on an on-premises Cloud, or in terms of money, when running on a pay-as-you-go public Cloud). For that, we leverage the already-existing policies of CLUES, but adapted to a Cloud scenario (instead of powering on and off physical nodes through Wake-on-LAN (Local Area Network (LAN)) or Intelligent Platform Management Interface (IPMI), VMs are deployed or terminated in a Cloud).

The Elasticity Manager (EM) is actually the aforementioned CLUES software, which runs in the head node (FE) of the cluster. Therefore, the cluster is self-managed and can scale in and out according to the elasticity rules without any user intervention. When the user deploys the cluster, the maximum number of nodes (VMs) is specified so that a reasonable cost per hour is never exceeded (when using a public Cloud). This means that these enhanced clusters automatically enter a low-cost mode (either energy or money) when no job workload is pending

or expected to be executed. Notice that for certain workloads (e.g. burst of jobs) the cost of re-deploying a new cluster does not pay off when compared to provisioning additional worker nodes, which typically involves less configuration and time.

2.2.2 Hybrid Scenarios

When trying to leverage both on-premises and public Cloud resources, hybrid scenarios arise (as depicted in Figure 2.2.c), in which VMs are deployed from different Cloud providers. For example, a virtual cluster is initially deployed on an on-premises Cloud and additional worker nodes are provisioned from another on-premises Cloud or a public Cloud to supplement the single virtual cluster with additional resources. Notice that the master node can either be deployed on the on-premises Cloud or on the public Cloud.

There are different scenarios in which provisioning from multi-cloud environments is beneficial. First, this approach can overcome a temporary shortage of computational resources within an on-premises Cloud. For example, when the requirements, in terms of number of nodes or their computational or memory capacities, of a virtual cluster exceed the capacities of an on-premises Cloud platform. A hybrid virtual cluster can span across an on-premises and a public Cloud to provide transparent Cloud bursting without affecting the users, which simply submit their jobs to be executed in the cluster through the LRMS. This is the case of our previous work [37], in which hybrid virtual clusters are employed to execute a parallel computationally intensive gyrokinetic plasma turbulence code running on such hybrid clusters with resources provisioned from an on-premises OpenNebula Cloud and AWS.

To create a common network among the VMs in disparate Cloud providers we provide support for Virtual Private Network (VPN)s with OpenVPN, where the OpenVPN server is automatically deployed and configured on the head node of the cluster. Alternatively, we also support automatic deployment of Secure Shell (SSH) tunnels, customised with *iptables* rules to decide when to channel the traffic through the tunnels.

2.2.3 Migration

Migrating virtual infrastructures is of interest both for datacenter administrators and for the owners of the virtual infrastructures themselves. On the one hand, datacenter administrators might require to decommission a physical node, perhaps because the SMART disk monitoring system alerts of an imminent failure. In this way, the ability to migrate virtual infrastructures enables to redistribute the virtual machines among the other physical nodes in the datacenter. For that, one

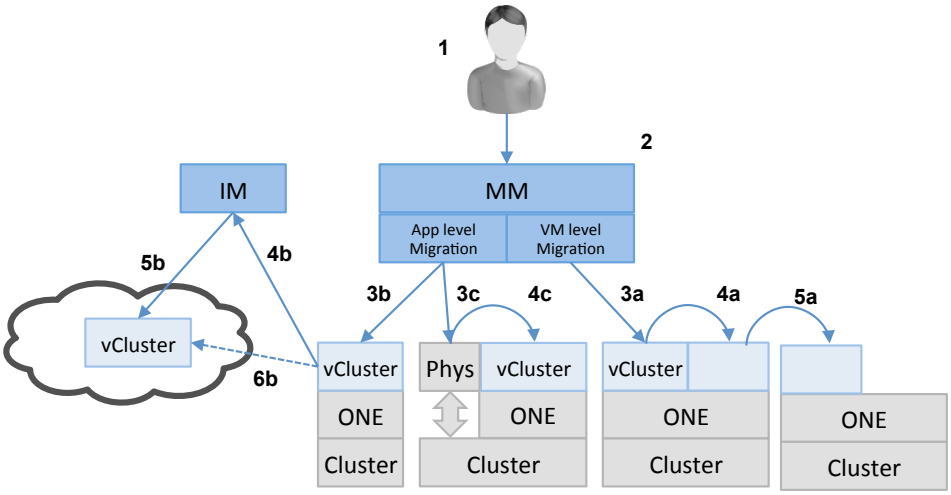


Figure 2.3: Migration schemes for virtual clusters.

can leverage the live migration capabilities available in hypervisors such as KVM or Xen so that VMs are migrated without any downtime or service disruption. Fortunately, CMPs such as OpenNebula leverage this ability to provide graphical tools to aid the sysadmin. However, migrating virtual infrastructures across Cloud providers is not a trivial task, where disparate hypervisors and platforms are employed. In the CLUVIEM project we address migration as shown in Figure 2.3.

First, the migration of virtual infrastructures can be achieved within the same Cloud on-premises (see arrows labeled a in Figure 2.3) by using live migration. We have assessed the capabilities of KVM to perform live migration across physical nodes of the same OpenNebula deployment without any downtime. Migration can also occur across different on-premises Clouds (as shown in $5a$ and across public Clouds (shown in arrows labeled b). For that, we deploy a replica of the virtual cluster into a different Cloud provider, coordinated by the Migration Manager (MM). Since clusters are created out of a high level language called Resource Application and Description Language (RADL); see [32] for details) it is possible to replicate the infrastructure into another Cloud provider by using the multi-Cloud capabilities of the IM. This involves deploying a new infrastructure with the same characteristics in another Cloud back-end. Transitioning from a physical cluster to a virtual one requires abstracting its hardware, software and data configuration to be expressed in RADL, what we intend to provide in a semi-automatic way but it is currently under research.

Second, the migration of running applications requires the introduction of application-independent checkpointing techniques in order to be able to resume a running ap-

plication on the target virtual machine instance. For that purpose we have been using BLCR [65] for Linux, a tool that introduces checkpoint capabilities both for sequential and parallel applications based on MPI. We use checkpointing both for migration of applications and as an application survival mechanism when using spot instances in Amazon EC2. A spot instance can be terminated if its price exceeds the bid of the user. For that, we developed a Checkpoint Manager that interacts with the Simple Linux Utility for Resource Management (SLURM) LRMS supporting BLCR in order to checkpoint the jobs both at periodic interval and considering the evolution of the prices of the spot instances. This way, interrupted jobs can be resumed in newly deployed instances, which may be on a different Cloud (with the same virtual hardware).

Migrating workloads, such as independent jobs that arise from HTC, can be efficiently achieved by deploying hybrid virtual clusters that dynamically remove and add nodes, from different Clouds, that are activated/deactivated from the LRMS so that jobs can be balanced across the working nodes without any user intervention, as performed in [37].

2.3 Discussion and Application Scenarios

These enhanced virtual clusters can be employed for many applications in which cost-effective cluster-based computing is required. In particular, we are focusing on the following scenarios. First, the non-linear and dynamic structural analysis of buildings, where it is required to accurately simulate how a building is affected by external dynamic loads, such as an earthquake. This involves a parallel MPI-based applications. Second, the execution of Monte-Carlo simulations to describe the trajectories of particles used in radiotherapy dosimetry and Positron Emission Tomography (PET) devices. Finally, the deployment of virtual clusters as educational infrastructures for HPC-related subjects in Master's Degree.

2.4 Conclusion

This paper has summarised the developments towards self-managed cost-effective elastic virtual clusters on hybrid Cloud infrastructures. So far, the developments of this vision are based on the open source EC3[2] tool, which enables to provision virtual hybrid elastic clusters that span public Clouds (AWS and Google Compute Engine) and on-premises CMPs (OpenNebula, OpenStack and any other Open Cloud Computing Interface (OCCI) compliant software), featuring checkpointing capabilities and spot instances support. Supporting OCCI enables the user to provision resources from EGI FedCloud, one of the largest scientific computing

[2]EC3 - `http://www.grycap.upv.es/ec3`

platforms. We have released an early version of this tool to the academic community together with the main underlying software components.

We expect to continue our early developments on migration of infrastructures and applications, which will introduce unprecedented flexibility for cluster-based computing.

Chapter 3

Virtual Hybrid Elastic Clusters in the Cloud

Published as

A. Calatrava, G. Moltó, M. Caballer, and C. De Alfonso. "Virtual Hybrid Elastic Clusters in the Cloud." in 8th Iberian Grid Infrastructure Conference (IberGrid 2014), pp. 103–114, 2014. ISBN: 978-84-9048-246-9

Abstract

Clusters of PCs are one of the most widely used computing platforms in science and engineering, supporting different programming models. However, they suffer from lack of customizability, difficult extensibility and complex workload-balancing. To this end, this work introduces virtual hybrid elastic clusters that can simultaneously harness on-premises and public Cloud resources. They can be fully customized, dynamically and automatically enlarged and shrinked (in terms of the number of nodes) and they offer migration capabilities to outsource workload from one datacenter to another (or to a public Cloud) in different situations. This is exemplified with case studies that involve a parallel computationally intensive gyro kinetic plasma turbulence code running on such hybrid clusters with resources provisioned from an on-premises OpenNebula Cloud and the AWS public Cloud.

3.1 Introduction

The usage of clusters of PCs as a computing facility is widespread in the scientific community with high success for both HPC and HTC. Nevertheless, these computing platforms suffer several drawbacks. On the one hand, physical clusters cannot be easily enlarged or shrinked, without downtime, according to the dynamic requirements of the organization. On the other hand, they lack the ability to provide a tailored execution environment customized for each application to be executed, specially when incompatible libraries are required by different applications running on the same cluster. Moreover, the ability to transparently migrate running jobs to be executed on other physical resources is not a trivial task, which is an important feature for effective workload balancing. Another important limitation is the large upfront investment together with the maintenance cost of such computing equipment for small and medium-sized research groups or organizations [58].

Traditionally, virtualization was not considered a viable option for HPC due to the overhead costs in I/O and network devices, but the major improvements in hypervisor technologies and virtualization have paved the way for Cloud computing. This paradigm can solve those disadvantages with customizable VMs that decouple the application execution from the underlying hardware and are dynamically provisioned and released on a pay-as-you-go basis [127]. This is the case of public IaaS Cloud providers such as AWS [12] or on-premises Cloud infrastructures deployed by CMPs such as OpenNebula [175], OpenStack [150] or Eucalyptus [147].

In the last years, the adoption of on-premises Cloud solutions for datacenters has increased [78], in part due to the maturity of the CMPs, which are becoming reliable enough for production-ready workloads. This enables to create virtual hybrid elastic clusters that ease extensibility and customizability in contrast to physical clusters. Virtualization also enables to partition a physical cluster into different virtual clusters specifically customized for the applications. Moreover, decoupling the application execution from the underlying hardware paves the way for migrating running applications from one hardware to another. Migration is convenient for several reasons. First of all, it enables to move workload from an overloaded datacenter (source infrastructure) to another (where the former or the latter could be a public Cloud infrastructure) to achieve workload balancing. Second, a planned maintenance on the source infrastructure might affect an application that has been running for a long time, so migration introduces the flexibility required in a modern datacenter.

This work focuses on harnessing resources from hybrid scenarios that simultaneously include resources from on-premises and public Clouds in order to create ready-to-use customized virtual elastic clusters that satisfy the dynamic computing requirements of an individual or organization. The remainder of this paper is structured as follows. First, section 3.2 covers the related work in this area. Next,

section 3.3 focuses on the main characteristics of the proposed cluster deployment architecture pointing out the main scenarios where the use of this type of clusters is beneficial. Then, the paper introduces the elasticity rules employed to automatically enlarge and shrink the cluster. Afterwards, two case studies are presented in section 3.4 that justify the benefits of the proposed approach. Finally, section 3.5 summarizes the main achievements and discusses the future work.

3.2 Background and Related Work

There are previous works in the literature that aim at deploying virtual clusters on Cloud infrastructures. For example, StarCluster [134] enables the creation of clusters in Amazon EC2 from a predefined configuration of applications (Open Grid Scheduler, OpenMPI, NFS, etc.). Along with StarCluster a plugin called Elastic Load Balancer has been developed that is able to enlarge and shrink the cluster according to the length of the cluster's job queue. This plugin typically runs on the local computer from which the cluster was deployed and requires permanent connection to the Cloud infrastructure, in order to create and destroy the VMs. Therefore, any disconnection on the client computer means a loss of control of the elasticity capabilities of the cluster. Instead, our proposal focuses on self-managed hybrid clusters, where the elasticity rules are embedded in the cluster and no external entity is required. ViteraaS [62] allows creating virtual clusters to manage the execution of user-defined jobs, but the user cannot remotely access the cluster. The standard distribution of Hadoop [28] 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 AMI and creates the computing nodes using another AMI, performing the required network configuration.

There are commercial solutions, like IBM Platform Dynamic Cluster [94] that aims at partitioning the owned resources to deliver each user a custom cluster with specific features. It has features such as live job migration and automated checkpoint restart. The drawback in this case is that this product is oriented to manage on-premises infrastructures and cannot be connected to commercial Cloud providers. CycleCloud [56] 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 Torque [2] 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.

All of the above examples only consider the usage of one Cloud infrastructure (no hybrid scenarios supported). Concerning the creation of clusters over hybrid Cloud infrastructures, the authors of works such as [138], [139] and [140] have analyzed

architectures, algorithms and frameworks to deploy HTC clusters over this type of infrastructures. They 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 works use a fixed number of on-premises nodes, and scale out the cluster using public nodes. However, workload migration from one infrastructure to another is not considered. In [120], [121], [66], 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 [22], 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.

An important point in a hybrid environment is the interconnection between nodes that are deployed in different Clouds. The authors of [106] introduced the concept of Sky Computing, which enables the dynamic provisioning of distributed domains over several Clouds. They pointed that one of the shortcomings of this approach is the need of trusted networking environments. The same authors of the previous work [182] study the inter-Cloud communication problem across independent Clouds in detail.

Our previous work in the field is EC3 [34], a tool that creates elastic virtual clusters out of computational resources provisioned from IaaS Clouds, but no hybrid scenarios are supported. These 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 dynamically and automatically scale in, according to some simple 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.

3.3 Virtual Hybrid Elastic Clusters

The objective of the proposed architecture is to ease the deployment and management of customized virtual hybrid elastic clusters whose computational resources are simultaneously provisioned from on-premises Clouds and from different public IaaS Cloud providers. In order to have all the nodes that compose the cluster in the same subnetwork, regardless of their physical location, it is convenient to use VPN techniques. A VPN enables a computer on a public network, such as the Internet, to exchange data as if it was connected to the private network.

Figure 3.1 shows the proposed architecture to deploy virtual hybrid elastic clusters. The user (or the system administrator of the datacenter) requests a cluster deployment to the IM) [7], a component that is in charge of contacting different

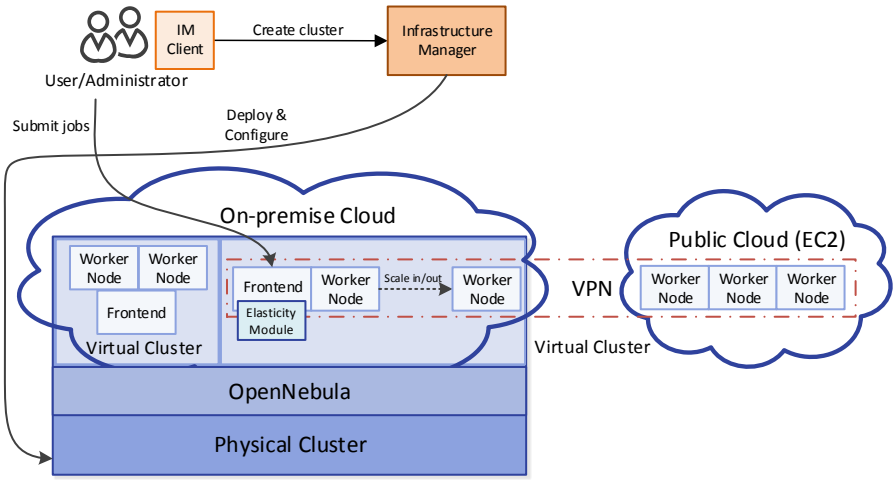


Figure 3.1: Proposed cluster deployment architecture.

CMPs in order to deploy the VMs that compose the virtual cluster. As a summary, the lifecycle of the virtual hybrid elastic cluster consist of three phases: (1) the creation of the infrastructure, (2) configuring the computing nodes to behave as a cluster and (3) managing the elastic cluster.

Firstly, the user provides the IM with the initial number of nodes of the cluster, and their distribution among the on-premises and/or public Clouds accessible with his credentials. The IM is in charge of phase (1). In the scenario shown in the Figure, the front-end (head node) is deployed on the on-premises Cloud, but it could also be deployed on a public Cloud. Two corner cases are also considered, which are deploying a virtual cluster fully on a public Cloud or fully on an on-premises Cloud. The latter complies with the requirements of research communities that require instant access to cluster-based computing with data that does not leave the boundaries of the organization.

Once the computing resources for the cluster have been provisioned, the Ansible tool is employed in phase (2) for the configuration via a set of high-level recipes. This requires to configure the VPN-based network. For that, we rely on OpenVPN [152] to implement network tunnels between each individual Cloud resource and the organization's private network. By default, the front-end of the cluster hosts a VPN server, to produce self-managed clusters that do not require external VPN services to work. However, our solution could also rely on the VPN services provided by the organization. The configuration process also installs and configures the Torque LRMS that is in charge of scheduling the execution of the jobs.

Moreover, the dependencies of the application that is going to be executed in the virtual cluster are installed and configured in this step.

When the virtual cluster is running, the users can connect via SSH to the front-end and execute their jobs. At this point starts phase (3). The workload of the cluster is periodically evaluated by the elasticity module, a component that can run inside the front-end (to produce self-managed clusters) or outside, and is in charge of monitoring the state of the LRMS queue to enforce a set of elasticity rules (described in section 3.3.1) to trigger actions in order to scale out (increase) or scale in (decrease) the number of nodes of the cluster. The user can choose from a set of elasticity rules as well as the maximum number of nodes of the cluster (growth limit).

From the point of view of the user, the use of a virtual hybrid elastic cluster enables proper customization of the execution environment, thus ensuring compatibility with the applications to be executed. The automatic elasticity, together with the workload balancing capability, enables to reduce the total execution time of the jobs by dynamically adapting the size of the cluster to the workload.

In addition, the system administrator is able to manage the resources of the virtual cluster with unprecedented flexibility due to the migration capabilities. For example, this enables a system administrator to temporarily outsource cluster-based workloads to a public Cloud (or to another datacenter running an on-premises Cloud) to deal with a planned outage.

3.3.1 Elasticity Rules

The elasticity module included in the architecture dynamically adds and removes worker nodes from the cluster by monitoring the front-end job queue. This module uses various policies to determine when to deploy additional VMs (scale out) in the Cloud or terminate them (scale in) based on the monitored information. The behaviour of this module is described in Algorithm 1.

The elasticity rules or policies can be proactive or reactive. Proactive rules can be employed if job execution patterns in the clusters are known, in order to deploy and configure the nodes just in time for the execution of the jobs arriving at the LRMS. However, proactive rules typically under-perform with stochastic job execution patterns. Therefore, this paper focuses mainly on reactive elasticity policies that deploy and relinquish resources based on the actual state of the cluster.

The scale out policies determine when it is necessary to increase the number of worker nodes of the cluster. Two policies are included: i) *On demand*, where for each job in the queue a worker node is deployed. Therefore, the jobs will wait for the deployment and contextualization of the node before they start their execution and ii) *Bursts*, this policy deploys a group of VMs for each job in the

Algorithm 1 Elasticity management

```
Require: Growth limit, l > 0, scale_out policy, scale_in policy
  while the front-end is running do
    Obtain the number of jobs in queue, j , and the total number of nodes, n
    if j > 0 and n < l then
      Apply scale_out policy
    end if
    if j == 0 then
      Obtain state of the nodes
      if some node state is free or offline then
        Apply scale_in policy
      end if
    end if
  end while
```

queue, assuming that if a job arrives at the LRMS there is an increased chance for other jobs to arrive shortly (thus including some proactivity). For HTC-based applications, such as BoT or parameter sweeps, this policy is expected to reduce the average waiting time of the jobs at the expense of an increased cost (economic, in the case of public Clouds or due to idle resources in the case of on-premises Clouds).

The scale in policies determine when it is necessary to decrease the number of worker nodes. Two policies are considered: i) *On demand*, to terminate the idle worker nodes when there are no pending jobs in the LRMS and ii) *Delayed shut-down*, where idle worker nodes are terminated after a certain amount of configurable time. This is of interest when using public Clouds that bill by the hour, where idle nodes are kept available for job executions before the hour expires, even if no jobs are available to be executed at the moment.

Notice that when there are no jobs in the LRMS for a period of time, all the worker nodes will be eventually terminated, regardless of the elasticity policies, thus resulting in a cluster with only the front-end running.

3.4 Case Study

In order to evaluate the benefits and impact and to validate the developed system on real scenarios, two case studies are proposed using a computationally intensive parallel (hybrid MPI/OpenMP approach) scientific application. The GENE [57, 85] version 11 (release 1.7) provides a state-of-the-art nonlinear gyrokinetic solver aimed to efficiently compute the microturbulence and the resulting transport coefficients in magnetized fusion and astrophysical plasmas. The application requires

an MPI-2 environment, Fortran and C compilers, and the BLAS and LAPACK libraries.

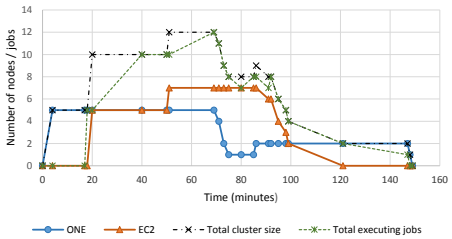
The infrastructure used to deploy the on-premises Cloud is composed by an heterogeneous blade-based system that has 4 kind of nodes: 2 x (2 quad-core L5430@2.6 Ghz, 16 GB), 2 x (2 quad-core multithreaded E5520@2.26 GHz, 16 GB), 6 x (2 quad-core multithreaded E5620@2.4 GHz, 16 GB) and 3 x (4 quad-core multithreaded E7520@1.86 GHz, 64 GB), with a total amount of 128 cores and 352 GB of RAM. The blade system is backed by a 16 TB Storage Area Network (SAN) connected via a private gigabit ethernet network. This system is managed by OpenNebula 4.4, using KVM as the underlying hypervisor. With respect to the public Cloud, we relied on AWS. The instance type chosen is `m1.medium`, with one (virtual) processor, 3.75 GB of RAM and 410 GB of HD. In this way, the VMs deployed on the on-premises Cloud have the same characteristics. The VMI used in the on-premises Cloud provides the same execution environment as the EC2 AMI employed (`ami-e50e888c`). Both images provide a pristine installation of Ubuntu 12.04 Long Term Support (LTS).

The case studies consist of jobs composed by 4 MPI processes in each node, using a maximum of 1000 MB per process. These processes communicate each other inside the node, but they do not communicate with external processes. The case studies have been downsized to get job executions in the order of minutes (instead of hours) to better focus on the dynamic cluster topology and job scheduling. The limit to the growth of the cluster has been fixed to 15 nodes.

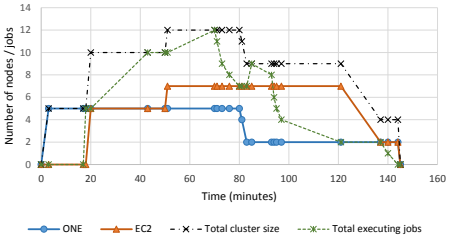
3.4.1 Cloud bursting

Our first case study focuses on automatic Cloud bursting, where a cluster running on an on-premises Cloud is automatically enlarged with computational resources provisioned from a public Cloud, with no service disruption. This enables to cope with an increased workload. For that, the user deploys the cluster and submits the jobs for execution. The cluster eventually becomes overloaded and, therefore, the elasticity module in the cluster that is monitoring the status of the queue decides to deploy new worker nodes (scale out) to handle the execution of the pending tasks. The new worker nodes can be on-premises resources, which are favoured, or/and public resources if no more resources can be allocated on-premises. We are going to assume that the user is restricted to 5 VMs simultaneously running on the on-premises Cloud, and the front-end is going to be able to execute jobs (but it cannot be shutdown unless the user terminates the cluster). The goal of this case study is to analyze the behaviour of the scale in/out policies described in section 3.3.1.

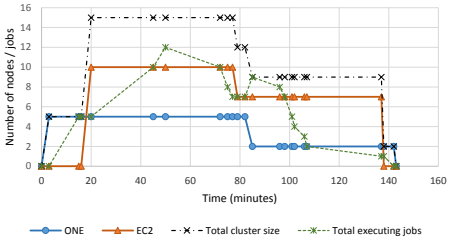
In the three cases, the workflow is as follows: the user requests an initial cluster of 5 nodes on the on-premises Cloud to the IM. When the initial cluster is deployed



(a) Scale out: on demand. Scale in: on demand



(b) Scale out: on demand. Scale in: delayed shutdown



(c) Scale out: bursts. Scale in: delayed shutdown

Figure 3.2: Evolution of cluster's size and nodes distribution for different policies.

and configured (minute 16-18), the user submits 10 jobs to the cluster. Regardless of the decision of the elasticity module, the user will submit two new jobs in minute 50. This behavior will be repeated in minute 85, submitting two more jobs to the cluster (14 jobs in total). It should be noticed that the average duration of each job performed in the case study is 55 minutes, regardless of the underlying infrastructure.

The bursts policy has been configured to deploy twice as much nodes as jobs were in the queue. The behaviour of the delayed shutdown scale out policy works as follows: the on-premises nodes are terminated 10 minutes after they finished executing their jobs, if there are no more jobs waiting in the queue. Since Amazon bills by the hour, the nodes deployed in Amazon EC2 are terminated minutes before the paid hour goes by.

Figure 3.2 shows the behaviour of the three possible combinations of the elasticity policies. Note that the combination of the bursts (scale out) policy with on demand (scale in) policy is not possible because the extra nodes that are deployed for future jobs would be shutdown immediately by the monitor system. The elasticity rules periodically examine the job queue (every 10 seconds for these tests), executing a policy and resizing the cluster if required.

Figure 3.2 shows that in (a) there are no idle periods in the worker nodes. The lifetime of the VMs is adapted to the workload of the cluster, but every time the cluster has a job waiting in the queue, a period of deployment plus contextualization is needed (4 periods in this case, minutes 4-17, 20-40, 51-69 and 85-92). In contrast, (c) has only 2 periods of deployment and contextualization (minutes 3-15 and 20-45), but the use of the nodes is not optimized, having idle machines the most part of the time. (b) is the intermediate solution, where the utilization of computing resources is better than (c) and there are 3 periods of deployment plus contextualization (minutes 3-17, 20-43 and 51-70). The contextualization process is faster in the ONE nodes than in the EC2 nodes because of the network, since the IM is deployed in the infrastructure as the on-premises Cloud deployment.

Notice that every time a node is terminated, the cluster needs another period of contextualization in order to reconfigure and restart Torque, what requires an average of 3 minutes. A new node form the public Cloud can be included in the cluster in 15 to 20 minutes, and it does not affect to the execution time of jobs because this process is done in parallel. Similarly, when a new node is added to the deployed infrastructure, the contextualization process mainly affects to the new nodes, that need to install all the software. The rest only need to reconfigure their `/etc/hosts` file and restart Torque, but it does not affect the execution of the jobs. The total execution time of the jobs is similar in the three combinations (142 minutes in (c), 144 in (b), and 148 minutes in (a). This is because the execution of the last job burst (composed by 2 jobs) in minute 85, needs almost an hour to be executed.

3.4.2 Maintenance period in a datacenter

Maintenance periods or planned outages are very common situations in a datacenter, and they might cause several inconveniences for the users of the datacenter resources that the system administrator must deal with. Therefore, the second case study is focused on the migration capabilities of the cluster, a very useful feature from the point of view of the system administrator.

First of all, when the user or system administrator requests a shutdown for some worker nodes, two different approaches can be used: i) *Aggressive shutdown*, that removes worker nodes without waiting for the completion of the jobs being executed, and ii) *Ordered shutdown*, which waits for the jobs to finalize their executions before shutting down the worker nodes.

An aggressive shutdown should not be used on tightly coupled parallel applications, where the loss of a worker node ruins the execution of the job unless application checkpointing is employed. However, for embarrassingly parallel independent jobs, this approach can be combined with a job resubmission system that enables to execute again the failed tasks. This, of course, affects the total execution time

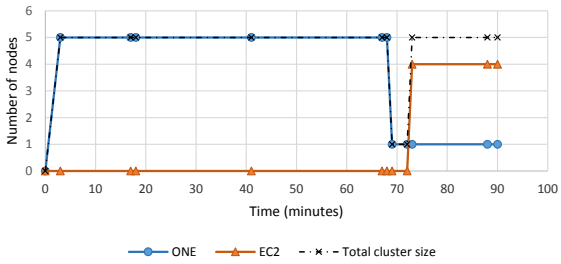


Figure 3.3: Ordered shutdown of nodes during a planned maintenance period.

of the jobs. In contrast, this total execution time is not affected by using the ordered shutdown, because the system waits until the end of the jobs execution to terminate the nodes. Therefore, we will use this latter approach to illustrate the migration capabilities of our developed system.

Figure 3.3 shows the evolution of the distribution of nodes when the cluster is asked to migrate part of its nodes. In this case study, the chosen policies where on demand (scale out) and delayed shutdown (scale in). It started with an initial cluster of 5 on-premises nodes, where the user is executing 5 jobs (minute 18). In the minute 41, the sysadmin requests the migration of 4 worker nodes to the IM client because of a maintenance period in the physical infrastructure. So, the system disables these worker nodes, by setting their state to `offline`, so that they are not assigned new tasks. This state change does not affect the execution of the current jobs, which terminate in minute 67-68. One minute later, the monitor terminates the nodes, reconfigures the cluster and deploys 4 new nodes in the public Cloud. As a result, the migration process is completed in minute 73, where the 4 worker nodes that originally composed the cluster are now allocated on Amazon EC2 resources. When the user launches new jobs (minute 90), they are going to be executed by the new nodes, so the migration process was completely transparent for the user.

3.5 Conclusions and Future Work

This paper has introduced a software architecture that abstracts the details of cluster deployment and configuration over hybrid Clouds. The system features the provision of virtual hybrid elastic clusters, composed by on-premises resources and public resources from public Cloud providers. Moreover, the system is able to configure these resources to support the execution of the applications, and to adapt the cluster's size and topology to the dynamic characteristics of the application and the needs of the datacenter. The benefits of the proposed architecture have been exemplified by the execution of a computationally intensive gyro kinetic

plasma turbulence application, that demonstrates the feasibility of this type of architectures into de scientific community.

The future work involves improving the migration capabilities of the cluster. Different schemes are going to be considered, from the migration of the virtual clusters to another physical infrastructure (involving the migration of the front-end), to live virtual machine migration. This will introduce an unprecedented flexibility to decouple cluster-based computations from the physical infrastructure on which the cluster was initially deployed. Moreover, this will enable datacenter migration, or the ability to migrate running computational resources from one datacenter to another. Also, we consider to develop new elastic policies that improve the performance of our virtual hybrid elastic clusters. Spot instances from Amazon EC2 are going to be considered in order to reduce the total cost over public Clouds.

Chapter 4

Self-managed Cost-efficient Virtual Elastic Clusters on Hybrid Cloud Infrastructures

Published as

A. Calatrava, E. Romero, G. Moltó, M. Caballer, and J. M. Alonso. "Self-managed Cost-efficient Virtual Elastic Clusters on Hybrid Cloud Infrastructures." in Future Generation Computer Systems, Volume 61, pp. 13-25, 2016. ISSN: 0167-739X
http://dx.doi.org/10.1016/j.future.2016.01.018

Abstract

In this study, we describe the further development of EC3, a tool for creating self-managed cost-efficient virtual hybrid elastic clusters on top of IaaS clouds. By using spot instances and checkpointing techniques, EC3 can significantly reduce the total execution cost as well as facilitating automatic fault tolerance. Moreover, EC3 can deploy and manage hybrid clusters across on-premises and public cloud resources, thereby introducing cloud bursting capabilities. We present the results of a case study that we conducted to assess the effectiveness of the tool based on the structural dynamic analysis of buildings. In addition, we evaluated the checkpointing algorithms in a real cloud environment with existing workloads to study their effectiveness. The results demonstrate the feasibility and benefits of this type of cluster for computationally intensive applications.

4.1 Introduction

The successful use of clusters of PCs as a computing facility is widespread in the scientific community for both HPC and HTC. However, these computing platforms have several drawbacks, such as the requirement for a large upfront investment and maintenance costs, which have major economic effects in small and mediumsized organizations. Moreover, the size of a physical cluster cannot be adapted easily to the application workload and they cannot provide customized environments for executing each separate application.

In recent years, the development of hypervisors and virtualization technologies have paved the way for cloud computing. This paradigm can address those problems with customizable VMs, which decouple the execution of the application from the underlying hardware, where they are dynamically provisioned and released [127]. Thus, depending on the resource usage and cost model, it might be convenient to deploy a virtual cluster instead of a physical one, as suggested in a previous study [58]. Virtual clusters in the cloud are highly beneficial for many computational workloads, but particularly for highly parallel tasks. These benefits include the ondemand provision of per-application customized clusters as well as the ability to dynamically increase and decrease the number of working nodes in the virtual cluster according to the current workload, as demonstrated in our previous study [34]. Our previous study led to the development of EC3[1] [34] as an open-source tool for the deployment of customized virtual elastic clusters on different on-premises platforms, such as OpenNebula [175] and OpenStack [150], and public cloud providers, such as AWS [12]. In the present study, we build on our previous research by introducing two significant features: (i) automatic checkpointing coupled with cost-effective mechanisms for providing transient computing capacity (referred to as spot instances in AWS); and (ii) the ability to deploy hybrid virtual clusters across on-premises and public cloud platforms, according to an elastic scheme that spans different cloud providers.

For the first feature, we exploit the cost-effective advantages provided by cloud providers in order to reduce the total execution cost. This is the case for spot instances, which is a cloud pricing scheme available in Amazon EC2, where the users decide the maximum price that they are willing to pay for an instance, with savings of up to 86% compared with on-demand instances [13]. The user bids on spare Amazon EC2 instances and runs them whenever the bid exceeds the current spot price, which varies in real-time based on supply and demand. This variation causes the instance to terminate if the spot price is higher than the bid by the user, thereby interrupting the execution of the job. This situation is referred to as an "out-of-bid" situation. Recently, Amazon has included spot instance termination notices [67], which provide a two-minute warning before the provider terminates the spot instance. This improvement is useful for some applications, but two

[1] EC3: `http://www.grycap.upv.es/ec3`

minutes may not be sufficient time to checkpoint big applications, such as scientific applications, which might require additional time to save their content. Therefore, this type of instance is available at a lower cost but at the expense of reduced reliability. Thus, checkpointing allows the job progress to be saved periodically before the spot instance is terminated by the provider, thereby facilitating job resumption from the last checkpoint. In the present study, we review, propose, and implement checkpointing algorithms for this purpose.

For the second feature, the coexistence of on-premises and public clouds has leveraged cloud bursting, where virtual clusters can be enlarged with resources outside the organization, and thus hybrid clusters can harness on-premises and public cloud resources simultaneously. This approach is highly advantageous because it allows users to seamlessly access cluster-based computing resources in addition to those available via their onpremises clouds. Other topologies can be considered for hybrid clusters when using virtual resources, such as heterogeneous clusters, where various nodes in the cluster have different hardware characteristics.

Therefore, in this study, we extend the capacities of EC3 to allow users to deploy self-managed cost-efficient virtual hybrid elastic clusters on top of IaaS clouds. The remainder of this paper is organized as follows. First, Section 4.2 reviews related research as well as the main contributions of the present study to the state of the art. Next, Section 4.3 focuses on the architecture and the new features included in EC3. In Section 4.4, we present two case studies that we conducted to assess the functionality and benefits of the new features incorporated in EC3. Finally, we give our conclusions in Section 4.5 and we discuss future research.

4.2 Related work

Previous studies have aimed to deploy virtual clusters on cloud infrastructures, e.g., StarCluster [134] is an open-source tool that provides clusters in Amazon EC2 based on a predefined configuration for applications (Open Grid Scheduler, OpenMPI, NFS, etc.). In addition, CycleCloud [56] is a commercial service provided by CycleComputing that deploys virtual clusters. However, both tools can only provide resources from Amazon EC2 so virtual clusters cannot be deployed through on-premises cloud platforms created with CMPs such as OpenNebula or OpenStack.

Elasticluster [203] can be employed to create virtual clusters on two cloud providers (Amazon EC2 and Google Compute Engine) as well as on-premises cloud platforms (OpenStack supported). The clusters can be scaled by the user but automated elasticity is not supported. Other tools for deploying virtual clusters have also been reported such as ViteraaS [62], which allows the creation of virtual clusters to manage the execution of user-defined jobs, but users are not provided with

direct access to the cluster. There are also commercial solutions such as IBM Platform Dynamic Cluster [94], which aims to partition on-premises resources to deliver each user with a custom cluster that has specific features. The features of this system include live job migration and automated checkpoint restart. However, this product was designed for the management of on-premises infrastructures and it cannot be connected to commercial cloud providers.

In terms of the creation of clusters over hybrid cloud infrastructures, previous studies [138], [139] and [140] have analyzed architectures, algorithms, and frameworks for deploying clusters over these infrastructures, where they analyzed the performance of virtual clusters deployed on top of hybrid clouds and obtained good results to demonstrate the feasibility of this type of deployment. These studies used a fixed number of on-premises nodes and they scaled up the cluster using public nodes. However, the migration of workloads among infrastructures was not considered. In [120], [121], [66], the Nimbus toolkit was employed to implement and evaluate an elastic site manager, which dynamically extends existing physical clusters based on Torque using computational resources provided by Amazon EC2 according to different policies. A similar approach was employed by [22], who investigated the benefits of using cloud computing to augment the computing capacity of a local infrastructure, although no details of the underlying technologies were given.

Regarding spot instances, many studies have attempted to develop predictive models of spot price variations, where some of the proposed solutions are based on Gaussian distributions [99] or Markov chains, such as [178] and [174]. However, other studies found that the spot price variation over time in Amazon EC2 did not seem to follow any particular distribution [123]. It was also observed [5] that Amazon might intervene with the prices artificially by setting a reserve price and generating prices at random, thereby further complicating the prediction of spot price variations.

Another field of research is the deployment of virtual clusters using spot instances. In [122], the economics of purchasing resources on the spot market were considered when handling unexpected load peaks in a cluster, but they did not consider checkpointing techniques. If the instance is terminated, the application is restarted from the beginning. This was the case in [42] where high bids were made rather than employing checkpointing strategies, but this solution can incur higher costs, thereby opposing the main advantage of spot instances. Moreover, Amazon has recently limited [11] the bids made by users to 10 times the on-demand price of the instance. Other solutions have involved deploying a cluster that fully comprises spot instances [184], but again it was assumed that the bid value was sufficiently large to avoid the spot instance being killed by Amazon. Finally, SpotMPI was presented in [177] as a toolkit to facilitate the execution of MPI applications on volatile auctionbased cloud platforms. This toolkit can monitor spot instances and bidding prices to automate checkpointing at the bidding price and automat-

ically restart the application after out-of-bid failures. However, this tool has the following limitations. First, it is based on StarCluster so it is restricted to AWS. Second, elasticity management for the clusters is not self-managed inside the cluster because StarCluster implements the elasticity using the Elastic Load Balancer plugin [135], which runs on the local computer from which the cluster was deployed, thereby requiring a permanent connection to the cloud infrastructure to create and destroy the VMs. Instead, we propose self-managed virtual clusters where no external entities are required for elasticity management.

In conclusion, to the best of our knowledge, no previous study has proposed a tool that integrates hybrid virtual clusters with the use of spot instances and checkpointing techniques, as well as featuring self-managed elasticity. EC3 provides a full-featured open-source development and a web-based interface, which allow users to deploy their own customized virtual clusters on AWS, OpenNebula, Openstack and EGI FedCloud.

4.3 Elastic Cloud Computing Cluster (EC3)

EC3 was developed to create virtual elastic clusters on top of IaaS clouds. These self-managed clusters have the ability to adapt the size of the cluster to the workload, thereby creating the illusion of a real cluster, but without requiring any investment beyond their actual usage. Therefore, they can scale up to a larger number of nodes (up to a maximum size specified by the user) depending on the number of jobs that need to be queued in the LRMS. Whenever idle resources are detected, the clusters scale up dynamically and automatically in order to reduce the costs when using a public cloud provider. The elasticity management is conducted by the front-end node of the cluster using CLUES [9]. More details of the elasticity management process are available in [34].

EC3 supported different cloud providers but the cluster had to be fully deployed in the same IaaS cloud. Therefore, EC3 did not provide support for hybrid clusters. In the present study, we describe the development of support for virtual hybrid elastic clusters in EC3, where on-premises resources are supplemented with public cloud resources to accelerate the execution process by providing further resources. Different instance types and the use of spot instances combined with on-demand resources are further cluster configurations supported by EC3.

Moreover, we modify EC3 to allow users to improve the cost/performance ratio. On-demand instances incur higher costs than spot instances, but the latter present higher risks at the expense of a lower cost and an increased provisioning time (the delay until the bid is accepted). In order to alleviate the risks of using spot instances, such as out-of-bid situations, EC3 employs checkpointing techniques. However, deciding when to perform a checkpoint to save the execution progress of

the jobs running on spot instances is not a trivial task, particularly for applications with large memory footprints, where the time required to perform a checkpoint cannot be neglected.

It is important to note that the user experience should be maintained regardless of the physical location of the cluster and the types of machines employed. Indeed, the user should be unaware that the virtual cluster actually comprises resources on top of one or more clouds where the infrastructure adapts its size dynamically to the workload. The user is provided with the IP of the cluster and an SSH client is employed to access the front-end node.

In the following subsections, we describe the overall architecture of the EC3 tool and its components.

4.3.1 Previously developed EC3 components

In order to deploy and configure the virtual cluster, we employ previously developed components and open-source software that are available to the community, as follows.

- **Resource Application Description Language (RADL)** [7]: A declarative language that allows users to describe the computational infrastructure required to run their applications.
- **Virtual Machine image Repository and Catalog (VMRC)** [46]: This component indexes the VMIs stored in different cloud VMI repositories. It also implements matchmaking algorithms to obtain a ranked list of VMIs that satisfy the aforementioned set of requirements (as described in the RADL document).
- **CLUES** [9]: This energy management system is used in our architecture to manage the automatic elasticity of the virtual clusters. Initially, CLUES was designed to work with physical clusters, but it can work with virtual resources via a cloud connector. CLUES implements the policies used to decide when to increase the capacity of the cluster and when to decrease the number of nodes. More details of the elasticity policies can be found in [34].
- **Ansible** [190]: This DevOps tool is used to perform the unattended execution of commands specified in a Yet Another Markup Language (YAML) document in order to automatically install the software dependencies. Therefore, this tool installs the required software packages so the VMs behave as a cluster and the execution environment of the applications is configured successfully.

- **Infrastructure Manager (IM)** [33]: The IM is in charge of contacting different CMPs in order to deploy the VMs that comprise the virtual cluster, the requirements of which are described in the RADL document. The IM also manages the contextualization of the VMs using Ansible. The IM uses a set of plugins to access a large number of cloud deployments and virtualization platforms. The IM currently provides access to OpenNebula, OCCI, Amazon EC2, Google Cloud, Microsoft Azure, Docker, OpenStack, and libvirt.

- **Local Resource Management System (LRMS)**: Two different LRMSs are currently supported by EC3: SLURM [100] and Torque + Maui. First, SLURM is an open-source resource manager, which provides a framework for starting, executing, and monitoring work on a set of allocated nodes. Second, the Torque Resource Manager [2] allows the control of batch jobs and distributed computing resources by using Maui as a job scheduler to coordinate the execution of jobs over the cluster.

- **Berkeley Lab Checkpoint/Restart (BLCR)** [65]: This is a tool for transparently checkpointing applications, including MPI applications, where the checkpoints are performed at the hybrid user/kernel level. BLCR does not require changes in the application code to perform a checkpoint. In EC3, BLCR is used to checkpoint applications running on spot instances, thereby saving the job state.

- **Network File System (NFS)**: In a spot instance environment, NFS is used to create a shared directory among the nodes of the cluster where the checkpoint files are saved. Thus, when a spot node is destroyed, the checkpoint file of the job under execution can be accessed to restart the job on another node.

- **OpenVPN [152] and SSH tunnels**: These two technologies provide connectivity between the front-end and the working nodes when some of the hosts are in a private network or behind a firewall. This situation typically arises in hybrid clusters.

4.3.2 Overall architecture

Figure 4.1 summarizes the main architecture of EC3. The deployment of the virtual cluster comprises three phases: (1) start a VM in the cloud to act as the cluster front-end, (2) configure the frontend, and (3) manage the cluster size automatically and configure new nodes according to the workload.

First, the user provides the EC3 client (using the Graphic User Interface (GUI) or CLI) with the following data to perform phase (1): a cluster name to facilitate the management of the cluster for the users (connecting via SSH, showing information, reconfiguring, etc.), a file that contains credentials for accessing the

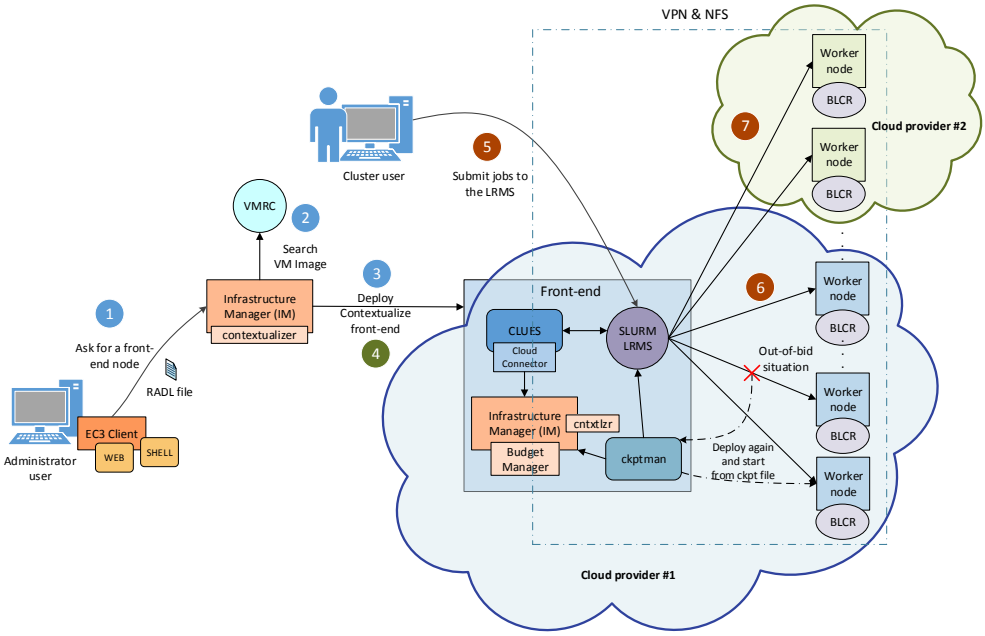


Figure 4.1: Proposed cluster deployment architecture. Phase 1 (blue), phase 2 (green) and phase 3 (brown) are also indicated.

different cloud providers, and the endpoint of the IM for deploying the front-end together with the RADL files (step 1 in Fig. 4.1). The user can employ the default RADLs provided by the tool (such as the instructions required to configure a SLURM cluster or the hybrid features), but the user can also include additional customized RADLs to configure the cluster for specific applications. Inside the RADLs, the characteristics of the nodes are given in terms of hardware, software, and configuration requirements.

Moreover, the user can specify the maximum number of nodes that comprise the cluster. The specification of a maximum number of nodes is also supported for a given type (e.g., the maximum number of nodes based on spot instances or the maximum number of on-demand nodes). The user can change these values during the lifecycle of the cluster via the reconfigure command in the EC3 client.

According to the data specified by the user, EC3 contacts the IM to deploy the front-end. First, the IM selects the appropriate VMI for the front-end. A particular user-specified VMI can be selected, or the VMRC can be contacted to choose the most appropriate available VMI (step 2) by considering the requirements in the RADL file. The IM then chooses the IaaS cloud provider and the type of instance (spot or on-demand if necessary) according to the requirements specified by the

user (step 3). In the RADL file, the user can specify whether spot instances will be used or not. It should be noted that the IM will only use spot instances if the actual spot price is lower than the on-demand price. Finally, phase 1 concludes by deploying the instance that will be used as the front-end node of the cluster.

Phase 2 starts when the aforementioned instance is available by installing and configuring all of the required software that is not already preinstalled in the VM (step 4). In this phase, all of the required software is installed to configure the instance as the front-end of the cluster, which involves deploying: (i) a new IM in the front-end for deploying the worker nodes; (ii) Ansible to configure the new nodes; (iii) CLUES to manage the elasticity of the cluster; (iv) the LRMS selected by the user; and (v) (optionally) additional software packages specified by the user. If the cluster needs to be configured as a hybrid, VPN or SSH tunnels must be configured to interconnect all of the nodes (selected by the user). Moreover, BLCR and NFS are configured if the user enables the use of spot instances.

Phase 3 starts after the front-end has been deployed and configured. At this point, the virtual cluster (composed only by the front-end node) becomes totally autonomous and users can submit jobs to the LRMS either from the cluster front-end or from an external node with job submission capabilities (step 5). The user will have the illusion of a cluster where the number of nodes is specified as the maximum size. CLUES monitors the working nodes and intercepts the job submissions as they arrive at the LRMS, thereby allowing the system to dynamically manage the cluster size transparently to the LRMS and the user by scaling up and down on demand. Similar to the deployment of the frontend, CLUES internally employs the IM configured in the front-end during phase 2 to deploy additional VMs for use as working nodes in the cluster (step 6). When these nodes are available, they are integrated automatically into the cluster as new available nodes for the LRMS.

Finally, step 7 represents a hybrid situation where some nodes in the cluster are deployed in another cloud provider to satisfy the requirements of the user. More details of the hybrid cluster configurations are considered in Section 4.3.5.

More details about the three phases of the deployment of the virtual cluster process can be found in A. This appendix contains two additional sequence diagrams that might help to understand the behaviour of EC3.

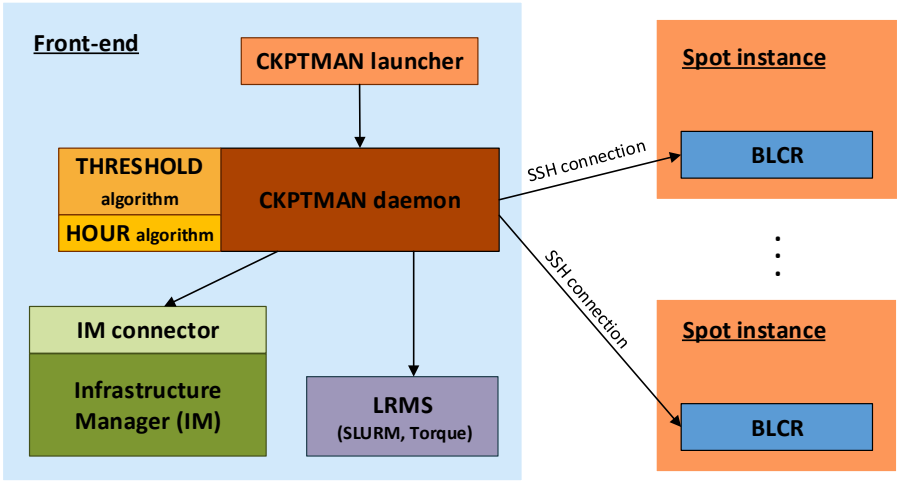


Figure 4.2: Structure of ckptman inside the front-end of the cluster and its relationship with the spot nodes.

4.3.3 Checkpointing Manager (ckptman)

Ckptman is a tool that was developed specifically for EC3, which automates checkpointing of the jobs running on spot instances in order to save as much of the job execution progress (and reduce the costs) as possible. Ckptman was designed to work with Amazon spot instances and the BLCR checkpointing tool within the EC3 cluster deployment tool. It also uses NFS to share the directory where the checkpoint files are saved. Thus, when a spot instance is terminated by Amazon, the checkpoint file is not lost and the job can be restarted in another node. It should be noted that the frontend node of the cluster is never deployed as a spot instance.

The structure of ckptman is illustrated in Fig. 4.2. The tool deployed in the front-end comprises a daemon to check the states of the jobs running in the nodes deployed using spot instances for every preset amount of time. The IM connector determines the nodes deployed using spot instances from the IM. In addition, it obtains the states of the jobs from the LRMS. Depending on the algorithm selected from those described in Section 4.3.4, a decision is made about whether it is necessary to perform a checkpoint or not. Checkpoints are made by invoking BLCR commands using SSH connections to the node where the checkpoint must be made. It should be noted that BLCR supports checkpointing MPI processes, so parallel jobs can also be checkpointed by ckptman.

When a node is destroyed by Amazon due to an out-of-bid situation, ckptman records this fact and enqueues the affected job again, as well as recovering the state of the job from its last checkpoint or from the beginning if no checkpoint file is found for that job. This might cause the deployment of a new node if no idle nodes are available in the cluster. In the next subsection, we described the checkpointing algorithms employed in previous studies and we explain the algorithms implemented in ckptman.

In Appendix B, two flowcharts about the behaviour of ckptman are included. These flowcharts provide more details about the initial creation of the dictionary that contains the relation about the spot nodes and the jobs that are in execution, and the control of checkpointing operations and state of nodes performed by ckptman.

4.3.4 Checkpointing algorithms

Several checkpointing strategies have been developed in previous studies for saving work immediately before an out-of-bid situation occurs, such as [199] and [103]. However, none of them were actually tested in a real environment so the the technologies used to produce the checkpoints were never described. In fact, the results presented in these previous studies were based on simulations. Some of them even considered infinite values for bids [107], which would make checkpointing unnecessary. Therefore, we describe the implementation of a solution and we propose a new algorithm, as well as presenting their assessments where we evaluated their performance in real scenarios. After analyzing the checkpointing strategies described previously based on the performance obtained in simulations, we selected two different checkpointing algorithms for implementation in our tool, as follows.

- **Hourly Checkpointing (`HOUR`):** Checkpoints are made just before the beginning of the next instance hour based on a checkpoint margin time. Amazon does not charge for any partial hours when it terminates the instances due to an out-of-bid situation, so this algorithm, which was proposed by Sangho et al. [199], will save a job process for which the user has already paid. The margin time required to perform a checkpoint can be adapted to the time required to perform the checkpoint for the user's application, which depends on the memory consumption by the application and other factors, such as the MPI communications used.

- **Threshold Checkpointing algorithm (`THRESHOLD`):** Checkpoints are made when there is an increase in the price of the spot instance within an interval. The lower limit of the interval is a fraction of the price determined by the user. This value is recalculated every 10 minutes or when a checkpoint is performed. The upper limit is the bid made by the user. Checkpoints are also made every hour.

The lower limit of the interval is:

$$Threshold = \frac{P_{\overline{x}} + U_{bid}}{2}$$

where $P_{\overline{x}}$ is the average of the prices in the last period of 10 minutes, and U_{bid} represents the bid of the spot price request.

However, it is necessary to avoid overloading the system due to massive amounts of checkpointing operations when fluctuations over the upper limit occur in some periods. Thus, when a checkpoint is made, the lower limit of the threshold is recalculated by:

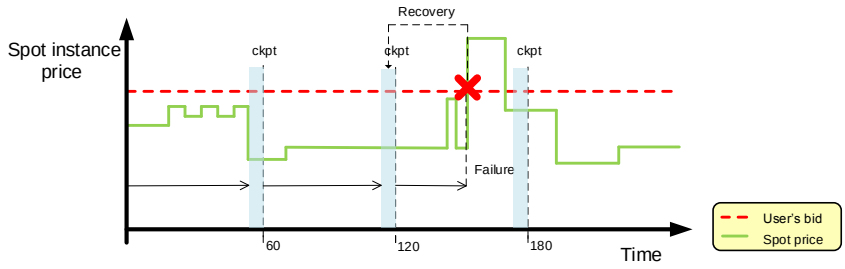
$$Threshold_{ckpt} = \frac{P_{ckpt} + U_{bid}}{2}$$

where P_{ckpt} represents the price that has caused the checkpointing operation. This formula is also used to calculate the initial threshold when the VM starts, where the value of P_{ckpt} represents the price when the instance is deployed.

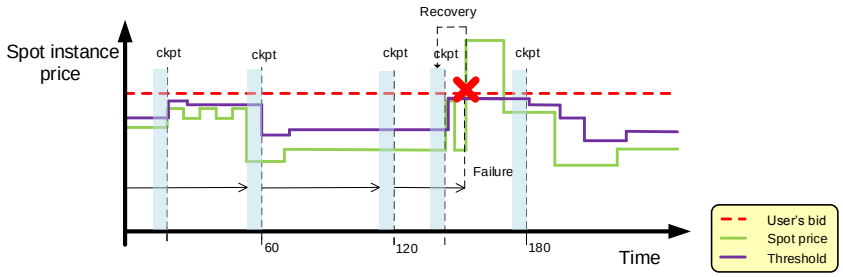
This can be considered as an improvement on the algorithm proposed in [103], for two main reasons. First, the hourly checkpoints can save the job under processing when there is an unexpected peak if the variation in the price is very high. Second, the new method for calculating the threshold better adapts the value to avoid multiple checkpoints during a variable period close to the user's bid. Algorithm 2 represents the threshold algorithm in a more programmatic manner.

Fig. 4.3 shows an example to illustrate the behavior of the algorithms implemented in ckptman with the same variation in spot prices. When an out-of-bid situation occurs, both algorithms can recover the job execution progress, but at different points. The `THRESHOLD` algorithm loses less of the job execution progress than the `HOUR` algorithm, but it also performs more checkpoint operations, although the threshold is recalculated to adapt to the price variations. Additional checkpointing algorithms can also be developed and easily introduced in ckptman[2].

[2]The source-code of ckptman is available at: `https://github.com/grycap/ckptman`



(a) Behavior of the `HOUR` checkpointing algorithm.



(b) Behavior of the `THRESHOLD` checkpointing algorithm.

Figure 4.3: Behaviour of the checkpoint algorithms implemented in ckptman in a simulated scenario.

4.3.5 Hybrid features

One of the objectives of the new processes added to EC3 is to simplify the deployment and management of virtual hybrid clusters when their computational resources are provided simultaneously by on-premises clouds and different public IaaS cloud providers. This is particularly important for the introduction of cloud bursting and it allows users to seamlessly access a larger amount of computing power.

Provisioning nodes across multiple clouds with private IP addresses hinders the connectivity among nodes in different clouds. Therefore, EC3 deploys a VPN server on the front-end, with which the worker nodes connect automatically. If the front-end is behind a firewall that prevents port forwarding, we also allow the deployment of reverse SSH tunnels on the affected nodes.

Algorithm 2 Threshold algorithm

Require: launch time, *launch_time*, of the node; user's bid, *u_bid*; checkpoint time margin, m
  $checkpoint$ = false
  Obtain actual time, *actual_time*
  $life\_time$ = *actual_time* - *launch_time*
  $remaining\_hour\_time$ = 3600 - $life\_time$ % 3600
  {Checkpoint if time is close to the next instance hour}
  if $remaining\_hour\_time < m$ **then**
    $checkpoint$ = true
  end if
  Obtain historic prices in the last 10 minutes from Amazon EC2, p
  $threshold$= $(\bar{p}$ + *u_bid*$)/2$
  {Checkpoint if the most recent price is greater than threshold}
  if $p[0] > threshold$ **then**
    $checkpoint$ = true
  end if
  if $checkpoint$ **then**
    $threshold$= $(p_l$ + *u_bid*$)/2$
  end if
  return $checkpoint$

According to the interconnection strategy selected, EC3 can create four different types of hybrid cluster configurations, as follows:

- **On-premises resources + public resources**: This is the clearest example of cloud bursting. The cluster comprises virtual nodes deployed inside the on-premises cloud of the organization. When the cluster needs to grow and no further resources are available inside the on-premises cloud (due to user quotas or an overloaded infrastructure), the new nodes are deployed in a public cloud. It should be noted that the nodes provided from the public cloud can be on-demand or spot.

- **On-demand resources + spot instances**: The cluster is deployed in a public cloud where nodes can be on-demand or spot. Spot nodes should have checkpointing capabilities in order to save the job execution progress in an out-of-bid situation.

- **Different instance types**: The cluster can comprise nodes with different characteristics (typically CPU and RAM), thereby forming an heterogeneous cluster. For example, in AWS, a cluster can comprise small (1 vCPU and 2 GB of RAM)[3] and medium (2 vCPU and 4 GB of RAM) instances. This

[3]Each vCPU is a hyperthread of an Intel Xeon core.

type of configuration is of special importance for heterogeneous parallel computing. The software configuration can be the same for all of the nodes comprising the cluster, but it is also possible to configure specific software packages for a particular type of node.

- **Different public or private Cloud providers**: the cluster is deployed across different cloud providers, thereby providing resources from multiple clouds.

It should be noted that all possible combinations of the four types described above are also supported by EC3. In particular, HPC applications should be considered when they run in hybrid scenarios. If all of the nodes running a job are not in the same cloud or data center, the performance can be compromised by high latency or low bandwidth when accessing the shared file system and performing explicit communications (e.g., as occurs in MPI applications). We prevent this by creating a nodes partition per cloud in the LRMS. For example, SLURM will enforce the assignment of all the nodes for a job from the same partition. However, we let the LRMS deal with the possible fragmentation of resources, i.e., the case where idle nodes cannot be assigned to a job because they are not in the same partition.

4.4 Case studies

In this section, we present two case studies. In the first presented in Section 4.4.1, we describe an assessment of the effectiveness of the tool based on a computationally intensive scientific application for the structural dynamic analysis of buildings. The results demonstrate the feasibility and benefits of this type of cluster for computationally intensive applications. Second, in Section 4.4.3, we present evaluations of the checkpointing algorithms in a real environment with real workloads to study their effectiveness.

4.4.1 Structural Dynamic Analysis of Buildings using EC3

In order to assess the effectiveness of a self-managed cost-efficient virtual hybrid elastic cluster on a cloud infrastructure, we present a case study based on the structural dynamic analysis of buildings. The structural dynamic analysis of buildings is required to accurately simulate how a building will be affected by external dynamic loads, such as an earthquake. This is a computationally intensive task and it can be tackled in an efficient manner by parallel computing on clusters of PCs. The simulations were executed using the structural simulator component of the Architrave software [160], which comprises an MPI-based HPC batch application that benefits from on-demand clusters via the cloud in order to provide an online service for structural analysis by a community of users. This structural simulator has some library dependencies, such as PETSc and SLEPc.

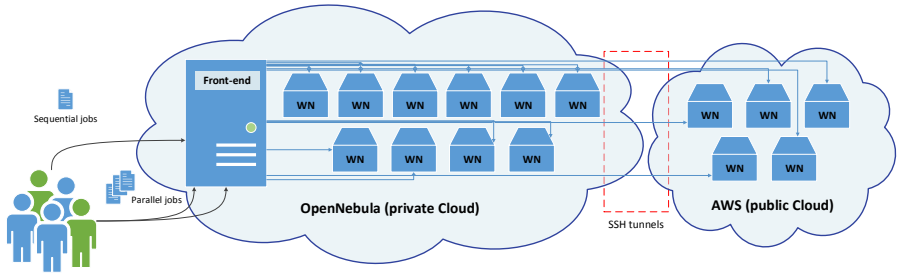


Figure 4.4: Scenario of the first case study: A hybrid virtual elastic cluster across an on-premises platform (OpenNebula) and public Cloud (AWS).

This case study involved a virtual infrastructure designed specifically for the structural dynamic analysis of buildings, where multiple users could access and execute their jobs. A simplified representation of the scenario is shown in Figure 4.4. We performed two different executions with two distinct cluster configurations: scenario (a) a hybrid infrastructure that comprised nodes from our on-premises cloud and public nodes from AWS, where we only considered on-demand instances; and scenario (b) a hybrid infrastructure that comprised nodes from our on-premises cloud and public nodes from Amazon EC2 using spot instances.

The job pattern submission used in both cases is represented in Figure 4.5, which we designed specifically to test the infrastructure elasticity and its ability to create hybrid clusters. The jobs were sequential (HTC) or parallel (HPC), so they could need more than one node, as well as having different duration, and thus we considered the job heterogeneity that the physical clusters had to handle. In particular, this case study comprised four types of jobs: (i) sequential jobs with an approximate duration of one hour in a single node, (ii) parallel jobs with an approximate duration of one hour in two nodes, (iii) sequential jobs with an approximate duration of two hours in a single node, and (iv) parallel jobs with an approximate duration of two hours in two nodes. As mentioned earlier, jobs that required more than one node were all performed by nodes in the same cloud using SLURM partitions that represented each cloud, which ensured high performance and low latency communication among these nodes. In total, 124 jobs were performed during each execution, i.e., 83 sequential jobs and 41 parallel jobs. The infrastructure used to deploy the on-premises cloud comprised eight dual processors with 14 core nodes (28 cores per node), with 64 GB of RAM and a shared storage system of 10 TB, which was backed up by a storage area network where the hard disks were stored as volumes. This system was managed by OpenNebula 4.8 using KVM as the underlying hypervisor. The public cloud was AWS. The instance type selected was `m1.medium`, with one (virtual) processor, 3.75 GB of RAM, and 410 GB of disk space. The VMI used in the on-premises cloud provided

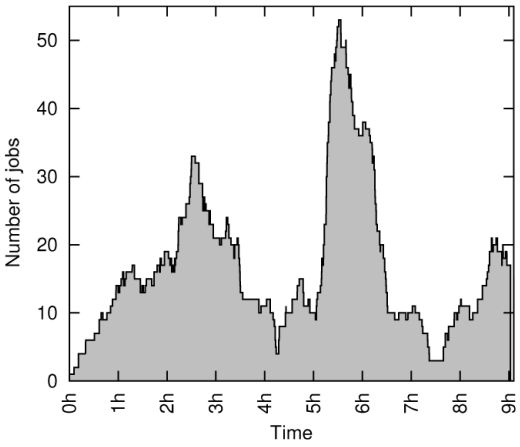


Figure 4.5: Job pattern submission during the case study. In total, 124 jobs were executed.

the same execution environment as EC2 AMI, i.e., an Ubuntu 12.04 LTS preconfigured with BLCR and SLURM. Thus, all of the VMs deployed on both clouds had the same characteristics. We selected a preconfigured VMI to accelerate the deployment time for the working nodes, as explained later, but it would also be possible to configure all of this software on-demand. For both executions, we used SSH tunnels to interconnect the nodes.

Moreover, we fixed a limit of 50 nodes (plus the front-end) as the size of the cluster. A maximum of 35 of these nodes could come from the on-premises cloud and the remainder (15 nodes) were from the public cloud.

4.4.2 Results and Discussion

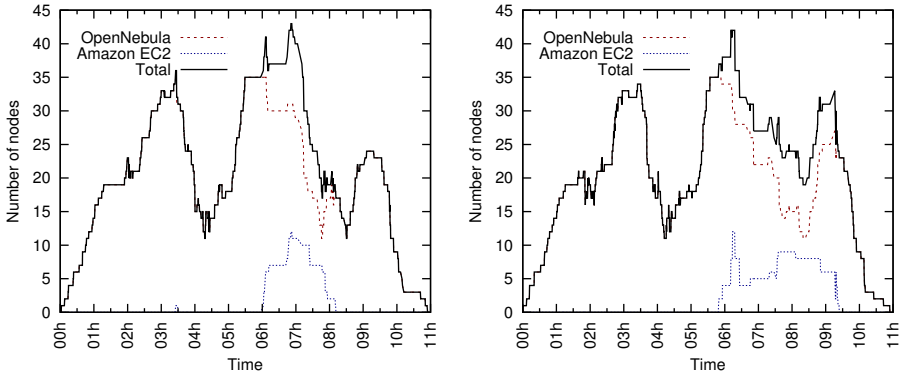
First, we analyzed the time differences in the deployment and contextualization processes for the types of VMs used in our case study. Table 4.1 shows the average times obtained for each step with VMs deployed in our on-premises cloud (second column), on-demand instances deployed in Amazon EC2 (third column), and spot instances from EC2 (fourth column). The results show that the deployment time using spot instances was considerably higher than that with on-demand instances because of the time required by AWS to complete the spot request (335 seconds in average). In addition, there were differences in the contextualization process according to the use of preconfigured VMIs (with only BLCR and SLURM preinstalled). Due to these differences, we decided to use a preconfigured VMI in the subsequent executions. It was still necessary to configure the SLURM configuration files, SSH tunnels, NFS system, and the application dependencies. However, the time consumption during the contextualization process was reduced significantly

	OpenNebula instance	EC2 On-demand instance	EC2 Spot instance
Deployment	40	45	335
Non-preconfigured VMI:			
Contextualization	661	698	702
Total average time	701	743	1037
Preconfigured VMI:			
Contextualization	196	232	337
Total average time	236	277	672

Table 4.1: Average deployment and contextualization times (in seconds) for nodes deployed in different clouds.

by starting from a VMI with BLCR and SLURM previously installed. Second, we present the results obtained when executing scenario (*a*) and scenario (*b*) using the job pattern submission in Fig. 4.5. For both executions, we used conservative elasticity policies to ensure the minimum costs for the infrastructure in terms of energy consumption (for the onpremises cloud) and budget consumption (for the public cloud). Thus, the selected *scale out* policy only deployed a new node (or two nodes if the job required them, such as parallel jobs in our case study) if there were no idle nodes in the virtual cluster. EC3 deployed nodes in the on-premises cloud until it reached the limit (35 nodes). If more nodes were required, they were deployed in the public cloud. The *scale in* policy removed a node from the infrastructure when it was idle for 60 seconds and no more jobs were queued in the LRMS. It is possible to select different scale in and out policies other than those provided in EC3, such as group-based starting of nodes or time blocks to destroy them; however, in this case study, we focused on the hybrid features and the cost savings obtained by using the spot instances and the checkpointing techniques. Further details of the elasticity policies can be found in [34].

During the first execution, as shown in Fig. 4.6(a), a hybrid infrastructure comprising nodes from our on-premises cloud (OpenNebula) and on-demand public nodes from AWS handled the execution of jobs. Three scale-out periods and three scalein periods could be observed. An accurate analysis of the results identified two overloaded periods in the on-premises cloud. The first occurred at 3 hours 20 minutes when the on-premises cloud had 35 nodes deployed and there was at least one job pending in the queue. EC3 decided to deploy a new node in the public cloud, but we can see from the graph that before the node deployed in EC2 was ready, a node in our private cloud completed the execution of its jobs and the pending job in the queue was automatically submitted to this node by the LRMS. EC3 did not receive new jobs during this period, so it terminated the instance provided from AWS.



(a) Hybrid infrastructure using on-demand instances.

(b) Hybrid infrastructure using Spot instances.

Figure 4.6: Behaviour of the Virtual Hybrid Elastic Cluster in both executions.

The second important feature in the graph started at 5 hours 30 minutes when the frequency of job submissions was less than one minute, thereby causing saturation of the infrastructure. All 35 of the nodes deployed in OpenNebula were executing jobs, so EC3 started to provide new nodes in AWS to execute the new jobs. At 6 hours, the working nodes deployed in OpenNebula finalized their execution and started to receive new jobs, thereby helping to overcome the saturation situation. It should be noted that the deployment and contextualization of new working nodes was conducted in parallel.

Figure 4.6(b) shows the results of the second execution, where EC3 configured a hybrid infrastructure that comprised nodes from our on-premises cloud and nodes from AWS using spot instances. The checkpoint algorithm selected for execution was *threshold*. We considered a basic opportunistic approach by setting the maximum bid price for the spot requests to the on-demand price for the same type of instance as 0.087$. Using this strategy, we ensured that a spot instance cost no more than the price for an on-demand instance because AWS guarantees that you will never pay more than your maximum bid price per hour. It was also possible that we might pay less per hour than our maximum bid price due to periodical adjustments in the price by AWS, where everyone pays the same spot price for a period regardless of whether their maximum bid price was higher.

As shown on Table 4.1, working nodes deployed with spot instances require more time before they are ready to execute jobs. This may explain the main differences observed between Fig. 4.6(a) and 4.6(b). In total, 29 checkpoints with an average duration of 160 seconds were completed during execution. This example shows that the 2-minute warning added recently by Amazon to notify the impending

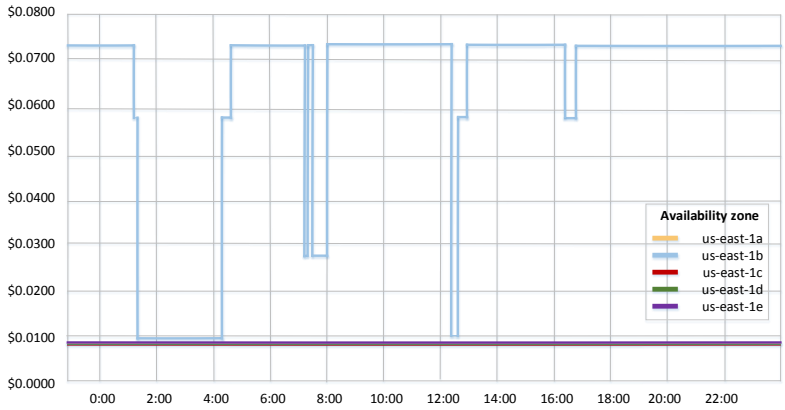


Figure 4.7: Spot price history corresponding to 8th April 2015, when the case study was executed, for *m1.medium* instance type.

termination of the spot instance is not sufficient. It should be noted that the IM selects the cheapest available zone for the spot instances request, which was *us-east-1e* in this case study. As shown in Fig. 4.7, this region was not affected by price variations during execution, so the checkpoints occurred hourly.

	On-demand execution	Spot execution
Seq. jobs avg. waiting time	7min 14s (434s)	10min 9s (609s)
Par. jobs avg. waiting time	14min 43s (883s)	17min 56s (1076s)
Total time of execution	10h 47min 12s (38832s)	10h 56min 21s (39381s)
Total cost of execution	2.349$	0.234$

Table 4.2: Details of time and price about the executions.

Table 4.2 shows the times and costs required for both executions, which indicate that the total time for execution was similar in both examples, but the cost was lower when we used spot instances, as expected. The actual on-demand price for a *m1.medium* instance was 0.087$, whereas the spot price for the same instance during our execution was 0.0081$. Thus, the total cost of execution using spot instances was 10% of the execution cost with on-demand instances. In addition, the differences in the job waiting time in the queue between sequential and parallel jobs was caused by the scheduling policy selected. When a new job arrives in the queue and requests two nodes (or more), CLUES automatically deploys the number of nodes requested by the job via the IM (if there are no available nodes in the cluster). If a second job arrives in the LRMS queue during the deployment process and it requests less nodes than the first job, its execution will probably start earlier than that of the first job. This is because the deployment and contextualization

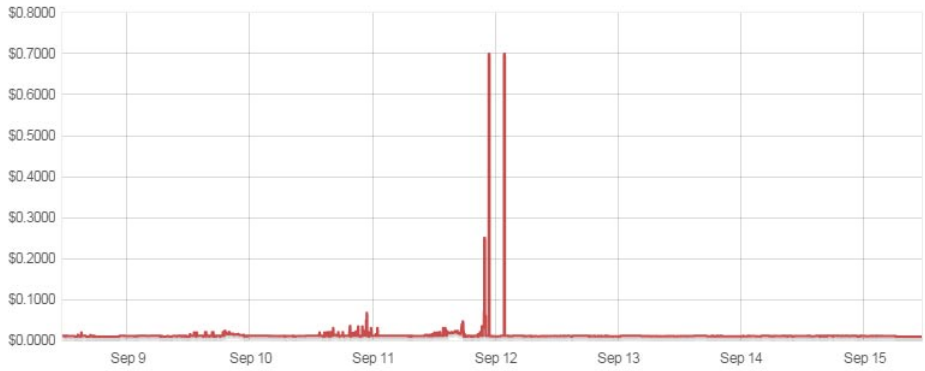


Figure 4.8: Spot price history corresponding to the period from 9th September 2015 to 15th September 2015, for *m3.medium* instance type in the availability zone *us-east-1c*. The OS is *Linux/UNIX (Amazon VPC)*.

processes for the new nodes are not finalized at the same time, and there is usually a difference of a few seconds between nodes launched at the same time, but the difference is sufficient to detect an active node before the others. Therefore, the LRMS detects an active node and sends it the second job. CLUES can also be configured to behave in a different way by waiting for the second node to become active without starting a new job on the first node.

4.4.3 Analysis of the checkpointing algorithms

In this subsection, we provide a detailed analysis of the effectiveness of the two checkpoint algorithms, which determine the best moment to make a checkpoint for an application running on a spot instance. We based our analysis on real workloads obtained from the Grid Workloads Archive [96], with the aim of testing the proposed algorithms in a real environment, thereby differentiating our research from most previous studies, which are typically based on simulations.

In order to rigorously compare the behavior and performance of the two checkpointing algorithms proposed in the present study (`HOUR` and `THRESHOLD`), we performed three different executions in a spot environment. The first did not use checkpointing techniques (`NONE`). The second used the `HOUR` algorithm and the third used the `THRESHOLD` algorithm. Thus, we had to use the same price variations for all of the executions in order to compare the results in an appropriate manner. Therefore, we selected a specific fragment of the spot price history provided by Amazon EC2, as shown in Fig. 4.8. The period used in the case study was September 9–14, 2015 for the *m3.medium* instance type in the availability zone *us-east-1c*. The user bid was 0.067$ and the on-demand price for this instance

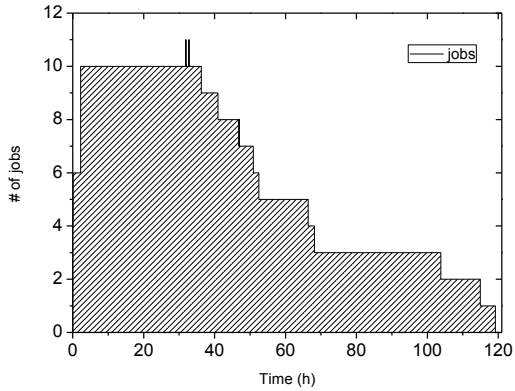


Figure 4.9: Workload of the case study, extracted from the *GWA-T-3 NorduGrid* dataset.

type employed the same opportunistic approach as the first case study. Finally, for comparative purposes, the fourth execution was an on-demand execution using the same type of instance, i.e., *m3.medium*.

The objective of this case study was to analyze the efficiency of the checkpointing algorithms under a spot scenario, so the configuration of the cluster had an on-demand *m3.large* frontend with spot *m3.medium* nodes, all of which were deployed in AWS. The maximum size of the cluster was fixed to 12 nodes according to the workload used. This real workload was a fragment extracted from the *GWA-T-3 NorduGrid* dataset provided by the Grid Workloads Archive (lines 4–16 in the *.gwf* file), as shown in Fig. 4.9 in terms of the size evolution of the cluster. In this dataset, the jobs were executed in a sequential manner [19]. Further parameters and values for this case study are presented in Table 4.3.

Task avg. time	191407s
Node deployment avg. time	672s
Checkpoint avg. time	160s
Recovery avg. time	[74, 746]s
User bid	0.067€
ckptman revalue time	30s

Table 4.3: Parameters and values for the second case study.

The recovery time of jobs depends on whether there is an available node for executing the re-submission of the job or not. If a node exists, an average of 74 seconds is required to recover a job from its checkpoint. However, if a node does not exist, EC3 needs to deploy a new node to execute the job, where time is required to

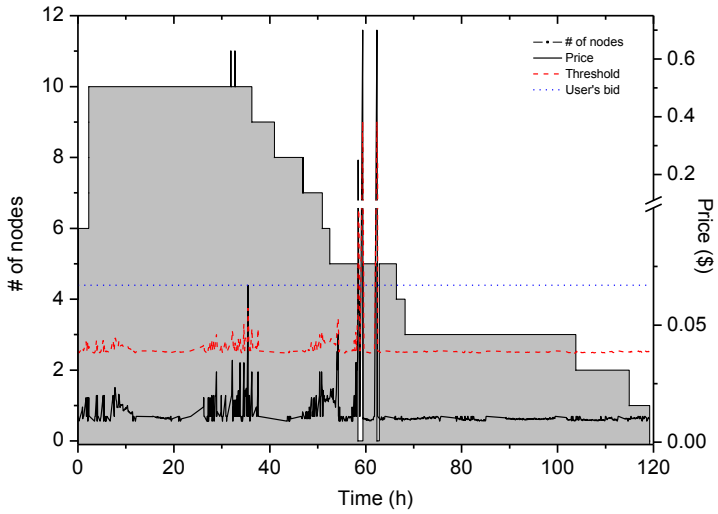


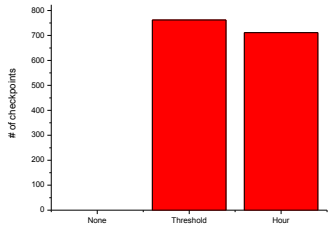
Figure 4.10: Threshold evolution of the `THRESHOLD` algorithm vs price history of *m3.medium* instance type. Also the evolution of the cluster in terms of nodes is represented.

deploy and configure this new node in addition to the detection and re-submission of the job (average of 746 seconds). This will occur when there is a large increase in the spot price that exceeds the user's bid because all of the nodes will be killed by Amazon in this out-of-bid situation. The *ckptman* revalue time shown in Table 4.3 is a configurable parameter, which indicates how often the jobs and nodes are reevaluated by *ckptman* to determine whether it is necessary to make a checkpoint.

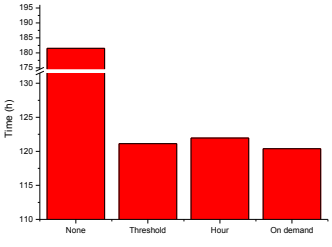
4.4.4 Results and Discussion

In this subsection, we analyze and discuss the results obtained from the four executions described above. First, Fig. 4.10 represents the threshold evolution of the `THRESHOLD` algorithm and the price history of the *m3.medium* instance type during the `THRESHOLD` execution. The evolution of the size of the cluster in terms of the nodes is also shown in this figure. Using this graph, we can obtain a general view of the scenario in the case study, which is applicable to the three spot executions performed. First, we can see that three important increases in the spot price destroyed the working nodes in the cluster (between 57 and 63 hours) due to an out-of-bid situation. Five jobs were affected and they had to be restarted. Second, for the `THRESHOLD` execution, we can see that the value of the threshold adapted continuously to the evolution of the price. Thus, a checkpoint was made due to the price increase at 36 hours. However, the increases during hours 57

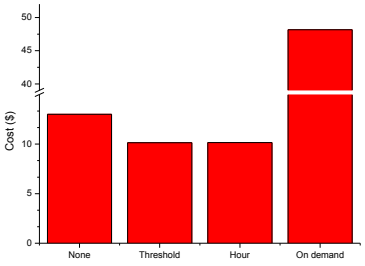
and 63 were so high that the virtual machines were killed before the algorithm could make checkpoints corresponding to the increases. However, the THRESHOLD algorithm also performed checkpoints every hour to avoid losing the job execution progress.



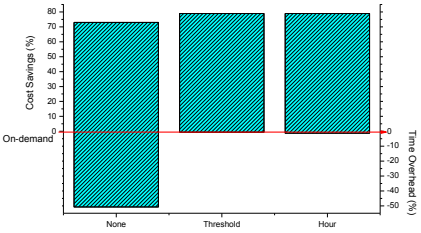
(a) Number of checkpoints per algorithm.



(b) Total execution time.



(c) Total cost of execution.



(d) Percentage cost savings and time overheads with respect to an on-demand execution.

Figure 4.11: Performance evaluation for the three checkpointing strategies.

To better analyze the four executions performed, Fig. 4.11 shows several graphs that compare the number of checkpoints, total time, and total costs. According to Fig. 4.11(a), we can see the number of checkpoints performed by our algorithms during execution. There was a difference of 7% between the HOUR and THRESHOLD algorithms. Obviously, NONE and on-demand executions did not make checkpoints. Fig. 4.11(b) shows the total execution time for the four executions. As expected, on-demand execution was faster than spot execution. In addition, the differences were very high with or without checkpoints during spot execution. Indeed, NONE execution had 50% higher time overheads compared with ondemand execution. By contrast, HOUR and THRESHOLD executions only had overheads of 1.3% and 0.63% with on-demand execution, respectively. The next graph shows the total cost of the executions represented in Fig. 4.11(c). The highest costs were with on-demand execution (48.17$), as expected. The second most expensive was NONE

execution (13.02$), while the HOUR and THRESHOLD executions had similar costs (10.16$ and 10.15$, respectively). Finally, Fig. 4.11(d) shows the performance of the algorithms in terms of the cost and time for on-demand execution. The time overheads represent the difference between the time required for on-demand execution and spot execution. The best results were obtained by the THRESHOLD algorithm, with savings of 78.94% and time overheads of 0.63% compared with on-demand execution. However, the HOUR algorithm also obtained good results, with savings of 78.92% and time overheads of 1.3%.

Based on this analysis of the execution data, we can make two important conclusions. First, we demonstrated the importance of using checkpointing techniques during spot execution. Spot execution can considerably reduce the costs but it can also significantly increase the total time required for execution. This increase can be reduced by using checkpointing techniques, as shown in other studies, such as [199] and [107]. Second, after comparing both of the algorithms proposed in this study (HOUR and THRESHOLD), we conclude that the THRESHOLD algorithm can reduce the costs and time overheads better, but its performance is strongly related to variations in the price history. Thus, if an increase that causes an out-of-bid situation occurs without a predecessor value within the THRESHOLD interval leading to an out-of-bid situation, then the algorithm cannot predict this increase. In this case, saving the job execution progress relies on the checkpoint performed within the last hour, and thus the differences between the two algorithms were not very high in our case study. However, the THRESHOLD algorithm is expected to perform better when the spot price history exhibits continual fluctuations before the out-of-bid situation. In this type of scenario, the use of the THRESHOLD algorithm is recommended. Furthermore, the HOUR algorithm is recommended for situations where the spot price remains steady until a rapid increase occurs in the spot price, thereby leading to an out-of-bid situation. In addition, given that partial hours are not charged for terminated spot instances, then the use of the HOUR algorithm is further recommended in this type of situation. This conclusion supports those given in [199].

4.5 Conclusions and future work

In this study, we described the further development of EC3 as a tool that produces self-managed cost-efficient virtual hybrid elastic clusters from the computational resources provided by multiple IaaS clouds. In particular, we focused on two topics: (i) hybrid clusters across on-premises and public clouds; and (ii) using the spot instances provide by AWS to achieve reliable low cost cluster-based cloud computing. We performed a case study based on a scientific application for the nonlinear dynamic analysis of buildings, which we executed using a hybrid virtual elastic cluster across an on-premises OpenNebula cloud and AWS. We also

assessed the threshold algorithm and other previously proposed algorithms using real workloads and real spot prices.

Our results demonstrated the ability of the clusters to adapt their size to the workload, as well as automatic cloud bursting to a public cloud, and significant savings due to the use of spot instances compared with on-demand instances, where the increased resilience was attributable to performing periodic and automatic checkpointing for the jobs. EC3 is open-source based on the Apache 2.0 License and hosted on GitHub[4], and a web application is also available for free (Elastic Virtual Clusters as a Service) to the community[5].

In future research, we aim to add migration capabilities to EC3. We plan to enable the migration of virtual clusters across cloud platforms, thereby introducing an unprecedented degree of flexibility for data centers, especially during planned outages where computing power can be outsourced temporarily to a public cloud. In addition, we may address the checkpointing of applications that also require the restoration of the file system's content at the time of checkpointing. We also intend to adapt the algorithms to other public cloud providers with similar features, which is the case for the preemptible VM instances provided by the Google Cloud Platform.

[4]EC3 at GitHub: `https://github.com/grycap/ec3`
[5]EC3 web GUI, available at `http://www.grycap.upv.es/ec3`

Chapter 5

eScience with a Galaxy web-service connected to an Elastic Cluster in the Cloud for Computational Biodiversity

Submitted as

M. Caballer, A. Calatrava, I. Blanquer, P. Chaumeil, A. Frank and JM. Frigerio. "eScience with a Galaxy web-service connected to an elastic cluster in the cloud for computational biodiversity." in Journal of Grid Computing, to appear in Special Issue on Science Gateways, 2016.

Abstract

As many other scientific challenges, the analysis of the impact of climate change in biodiversity requires assembling big databases and integrating tools for processing and modelling to analyse data and derived results. In this paper, we address the development of an e-Science service that permits an user to select the datasets he/she wishes to study, as well as a set of tools for running the study, leveraging elastic cloud computing capabilities. In our case, we provide a service that exposes a user-friendly Galaxy interface that submits jobs on an elastic queue based on a virtualised cluster that automatically deploys and undeploys resources as needed, improving current cloud-computing capabilities of Galaxy. Moreover, the approach uses a set of services for the automatic configuration and contextualisation of the resources, which make the virtual infrastructure platform-agnostic, so it can be deployed in different IaaS cloud providers (both public and on-premise clouds). An experiment of a workflow on taxonomy for computing the dif-

ferences or dissimilarities between molecular markers associated to specimen has been successfully tested.

5.1 Introduction

The short-term and long-term impact of climate changes upon biodiversity is now acknowledged [24]. It has significant consequences on environment quality, human health, and global food supply. Describing biodiversity patterns and modelling the processes shaping them is a key activity for mitigating these effects. This challenge requires assembling big databases on biodiversity observations and environmental measures, as well as the integration of a diversity of tools for data post-processing and analysis, data mining, machine learning and modelling to analyse data and derived results.

Cloud infrastructures can be used to address the computational needs in multiple domains, including scientific applications. Cloud computing has proven to fit part of the challenges posed by scientific research [197]. Public cloud infrastructures have intensively focused on supporting science, through the availability of large data sets ([15], etc.), the offering of customised VMIs for scientific computing ([79, 50] or the development of scientific support programmes ([130]). Research community clouds (such as EGI Federated Cloud [70]) provide the scientific community with both resources and customised Virtual Appliances (VAs) for biocomputing. However, the use of public or on-premises IaaS clouds require non-trivial system administration skills. If a single VM is needed, custom configurations available in public clouds may be sufficient to meet users' requirements. Scientific users can easily deploy those single instances and use them in a pay-per-use on-demand fashion. Despite more complex deployments are possible (virtual clusters of AWS, Data analytics based on Hadoop) [165], customisation is cumbersome. When multiple VMs are needed, users have to configure shared directories, install batch queue systems, manage user accounts and customise software.

The challenge we address in this study is to provide an e-Science service that enables a user to select the datasets he/she wishes to study, as well as a set of tools for running the study, leveraging elastic cloud computing capabilities without caring about the underlying infrastructure. Currently, most of data sets related to biodiversity are based on the output of next-generation sequencing, which delivers huge datasets of several GB. In the last years, many organisations started offering services to the community in similar Biocomputing disciplines, since these analysis can no longer be performed in a reasonable time on researchers' own computers [128]. We aim in this paper to assist the long-tail of science users who have not access to an intensive computational facility and who had a reduced budget for computation.

The work in this paper discuss the implementation of a service that exposes a user-friendly Galaxy interface[1] [81, 82] for uploading data, selecting analysis tool, defining options and parameters and submitting the execution of jobs on an elastic queue based. This elastic queue is based on a virtualised cluster that automatically deploys and undeploys resources as needed, improving current cloud-computing capabilities of Galaxy. In this multi-tenancy scenario, the capability of adjusting the computing capacity to the actual workload will reduce costs (by switching off unused resources accordingly) without a penalty on the quality of service. The approach uses a set of services for the automatic configuration and contextualisation of the resources, which make the virtual infrastructure platform-agnostic, so it can be deployed in multiple IaaS cloud providers (both public and on-premise clouds).

The article describes an experiment with a workflow with an input dataset of 10K reads of shorts sequences, from a data set of 300K sequences, where pairwise distances are computed through an exact algorithm of sequences comparisons by local alignment, where a nested aggregative clustering is built from the pairwise distance matrix. This provides an insight on diversity structure of dataset. Intensive part of the computation is on computation of the distance matrix, and it has been parallelised on the elastic cluster. As the feasibility of such a workflow is demonstrated by this running example, we will pursue in both direction of diversification of tools on one hand, and scaling up the size of data sets on the other. The experiment defines an irregular job submission pattern of 75 jobs of different sizes and analyses the behaviour of the virtual cluster.

The rest of the paper is structured as follows. First, section 5.2 discusses other approaches for science gateways for computational biodiversity and tools to deploy virtual clusters on Cloud infrastructures. Then, section 5.3 analyses the solution proposed in this paper, describing the architecture to automatically deploy the Galaxy software tool over a virtual elastic cluster. Later, section 5.4 focuses on a use case for biological diversity, from a set of molecular markers, associating genetic and inter-specific diversity over a Galaxy virtual elastic cluster deployed with our proposed architecture. A discussion of the results is also presented in this section. Finally, we present concluding remarks in section 5.5.

5.2 Related work

The main aim of the work has been to provide a way to deploy a well-stablished scientific gateway such as Galaxy on an elastic cloud IaaS, covering the complete configuration and customisation of the environment and including the installation of new software and interface modules for Galaxy. Therefore, we will first analyse

[1] `http://galaxyproject.org`

the tools available for Virtual Cluster Deployment and then the scientific gateways on the field.

There are previous works in the literature that aim at deploying virtual clusters on Cloud infrastructures. For example, StarCluster [134] with the Elastic Load Balancer [135] plug-in enables the provision of clusters in Amazon EC2 from a predefined configuration of applications (Open Grid Scheduler, OpenMPI, NFS, etc.). The caveat of this plugin is that it requires the StarCluster User Interface (UI) to be connected to the Cloud infrastructure, in order to create and destroy the VMs. ViteraaS [62] allows the creation of virtual clusters to manage the execution of user-defined jobs. The main problem is that ViteraaS does not allow the user to remotely access the cluster. Instead, ViteraaS is only devoted to execute jobs as done in classic Grid approaches without providing access to a cluster.

There are also commercial solutions, like IBM Platform Dynamic Cluster [94], that aims at partitioning on-premises resources to deliver each user a custom cluster with specific features. It supports live job migration and automated checkpoint restart. The drawback in this case is that this product is oriented to manage on-premises infrastructures and cannot be connected to commercial Cloud providers. In this way, CycleCloud [56] is a commercial service provided by CycleComputing that deploys virtual clusters. However, it only works on Amazon EC2 and, therefore, virtual clusters cannot be deployed on on-premises Cloud platforms created with Cloud Management Platforms such as OpenNebula or OpenStack.

The summary of these generic tools is that there is no general framework that enables the creation and management of elastic clusters in general IaaS deployments. Most of them provide a virtual cluster that comprises a fixed number of nodes, other solutions are oriented to a specific LRMS or they 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, etc.).

There are other solutions that provide web portals, initially created to launch grid jobs that has evolved to access Cloud resources as Distributed Infrastructure with Remote Agent Control (DIRAC) [75, 84], Catania Science Gateway Framework (CSGF) [20] or the Italian Grid Infrastructure (IGI) Web portal [26]. In [26] the authors present a portal to launch jobs to Grid and Cloud resources. The Cloud port-let used in this work enables to access OpenNebula, OpenStack and any OCCI compliant Cloud provider. DIRAC is a software framework for distributed computing providing a complete solution to user communities requiring access to distributed resources. It allows to aggregate computing resources of different source and nature, such as computational grids, clouds or clusters, transparently for the end users. Through the some extensions it supports any OCCI compliant Cloud provider or Amazon EC2. The CSGF allows users to execute applications on the EGI Federated Cloud [70] through web portals. All these works do not enable the contextualisation the VMs so all the needed software must be pre-configured

to execute a specific set of jobs. Furthermore they have a reduced list of supported Cloud providers.

The TeraGrid Science Gateway program [188] offered solutions well suited for the integration of research computing facilities with user-friendly interfaces. Several solutions have been developed in the frame of biodiversity and evolutionary genetics, such as CyberInfrastructure for Phylogenetic RESearch (CIPRES) [131]. However, those solutions directly connect to the infrastructures through the meta scheduling services that enable them to submit jobs. This prevents users to deploy legacy solutions as they were running on their own resources. IaaS clouds have been used to provision and deploy such systems in those conditions. However, multi-tenancy of the cloud IaaS (and therefore the fair sharing or effective costs) depends on releasing resources when they become idle.

In this paper we focus on deploying an existing scientific gateway (Galaxy) on an IaaS, enhancing this gateway with self-adaptive scaling of resources. There are existing solutions for deploying Galaxy on the cloud ([77, 4]). In this approach, AWS EC2 AMIs are available with a standard full configuration of the Galaxy portal so they can be easily deployed on AWS cloud. Galaxy can deploy a cluster of Galaxy instances on AWS EC2 using the user's credentials and configure them. A shared directory and a batch queue are created and configured by means of CloudMan. It can be used also to deploy similar clusters on OpenNebula or OpenStack. However, this approach lacks from two main capabilities. First, it needs pre-configured VMIs, which should be properly maintained, and any extra configuration (e.g. own Galaxy modules or data) should be copied in the images or manually configured later. Elasticity is limited, enabling the user to manually trigger the boot or shutdown of a Galaxy node. Other e-Infrastructures, such as EGI Federated Cloud ([70]) expose an OCCI ([129]) standard interface, which is not supported by CloudMan. The use of pre-configured VMIs carries the burden of maintaining them, and the manual scalability cannot be applied in scenarios where the workload is dynamic and variable.

Globus Genomics has also integrated Galaxy Workflows ([118]) with their authentication, queue scaling and data transfer systems. The use of Globus Online offers an efficient solution for geographically remote connections to filesystems. The limitation of the use of NFS is then reduced. The scaling capabilities are bound to the queue length and job waiting time, acting over the available queue resources. The solution of Globus Genomic is pertinent for their use within Amazon AWS, but it could not be applied to on-premise clouds or other federated clouds as contextualisation depends on the VM image and the scaling uses EC2-specific primitives. In our case, the use of Ansible ([90]) and the capability of being platform-agnostic of EC3 ([34, 41]) enables deploying the same infrastructure on different resources and flavours. Moreover, the transition to container-based working nodes is straightforward as it is also supported by EC3.

Therefore, we propose in this work to use EC3, a platform-agnostic, self-scalable framework that eases the deployment of elastic virtual clusters in any computing platform. This framework makes use of installation recipes, a description of the virtual hardware, a service that monitors the length of the batch queue and a set of plug-ins to deal with multiple cloud IaaS back-ends. Furthermore it provides an easy-to-use web interface that enables non-advanced users to launch a virtual cluster using a wizard following a set of simple guided steps.

5.3 Galaxy Virtual Elastic Cluster

In this paper we provide a distributed architecture that integrates an elastic cluster providing virtual machines on-demand for computationally intensive methods. The deployment of the virtual machines, as well as the execution of the processing is triggered from a user-friendly interface on a Galaxy server.

The solution proposed is based on the EC3 tool. EC3 is a tool to create elastic virtual clusters on top of IaaS clouds, both public or on-premise ones. EC3 integrates with the IM [8, 33] (a tool that eases the access to IaaS clouds) and CLUES [9] (an energy-aware cluster management system). It provides recipes to deploy Torque (optionally with Maui), SLURM, SGE, HTCondor and Mesos clusters that can be self-managed with CLUES. It also provides a set of recipes to install other tools as NFS, Octave, etc. In the context of this paper, we developed two recipes for Galaxy that enable launching a Galaxy elastic cluster with a set of user defined tools integrated (Next Generation Sequencing (NGS) alignment and the biodiversity analysis tools, as well as the reference data). It is important to outline that the recipes include the customisation of the portal interface on the fly.

EC3 starts with a single front-end node, and working nodes will be dynamically deployed and provisioned to fit increasing load (number of jobs at the LRMS). Working nodes will be powered off when they are idle. This introduces a cost-efficient approach for Cluster-based computing.

5.3.1 Architecture

The architecture is depicted in Figure 5.1. In the first step (1) the user requests a cluster indicating the software dependencies needed for his application, that will be automatically installed and integrated by the platform. The user can select either the CLI tool[2] or the Web interface[3] offered by EC3 to request the cluster deployment. In the Web application (Figure 5.2), the user has a user-friendly wizard application that guides the user through a set of steps to customise and launch the virtual elastic cluster. First, it is necessary to choose the cloud provider in which

[2] `https://github.com/grycap/ec3`
[3] `http://www.grycap.upv.es/ec3`

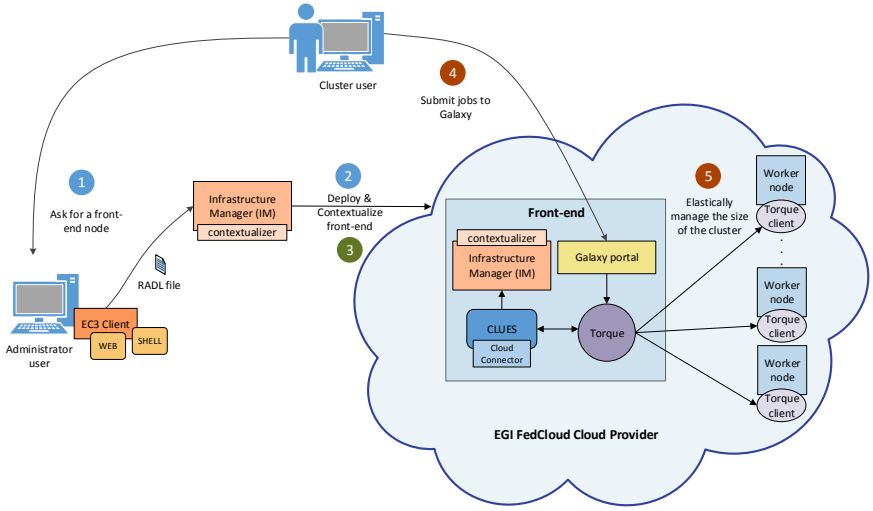


Figure 5.1: System Architecture.

the cluster will be deployed. Currently, the web interface supports deployments in Amazon EC2, OpenNebula, OpenStack and EGI FedCloud, but more providers are supported by the IM, which can be selected from the CLI interface. The wizard application shows the specific fields for each Cloud provider, where the user can specify the LRMS system to use (Torque, SGE, SLURM, HTCondor or Mesos) and a set of tools that will automatically be installed in the cluster (in our case, we have to select Galaxy). Finally, the user has to specify the maximum number of nodes the cluster can reach and push the *submit* button to launch it. A RADL document [8, 33] with all the details of the required virtual cluster (hardware, software and configuration steps) is automatically generated by the web service and it is sent to the IM service. It will be in charge of deploying the front-end VM and configure it according to the RADL info (steps 2 and 3). This process can take between 10-20 min, depending on the complexity of the applications to install. The CLI interface requires the RADL document that describes the cluster. For that, a set of predefined RADL recipes is distributed together with the EC3 code, that facilitates this task, but this option is only recommended for experienced users.

Once the front-end of the cluster is deployed and fully configured, the user has a fully working instance of Galaxy with the previoulsy selected tools automatically configured. As a result of this process, the user is provided with the front-end IP of the cluster, where he/she can now connect via SSH or, in our case, via a web client to the Galaxy portal port (8080). Now, the cluster is ready to compute the tasks of the user (step 4). When any of the Galaxy tools are used, a job is submitted to the underlying Torque LRMS subsystem. When a job arrives to the

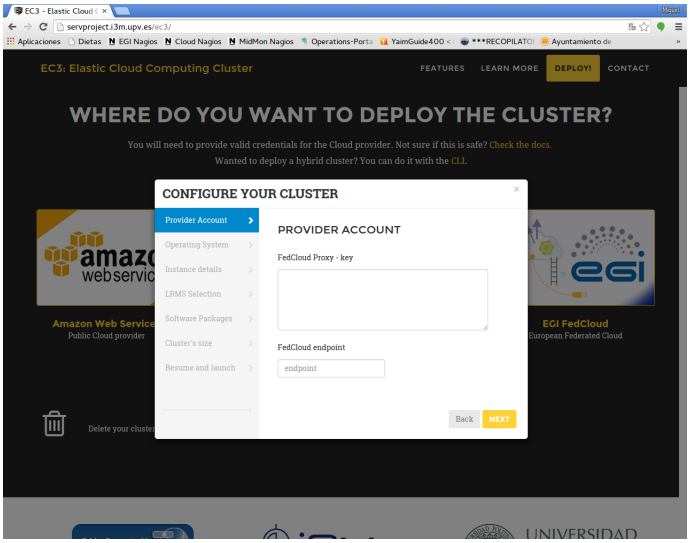


Figure 5.2: EC3 Web interface.

LRMS queue, the CLUES system automatically detects it. If no working nodes are available, it will deploy and configure one. There are different policies that can be fine-tuned, to deploy several nodes if more the one job arrives or to deploy nodes by groups. The nodes will be automatically configured and integrated in the cluster (step 5). Take into account that the initial deployment process only deploys and configures the front-end of the cluster, all the working nodes are then managed by CLUES, powering them on and off when required, thus saving energy/costs of unused resources and providing an automated-elastic infrastructure.

Automatic Configuration of Virtual Appliances with Infrastructure Manager

The IM is a tool that eases the access to IaaS clouds by automating the selection of the rightmost VMI [45], and facilitates the deployment, configuration, monitoring and maintaining of VAs. The IM provides an abstraction layer to be interoperable with different IaaS cloud back-ends. This layer has been designed using a plug-in scheme that currently supports libvirt, Docker containers, Kubernetes, OCCI, OpenStack, OpenNebula, Amazon EC2, Google Cloud and Windows Azure. This set of plug-ins enables the compatibility with a large number of cloud deployments and virtualisation platforms. This approach prevents vendor lock-in and facilitates the seamlessly migration from a simple virtualisation system to a large-scale cloud deployment.

It integrates Ansible as the contextualisation system to enable the installation and configuration of any software dependency. Despite that multiple cloud providers offer services for the deployment of software configurations (e.g. Heat from OpenStack or the Software Management and Catalog service of OpenNebula), IM recipes are compatible with all the platforms supported via the previously described plug-ins.

In our proposed architecture, the IM becomes a key tool that is in charge of contacting the cloud provider selected by the user and deploying the VMs that compose the virtual cluster. Moreover, all the virtual resources deployed are automatically configured, transparently to the user, that receives a customised ready-to-use cluster with Galaxy and Torque properly configured.

Elastic management with CLUES

CLUES is an energy management system for 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 (Torque, SLURM, Mesos, SGE or HTCondor) or a Cloud infrastructure management system, by means of different connectors. CLUES is also integrated with the physical infrastructure through different plugins, so that nodes can be powered on/off using different techniques (Wake-on-Lan, IPMI, etc). In the case of Cloud infrastructures it has a plugin to deploy and undeploy VMs, instead of powering on/off physical nodes. This is the case in our proposed architecture, where CLUES acts as a elasticity manager, deploying and terminating cloud instances by terms of the IM.

CLUES implements different policies that aim at balancing the trade-off that arises when trying to minimise the waiting time for the jobs (which involves a larger number of available nodes) and the minimisation of a Cloud infrastructure cost (which involves a reduced number of nodes, which generate a cost). The implemented policies for a Cloud environment are deeply described in [34].

5.4 Use case

With the advent of massive genomic sequencing, evolutionary genomics and biodiversity genomics have shaken the foundations of many traditional specimen classifications. Many different techniques are applied to cross-correlate the genetic markers of specimens to identify relations and divergences among species. Those techniques usually involve the cross-alignment of Deoxyribonucleic Acid (DNA) regions and the use of clustering techniques to group similar species. Those techniques are computationally expensive and combine both user-specific and common

reference data. In the next subsections, we describe a use case centered in the area of biological diversity.

5.4.1 Application domain

The connection built in this work between a user-friendly Galaxy interface and an elastic cluster focuses on the study of biological diversity, from a set of molecular markers, associating genetic and interspecific diversity. The goal of the experiment is to build cloud points where one point is a molecular marker of a specimen. Then, to recognise patterns in these clouds as geometric objects, and to induce diversity patterns as shapes of these geometric objects. This links biodiversity studies with computational geometry. The method used is to compute differences (or dissimilarities) between markers attached to specimen, and study the patterns in the set of pairwise distances. We provide a tool `disseq` – included in the pipeline – to compute exactly pairwise distances between sequences, from exact score of global or local alignment ([143, 171]; see as well [86]). These distances will be starting point for different tools and methods:

- Multidimensional scaling, which will be presented here as an example
- Nested aggregative clustering,
- Graph based clustering.

These programs are installed, and automatically integrated with the Galaxy portal by means of EC3.

5.4.2 Tools which are deployed

The most time consuming step is the computation of pairwise distances. Indeed, computing a distance between two sequences of respective lengths p and q is in $O(p \times q)$ complexity in time, hence $O(n^2pq)$ for the full matrix. There exists very efficient algorithms for computing first eigenvalues and eigenvectors of a symmetric matrix in Multidimensional Scalling (MDS). Nested aggregative clustering with simple linkage can be executed in quadratic time with n [142], and graph-based clustering can be executed in linear time. The first step in the service deployed on the elastic cluster is computing the pairwise distance matrix. It is the most intensive computation. All other post-treatments can be executed afterwards. These tools are presented here with more detail:

- Multidimensional scaling, the objective of which is to build a point cloud in a low-dimensional Euclidean space, where each point is a specimen, and such that the geometric distances between points is as close as possible from

the local alignment distance between specimen. Several extensions of MDS exist, see e.g. [98] and [164] nonlinear mapping are available.

- Nested aggregative clustering, which is classically linked with designing Operational Taxonomic Units (OTU), where simple linkage method if often privileged [167]. Very efficient quadratic algorithms exist [142]. Let us denote that, if a tree produced by a nested aggregative clustering is cut at a given level, then there is a bijection between disconnected subtrees, and connected components of a graph induced by distances. An advantage of this observation is that connected components can be computed in linear time, and such an approach can be efficient for large numbers of specimen when scaling up.

- Graph based clustering: a pairwise distance matrix between specimen being given, as well as a gap α, a graph can be defined where vertices are the specimen, and there is an edge between vertices i and j if, and only if, $d(w_i, w_j) < \alpha$. Ideally, a taxon at a given level α should correspond to a clique in the induced graph, and in practice, cliques are imperfect, but connected components yield results in accordance with taxonomy in supervised clustering (i.e. when taxonomy is known).

5.4.3 Dataset

The dataset used in this experiment has been produced by sequencing a sample of a freshwater diatoms community in a Swedish river, in 2013. Sequencing occurred in Plateforme Génome Transcriptome de Pierroton (PGTP), in 2014. The dataset consist in 3.10^5 reads, approximatively 300 base pairs long each. It is not unrealistic to produce a full pairwise distance matrix, as it contains $\approx 10^{11}$ floats, i.e. $\approx 300\,Mo$. However, as the objective of this work is to address the IM of an EC3 framework from a user-friendly Galaxy server, we have selected a subsample only, large enough to require the deployment of several virtual machines, but small enough for this number to be reasonable. Scaling up the dimension of this calculation with EC3 and Galaxy is deferred to further work. Hence, a random subsample of 10^4 reads has been selected, on which computations have been performed. Such a figure has been selected as it is beyond the usual time and memory resources usually mobilised by softwares running on laptops for molecular based analysis (without heuristics for accelerating time), which run fluently up to a few thousands reads. Dataset has been uploaded from users desktop by the *upload function* of the Galaxy interface.

5.4.4 Launch elastic Galaxy web-service with EC3

To launch the elastic Galaxy web-service with EC3 with the CLI tool the user must execute a command line similar to that:

```
$ ec3 launch galaxy_cluster ubuntu-fc-cesnet torque nfs maui \
  galaxy galaxy-inra-tools -a auth.dat
```

In this case the *ubuntu-fc-cesnet* template has been used to launch the cluster in the CESNET site of the EGI FedCloud infrastructure. Previously the user has to create the *auth.dat* file with his credentials to access the cloud provider, in this case a proxy file to the EGI infrastructure.

In the Web application (Figure 5.2) the user has a wizard with a set of steps to easily launch a virtual cluster. In the case of EGI FedCloud infrastructure the user has to specify his/her credentials to access the cloud provider (a proxy file), the cloud site endpoint, the identifier of the Virtual Machine Image (typically a basic linux distribution image), the type of instance (flavour) for the font-end and worker nodes. Furthermore the user has to select the all the required tools to make the Galaxy web-service work: NFS, Maui and Galaxy. The Galaxy tools option has been also selected as it will install and configure all the required components to make the previously described tools available to the deployed Galaxy instance.

In some seconds the application will show the IP of the front-end node, the credentials to access with SSH and a cluster name, that can be used to delete the cluster using the web interface. Then the user must wait for the contextualisation processes to finish. This test took 12:29 to launch and fully configure the front-end node.

As it has been described in previous sections, CLUES automatically detects when a job is submitted to Galaxy and it eventually boots up new VMs and integrates them with the LRMS front-end. This process introduces an overhead on the job submission in case that there are no active nodes. As in the case of the front-end, the configuration time depends on the complexity of the applications installed. In this case it took 5:27. This time may vary depending on the number of simultaneous Working Node (WN) booted-up and the contextualisation restarts. If a new WN is booted up before the configuration process of a previous one has finished, the configuration for both WNs will be restarted, although the configuration time in the previous machine will take shorter, as it has been partially performed. Ansible guarantees that even if a configuration process is restarted, the final result will be the same (idempotency). Different policies can be used with CLUES (the elasticity manager) to minimise this overhead, as switch on a block of nodes, or have a set of nodes always on, etc. In this case the Multi Dimensional Scaling tool has been used. Initially, the job remains queued until the a virtual node is deployed, then it is executed and finally it successfully finishes. Figure 5.3 shows

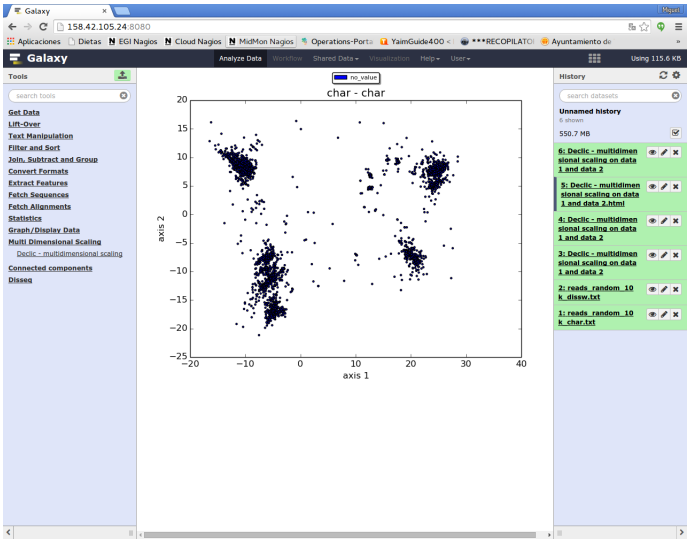


Figure 5.3: Projection of points cloud on first two axis. There are 10^4 points with a definite clustered pattern, in accordance with clustered structure of taxonomy.

the final results of the execution of the job. Dataset and algorithm for this calculation are presented in next section for sake of completeness. When jobs finish and the nodes become idle for some time they will automatically be undeployed automatically from the cloud provider to avoid unnecessary costs.

A whole demonstration of the functionality of the EC3 tool has been recorded in a set of video tutorials[4].

5.4.5 Experimentation

In order to analyse the behaviour of the elastic cluster back-end, we performed an experiment that involved the submission of 75 jobs related with the use case that have a short execution time (from 2 to 8 minutes long), scattered along the time. Jobs were submitted through an script so it can be reproduced. The initial conditions were a galaxy cluster with up to 10 nodes with no worker node powered on. The cluster was created directly using the EC3 Cluster as a Service web portal. The idle time was fixed to 3 minutes, so a WN with no jobs for more than 3 minutes is automatically powered off.

Figure 5.4 shows the behaviour of the experiment. The green line depicts the accumulated number of jobs submitted. This line gives the information of the

[4] `https://www.youtube.com/playlist?list=PLgPH186Qwh_1IOesmaTLjd35Q-QqdWf9k`

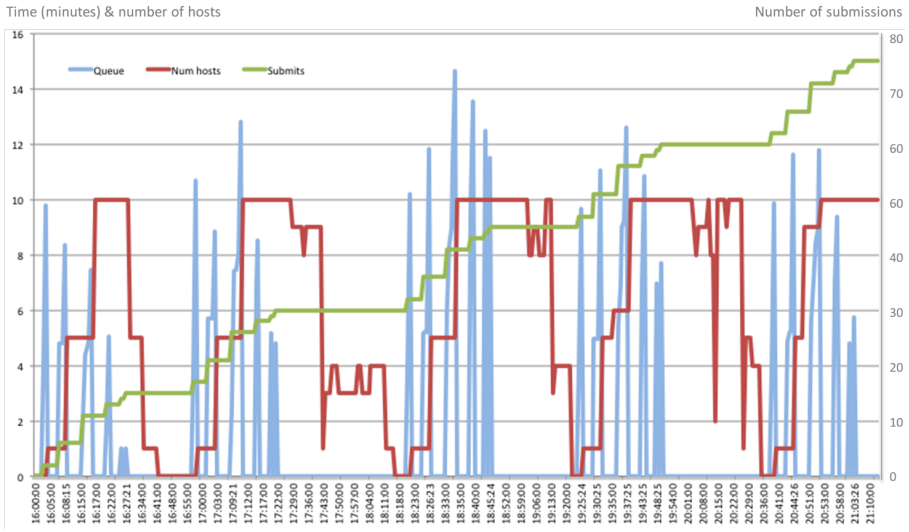


Figure 5.4: Evolution on time of number of jobs and VMs powered on.

submission pattern. It can be easily seen that submissions are concentrated in 5 blocks, with a variable idle time among them. The idle time was long enough to guarantee that a reasonable fraction of the working nodes become empty and they are powered off. The number of submissions is shown in the right axis.

The red line describes the number of powered on WN at a specific point in time. The system starts with all the nodes powered off, and progressively they are powered on and off depending on the workload. The left axis shows the number of powered on nodes. It can be seen that nodes are up and running few minutes after the submission. Some instabilities appear due to self reconfiguration actions of the cluster and external network traffic of the production platform. It can be seen that the number of hosts powered on depends on the number of submission job.In some cases, 5 nodes are simultaneously powered on. The usage time of the whole cluster is 39%. This means that considering the 10 working nodes along the time (nodes*time) 61% of the potential cost of the infrastructure has been saved (assuming a pay a you go model with the granularity of minutes). The active time includes idle time for the WNs before they are powered off.

Blue line shows the waiting time in minutes (units on the left axis). It can be seen that the queueing time is longer at the beginning of each submission block, decreasing as the resources are being powered on. Then, queuing time is reduced. Note that the submission blocks have more jobs (15) as resource available (10), and depending on the time required for setting up the clusters, the waiting time

may change. The cluster is installed on a production infrastructure that is affected by external network traffic.

5.5 Conclusion

In many application domains, e-Science is growing up as grid or cloud computing for computationally intensive processes involving massive data sets. Technology deployment behind requires high skills in computer sciences. However, one goal of e-Science is to promote and foster collaborative works in application domains, by sharing data sets, pipelines, and access to material generated by those processes. This work is run in application domains, and lack of computing science skill may be an obstacle. Therefore, we have implemented a service that enables the automatic deployment on the cloud of an user-friendly Galaxy interface that submits jobs on an elastic queue based on a virtualised cluster that automatically deploys and undeploys resources as needed, improving current cloud-computing capabilities of Galaxy. A working example has been implemented in the domain of computational biodiversity, more precisely investigations for patterns of biodiversity, from molecular data sets of freshwater diatoms.

Chapter 6

Towards Migration of Virtual Clusters across Clouds

To be submitted as

A. Calatrava, G. Moltó, "Towards Migration of Virtual Clusters across Clouds" to Concurrency and Computation: Practice and Experience, 2016

6.1 Introduction

Migration is the act of transferring, partly or fully, a running application together with its progress from one infrastructure to another. Migration introduces an unprecedented degree of flexibility for a datacenter administration, with the ability to decouple the execution from the underlying hardware (see Figure 6.1). For example, this enables a system administrator to temporarily outsource cluster-based workloads to a public Cloud (or to another datacenter running an on-premises Cloud) to deal with common situations in a datacenter, such as a planned outage, experimented failures in the hardware or if an actualization is scheduled. A scheduled downtime might affect a long-running scientific application. By migrating the application or even the complete virtual cluster of VMs where the application is being executed to another virtual infrastructure, the application can resume its execution. The migration mechanism can be used to provide fault tolerance in a highly available system. This migration should be transparent to the guest operating system, applications running on the operating system, and remote clients of the virtual machine. It should appear, to all involved parties, that the virtual machine did not change its location.

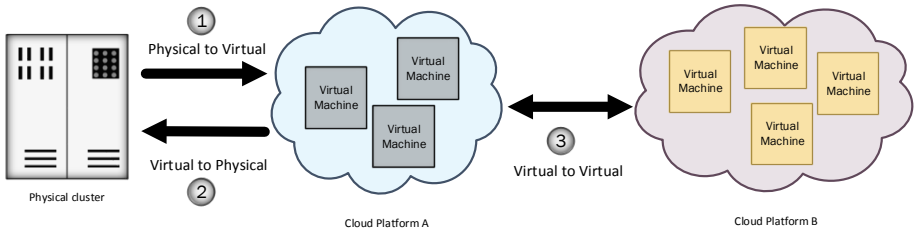


Figure 6.1: Migration scenarios in a datacenter.

Migration can be applied at three different levels (Figure 6.2). The first level is Application Migration. This schema consists on migrating the applications running inside of the VMs. This procedure typically requires checkpointing techniques in order to save a snapshot of the running application to be resumed in another location. Since we are able to reproduce the same precise software environment at destination (via appropriate contextualization) the application can resume execution in a newly provisioned VM. Checkpointing operations write to disk checkpoint files from which to resume execution. Then, these files have to be transferred together with the application in order to resume execution at destination. The major advantages of this approach are lower communication overhead and the possibility of migrating services between physically distributed locations and incompatible hypervisors or physical nodes involved.

In the middle of the pyramid we found Virtual Machine Migration, where (a set of) VMs are migrated to another hardware nodes (either inter-cluster or intra-cluster) with the help of the underlying hypervisor. Migration can be performed online or offline. Live migration of Virtual Machines consists on moving a running VM from one physical host to another minimizing the downtime perceived by the VM users. This concept was first introduced by Clarke et al. [48], as opposed to offline VM migration, where a VM has to stop running before copying the complete state to another machine. Intra-cluster live migration is supported by the major hypervisors, while inter-cluster live migration is supported by Xen and to some extent by KVM.

Finally, the last level of migration is Virtual Cluster Migration, which involves transferring the computing nodes (and also the front-end) from one source Cloud infrastructure to a destination Cloud infrastructure, even across different Cloud providers. For that, we can deploy a cluster in the destination datacenter with the same precise configuration as the original virtual cluster. This includes searching the most similar VMI. Then, jobs can be migrated from the original virtual cluster to the new one by using application migration techniques.

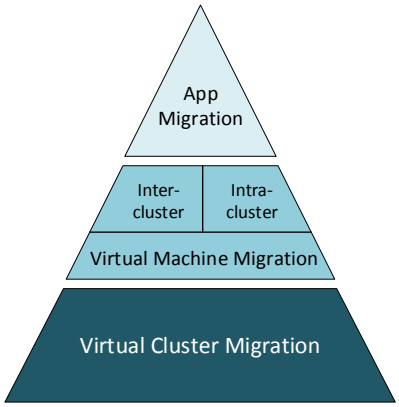


Figure 6.2: Migration hierarchy represented in a pyramid.

This chapter aims at developing the methodology and software tools required to migrate (partly or fully) virtual clusters and their workloads across different Cloud infrastructures. In particular, we want to provide support for virtual cluster migration to EC3, a tool previously developed by the authors, to create and manage virtual hybrid elastic clusters. After the introduction, the remainder of the chapter is structured as follows. First, section 6.2 covers the state of the art in the context of migration. Next, section 6.3 exposes and analyzes the proposed architecture to provide migration capabilities to EC3. Then, section 6.4 addresses a proof of concept of the developed migration capabilites. Finally, section 6.5 summarizes the chapter and points to future work.

6.2 Background and Related Work

As stated in the introduction, there exists several levels of migration. Therefore, it is necessary to analyze the related work in the field of migration making distinctions in the different types of migration: application migration, virtual machine migration and virtual cluster migration. The following subsections analyze and discuss each one of them.

6.2.1 Application migration

Application migration is the act of transferring a running application between two machines. This type of operation involves saving, migration and recreation of the process state as well as the reconstructing of any inter-process communication channels to which the migrating process is connected. This is not a new concept, as we can see in some works like [132] where the authors reviewed the field of process migration by summarizing the key concepts and giving an overview of the most important implementations at that time. Nowadays some works still continue focusing their attention on migration strategies, like [200] where the authors attempt to provide a systematic reference for developers to leverage off among different migration strategies for seamless application mobility. They consider four dimensions of design concerns in application migration in order to classify the different approaches: temporal, spatial, entity and other concerns. Other researchers are doing efforts in improving migration algorithms. For instance, the work described in [101] presents the Lightweight Live Migration (LLM) mechanism to integrate the migration of the whole system but, applying input relay techniques for service replication, in order to reduce the overheads caused by the migration process.

Migration can be used in combination with checkpointing, a mechanism by which an application stores its state periodically to the disk. The most common use of checkpointing is in fault tolerant computing, where the goal is to minimize loss of CPU cycles when a long-running application crashes before completion. By checkpointing a program's state at regular intervals, the amount of lost computation is limited to the interval from the last checkpoint to the time of crash. Research in this class of checkpoint algorithms and systems has been ongoing for at least last 20 years. For example, in [186] the authors describe the implementation and applications of the Unix checkpointing library libckp, one of the first libraries that considered user files as part of the process state. In recent works, after some studies about the performance of checkpointing algorithms [149], the aim is to optimize these algorithms in order to reduce the impact in the programs and improve the total execution time of the scientific applications [21], [193].

The checkpointing technique can be used in grid environments too, as shown in [125], where it is combined with job migration to recover from system failure over the grid. Another area of work is parallelism, where high-performance applications must be able to tolerate inevitable faults [195]. For instance, in [92] the authors present the design and implementation of an infrastructure to support checkpoint/restart fault tolerance in the OpenMPI project.

With regard to functional tools and libraries of checkpointing, several of them have been developed for the Linux/Unix family of operating systems. These checkpointing utilities may be divided into two classes, those which operate in user space, and kernel-based checkpointing packages. User space checkpointing packages are highly

portable and can typically be compiled and run on any modern Unix. Examples of tools that implement this type of approach are CryoPID [55], that allows the user to capture the state of a running process in Linux and save it to a file, DMTCP [16], a tool to transparently checkpoint the state of multi-threaded and distributed applications, or CPPC [163], a checkpointing tool focused on the insertion of fault tolerance into long-running message-passing applications. HTCondor [91] offers user-level checkpointing and other tools for batch job scheduling. In addition to writing a checkpoint at vacate time, HTCondor can be configured to write checkpoints periodically while the job is executing, for additional reliability. OpenVZ [153] is a kernel-based checkpointing package that allows live migration of virtual private server. There are hybrid kernel/user implementations, like BLCR [65], whose goal is to provide a robust, production quality implementation that checkpoints a wide range of applications, without requiring changes in the application code.

Focusing on improving resource utilization in clouds, CMPs could suspend/migrate some running jobs/virtual machines to reserve resources for other resources. One possibility is to migrate the whole VM, but the cost of this operation can be very high due to the large footprint of VMs. That is the reason why application level migration appears as a possible solution. Moreover if the application is running directly in one of the physical nodes, the application migration is the best solution. In [144] and [145] the authors propose the Event-Based Checkpointing tool (EBC), in which users can easily checkpoint and restart running programs in cloud systems. EBC is based on event-driven architecture and supports both sequential and parallel programs like MPI programs.

6.2.2 Virtual Machine migration

Another approach of migration is live migration of VMs, which consists on moving a running VM from one physical host to another. This concept was first introduced by [48], as opposed to offline VM migration, where VMs have to stop running before copying the complete state to another machine. The authors propose a model, known as pre-copy live migration, in which the memory state of the VM is sent between two machines while the VM is running. Memory pages are sent, keeping track of the ones that get modified and must be resent later. After all the state has been transferred and only a small quantity of modified pages needs to be resent, the VM is stopped for a small fraction of time in order to finish the migration operation. This model assumes that the disk state is either shared in a SAN or mirrored in a Redundant Array of Independent Disks (RAID).

A number of improvements have been proposed on the original algorithm. Svärd et al. [176] evaluate techniques for compressing the changes in memory that must be sent between nodes in order to minimize the overhead. In [102] the authors argue that a VM with high load that make many changes to memory may be

very inefficient or unable to perform live migration. In order to alleviate this problem, the authors propose to reduce the CPU frequency of a migrating VM in order to lower the rate of memory changes, without disrupting the service. Lui et al. [114] propose an alternative memory-to-memory VM migration algorithm that uses trace and replay of operations, claiming a much lower migration overhead. The work in [97] proposes another alternative VM migration algorithm that uses remote direct memory access, bypassing Transmission Control Protocol (TCP) in network operations and reducing the data transmission overhead. Besides these works, another technique appeared in the literature to perform live migration of VMs, known as post-copy [89]. Post-copy migration defers the transfer of a VM's memory contents after its processor state has been sent to the target host while CPU and device state are transferred immediately to the destination host. This migration scheme reduces the downtime and total migration time but incurs service degradation due to page faults which must be resolved over the network by the source host.

Other works have focused on studying the applications and implications of VM live migration. The authors in [185] perform an experimental evaluation of the impact of live migration in the performance of applications running inside. In [117], the authors propose using live migration to move a running VM out of a physical node, performing software maintenance on it and then move the VM back on the node. The work presented in [169] studies the techniques that can be applied to both pre-copy live migration and post-copy live migration to better support migration of memory intensive applications. Finally, the study described in [194] proposes a scheme to reduce power consumption in clusters using live migration. The schema involves consolidating the cluster load in fewer machines and putting to sleep inactive machines.

The concept of reducing energy consumption by consolidating VMs in a cluster using live migration is one of the most prominent applications of live migration in current datacenters. Liao et al. [110] introduce a framework for mapping VM to physical machines under resource constraint in order to minimize energy consumption and cost, using migration to consolidate VMs. The work in [74] proposes a methodology for controlling consolidation of VM in datacenters, minimizing migration of VM with steady load to avoid migrating machines back and forth. The study in [25] proposes heuristics with historical usage data to drive the consolidation of VM in order to optimize the process. Graubner et al. [72] propose a novel energy-efficient VM migration and consolidation method, and evaluates a implementation in Eucalyptus. Finally, the work presented in [115] introduces a model of the performance and energy consumption of the migration process itself, in order to account for this operation in the calculations of the energy efficiency of a consolidation process.

The migration schemes, algorithms and applications discussed so far apply to intra-cluster migration. They assume shared storage for migration operations and

contemplate consolidation operations inside a single cluster. Although VM live migration across physically distributed datacenters (inter-cluster migration) incurs in a much higher overhead than intra-cluster migration, it has also been discussed in the literature. The study of Travostino et al. [181] was the first work to propose live migration of VM across Wide Area Network (WAN) network shortly after VM live migration was introduced. The authors motivate the live migration of VM between physically distributed datacenters, and argue that the performance obtained is competitive. Wood et al. [189] introduce CloudNet, a framework for the pooling of Cloud resources from different datacenters in a unique virtual infrastructure. Disparate resources are configured in a unique VPN and VM can be allocated and migrated between machines at different locations as if they were in the same network.

Another approach that can be applicable to this problem is the mechanism to provide fault tolerance to VMs. In this field, the COarse-Grained LOck-Stepping (COLO) [63] solution arises. COLO is a high availability solution for providing application-agnostic fault tolerance for non-stop services. Both primary VM and secondary VM run in parallel. They receive the same request from the client, and generate a response in parallel too. If the response packets from both machines are identical, they are released immediately. Otherwise, a VM checkpoint (on demand) is conducted. Currently, projects like ORBIT [154] are working on this approach, trying to integrate it with OpenStack to offer high performance fault tolerant VMs.

6.2.3 Virtual cluster migration

The last approach of migration covered in this chapter refers to virtual cluster migration. This action involves transferring the computing nodes, together with their running applications and state, from one Cloud infrastructure to another. This process can imply different Cloud providers in the source and destination. Moreover, the front-end machine can also be involved in the migration operation.

Some works in the literature have discussed virtual cluster migration under a Cloud scenario. For example, the work presented in [29] allows transparent migration of VMs across the datacenter, based on the coordinated use of Network Address Translation (NAT) rules and Address Resolution Protocol (ARP) proxying, for the problem of transparently migrating VMs across multiple IP subnets within a single datacenter. In other study, the framework VC-Migration [196] was developed to control the live migration of virtual clusters, applying different migration strategies based on the well-known solutions to migrate a single VM [124]. The results of the experiments performed with this framework show the main limitation of migrating a virtual cluster, i.e., the large amount of data that needs to be transmitted over a limited network bandwidth.

Tools like VNsnap [105] have appeared in the literature to present a solution to take snapshots of infrastructures composed by VMs connected by a virtual network. This tool is implemented on top of the Xen hypervisor and has the ability to take snapshots of this type of infrastructures, which include images of the VMs with their execution, communication, and storage states, to later restore them without requiring any modifications to the applications, libraries or guest OSs. The work described in [162] introduces Shrinker, a system to live migrate virtual clusters across networks. Shrinker is based in the principle that VMs with similar configuration have much of their state (disk and memory) in common, and hence this common information can be migrated only once. Performance evaluations for a KVM implementation are included in the study.

Migration capabilities also allowed hybrid schemes in which computing nodes running in an on-premises infrastructure are switched-off while working nodes are deployed in a public Cloud to replace them, as shown in our previous work [37]. In this scenario, the front-end is not migrated, only the working nodes are marked as migratable nodes that can be immediately powered off and deployed in the new infrastructure, loosing the progress of the applications running inside of them, or the system can wait until the application running inside the node finishes and then terminate the nodes, without loosing the job progress.

The following sections of this chapter describe the proposed architecture to perform virtual cluster migration, that implies also checkpointing operations at the application level to migrate jobs running inside the virtual cluster without loosing its work progress, and VM migration.

6.3 Proposed solution for virtual cluster migration

In this subsection we propose virtual cluster migration achieved by using checkpointing technologies to migrate applications running on the cluster and the replication of the original virtual infrastructure in the cloud destination, transferring only the checkpointed data of the running applications, instead of transferring the complete cluster. This approach aims at reducing the overheads in the network caused by migrating virtual machines across different Clouds, as we discussed in the related work section above, improving the performance of this type of operations. This will give a powerful degree of flexibility to move stateful long-running jobs due to opportunistic or maintenance issues across infrastructures, being able to take advantage of a price reduction in the case of public Clouds (e.g. spot instances in Amazon EC2) or to cope with common problems presented in a datacenter, like an outage, experimented failures in a physical cluster or scheduled updates by e.g., temporarily moving the workloads to a public Cloud without restarting the applications from the beginning.

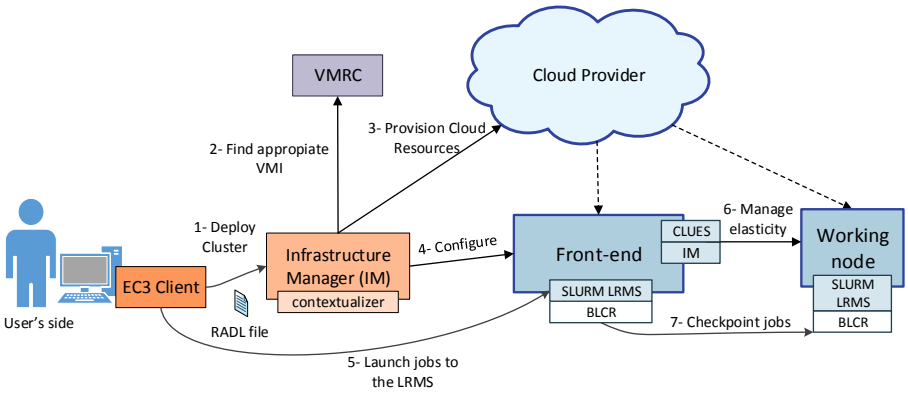


Figure 6.3: Architecture to deploy a migratable cluster supporting the SLURM LRMS.

The proposed solution is based on the usage of EC3 to clone the original infrastructure and the usage of the BLCR tool [65] to checkpoint the applications running on the cluster and restore them in the new cluster destination. For that, four main steps need to be covered: i) checkpoint the running applications, ii) clone the original cluster with the primary configuration in the same or another cloud provider platform, iii) copy those checkpoint files to an external storage system, like Amazon S3 and iv) download checkpointing files in the cloned cluster and restore the jobs at the destination.

Figure 6.3 represents the architecture of a migratable cluster deployed by EC3. The deployment of the migratable virtual cluster starts when the user requests it to EC3. Immediately after, EC3 contacts with the IM to start a VM in the cloud provider to act as the cluster front-end. Once the VM has been successfully deployed, the configuration phase starts by means of Ansible, that is in charge of configuring the machine to behave as the front-end of a cluster. For a migratable cluster, it will install and configure SLURM [100] to act as the LRMS of the cluster, together with BLCR and configure them properly to facilitate checkpointing operations, as required by SLURM. Finally, EC3 always installs CLUES and the IM to perform the automated elasticity of the cluster. Once the configuration process finishes, the front-end is ready to receive the user's jobs. CLUES is able to detect when the jobs reach the LRMS and no available nodes of the cluster exist to satisfy the job requirements, thanks to the plugin specifically developed for SLURM. When this situation occurs, CLUES contacts with the IM installed in the front-end node and requests the deployment and configuration of a new working node to execute the job. Thus, the cluster is able to manage the cluster size automatically and configure new nodes according to the workload.

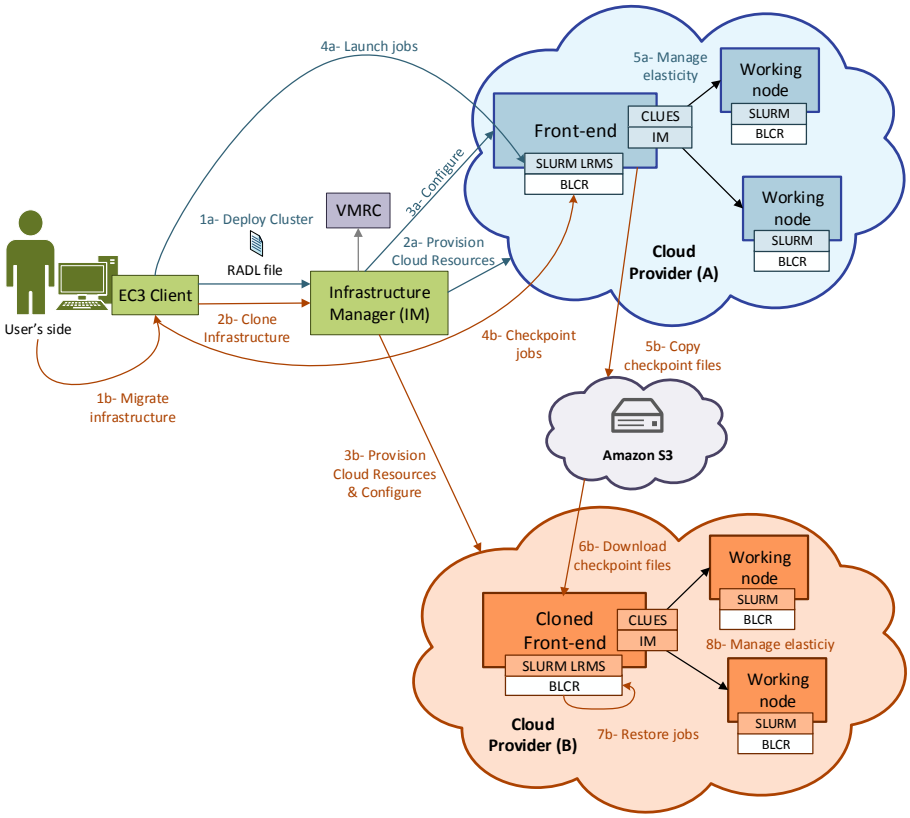


Figure 6.4: Workflow of a virtual cluster migration across Clouds.

6.3.1 The migration process

Figure 6.4 represents the steps carried out in order to migrate the virtual cluster from one Cloud infrastructure to another. In the figure, for the sake of completeness, the deployment and usage of the cluster is represented in blue (explained in more detail in the section above), while the process of migration of the cluster is represented in orange.

In the migration scenario, an interaction begins when the administrator/user requests the migration of the cluster to the EC3 client (1b). Then, EC3 starts the process to clone the original infrastructure to the destination Cloud infrastructure specified by the user (2b). This process includes recovering the primary configuration of the cluster (i.e. the RADL originally employed) to deploy another cluster with identical features (hardware, software and configuration requirements). With

this data, EC3 contacts with the external IM and requests the deployment of a new cluster front-end, identical to the original front-end (3b). Immediately after requesting the new deployment, EC3 checkpoints the running applications terms of BLCR (4b). This operation produces several checkpoint files that need to be saved before the old infrastructure is destroyed. For that, all the checkpointed data is copied into a bucket in Amazon S3 (5b), specifically created for this operation. The usage of S3 provides fault tolerance to the solution, due to the replication algorithms that S3 internally employs. Once the cloned cluster is ready, EC3 can copy those checkpoint files from S3 to the new cluster (6b) and restore them at the destination (7b) calling the *scontrol* SLURM operation, configured with BLCR. Thus, the process of migration is completed, and the applications running on the original cluster are now restored in the cloned cluster, restarting their execution from the checkpoint without loosing their progress.

Notice that this process must be transparent for the applications, that need to find the same execution environment in both clusters. The IM needs to find an appropriate VMI in the destination infrastructure to be configured to produce exactly the same VM configuration in both clusters. For that, we use the VMRC service [45], a cloud agnostic catalog of VMIs. This system offers matchmaking capabilities that allows the IM to obtain the VMIs that best fit the application requirements.

6.4 Case study

In this section, we present a proof of concept of the migration process. The objective is to analyze the performance of the migration operation with EC3. For that, we are going to migrate a virtual cluster running inside an OpenNebula on-premises cloud to the AWS public provider. We will execute MrBayes [141] to perform a Bayesian phylogenetic analysis of combined data that will be checkpointed and migrated to another platform to continue with its execution. This application is also used in chapter 7, where more details about it are given. The virtual cluster will be configured with SLURM acting as LRMS and BLCR, correctly configured to work together. Moreover, an NFS system is configured in the cluster to access to the checkpoint files from the front-end and all the working nodes. Finally, the AWS CLI [1] client is installed to work with Amazon S3.

The infrastructure used to deploy the initial on-premises cluster is a Cloud platform that comprises eight dual processors with 14 core nodes (28 cores per node), with 64 GB of RAM and a shared storage system of 10 TB, which was backed up by a storage area network where the hard disks were stored as volumes. This system was managed by OpenNebula 4.8 using KVM as the underlying hypervisor. With respect to the destination infrastructure, we rely on Amazon Web Services as the

[1] AWS CLI: `https://aws.amazon.com/cli/?nc1=h_ls`

public Cloud provider for the migration. The instance type chosen is `m1.small`, with one (virtual) processor, 1.7 GiB of RAM and 160 GiB of disk. Finally, the VMI used in the on-premises cloud must provide the same execution environment as the EC2 AMI. Thus, we will rely on Ubuntu 14.04 LTS images previously indexed, not stored, in the VMRC.

Deployment avg. time	1142s
WN deployment avg. time (ONE)	520s
WN deployment avg. time (EC2)	1460s
Clone avg. time	3510s
Checkpoint avg. time	1s
Data upload to S3	3s
Data download of S3	3s
Recovery avg. time	6s
Total migration time	4975s

Table 6.1: Times involved in the process of migration.

Table 6.1 shows an itemization of the main times involved in the process of migration. First, the deployment of the initial front-end of the original cluster takes an average of 1142 seconds. This time includes from the request to the cloud provider for a VM to the creation of the VM and the contextualization of the VM to behave as a front-end of a SLURM cluster configured with BLCR. The time required for deploying a new working node to the cluster when jobs arrive to the LRMS is 520 seconds (on average).

Once the cluster is running and several jobs are in execution, a request to migrate the cluster arrives to EC3. Immediately, the process to clone the original cluster starts. This process includes searching an equivalent VMI for the deployment of the new cluster in the destination Cloud infrastructure, recover the original configuration recipes and, finally, deploy and contextualize the new front-end of the cloned cluster. This process takes in Amazon EC2, with a `m1.small` instance, an average time of 3510 seconds. This time can be reduced by using preconfigured VMIs, but in the case of migration, it is very difficult to find the same preconfigured VMI in different Cloud providers, unless the user has already created the preconfigured VMI in both Cloud infrastructures. Instead, our approach can use vanilla VMIs where only the basic Operating System is available and all the applications are dynamically installed, together with the configuration applied. Another solution might be to increase the features of the VM by selecting, for example, a `m1.medium` instance, that can reduce the times compiling and installing SLURM and BLCR tools. which is a very time-consuming process.

At the same time, the system is creating checkpointing files for all the tasks running inside the cluster, to subsequently upload them to Amazon S3 (the times shown in Table 6.1 are referring for only one job, thus, in our case, it takes only 4 seconds per task). This way, the long process of cloning the original infras-

tructure is partly alleviated by performing these tasks in parallel. The size of the checkpointing files for this case were of 4.22 MB. Notice that the times to checkpoint and recover a job, together with the upload and download actions done with S3, depend directly on the application that is running. The more complex the application is (more memory consuming, processes, connections, etc), further time is needed to perform all these operations. For example, for the Architrave application presented in chapter 4, an average time of 160 seconds was required to checkpoint the application, and a recovery time of approximately 70 seconds. However, in our case, the MrBayes application is exhaustive in CPU (98%) but not in memory (with a 7% consuming percentage).

Finally, once the front-end of the cloned cluster is ready, the migration process recovers the checkpoint files of the jobs running in the original cluster from Amazon S3 (an average of 6 seconds per job), and restore the jobs from their checkpoint. This action will cause the automatic detection of new job requests arriving to the LRMS queue, thus starting the process of launching new working nodes to attend the petitions (that in Amazon EC2, it takes an average of 1460 seconds per node, but they can be deployed in parallel). Thereby, the migration process of the whole virtual cluster is completed in approximately 5000 seconds, without affecting the job execution, i.e. transparently for the jobs and without loosing the job progress.

6.5 Conclusions and Future Work

Migration introduces an unprecedented degree of flexibility for a datacenter administration, with the ability to decouple the execution from the underlying hardware. Thus, this chapter has introduced an approach to migrate virtual clusters deployed over Cloud infrastructures. The early developments presented in this chapter have been integrated in EC3. The *clone* operation for EC3 is already developed, documented and integrated in the master branch of EC3 in GitHub. Also, the plugin for SLURM is successfully developed and tested, and also forms part of the latest release of CLUES. A brief test has been presented also in this chapter, that proves the ability of EC3 to migrate virtual clusters together with their workloads.

However, we can progress in the proposed solution. As future work, we want to improve the migration capabilities of EC3 by using containers. Containers are a light-weight solution to encapsulate applications. These applications running in containers can be migrated across infrastructures by using checkpointing techniques applied to containers, by means of existing tools like OpenVZ [133] or Checkpoint/Restore In Userspace (CRIU) CRIU [54]. Our idea is to use an LRMS that supports the execution of the applications inside containers, such as Apache Mesos [88] together with Docker [60], and checkpoint the containers rather than the applications. This solution eliminates the problems caused by most of currently available checkpointing tools, that need to compile the application against their

checkpointing libraries, what might cause incompatibility problems. However, the support of checkpointing in containers is still a work in progress. The official Docker GitHub account has several open issues to officially support checkpointing operations, but they are still in progress [148]. Our proposed solution will provide container-based cluster migration once the current developments in Docker will be available. This process will be of special interest for the scientific community, since no other tools currently exists that support virtual cluster migration.

Chapter 7

Container-based Virtual Elastic Clusters

Submitted as

C. De Alfonso, A. Calatrava, G. Moltó, "Container-based Virtual Elastic Clusters" to the Journal of Systems and Software, 2016

Abstract

eScience demands large-scale computing clusters to support the efficient execution of resource-intensive scientific applications. VMs have introduced the ability to provide customizable execution environments, at the expense of performance loss for applications. However, in recent years, containers have emerged as a light-weight virtualization technology compared to VMs. Indeed, the usage of containers for virtual clusters allows better performance for the applications and fast deployment of additional working nodes, for enhanced elasticity. This paper focuses on the deployment, configuration and management of Virtual Elastic computer Clusters (VEC) whose nodes are hosted in containers running on bare-metal machines. The open-source tool Elastic Cluster for Docker (EC4Docker) is introduced, integrated with Docker Swarm to create auto-scaled virtual computer clusters of containers across distributed deployments. We also discuss the benefits and limitations of this solution and analyse the performance of the developed tools under a real scenario by means of a scientific use case that demonstrates the feasibility of the proposed approach.

7.1 Introduction

eScience involves the execution of complex HTC, HPC applications and long-running workflows. This requires a significant amount of computing power and memory capacity that can be only obtained via distributed computing. Indeed, large-scale Distributed Computing Infrastructures (DCIs), such as the EGI[1] have been tremendously successful in supporting the computational requirements of many scientific communities across Europe [183, 43]. However, one of the main limitations of Grid infrastructures is that applications have to be ported to the execution environments provided by the machines involved, what results in a rigid structure composed by several VOs that support a set of applications. This inability to provide customized execution environments for applications is addressed by Cloud Computing by means of VMs that encapsulate the Operating System (OS)) together with the user application and its dependences in a VMI that can be run on a physical machine by means of a hypervisor.

Indeed, the ability to provide ubiquitous, on-demand network access to a set of configurable computing resources, according to the NIST definition [127] of Cloud Computing, has paved the way for the rise of many public Cloud providers (such as AWS[2], Microsoft Azure[3] or Google Cloud Platform[4]), different CMPs (such as OpenNebula or OpenStack) and even initiatives to create large-scale community Clouds (e.g. EGI Federated Cloud[5]). Cloud computing has provided researchers with access to unprecedented customizable computing resources, either on-premises or on public Clouds. However, these computing resources still require a coordinated use for applications to efficiently use them. For that, LRMS such as Torque [2], SLURM [100] or HTCondor [179] are job schedulers that are commonly used to dispatch jobs across nodes [6]. Indeed, computing clusters are still widely-used computing facilities to support the execution of many types of applications.

VEC deployed on Cloud infrastructures have introduced many benefits when compared to physical clusters, as we addressed in our previous work [34], avoiding upfront investments and the ability to adapt the execution environment to the applications (and not viceversa). This work was later extended to create EC3[6] [41] an open-source tool to create self-managed cost-efficient virtual hybrid elastic clusters across Clouds that is currently offered as a free online service, being used for scientists to provision their own clusters on public, on-premises and federated Clouds.

[1] European Grid Infrastructure: `http://www.egi.eu`
[2] Amazon Web Services: `https://aws.amazon.com`
[3] Microsoft Azure: `https://azure.microsoft.com`
[4] Google Cloud Platform: `https://cloud.google.com`
[5] EGI Federated Cloud: `https://www.egi.eu/federation/egi-federated-cloud/`
[6] EC3 (Elastic Cloud Computing Cluster): `http://www.grycap.upv.es/ec3`

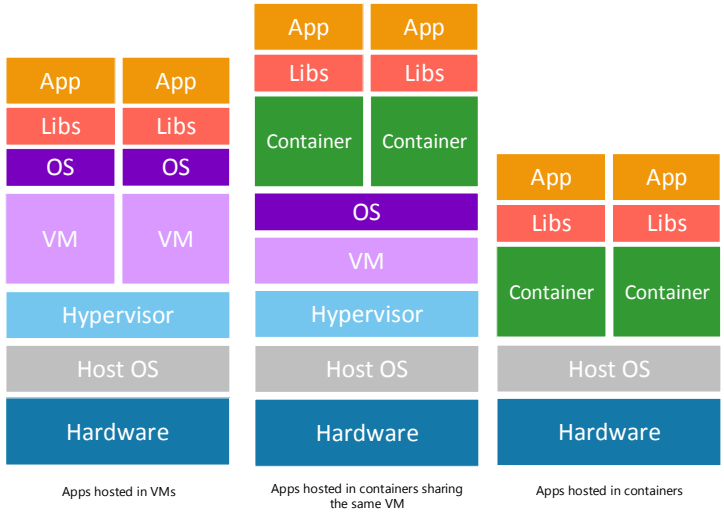


Figure 7.1: Virtual Machines and Containers possible architectural configurations.

In the quest for increased performance with respect to virtualisation techniques, Linux containers appeared as a lightweight alternative to VMs. Linux containers enable to run multiple isolated processes in a host without the overhead caused by the hypervisor layer introduced by VMs. While hypervisors provide hardware abstraction, container-based virtualization is characterised by multiple isolated user spaces running at the operating system level (see Figure 7.1). This provides process isolation at a fraction of the overhead introduced by the hypervisor. Container-based virtualization proved to be an alternative to traditional hypervisor-based systems, as it reduces the overhead caused by VMs in CPU, memory and storage, as described in [73] and [166]. Linux containers can be run on top of VMs to achieve multi-tenant isolation using the VM as the boundary of security and containers as the boundary of resource allocation to applications. However, the main benefits of containers arise when used on bare metal, in order to obtain increased performance compared to VMs. Among the different existing container platforms, Docker[7] stands out as a software containerization platform that can encapsulate an application in a complete filesystem that contains all the dependences required to be executed (code, runtime, system tools and libraries, etc.). This guarantees portability across multiple platforms, regardless of the execution environment.

Our hypothesis is that container-based technology can be effectively integrated with cluster-based computing to create virtual computer clusters of Docker containers with the very same functionality as virtual clusters of VMs, and physical

[7]Docker: `https://www.docker.com`

clusters of PCs, but with enhanced capabilities that include: i) improving the performance of resource-intensive applications that will run isolated on bare metal; ii) improving the elasticity of the cluster, by reducing the time required to spawn and terminate additional containers; iii) supporting customised execution environments via low-footprint images and iv) the ability to access specific hardware devices such as General-Purpose Computing on Graphic Processing Units (GPGPUs).

Therefore, this paper introduces an architecture to deploy container-based virtual computer clusters that feature automated elasticity and the ability to provide customised virtual execution environments across a bare-metal backend on which containers managed by a Container Orchestration Platform (COP) are executed. Several computer clusters customised for the execution of different scientific applications can be provisioned to share the same physical computing backend. This provides increased resource utilisation and performance while maintaining isolation across workloads coming from different clusters.

To this aim, this paper describes EC4Docker[8], an open-source tool to deploy, configure and manage container-based virtual computer clusters that can be run on bare-metal nodes (as well as on VMs). These virtual computer clusters expose the very same user interfaces expected by users (accessed via SSH, supporting a LRMS, etc.) but they are completely backed by Docker containers that are dynamically deployed, depending on the workload, across a distributed Docker Swarm [116] backend that can be deployed either on bare metal or on public and on-premises Clouds.

After the introduction, the remainder of the paper is structured as follows. First, section 7.2 introduces background information and covers the state of the art related to containers, revising existing tools, performance studies and clustering solutions of containers. Next, section 7.3 exposes and analyses the proposed architecture to deploy these container-based virtual computer clusters. Then, section 7.4 addresses different scenarios in which the proposed solution is evaluated and analyses the significant benefits of these approach. Finally, section 7.5 summarises the paper and points to future work.

7.2 Background and Related Work

According to Buyya [31], a computer cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single integrated computing resource. The key components of a cluster include:

[8] EC4Docker is available in `https://github.com/grycap/ec4docker`

1. Multiple Computers. Typically one of them (named the "front-end node") acts as an entry point to the computer cluster and the others execute the jobs (named the "working nodes").

2. Operating Systems OS. In scientific computing, the most common operating systems are Linux or Unix-based.

3. Interconnection network. The computers interact among them through a local network. There may exist different networks specialised for different tasks (e.g. data, parallel processing, etc.) based on different technologies for computers to communicate (e.g. Myrinet, GbE, 10GbE, etc.).

4. Cluster middleware. The cluster middleware, also known as LRMS, is a set of tools to use the cluster as a single computing entity. These tools carry out the whole lifecycle of executing a job in the cluster (e.g. staging the files in the working nodes, starting the applications, retrieving the resulting files, etc.). Some examples of well known LRMS are Torque, SLURM, or LSF.

5. Parallel programming environments. Applications typically use well-known libraries to communicate processes. Some examples are OpenMPI, LAM/MPI or MPICH, which support the MPI standard. These libraries are usually optimised for the specific network interfaces (e.g. SCI, Myrinet, etc.).

6. Applications. These are the user applications executed in the computer cluster.

The main interface employed by the users of the cluster is an interactive session to the front-end node in order to submit jobs to be executed on the working nodes [159]. Indeed, computer clusters used to be huge physical infrastructures, but advances in virtualization technologies and Cloud computing paved the way for VCs to appear. A VC is comprised of VMs and a virtual networking environment. The other components in a VC are the very same that those used in the physical cluster. These VCs can be deployed in on-premises infrastructures or in commercial public Clouds.

A VC relies on VMs even if they are not used (i.e. they are idle). These idle virtual working nodes are a problem in a Cloud environment because (a) in case the cluster is deployed in an on-premises infrastructure, other users cannot take advantage from the unused resources allocated to the VC, or (b) in case the cluster is deployed in a public Cloud, the unused resources result in an economic cost for the user. An Elastic Virtual Cluster (EVC) avoids wasting either resources or money, by destroying the idle working nodes and deploying them again when they are needed. In order to implement an EVC, an elasticity manager is required to take care of creating or destroying the working nodes, depending on the workload.

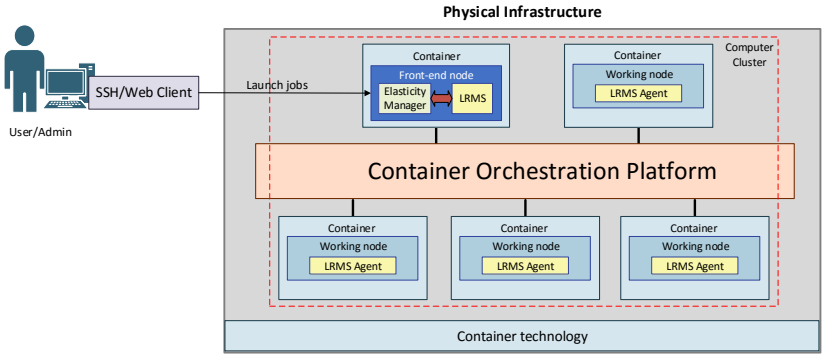


Figure 7.2: Generic architecture to deliver container-based virtual elastic computer clusters deployed on a computing infrastructure managed by a Container Orchestration Platform.

The work described in this paper is a step forward on computer cluster virtualization, that builds on container-based virtualization to reduce the performance penalty introduced by VMs. The goal for a container-based EVC is to provide the users with computer clusters to be used as if they were physical computing clusters, with the added value of using containers instead of VMs. Therefore, the requirements for the container-based EVC is to preserve the very same environment and usage patters that are commonly used in this computing platforms, i.e. the software stack: the OS, the cluster middleware, the parallel environments and the applications, as shown in Figure 7.2.

The next section includes a review of related works about the different technologies that lie within the scope of this work.

7.2.1 Related work

Containers

Container technologies have gained significant momentum in the last years, introducing changes in the way applications are built, shipped, deployed, and instantiated [126], [155]. There exists different software available to create Linux containers, as is the case of LinuX Containers (LXC) [112] and LXD [44], rkt [53], OpenVZ [153], Linux-VServer [173] and Docker [60]. In particular, Docker turned containers into a mainstream technology, contributing: i) Docker Hub, a global shared repository of Docker containers; ii) a procedure to create Docker images out of Dockerfiles and iii) the usage of a layered file system that reduces the footprint

of Docker images. Docker containers use *cgroups*, a feature in the Linux kernel that allows to constrain the resources (e.g. CPU, memory and network) consumed by a process together with *namespaces* to provide processes with their own view of the system. In our case, the containers will correspond to the working nodes that compose the VC.

Container Orchestration Platforms

The ecosystem of applications around Docker has exploded in the last years [157], with contributions in many areas such as Continuous Integration/Continuous Delivery (CI/CD), application packaging and COPs. Indeed, there are many applications to manage the execution of containers across multiple hosts. For example, Kubernetes [109] is an open source orchestration system for Docker containers. It handles scheduling onto nodes in a compute cluster and actively manages workloads to ensure that their state matches the user's declared intentions. The scheduling in Kubernetes is based in *Pods*. These are groups of containers that are deployed and scheduled together. Pods form the atomic unit of scheduling in Kubernetes, as opposed to single containers in other systems. Containers within a pod share an IP address, and labels can be used to identify each group of containers. Apache Mesos [88] can be used to deploy and manage applications inside containers in large-scale clustered environments. The architecture of Mesos is designed to be high-available and for that uses ZooKeeper. Mesos, in combination with a job system like Marathon [119] or Chronos [47], takes care of scheduling and running jobs and tasks, that can be run in containers or directly in the nodes of the cluster. Finally, Docker Swarm [116] represents the native clustering approach proposed by Docker, which "provides native clustering capabilities to turn a group of Docker engines into a single, virtual Docker Engine". This way, a container-based VC can be easily created on top of virtual or physical resources. The architecture of Docker Swarm consists on each host running a Swarm agent and one host running a Swarm manager. The manager is responsible for the orchestration and scheduling of containers on the hosts. Moreover, Docker Swarm can be run in a high-availability mode where either etcd, Consul or ZooKeeper is used to handle fail-over to a back-up manager. We opted for Docker Swarm due to its easy integration with the Docker CLI.

Notice that COPs are used to manage the execution of containers in a cluster. The user describes the container, and the COP selects which of the physical host is going to perform the execution of the container. Therefore, these tools represent for containers a similar concept than a LRMS (e.g. Torque, SLURM, etc.) is for jobs in a computer cluster.

Notice that one could use the interfaces provided by a COP to directly deploy containers that run their jobs on a set of computing resources. However, this approach would be disruptive for traditional users of computer clusters since their

usage patterns would significantly change. They would have to use client-side tools to interact with such COPs and deal with data staging in the COP, instead of performing an interactive session via SSH to the cluster. Instead, the clusters deployed via EC4Docker maintain the very same user experience and interfaces exposed by traditional computer clusters (e.g. SSH-based access to the front-end node).

Reducing overhead of VMs using Containers

There are studies in the literature that analyse the overhead of containers for the execution of applications. In [73], the authors explore the performance of traditional VM deployments and contrast them with the use of Linux containers (using Docker). Several benchmarks are used to demonstrate that containers result in equal or better performance than VMs in terms of CPU, memory and storage. The study covered in [161], analyzed the performance of three well known open-source tools (KVM, OpenVZ, and Xen) in the context of HPC. The results showed that the solution that offers near native CPU and I/O performance was OpenVZ. Other works in the literature have also analyzed the performance of containers to execute scientific applications and workflows, such as [27] and [202]. Skyport [80] utilizes Docker containers to execute scientific workflows instead of VMs, reducing the overhead caused by VM virtualization. Also, analysis of the requirements of the applications to be executed in containers have been performed [170]. Because container-based virtualization works at the operating system level, all instances (containers) share the same operational system kernel. That is why container-based virtualization has a weaker isolation when compared to hypervisor-based virtualization [192]. In order to guarantee the resource isolation between the host system and the containers running on, such a system implements kernel namespaces. However, using containers for security isolation might not be a good idea [158]. The only way to have real isolation with Docker is to either run one container per host, or one container per VM, at the expense of a performance overhead. Nevertheless, for security reasons, it might be worth sacrificing the performance of a pure-container deployment by introducing a VM to obtain true isolation.

Containers can run on VMs too, although such double virtualization imposes performance overheads. In [3] authors investigate container-based technology as an efficient virtualization technology for running high performance scientific applications. They used Docker containers and VMs created using OpenStack to execute a molecular modeling simulation software. Results show that container-based systems are more efficient in reducing the overall execution times for HPC applications, because they can be deployed in a remarkable minor time and have better memory management for multiple containers running in parallel.

Virtual computer Clusters

Concerning the use of VC, several well-known tools already exist in the literature to deploy them, such as StarCluster [134], Elasticluster [203] and EC3 [41], but all of them are based on the deployment of VMs. Concerning the creation of VCs based on containers, studies like [191] analyzed and compared some of the container technologies available to the community (Linux-VServer, OpenVZ and LXC) from the point of view of MapReduce workloads, executing several benchmarks to test their performance and manageability. The results show that container-based systems reached near-native performance though LXC offers the best relationship of performance and isolation. The study covered in [87] present the results of deploying Docker containers in a cluster environment when compared to the KVM hypervisor and an evaluation of its suitability as a runtime for high performance parallel execution. The results showed that containers can be used to tailor the runtime environment for an MPI application without compromising performance, and provide better Quality of Service for users of scientific computing. The developers of a Linux-VServer address in [156] a container-based cluster management platform in which Docker and HTCondor work together to execute scientific workflows. The results obtained from executions of a Monte-Carlo simulation showed that Docker had a near native performance comparing with a hypervisor-based virtualization solution.

To our knowledge, there are no works in the literature that feature the adoption of Docker containers to create VCs that provide users with the very same execution environment (e.g. LRMS, client tools) typically available in both physical clusters and virtual clusters of VMs. This pioneer approach allows users to access well-known computing facilities, i.e. clusters of PCs, on top of the lightweight virtualisation provided by containers in order to take profit from enhanced performance and fast elasticity.

7.3 Elastic Cluster for Docker (EC4Docker)

EC4Docker is an open-source tool that deploys Docker Container-based Virtual Elastic computer Clusters (CVEC). The cluster delivered by EC4Docker consists of a Docker container that acts as the front-end node of the cluster, and a set of containers that act as the working nodes. The front-end container behaves as a regular front-end in a cluster: it is accesible by SSH, has installed a LRMS such as Torque or SLURM, and it shares its file system to the working nodes using NFS. The working nodes of the EC4Docker cluster are also containers that behave like regular working nodes in a cluster: they are accesible from the front-end using password-less SSH, they are integrated in the LRMS, and they mount the shared file system.

The novelty of EC4Docker is that the front-end of the cluster is able to create and to destroy the internal nodes depending on the workload. This ability is possible due to: i) the integration of the CLUES[9] [9] elasticity manager that decides when to power on or off the internal nodes and ii) a plugin for CLUES that has been developed for EC4Docker, that makes it possible to translate the commands to power on and off of the internal nodes into the proper Docker instructions that create and destroy Docker containers. This plugin takes advantage from the ability of Docker to be used remotely by exposing its Application Programming Interface (API) through a standard TCP/IP socket.

The concept of a container-based cluster as-is may be useful for prototyping, as the containers are conceived to be ran in a specific host. However, in the case of EC4Docker, it is integrated with Docker Swarm, which behaves as a scheduler that manages a set of Docker hosts as a single entity. It works together with a discovery service, such as Consul[10], that provides high availability to the underlying Docker Swarm cluster. Using Docker Swarm, when a Docker container is created, it is deployed in any of the Docker hosts managed by the Swarm. Using this combination of EC4Docker and Docker Swarm, it is possible to deploy the containers that build the CVEC across multiple hosts which, by the way, can be either physical or VMs. It is important to point that other COPs could be employed instead of Docker Swarm, such as Kubernetes or Apache Mesos as well as managed services for the deployment of containers such as Amazon EC2 Container Service[11].

7.3.1 Features of the Container-based Virtual Elastic Cluster

As stated earlier, using EC4Docker, the users are delivered a computer cluster with the tools that they typically use, and they do not need to change the way of interacting with the cluster. They access the cluster using SSH, where they find the LRMS to which jobs can be submitted as usual. The LRMS is not aware of any container and the applications require no modifications.

However, even experienced users in traditional computing clusters can benefit from the CVEC, because these are useful to create the specific execution environment for their applications. Docker containers are commonly employed to ease the distribution of applications: using *Dockerfiles* in Docker, users can create the container images that include their application along with the required libraries, the most appropriate OS distribution, etc. Starting from that Docker image, the administrator will include the EC4Docker *Dockerfiles* that will create the EC4Docker CVEC that will be delivered to the user.

[9] CLUES: `https://github.com/grycap/clues`
[10] Consul: `https://www.consul.io`
[11] Amazon EC2 Container Service: `https://aws.amazon.com/es/ecs/`

Using this approach, although the underlying infrastructure is shared by all the CVEC, different configurations can be employed. For example, a cluster based on Ubuntu 16.04 and the Torque LRMS can coexist and share the same underlying computational resources with a Scientific Linux cluster whose jobs are scheduled by SLURM. It is important to point out that this feature can be very beneficial for the execution of software applications that are incompatible with each other, without needing to physically isolate the resources. Therefore, the bare-metal physical nodes are shared by all the clusters deployed in the infrastructure, where the container-based working nodes will be deployed to execute the jobs of each cluster.

EC4Docker is not only useful for CPU-oriented applications. In case the applications require access to specific devices, such as GPGPUs, it is possible to instruct EC4Docker to allow the Docker containers to access these devices. On the one hand, in the case of homogeneous configurations where all the physical nodes have a GPGPUs, EC4Docker can be instructed to automatically mount that device inside the container to expose it to applications. In this case, EC4Docker will use the Docker mechanisms to enable the applications to use the GPGPUs available in the physical hosts. For this, the container has to support the specific libraries and drivers required to use the GPGPUs. On the other hand, in the case where only a subset of the physical nodes have a GPGPUs, remote Compute Unified Device Architecture (rCUDA) [64] can be used in order to turn those nodes into servers that provide Graphic Processing Unit (GPU) services to the container nodes that actually execute the applications. The applications do not require source code modification since the rCUDA runtime takes care of the details of routing requests to the specific hardware device.

7.3.2 Behaviour of a container-based virtual elastic cluster

Figure 7.3 describes the designed architecture employed to deploy CVECs on top of a physical infrastructure, as an instantiation of the general architecture shown in 7.2. Therefore, the workflow to create the CVEC follows the next steps:

1. *Preparation of the Docker images.* The preparation of a CVEC starts with the creation of the Docker images that will be used to create the front-end and the working nodes, and its instrumentation using the EC4Docker Dockerfile fragments.

2. *Creation of a network for the container.* The CVEC needs a network for containers to communicate. In case of using Docker Swarm, an *overlay* network that spans across the different sites is required. This overlay network enables different hosts to become part of a swarm and assign non-overlapping IP addresses to their containers to enable communication among them. There is the option of using a single overlay network shared among all the containers

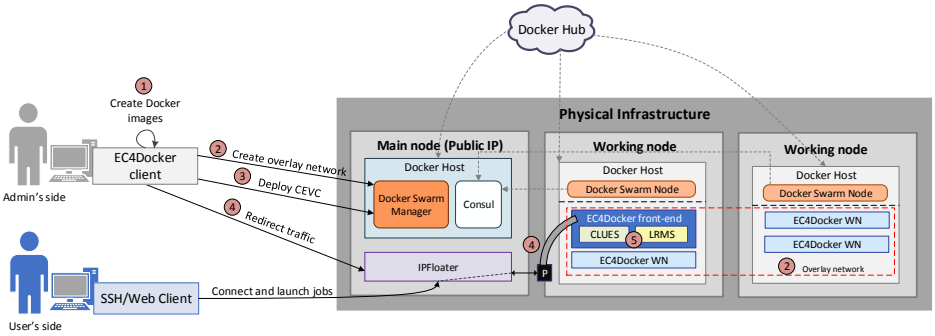


Figure 7.3: Architecture of a container-based virtual elastic cluster deployed on top of a physical infrastructure and managed with Docker Swarm and EC4Docker.

from all the CVECs, or to create per-cluster networks in order to isolate the different CVEC. The overlay network is used to virtualize the interconnection network for the creation of the CVEC.

3. *Creation of the CVEC.* The creation of the cluster consists of deploying the container that will act as the front-end of the CVEC. Since we want the computer clusters to span across multiple hosts, the request to create the container will be submitted to the Docker swarm front-end. A container will be instantiated out of a Docker image created from the EC4Docker Dockerfiles, which include an installation of CLUES and the LRMS chosen by the end-user (SLURM or Torque). The containers are used to virtualize the working nodes for the creation of the CVEC.

4. *Enable external access to the cluster.* In order to access the cluster using SSH, the IP address of the front-end node of the CVEC is required. However, the IP addresses in the Docker swarm cluster will be private to the overlay network for the cluster and, therefore, they are not accesible from outside networks. To solve this problem, we use IPFloater[12], a tool able to redirect the traffic from a public IP to a private IP inside a LAN, thus, simulating the floating IPs offered by OpenStack [151].

Once the workflow has finished, the user is provided with the IP address of the front-end node of the CVEC.

Then, the end users can connect to the cluster via SSH or by means of a web browser (in case of accessing a web application like the Galaxy Portal [82]) and submit their jobs to the selected LRMS as they would do with a physical cluster.

[12]IPFloater is available at `https://github.com/grycap/ipfloater`

The CVEC deployed using EC4Docker dynamically manages the size of the cluster, with the novelty of running the jobs that are going to be executed in the container-based working nodes, instead of using the traditional VM-based working nodes. This way, jobs will enjoy the advantages of light-weight virtualization with a reduced overhead in CPU and memory.

The self-managed elasticity is carried out by CLUES (step 5 in Figure 7.3), that forms part of the container image used by EC4Docker to deploy the front-end of the cluster. CLUES running inside the EC4Docker container detects job submissions to the LRMS in the container-based cluster. Then, if there are no available nodes to satisfy the requirements of the job, it requests an EC4Docker node container to the Docker Swarm Manager. This container will be deployed by Docker Swarm in one of the bare-metal nodes that compose the infrastructure, and will act as a container-based working node of the computer cluster, automatically integrated in the LRMS. The EC4Docker node container will be also connected to the overlay network specifically created for the CVEC, interconnecting the new container with the rest of the CVEC.

As we have mentioned, the scheduling of the location of the containers that represent the cluster is carried out by Docker Swarm. Docker Swarm works with rankings to decide where to execute the container. The node with the highest ranking is the one that is chosen to run the new container. The policies offered by Docker Swarm are: *spread* (default), *binpack* and *random*. The first two policies care about the number of containers deployed in the node and the CPU and RAM free for each node, while the latter policy (*random*) simply returns a random value for each node. Through the *spread* policy, the node chosen to host the new container depends on the number of containers running on the node, regardless of their status. With the same resources (CPU and RAM), the node that has fewer containers will run the new container. The *binpack* policy, on the other hand, tries to pack the containers in a node, trying to leave free enough space in other nodes to hold containers with higher requirements. Thus, it avoids fragmentation. It is noteworthy that, for all the policies, if all nodes get the same ranking, the election is performed randomly.

7.3.3 Elasticity Rules

As stated earlier, elasticity in EC4Docker is managed by CLUES. This software implements different policies that aim at balancing the trade-off that arises when trying to minimize the waiting time for the jobs (which involves a larger number of available nodes) and the minimization of the infrastructure cost, which involves a reduced number of nodes, which generate a cost in electricity (for physical infrastructures) or in resources (for public cloud providers). In the context of containers, the creation of a container results in less available resources for the

subsequent containers deployed on the same host. Therefore, it is important to submit the containers only when they are really necessary.

The policies implemented by CLUES can be divided in two groups: the policies used to decide when to increase the capacity of the cluster (scale-out) and those used to decide when to decrease the size of the cluster (scale-in). Regarding the scale-out policies, CLUES can interact with the LRMS at two levels. On the one hand, it intercepts the submitted jobs before they reach the LRMS. On the other hand, CLUES also monitors the queued jobs at the LRMS to check if these jobs require additional nodes to be added to the cluster. The policies available are:

- **1:1 start**. For each job launched, if no working nodes are available for its execution, then a new node is deployed. Therefore, the jobs will wait for the deployment of the node before they start their execution.
- **Group-based start**. Every time a new 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 over-provisioning a larger number of nodes, the waiting time of the subsequent jobs will be reduced.

In order to decide when to shutdown a node (scale-in policies), the strategy is to remove a node from the computer cluster when it has been idle for a specified amount of time. The selection of this time depends on the workload of the computer cluster and it is important to achieve a good trade-off between the used resources and the waiting time of the jobs. These are the available strategies:

- **Queued jobs**. Idle working nodes are terminated when there are no pending jobs in the LRMS.
- **Delayed shutdown**. Idle working nodes are terminated after a certain amount of configurable time. This is of interest when using public Clouds that bill by the hour, where idle nodes are kept available for job executions before the hour expires, even if no jobs are available to be executed at the moment.
- **Keeping some nodes always active**. The computer cluster will have a set of nodes deployed waiting for jobs. This way, the computer cluster tries to prevent incoming jobs from waiting while nodes are started.

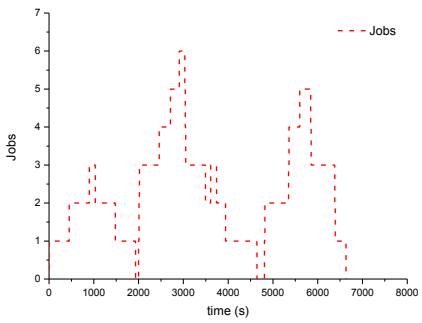
7.4 Case study

In order to assess the effectiveness of the self-managed CVECs deployed with EC4Docker, we present a case study based on a bioinformatics community of users that need to execute several scientific tasks for their research. In particular, the application used is MrBayes [141] (Bayesian Inference of Phylogeny). MrBayes is a program for Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models. MrBayes uses Markov Chain Monte Carlo (MCMC) methods to estimate the posterior distribution of model parameters. MrBayes has several dependencies in order to work properly, like an MPI implementation or the Beagle library. For this, we installed, among others, OpenMPI together with the *gcc* compiler. The case study also analyzes the performance of containers comparing to VMs, trying to prove the advantages of light-weight virtualization in contrast with traditional virtualization based on VMs.

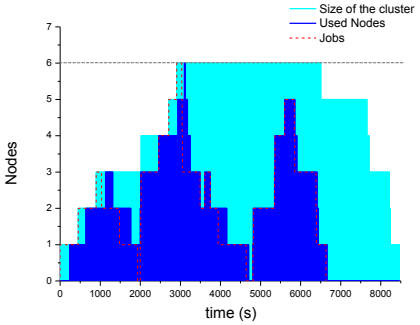
The overall scenario consists of three different executions of the same job pattern submission (represented in Figure 7.4 (a)) on two different computing scenarios, but on top of the very same physical resources. On the one hand, scenario a) involves a container-based virtual computer cluster managed by EC4Docker. All the containers submitted during this execution were limited to 1 CPU and 1 GiB of RAM. On the other hand, scenarios b) and c) involve a VM-based virtual computer cluster deployed on an OpenNebula on-premises Cloud by means of EC3. Each one is configured with different idle times to trigger the scale-in policy: scenario b) is configured with a maximum value for idle nodes of 1800 seconds (30 min.), and scenario c) will power off nodes that were idle for more than 600 seconds (10 min.). Each VM deployed has 1 CPU and 1 GiB of RAM. The VMI employed is based on Ubuntu 14.04 LTS. In this case, two different executions for each configuration were carried out, one in which the software is dynamically deployed on vanilla VMs and the other in which the software (SLURM, OpenMPI, NFS, MrBayes and its dependencies) is pre-installed in the VMI, thus reducing the time for contextualization, i.e., installation and configuration of the software applications. This last option was the one chosen to compare with the execution of the container-based cluster, since Docker containers are created out of pre-configured Docker images.

The physical infrastructure used to deploy the case study is the same for both scenarios for the sake of a fair comparison. It comprised eight physical nodes with a total of 224 cores (28 cores per node), 512 GB of RAM (64 GB of RAM per node) and a shared storage system of 16 TB. For the scenario a) we deployed Docker Swarm and the main node includes the IPFloater tool in order to associate a public IP to each container-based front-end. In scenarios b) and c), an OpenNebula 4.8.0 on-premises Cloud deployment is used.

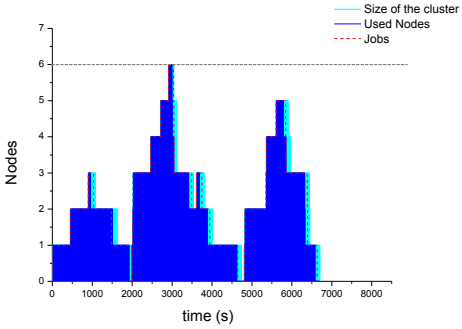
The limit size of the cluster was fixed to 6 nodes in all scenarios. A total of 15 Bayesian tasks with an average duration of 17.5 minutes is executed for each test. The dataset employed is *cynmix.nex* [172], a partitioned dataset consisting of data from four genes and morphology for 30 taxa of gall wasps and outgroups. The number of generations has been fixed to 170.000. The following subsections describe and analyze the obtained results for this case study.



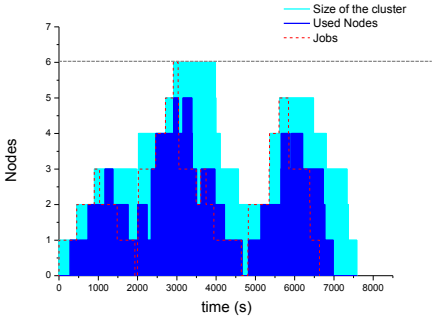
(a) Job pattern submission of the case study.



(b) Execution on VMs (EC3 with idle time for scale-in set to 30 min).



(c) Execution on containers (EC4Docker).



(d) Execution on VMs (EC3 with idle time for scale-in set to 10 min).

Figure 7.4: Execution results for both container-based cluster (a) and VM-based cluster (b) where *light blue* represents the number of virtual nodes deployed, *dark blue* depicts used nodes executing jobs and the *red* dashed line indicates the job pattern submission. The upper *grey* dotted line represents the limit size of the cluster, fixed to six nodes.

7.4.1 Results and discussion

First, we analyzed the time differences in the deployment and contextualization processes for both containers and VMs used in our case study. Table 7.1 shows the average times for the deployment, configuration and execution times for the three scenarios. As we expected, the total average times for both the front-end (FE) and working nodes (WN) were considerably higher with VMs, even if we use a preconfigured VMI with SLURM, NFS, OpenMPI and MrBayes dependencies previously installed. In the last case, it was still necessary to configure the SLURM configuration files, NFS system, and the application MrBayes, that takes an average time of [335-340] seconds in the case of the front-end and [284-285] seconds for the working nodes. Even so, the time consumption during the contextualization process was reduced significantly by starting from a preconfigured VMI (about 65%). However, preparing a customized VMI is not a trivial task so non-experienced users would refrain from using EC3 if they are required to prepare their own VMIs.

In contrast, creating a Docker container image from a Dockerfile is a much easier process than building a VMI. It is necessary to take into account that the container times shown in the table do not consider the time required to generate the container image from the Dockerfile, since this task only needs to be performed once by the administrator or the user. It is worth to point out that the time needed to create the container image is equivalent to the contextualization time employed by a non-preconfigured VM. Moreover, the time to pull the container images if they are stored in Docker Hub has not been included in the table, as this is performed only once, but it took an average of 150 s. in our tests.

	Scen. a)	Scen. b)		Scen. c)	
		Prec.	Non prec.	Prec.	Non prec.
Deployment avg. time	2	35	35	35	35
Active SSH avg. time	1	30	30	30	30
Total avg. time machine ready	3	65	65	65	65
FE contextualization avg. time	0	340	830	335	853
Total avg. time FE ready	3	405	895	400	918
WN contextualization avg. time	0	219	702	220	684
Total avg. time WN ready in LRMS	16	284	767	285	749
Job avg. waiting time	15.25	101	305	209	449
Job avg. execution time	994	1076	1083	1064	1128

Table 7.1: Time analysis, in seconds, for the different phases of the scenarios. Scenario a) refers to the container-based execution, scenario b) refers to the VM-based execution with an idle time configuration of 30 minutes and scenario c) refers to the VM-based execution with an idle time configuration of 10 minutes. In b) and c) the tests are carried out with preconfigured VMIs (Prec.) and without preconfigured VMIs (Non prec.).

Second, we present in Figure 7.4 the results obtained from the execution of the job pattern submission show in Figure 7.4(a). Scenario a) is represented in Figure 7.4(c), scenario b) is shown in Figure 7.4(b)) and scenario c) is addressed in Figure 7.4(d). For the three executions, we have used conservative elasticity policies to ensure the minimum costs for the infrastructure in terms of energy and resources consumption. Thus, CLUES has been configured to power on nodes according to the *1:1 start* strategy, i.e. when a job arrives to the LRMS and there is no available node to execute it, a virtual node is deployed. On the other hand, the power off policy selected was *delayed shutdown*, destroying nodes when they are idle for 2 minutes, for the scenario a) execution, 30 minutes for the scenario b) execution and 10 minutes for the scenario c). The differences in time for powering off a node are based on the time that a new virtual node needs to be ready for task execution (16 seconds in case of a container node and 285 seconds in average for a VM node). Scenarios b) and c) involve the same execution but the variations in the idle time to trigger the scale-in policy introduced differences in the behaviour of the cluster, as it can be appreciated in the figures.

Based on the results represented in Figure 7.4 we can highlight that the container-based cluster deployed in scenario a) fits almost perfectly to the workload of the computing cluster. Indeed, containers only take a few seconds to be ready to execute the jobs of the cluster since the contextualization process is not required, and starting a container is faster than booting a VM. Therefore, the average time that a job is queued up at the LRMS, i.e. in PENDING state, does not exceed 15 seconds.

In contrast, in scenarios b) and c) we can easily denote the differences deploying a node, that takes an average of 285 seconds to be ready and detected by the LRMS as an eligible node to execute jobs. This situation is represented in Figure 7.4(b),(d) in light blue, and covers the time needed to deploy a new VM, obtain SSH access to it and contextualize the job execution environment. For example, in Figure 7.4(b), for the first job this requires the initial 280 seconds of the execution. This situation is repeated for the subsequent jobs that arrive to the LRMS, when no available nodes are in the cluster. However, once the cluster is fully deployed, new jobs do not need to wait for additional nodes to be deployed. Instead, they just wait for other tasks to finish. This fact helps reducing the total average time of jobs waiting in the LRMS queue, which is 101 seconds. However, the resources are not properly exploited, because most of the nodes were idle a long period of time.

This situation can be better addressed by reducing the idle time allowed for nodes as it is done in scenario c). In this case, the available resources are better used, but the total time of execution increases (6989 seconds) like the job average waiting time (209 seconds) in contrast with the other two scenarios a) and b). However, despite the differences in the time required to provision new nodes in all scenarios, the total execution time in scenarios a) and b) is very similar. Container-based

execution (scenario a)), requires 6579 seconds to complete all the submitted jobs while VM-based execution (scenario b)) takes 6661 seconds. Note that, on the one hand, scenario b) is significantly impacted by requiring to deploy additional nodes (VMs) at the beginning but the deployment and configuration of the nodes is produced concurrently. However, once the new nodes are up and running, jobs can be processed on a first-come-first-served basis. On the other hand, scenario c) is also impacted by the initial deployment of new VMs. However, the infrastructure does not maintain the nodes active and more time dedicated to deploy nodes os needed during the execution. These facts reveal that in a VM-based execution, increasing the time that nodes are idle, reduces the total execution time (no extra time is dedicated to deploy nodes to scale out) at the expense of wasting computational resources. Also, if the idle time to trigger a scale in operation is reduced, the total time of execution increases (due to the extra time required to provision additional nodes, which increases the job waiting time) but the computational resources are better used.

It is of special relevance the differences in the average time for a single job execution, that is an 8.2% faster in the containers deployed in the scenario (a) (994 seconds), than in VMs ([1064-1128] seconds). This fact confirms the higher overheads in CPU and memory that VMs suffer, comparing with the light-weight virtualization introduced by Docker containers.

Figure 7.4 does not represent the time required to deploy and configure the front-end of the cluster. This data is presented in Table 7.1, where for a container-based cluster this task only requires deploying a container in Docker Swarm and requesting a redirection to IPFloater (3 seconds). Meanwhile, for a VM-based cluster, this task involves the creation of a new VM in OpenNebula, wait until the SSH of the VM is active and complete the contextualization process ([895-918] seconds in average for a non-preconfigured VMI and [335-340] seconds for a preconfigured VMI).

All the analyzed results suggest that containers are a proper solution to execute groups of short HTC tasks, like BoT early outweighs the execution time of the tasks. HPC tasks can also benefit from the reduced overheads that arise when using containers. In contrast, for longer tasks, contextualization time may become negligible with respect to the total execution time and, therefore, these tasks can take advantage of the unlimited resources offered by Cloud Computing platforms in the shape of VMs.

7.5 Conclusions and Future Work

This paper has analyzed the feasibility of using Docker containers to support the creation of virtual elastic computer clusters for the execution of scientific applications. These clusters maintain the very same interfaces for end users but benefit from the reduced overheads introduced by containers. For this, we introduced the open-source EC4Docker tool to support the deployment of such clusters on a Container Orchestration Platform managed by Docker Swarm.

We have demonstrated the feasibility of adopting containers to execute scientific applications, introducing two main advantages when compared to traditional VMs: i) the low deploying times for new working nodes, and ii) potential reductions in the overhead caused by VMs in CPU, memory and storage, offering near-native performance. Moreover, from the discussed case study, we can conclude that container-based virtual clusters are an appropriate solution for the execution of short HTC tasks.

Future work involves the automatization of the generation of the container images that EC4Docker uses to deploy the cluster. Currently, the administrator or the users need to generate their own images including the Dockerfile provided with EC4Docker in order to deploy their own applications in the container cluster environment. A service will be implemented to facilitate this process for non-experienced users.

Chapter 8

EC3aaS: EC3 as a Service

"I have always wished for my computer to be as easy to use as my telephone; my wish has come true because I can no longer figure out how to use my telephone."
Bjarne Stroustrup, C++ creator (1990).

EC3 as a Service (EC3aaS), is a web service offered to the community to facilitate the usage of EC3 to non-experienced users. Anyone can access the website [39] and try the tool by using the user-friendly wizard to easily configure and deploy Virtual Elastic Clusters on multiple Clouds. The service does not require any account to use it. The user only needs to choose the Cloud provider and provide its credentials to allow EC3 to provision VMs from the underlying Clouds on behalf of the user. The website features the documentation of EC3. Therefore, this service facilitates the usage of EC3 to non-experienced users.

EC3aaS has been developed with Bootstrap [30], a free and open-source framework for creating websites and web applications. This way, the underlying technologies used are HyperText Markup Language (HTML) 5.0, Cascading Style Sheets (CSS) and JavaScript.

8.1 Configuration and Deployment of a Cluster

In order to configure and deploy a Virtual Elastic Cluster using EC3aaS, a user accesses the homepage and selects "Deploy your cluster!" (Figure 8.1). With this action, the web page will show different Cloud providers supported by the web interface version of EC3. Notice that not all the Cloud providers supported by EC3 appear in the website, only the most important providers in the field of research are currently supported by the web version. Users that want to use another supported

Figure 8.1: Homepage of EC3aaS.

Cloud provider, such as Microsoft Azure or Google Cloud Engine, are encouraged to use the CLI interface.

The first step, then, is to choose the Cloud provider where the cluster will be deployed (Figure 8.2). When the user chooses one of the offered providers (Amazon EC2, OpenNebula, OpenStack or EGI FedCloud), a wizard pops up (Figure 8.3). This wizard will guide the user during the configuration process of the cluster, allowing to configure details like the operating system, the characteristics of the nodes, the maximum number of nodes of the cluster or the pre-installed software packages. Specifically, the general wizard steps are:

1. **Provider account:** Valid user credentials are required to access to the resources of the Cloud provider chosen. Additionally, in the OpenNebula, OpenStack and EGI FedCloud wizards, the endpoint of the server is also required. The use of temporary credentials is recommended.

2. **Operating System:** the user can choose the OS of the cluster, by using a select box where the most common OS are available or by indicating a valid AMI/VMI identifier for the Cloud selected.

3. **Instance details:** the user must indicate the instance details, like the number of CPUs or the RAM memory, for the front-end and also the working nodes of the cluster. In case of using Amazon EC2, a select box is provided with the instance types offered by this provider. In case of using EGI FedCloud, the user must indicate the instance type desired from the ones available in the endpoint selected.

4. **LRMS Selection:** the user can choose the Local Resource Management System preferred to be automatically installed and configured by EC3. Currently, SLURM, Torque, Grid Engine and Mesos are supported.

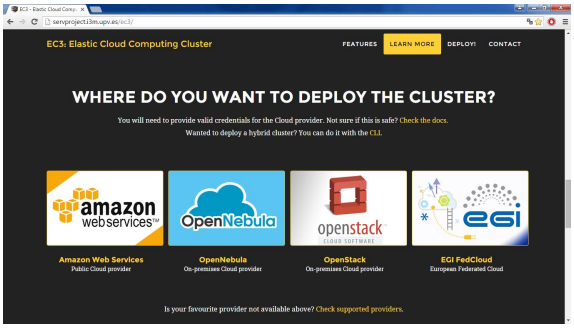


Figure 8.2: List of Cloud providers supported by EC3aaS.

5. **Software Packages:** a set of common software packages is available to be installed in the cluster, such as Docker Engine, Spark, Galaxy, OpenVPN, BLCR, GNUPlot, Tomcat or Octave. EC3 can install and configure them automatically in the contextualization process. If the user needs another software to be installed in his cluster, a new Ansible recipe can be developed and added to EC3 by using the CLI interface.

6. **Cluster's size:** the user can introduce the maximum number of nodes of the cluster, without including the front-end node. This value indicates the maximum number of working nodes that the cluster can scale.

7. **Resume and Launch:** a summary of the chosen configuration of the cluster is showed to the user at the last step of the wizard, and the deployment process can start by clicking the *Submit* button.

Notice that not all the Cloud providers need or allow to configure the same data. Because of that, the wizards are customized for each Cloud provider. For example, in the EGI FedCloud, a new wizard step appears, in order to collect the data about the MyProxy account of the user, a particularity of this Cloud provider. At the same time, the credentials needed to contact with the Cloud providers have different formats, depending on the provider selected. This is another reason why the wizards are customized for each one. Moreover, the instance options (CPU, RAM and disk) are presented different for each provider. Since Amazon EC2 and EGI FedCloud offer a predefined list of instance types, OpenNebula and OpenStack let the user indicate the values of CPU and RAM for the instance.

Finally, when all the steps of the wizard are filled correctly, the submit button starts the deployment process of the cluster. Only the front-end will be deployed, because the working nodes will be automatically provisioned by EC3 when the workload of the cluster requires them. When the virtual machine of the front-end

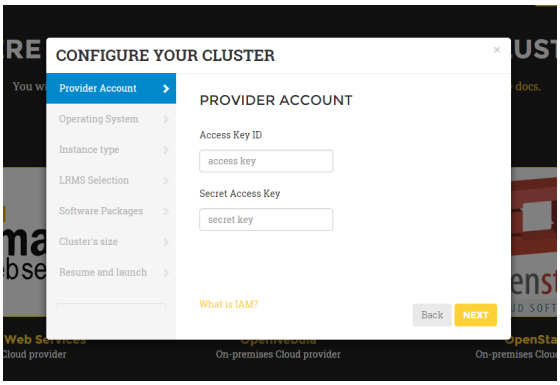


Figure 8.3: Wizard to configure and deploy a virtual cluster in Amazon EC2.

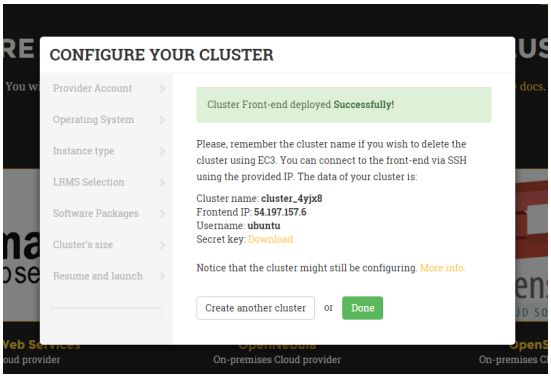


Figure 8.4: Information received by the user when a deployment succeeds.

is running, EC3aaS provides the user with the necessary data to connect to the cluster (Figure 8.4) which is composed by the username and password to connect to the cluster, the front-end IP and the name of the cluster. The user must keep this data during the lifetime of the cluster, since it is used also to terminate it. The cluster may not be configured when the IP of the front-end is returned by the web page, because the process of configuring the cluster is a batch process that takes several minutes, depending on the chosen configuration. However, the user is allowed to log in the front-end machine of the cluster since the moment it is deployed. To know if the cluster is configured, the command *is_cluster_ready* can be used. It will check if the configuration process of cluster has finished.

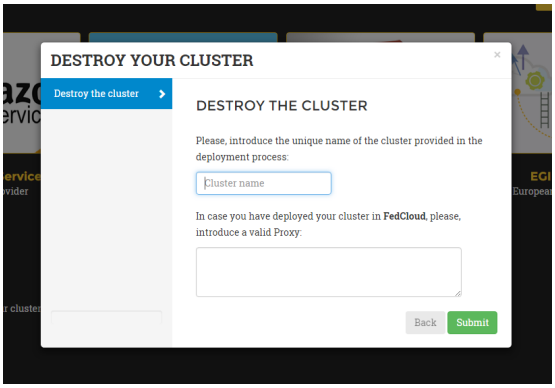


Figure 8.5: Wizard to delete a cluster.

Notice that EC3aaS does not offer all the capabilities of EC3, like hybrid clusters or the usage of spot instances. Those capabilities are considered advanced aspects of the tool and are only available via the CLI.

8.2 Termination of a cluster

To delete a cluster the user only needs to access the EC3aaS webpage, and click on the trash icon (situated underneath the provider buttons in the deployment section of the website) and indicate in the wizard (Figure 8.5) the cluster name provided to the user in the deployment phase. The cluster name is a string composed by the word *cluster* followed by a random string of five characters (including numbers and letters). This cluster name is unique and allows EC3 to identify the cluster of the user without using an user account. Moreover, in case the user has developed the cluster in the EGI FedCloud, a valid proxy will be required in order to destroy the cluster.

When the process finishes successfully, the front-end of the cluster and all the working nodes had been destroyed and a message is shown to the user informing the success of the operation. If an error occurs during the deleting process (for example, the indicated cluster name does not exist), an error message is returned to the user.

8.3 Other available materials

In addition to the webpage together with the EC3aaS service, where its features, architecture, etc. are explained in depth, EC3 also has several public materials available on the Internet (Figure 8.6). The source code of EC3 is available to the community in its GitHub page [40], under an Apache 2 license. That repository includes both the current developments of EC3, and the documentation about it. Anyone can easily clone the repository and use the EC3 CLI interface.

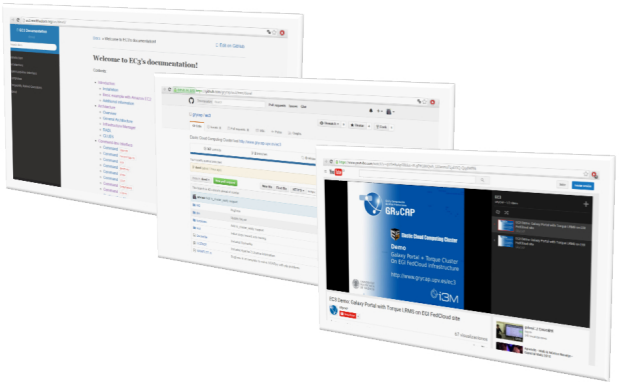


Figure 8.6: EC3 webpages in ReadTheDocs, GitHub and Youtube.

Moreover, a Docker image that includes the installation of EC3 is available in the Docker Hub repository [51]. This image enables a user to easily deploy a container to use EC3 commands to deploy and configure virtual clusters. The documentation of EC3 can be also found in ReadTheDocs [38], a well-known repository that publicly hosts software packages's documentation.

Finally, EC3 offers several video-tutorials avaliable in the Youtube Channel of the Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) research group [10]. The video-tutorials cover examples using the EC3 CLI interface and also the web interface offered by the EC3aaS version. All the links to those materials are available in the *Learn more* section of the EC3's webpage.

8.4 Use Cases

The usage of EC3aaS has facilitated the access to cloud computational resources for researchers of fields devoid of expert computer science skills. For example, in the Bijvoet Center for Biomolecular Research (the Netherlands)[1], the researchers that have developed software tools for the modelling of biomolecular complexes, are using EC3 to expose them as virtualized services in the EGI FedCloud platform, inside the INDIGO-DataCloud project [201].

In particular, the tool High Ambiguity Driven protein-protein DOCKing (HADDOCK)[2], an integrative, information-driven approach for modelling biomolecular complexes, is offered nowadays through a user-friendly web interface deployed both on local sources and distributed grid resources. The case study for HADDOCK Portal involves the virtualization of the HADDOCK web portal and the required computational infrastructure underneath it using EC3. The aim is to be less dependent on local hardware and to facilitate the deployment of the software at other sites.

Moreover, the tools PowerFit[3] and DisVis[4] are two other softwares recently developed by this research group, for automatic rigid body fitting of biomolecular structures in Cryo-EM densities and for visualization and quantification of the accessible interaction space of distance restrained binary biomolecular complexes, respectively. PowerFit and DisVis case studies involve the deployment of these softwares into VMs with all their dependencies, so that the end user will not deal with the installation procedure, and EC3 seems to be a proper solution to achieve it.

As we have presented, chapter 5 reflects another use case of the EC3aaS service, where researchers from the National Institute for Agricultural Research (INRA)[5], in France, whose research is centered in the field of biodiversity, use EC3 to deploy a Galaxy cluster to perform their studies.

These are examples of the usage of EC3 in areas of knowledge different from computer science. The next chapter lists the projects where EC3 has been integrated and also its usage in the academia.

[1] Bijvoet Center for Biomolecular Research: `http://bijvoet-center.eu/`
[2] HADDOCK software: `http://haddock.science.uu.nl/services/HADDOCK2.2/`
[3] PowerFit software: `https://github.com/haddocking/powerfit`
[4] DisVis software: `https://github.com/haddocking/disvis`
[5] INRA: `http://www.inra.fr/`

Chapter 9

Discussion and Results

"Afronta tu camino con coraje, no tengas miedo de las críticas de los demás. Y, sobre todo, no te dejes paralizar por tus propias críticas." Paulo Coelho, Maktub (1994)

Throughout this thesis, different tools have been produced that contribute significantly to the state of the art of HPC on Cloud platforms. This thesis has produced technology that is having an impact on the communities that require cluster-based computing, like datacenters and scientific communities, which are now able to easily outsource part of their computing power to Cloud platforms and to take advantage of the virtual hybrid elastic clusters capabilities in order to reduce the costs of the computing. Moreover, the ability to present these capabilities in a friendly way to the users, has facilitated the usage of Cloud computing to researchers from other areas of knowledge, that are not experts in computing science.

Therefore, the author has been pleased with the collaboration with different projects and tools, enriching both parties. On the one hand, contributions to relevant tools of the research group where the PhD student was integrated have been carried out. These contributions have helped to improve existing tools and also have provided experience and knowledge to the researchers involved. On the other hand, the main tool developed in this thesis has been integrated into several national and international projects as part of the solution to larger problems. This outlines the quality of the solution developed in this thesis.

The most significant results of the research carried out in this project have been published in several conferences and journals of remarkable relevance inside the area of knowledge. These publications denote the quality of work performed, and contribute to the dissemination of its results inside the community. Next

Figure 9.1: Flyers of EC3 used for dissemination purposes.

subsections summarize the tools developed during the project and the principal contributions to the area. Also, research projects involved in the works developed in this project are presented. Finally, publications and contributions in national and international conferences are listed, as well as published papers in journals of relevance in the area.

9.1 Software and tools developed

- **Elastic Cloud Computing Cluster (EC3):** this tool is the main development of the project. EC3 allows its users to deploy virtual hybrid elastic clusters on top of IaaS Clouds. These clusters are self-managed since they have the ability to auto-scale its size (in terms of number of nodes) according to their workload. Also, they are automatically configured, according to the requirements of the user application. Moreover, as we have discussed in this document, these clusters are cost-efficient, since they can be deployed using spot instances that significantly reduce the total cost of executions [13]. Currently, EC3 supports the deployment of virtual clusters in OpenNebula [175], Amazon EC2 [12], OpenStack [150], OCCI [129], EGI FedCloud [70], LibCloud [18], Docker [60], Microsoft Azure [130], Google Cloud Engine [83] and LibVirt [111], thanks to the IM, that is used in EC3 to deploy resources in the Cloud. EC3 supports the following LRMS: SLURM [100], Torque [2], SGE [168], Mesos [88], and HTCondor [91], covering several of the most used resource management tools. Recipes for several software packages are offered together with the EC3 source code, like Galaxy, Docker, OpenVPN, Octave or BLCR, among others.

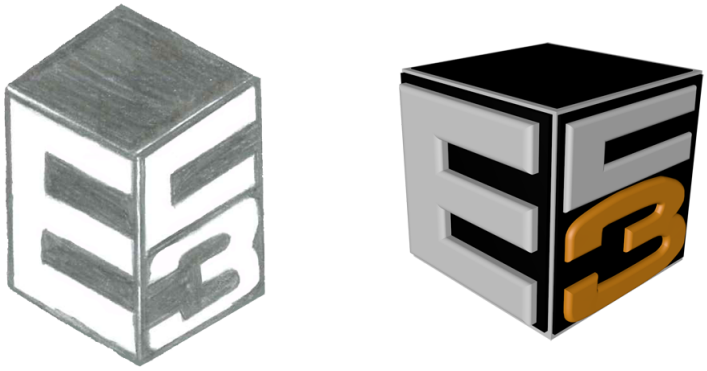


Figure 9.2: EC3 logo: sketch (left) and final version (right).

Branding activities have been performed with EC3 to publicise and promote it inside the research community. Therefore, EC3 has its own logo (see Figure 9.2) and even some merchandising products, like T-shirts (see Figure 9.3) and flyers (Figure 9.1), that have been used in the conferences where the author has participated.

- **EC3 as a Service (EC3aaS):** this is a web-based interface for EC3 for users to easily deploy virtual elastic clusters without any installation and by using just a web browser. For researchers that are not specialized in computing science, but they need to use virtual resources to perform their research, this web-interface is the best choice. It was deeply described in chapter 8.

- **Checkpointing Manager (*ckptman*):** this is a tool to automate the checkpointing in spot instances. It works together with the BLCR [65] checkpointing tool, which is used by *ckptman* to perform the checkpointing operations, and with the IM [33], which is used to know the current state of the cluster. *ckptman* also implements two checkpointing algoritms (`HOUR` and `THRESHOLD`) to determine the best moment to perform a checkpoint of an application without saturating the system. It has been integrated with EC3 by developing an Ansible recipe that installs and configures the tool in a cluster. An example of use of this tool has been presented in chapter 4.

Figure 9.3: Design of the t-shirt of EC3 used in the EGI Community Forurm 2015.

9.2 Publications and contributions

As a result of the work carried out in this PhD project, the author has participated in several international and national conferences in the area. Contributions to the state of the art through publications in journals have also been performed. All of the contributions obtained so far from this thesis are detailed below.

National conferences:

- 1º Encuentro de estudiantes de doctorado de la Universitat Politècnica de València (June 12 2014, Valencia, Spain): in this event we participated with a poster titled as the doctoral thesis "Computación Científica de Altas Prestaciones sobre Plataformas Cloud Híbridas" [93]. The poster was chosen by the jury among more than 120 posters as one of the top ten posters of the conference. This gave to the author the chance of an oral exhibition of the work.

International conferences:

- 3rd IEEE International Conference on Cloud Computing Technology and Science, November 29-December 1, 2011, Athens, Greece): In this conference, we presented a paper called "Combining Grid and Cloud Resources for Hybrid Scientific Computing Executions" that formed part of the master's project of the PhD candidate. This conference is ranked as C in the CORE ranking [52]. The complete reference of the publication can be found in [35].

- Conference "8th Iberian Grid Infrastructure Conference", IBERGRID 2014 (September 8-10, 2014, Aveiro, Portugal): in this conference we participated with an article that described the early versions of the main tool being developed in the thesis, EC3. The paper, entitled "Virtual Hybrid Elastic Clusters in the Cloud" [37] described the main characteristics of the architecture developed for the deployment and management of virtual hybrid elastic clusters (consisting of resources from different Cloud providers). It also presented the results obtained for the execution of a use case in which a parallel computationally intensive gyro kinetic plasma turbulence application was involved.

- Conference "10th IEEE International Conference on e-Science" (October 20-22, 2014, Guarujá, Brazil): in this indexed conference with a CORE A ranking [52], a poster summary of the thesis, entitled "High Performance Scientific Computing over Hybrid Cloud Platforms" was presented. The author also participated in the lightning session, with a two-minute short talk about the work done. This congress was attended thanks to a grant obtained in competitive tendering.

- Conference "8th IEEE International Conference on Cloud Computing" (June 27 - July 2, 2015, New York, USA): to this congress we participated with a short paper, entitled "Towards Migrable Elastic Virtual Clusters on Hybrid Clouds" [36] describing the research work carried out in the CLUVIEM project, a project in which the PhD candidate was actively involved, because it is perfectly aligned with the contents of the thesis. The work presented during the conference, where we also will come with a poster summary of the work performed. This is an indexed conference with a CORE B ranking [52].

Demonstrations:

- Conference "EGI Community Forum 2015" (November 10-13, 2015, Bari, Italy): in this conference we participated with a demo of the tool developed in the thesis, EC3 , which was named "Deploying Cost-Efficient Virtual Elastic Clusters across Multi-Clouds" [59]. The demo was honored by the conference organizers with the "Best Demo Award".

Journal articles:

- Lecture Notes in Computer Science (LNCS): in the volume 7851 of the series Lecture Notes in Computer Science, the author, together with other researchers, published a paper called "A Service-Oriented Architecture for Scientific Computing on Cloud Infrastructures" [136], that was selected from the conference 10th International Meeting on High-Performance Computing for Computational Science (VECPAR 2012). The congress is an indexed conference with a CORE B ranking [52] (when the work was published). This work formed part of the master's project of the PhD candidate.

- Future Generation Computer Systems: the main developments of this PhD project were published in this journal, under the title "Self-managed Cost-efficient Virtual Elastic Clusters on Hybrid Cloud Infrastructures" [41]. This journal has a impact factor of 2.786 and it is classified in the first quartile (Q1) of the JCR for the topic "Computer Science, Theory & Methods" in 2016 (when the paper was published).

- Journal of Grid Computing: together with researchers from INRA, UMR BioGeCo and INRIA, a paper entitled "eScience with a galaxy web-service connected to an elastic cluster in the cloud for computational biodiversity" was sent to the Special Issue on Science Gateways of this journal. It is now under its second review. The Journal of Grid Computing has a impact factor of 1.507 and it is classified in the first quartile (Q1) of the JCR for the topic "Computer Science, Theory & Methods" and in the second quartile (Q2) for the topic "Computer Science, Information Systems" in 2016 (when the paper was submitted).

- Journal of Systems and Software: the motivation to work with containers together with the developments to produce EC4Docker are covered in the paper "Container-based Virtual Elastic Clusters" that was sent to the Journal of Systems and Software. It is now under its second review. The Journal of Systems and Software has a impact factor of 1.424 and it is classified in the second quartile (Q2) of the JCR for the topic "Computer Science, Theory & Methods" (when the paper was submitted).

9.3 Projects where EC3 is integrated

EC3 has been integrated in the developments of several international and national projects. Those projects are:

- **INDIGO-DataCloud [95]:** INtegrating Distributed data Infrastructures for Global ExplOitation is a project funded under the Horizon2020 framework program of the European Union, which has received a budget of 11M € and integrates the collaboration of 26 partners. It aims at developing a data and computing platform targeting scientific communities, deployable on multiple hardware and provisioned over hybrid (private or public) e-infrastructures. By filling existing gaps in PaaS and SaaS levels, INDIGO-DataCloud will help developers, resources providers, e-infrastructures and scientific communities to overcome current challenges in the Cloud computing, storage and network areas. EC3 is being currently integrated in the INDIGO-DataCloud european project, together with the developed CLUES plugins for Mesos and HTCondor.

- **EGI-Engage:** The EGI ENGAGE [68] project (Engaging the Research Community towards an Open Science Commons) started in March 2015, co-funded by the European Commission for 30 months, as a collaborative effort involving more than 70 institutions in over 30 countries. EGI-Engage aims to accelerate the implementation of the Open Science Commons by expanding the capabilities of a European backbone of federated services for compute, storage, data, communication, knowledge and expertise, complementing community-specific capabilities.

  In January 2016, the EGI Inspire Newsletter (Issue 22) published an article about EC3 [1]. Moreover, EC3 is being integrated with the EGI Long Tail of Science (EGI LToS) [69], an easy-to-use platform for researchers to access compute, storage and applications services in order to carry out data/compute intensive science and innovation.

- **EUBra-BIGSEA:** The project EUrope-BRAzil Collaboration on BIG Data Scientific REsearch through Cloud-Centric Applications aims at providing services in the Cloud for the processing of massive data coming from highly connected societies, which impose multiple challenges on resource provision, performance, Quality of Service and privacy. The EC3 tool is being used in this project as the main solution to deploy the virtual infrastructure used to perform massive data analysis. EUBra-BIGSEA is funded in the third coordinated call Europe–Brazil, by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 690116, together with the Ministry of Science, Technology and Innovation (MCTI) of Brazil.

- **CLUVIEM:** The project Migrable Elastic Virtual Clusters on Hybrid Cloud Infrastructures, with reference TIN2013-44390-R aims at the usage of virtual hybrid elastic clusters. The general objectives of this project are the elastic management of hardware infrastructures, the management of hybrid elastic virtual clusters, the migration of virtual clusters and workloads, the energy efficiency and the application use cases to showcase the benefits of the approach. The main objectives of this national project are perfectly aligned with the contents of this thesis project, so all the developments explained in this document are part of the solution of CLUVIEM. This project is financed by the Spanish Ministry of Economy and Competitiveness.

9.4 EC3 in the academia

EC3 has also been integrated in academia. Specifically, in the Master's Degree in Parallel and Distributed Computing (MUCPD)[1], the Master's Degree in Computer Engineering (MUIINF)[2], the Master's Degree in Information Management (MUGI)[3], and the Master's Degree in Big Data Analytics (MUBDA)[4], all of them imparted at the "Universitat Politècnica de València" in the academic course 2015-2016. Those subjects are specially related to distributed computing and cloud computing, such as *Servicios en la Nube*. This subject introduces students to Cloud Computing with a wide panoply of services and tools that enable to leverage this technology in many scenarios. One of the tools addressed is EC3 for its ability to provide multi-purpose cluster-based computing on top of Cloud infrastructures. Students learn the intricacies of computing in the Cloud and find out an easy-to-use interface to provision their own clusters on AWS.

Moreover, the developments in this thesis project have derived some final degree projects and final master projects, briefly summarised as follows:

- Development of a Representational State Transfer (REST) API for EC3 to ease the integration for other developers.
- Development of an elasticity plugin for CLUES integrated in Docker Swarm, to achieve an elastic back-end for container management.

9.5 Contributions to other projects and tools

During the development of this PhD thesis, the author has collaborated with software projects carried out by the research group where she was integrated, the GRyCAP. Those projects and the contributions are detailed below.

- **CLUES:** CLUster Energy Saving CLUES [9] 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. CLUES is integrated with the physical/virtual infrastructure by means of different plug-ins. Those plug-ins connect with the underlying resource management system in order to obtain the info about the infrastructure so that nodes can be powered on/off using green techniques. The PhD candidate has been collaborating with the developments of CLUES2 by the implementation and

[1]MUCPD: `http://www.upv.es/titulaciones/MUCPD/`
[2]MUIINF: `http://muiinf.webs.upv.es/`
[3]MUGI: `http://mugi.webs.upv.es/`
[4]MUBDA: `http://bigdata.inf.upv.es/`

testing of two new plug-ins for the SLURM and Mesos (including Chronos and Marathon schedulers support) resource management tools. The use of these LRMSs is currently widespread in the community, so the development of such plug-ins expand the range of action of CLUES2.

- **Infrastructure Manager (IM):** IM [33] is a tool that deploys complex and customized virtual infrastructures on IaaS Cloud deployments (such as AWS, OpenStack, etc.). It eases the access and the usability of IaaS clouds by automating the VMI selection, deployment, configuration, software installation, monitoring and update of the virtual infrastructure. It supports APIs from a large number of virtual platforms, making user applications cloud-agnostic. In addition it integrates a contextualization system to enable the installation and configuration of all the user required applications providing the user with a fully functional infrastructure. The author of this thesis has been actively involved in this software tool, testing its functionality and also adding new features to it, like the support for spot instances in AWS. Also, she has been involved in the developments needed to support the definition of different types of working nodes inside an RADL, thus allowing the creation of hybrid clusters.

- **Elastic Cluster for Docker (EC4Docker):** this tool [71] provides a simple elastic cluster whose nodes are containers. There exists a front-end that can be accessed by ssh, and the internal working nodes are powered on or off according to the needs (if the nodes are not used for a while, they are powered off, and they are powered on if they are needed). However, it was primarily conceived to work in a single machine. The author of this PhD has been working on this tool, adapting it to use Docker Swarm, a simple tool which controls a cluster of Docker hosts and exposes it as a single virtual host. The tool now can take advantage of a pool of resources, creating a container-based virtual cluster on top of a physical cluster, not in the same machine.

Chapter 10

Conclusions

"Incluso un camino sinuoso, difícil, nos puede conducir a la meta si no lo abandonamos hasta el final."
Paulo Coelho, Maktub (1994)

In this chapter, to conclude the Thesis, a reflection of the work carried out is presented. Also, the main ideas for the future work are pointed out. Finally, the fundings that have made possible the development of this project are enumerated.

10.1 Conclusions

Nowadays, in many areas of knowledge, eScience depends on grid and cloud computing solutions for computationally intensive processes involving large data sets. Technology deployments behind these applications requires high skills in computer science. However, one goal of eScience is to promote and encourage collaborative works in application domains, by sharing data sets, pipelines, and access to material generated by those processes. This thesis has focused on facilitating high performance scientific application execution over hybrid cloud platforms.

This thesis has presented the developments towards migratable self-managed cost-effective virtual elastic clusters on hybrid Cloud infrastructures. So far, the developments of this vision are represented on the open source EC3 [39] tool, which abstracts the details of cluster deployment and configuration over hybrid Clouds and enables to provision virtual hybrid elastic clusters that span across public Clouds (AWS, Microsoft Azure and Google Compute Engine) and on-premises CMPs (OpenNebula, OpenStack and any other OCCI-compliant software), featuring checkpointing capabilities and spot instances support. Moreover, the system is

able to self-configure these resources to support the execution of the applications, and to adapt the cluster's size and topology to the dynamic characteristics of the application and the needs of the datacenter. It also supports OCCI, that enables the user to provision resources from EGI FedCloud [70], one of the largest scientific computing platforms. Finally, migration features have been presented to perform (partly or fully) virtual cluster migration, integrated with EC3, that provide an unprecedented degree of flexibility to virtual elastic clusters.

Another point addressed by this thesis was a detailed analysis of the advantages of containers compared to VMs as a viable alternative to deploy virtual clusters. Important Cloud providers, such as AWS, are beginning to offer containers [14] as a solution for deploying applications in the Cloud. Thus, this issue was of special interest. EC4Docker was presented as a tool to facilitate the execution of user applications inside a container-based virtual cluster deployed on top of physical resources, trying to facilitate the user experience.

Moreover, the author has implemented a service [39] that enables the automatic deployment of virtual elastic clusters on the cloud through an user-friendly web interface. This service facilitates the usage of EC3 to non-experienced users. EC3 also supports a designed branding as a product. In addition to the web application, EC3 has its own logo, a repository in Github [40], where its source code is available under an Apache 2.0 license, and video-tutorials of the CLI and web GUI interfaces are available in the Youtube channel [10] of the GRyCAP research group.

The benefits of the proposed solutions have been exemplified by the execution of several case studies that involve high performance scientific applications. For example, a computationally intensive gyro kinetic plasma turbulence application has been executed in a virtual hybrid elastic cluster, that demonstrates the feasibility of this type of architectures into the scientific community. Another case study was executed that involved a scientific application to perform the nonlinear dynamic analysis of buildings executed in a hybrid virtual elastic cluster across an on-premises OpenNebula Cloud and AWS. The results show the ability of the clusters to adapt their size to the workload and the automatic Cloud bursting to a public Cloud. Also, the checkpointing algorithms implemented by *ckptman* have been tested under a real scenario using real workloads and real spot prices. The results of this study show the significant savings that suppose the use of spot instances in contrast to on-demand instances, with the increased resilience that arises from periodic and automatic checkpointing of the jobs.

A working example of a Galaxy interface, that submits jobs to an elastic queue based on a virtualized cluster that automatically deploys and undeploys resources as needed, has been implemented in the domain of computational biodiversity, allowing the research for patterns of biodiversity, from molecular data sets of freshwater diatoms. Researchers in the area of biodiversity did not need a deep knowledge of the underlying technologies to benefit from it, because EC3 hides

the technical aspects of the underlying infrastructure. Finally, a comparative case study of both solutions presented in this thesis to deploy virtual clusters, containers and VMs, was performed by using a bioinformatics application for Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models (MrBayes). The study allowed to determine which type of jobs fits better for a container solution an which of them for a VM-based solution.

We can conclude affirming that all the objectives proposed at the beginning of this thesis have been successfully covered. Moreover, the results obtained during the thesis have been timely published in relevant journals and conferences of the area. The thesis has produced the technology required to turn a datacenter into a hybrid infrastructure that can dynamically provision from public Cloud providers, transparently to their users, which can take advantage from its benefits without a deep knowledge.

10.2 Future work

Future work involves the maintenance of the current software and services offered to the community as well as attracting new users to EC3. To achieve that, different new developments are proposed. Regarding spot instances, we plan to adapt the current algorithms of *ckptman* to other public Cloud providers with similar features. This is the case of the preemptible VM instances provided by the Google Cloud Platform.

It is also of our interest to improve the migration capabilities of EC3. On the one hand, the advent experimented by light-weight virtualization in recent years has accelerated the developments regarding container technology. Furthermore, applications running in containers can be migrated across infrastructures by using checkpointing techniques applied to containers. Recent works in the literature aims at performing checkpoint and restore operations with containers, based on previous developments in tools like OpenVZ [133] and most recently CRIU [54]. This early works open the possibility for introducing migration of container-based workloads across multiple Clouds. On the other hand, EC3 supports the usage of containers to deploy applications by using Docker and Mesos. Moreover, CLUES is able to detect when the jobs reach the LRMS and no available nodes of the cluster exist to attend the job requirements, thanks to the plugin specifically developed during this thesis for Apache Mesos. This plugin is able to detect jobs submitted directly to Mesos, in addition to jobs submitted to Chronos and/or Marathon. Thus, this plugin provides CLUES with the ability to detect jobs at three levels at the same time, completely controlling the infrastructure and providing an elastic Mesos platform. The proposed solution is based on the usage of EC3 to clone the original infrastructure and the usage of CRIU and Docker to checkpoint the containers on where the application is running and restore them in the new cluster

destination. The first developments to support Docker checkpointing operations are available online [61], but still have not been merged upstream in the official Docker repository.

Also, the integration of a REST API for EC3 will be considered. REST, REpresentational State Transfer, is a type of architecture for web development that relies entirely on the HTTP standard. REST allows us to create services and applications that can be used by any device or client who understands Hypertext Transfer Protocol (HTTP). This new feature will provide a new way to use EC3, approaching it to the current most used software technologies that usually offer this type of API to interact with the software (like Mesos or Marathon currently do).

Moreover, we want to increase the list of offered Cloud providers in EC3aaS. Since EC3 currently supports the deployment of virtual clusters in more Cloud providers than the ones offered by EC3aaS (such as Google Cloud Platform or Microsoft Azure), the idea is to offer the opportunity to deploy these virtual elastic clusters in those platforms also through the website. This involves the development of new wizards customized for those new providers. This proposal will help attracting new users to EC3. Another measure that can be performed is preparing more video-tutorials of EC3 to be offered in the Youtube channel of the research group. This type of material is always a highly valued support for the users. Thus, new video-tutorials will cover examples using the EC3 CLI interface to deploy hybrid clusters and also examples with other providers using EC3aaS.

Finally, after the good results obtained with EC3aaS, we plan to develop a web interface for EC4Docker, to facilitate its usage for non-experienced users. This service will offer the automatization of the generation of the container images that EC4Docker uses to deploy the cluster. Currently, the administrator or the users need to generate their own images including the dockerfiles provided with EC4Docker in order to deploy their own applications in the container cluster environment. A web portal will be implemented to facilitate this process, together with the facilitation to deploy, configure and manage virtual elastic clusters based on containers.

10.3 Fundings of this project

This work has been developed under the support of the program “Ayudas para la contratación de personal investigador en formación de carácter predoctoral, programa VALi+d”, grant number ACIF/2013/003, from the Conselleria d’Educació of the Generalitat Valenciana.

Also, the author wishes to thank the financial support received form The Spanish Ministry of Economy and Competitiveness to develop the project “CLUVIEM”

(Migrable Elastic Virtual Clusters on Hybrid Cloud Infrastructures), with reference TIN2013-44390-R.

Appendix A

Sequence diagrams of EC3

This appendix includes two sequence diagrams that represent the three phases explained in the "Overall architecture" section 4.3.2 of the chapter 4. On the one hand, Figure A.1 of this annex represents the sequence diagram of the deployment and configuration of the frontend (phases 1 and 2). As in Figure 4.1 of the chapter 4, the diagram represents the interaction of the main components and actors that perform the deployment of the cluster: EC3 client, the RADL file, the Infrastructure Manager (IM), the VMRC catalog of images, Ansible (in charge of configure and install in the frontend the LRMS, CLUES to automate the elasticity, and other packages if needed, like BLCR if spot instances are going to be used) and the Cloud provider.

On the other hand, Figure A.2 of this annex represents the sequence diagram when a new job arrives to the cluster and no nodes are available. This is related to phase 3 and represents the elasticity management of the cluster, where the interaction between the LRMS, CLUES and the IM is showed.

These two diagrams are included in the documentation of our tool, available in [38].

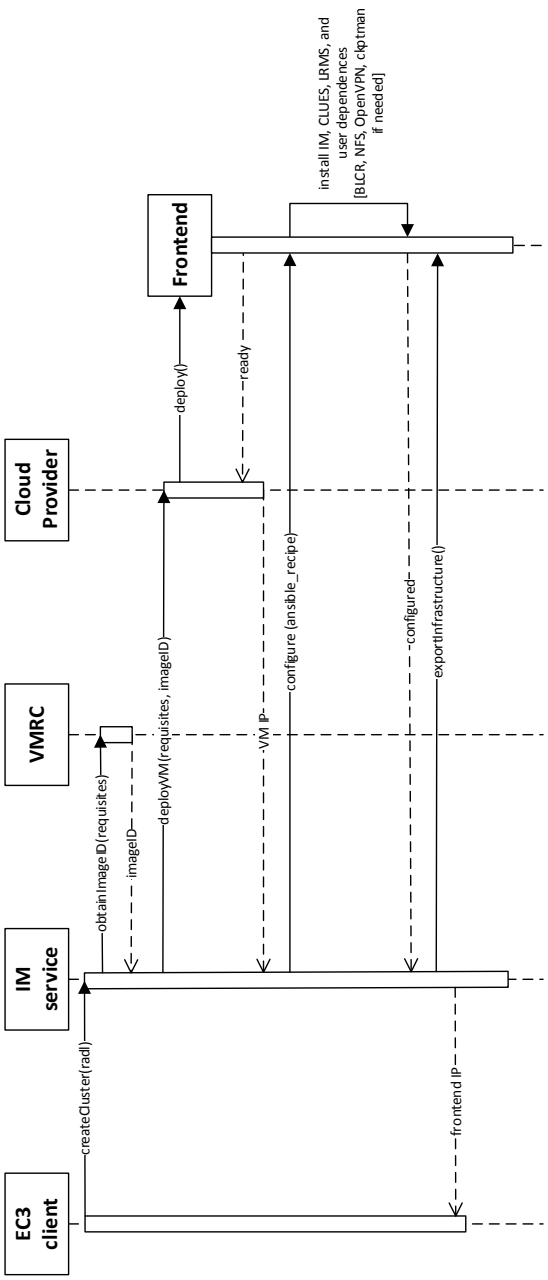


Figure A.1: Sequence diagram of the deployment and configuration of the frontend (phases 1 and 2).

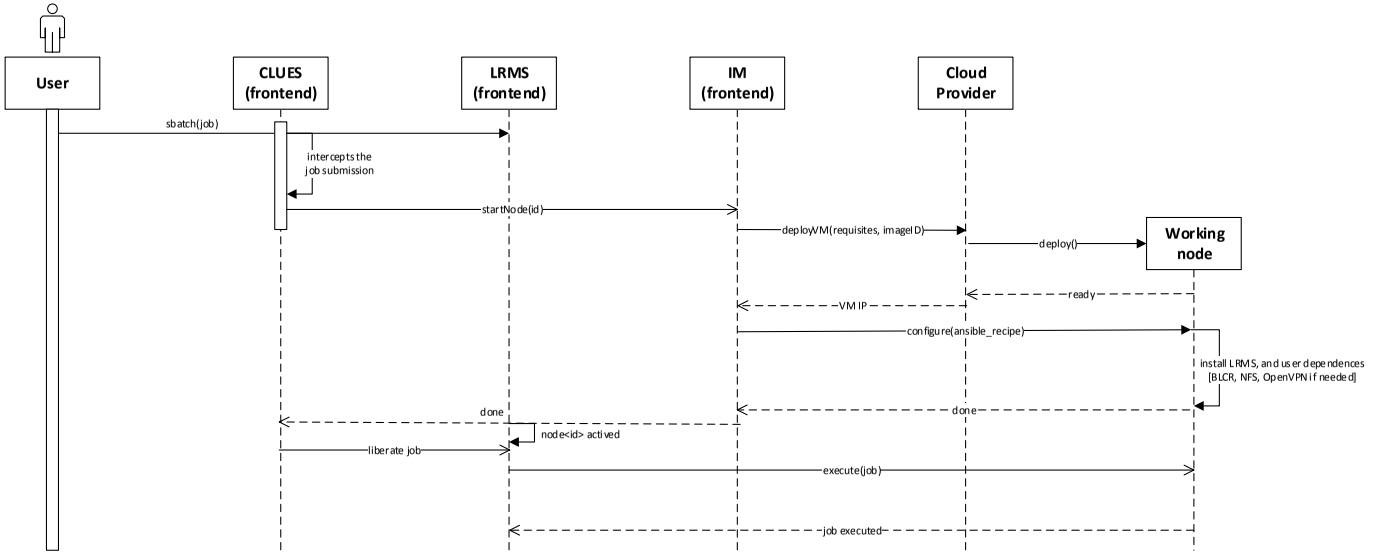


Figure A.2: Sequence diagram that represents the arrival of a new job to the cluster and no nodes are available (phase 3).

Appendix B

Flowcharts of ckptman behaviour

This appendix includes two flowcharts that represent the behaviour of ckptman in its two principal processes. On the one hand, Figure B.1 represents the logic of ckptman applied each time the dictionary of spot nodes and its relation with the jobs in execution is updated. If a new node has appeared and it is not in the dictionary, ckptman saves its reference. In the same way, if a new job in execution has been appeared to the LRMS, ckptman checks if it is running inside a spot node. If the answer is positive, ckptman checks in which spot node the application is being executed and saves this data in its dictionary. This schema is also applied the first time ckptman is executed.

On the other hand, Figure B.2 points out the process followed by ckptman to control the checkpointing operations and the state of the nodes that compose the cluster. With the spot nodes dictionary updated (applying the process showed in figure B.1), ckptman checks if for each spot node, it has a running application associated. If the answer is positive, ckptman checks if the spot node is still alive. In case the spot node is dead, ckptman restarts the application from its last checkpoint (if exists). If not, ckptman will use the checkpointing algorithms presented in 4.3.4 in order to know if a checkpoint operation must be performed, and perform it in case the answer is positive.

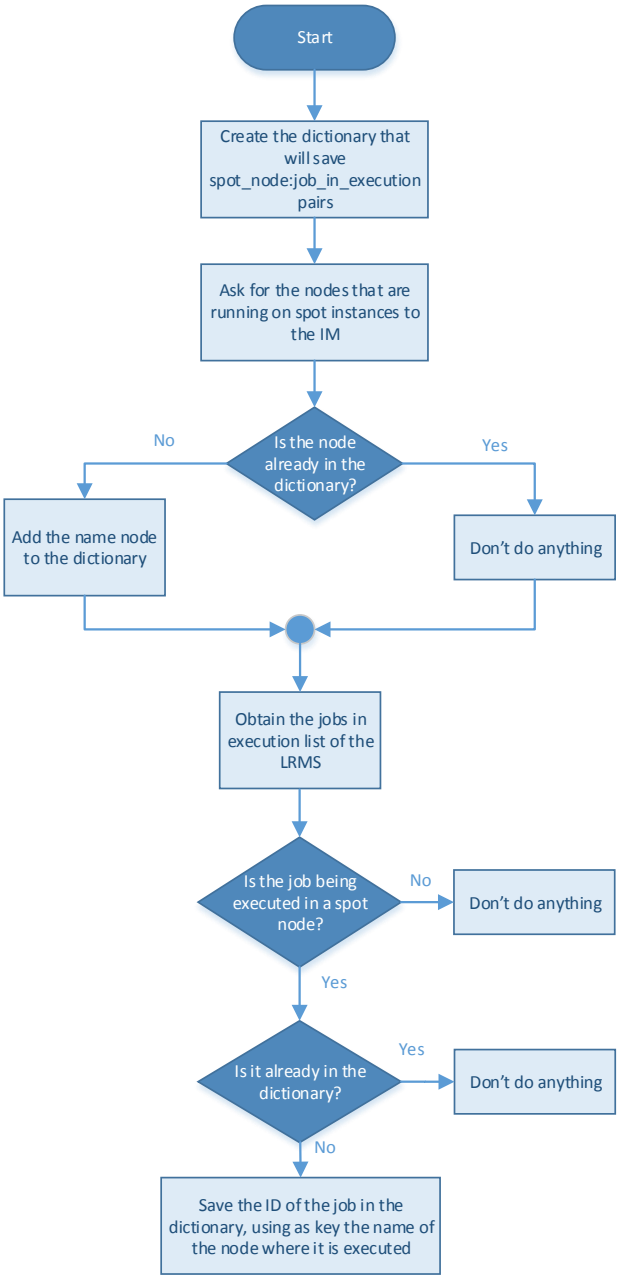


Figure B.1: Creation and update of the spot nodes dictionary in ckptman.

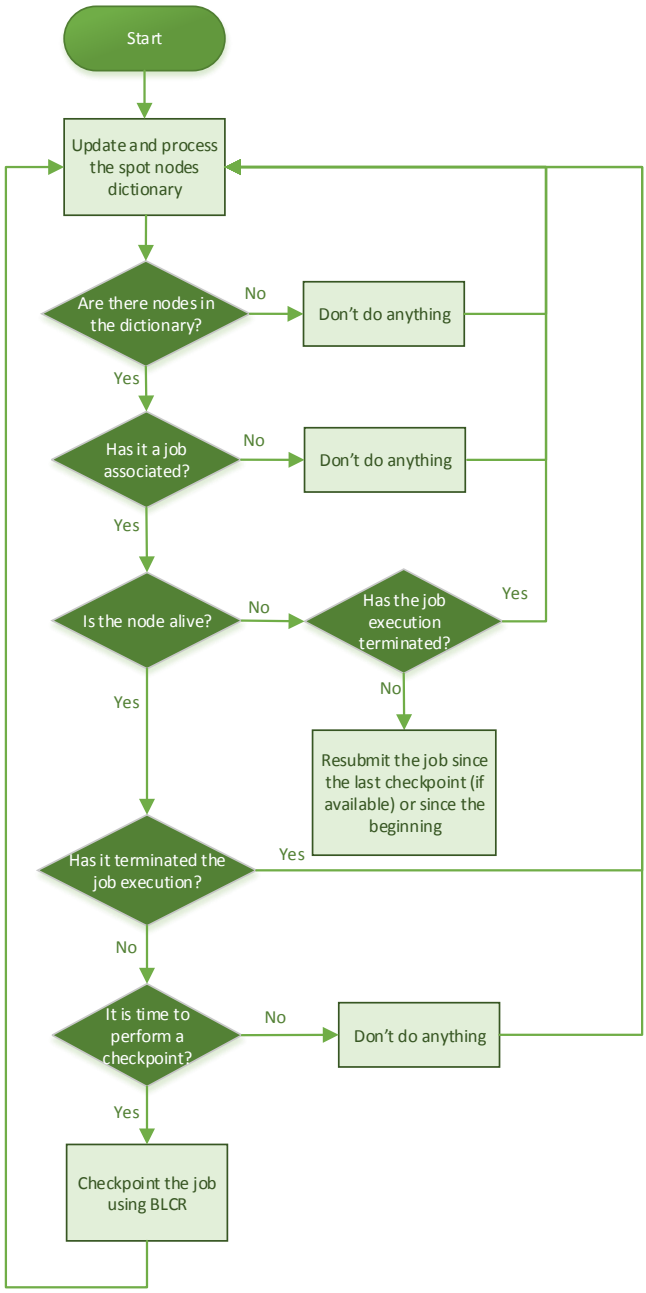


Figure B.2: Control of checkpointing and state of nodes in ckptman.

Bibliography

[1] EGI Newsletter (Issue 22) January 2016. *Custom elastic clusters to manage Galaxy environments*. http://www.egi.eu/news-and-media/newsletters/Inspired_Issue_22/Custom_elastic_clusters_to_manage_Galaxy_environments.html. [Online; accessed 27-January-2016].

[2] AdaptiveComputing. *Torque Resource Manager*. http://www.adaptivecomputing.com/products/open-source/torque/. [Online; Accessed: 22-June-2016].

[3] T. Adufu, Jieun Choi, and Yoonhee Kim. "Is container-based technology a winner for high performance scientific applications?" In: *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. 2015, pp. 507–510. DOI: 10.1109/APNOMS.2015.7275379.

[4] Enis Afgan et al. "Galaxy CloudMan: delivering cloud compute clusters". In: *BMC bioinformatics* 11.Suppl 12 (2010), S4.

[5] Orna Agmon Ben-Yehuda et al. "Deconstructing Amazon EC2 Spot Instance Pricing". In: *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 304–311. ISBN: 978-0-7695-4622-3. DOI: 10.1109/CloudCom.2011.48. URL: http://dx.doi.org/10.1109/CloudCom.2011.48.

[6] Sang Un Ahn, Sang Oh Park, and Jin Kim. "Profiling Job Activities of Batch Systems in the Data Center". In: *2016 International Conference on Platform Technology and Service (PlatCon)*. IEEE, 2016, pp. 1–5. ISBN: 978-1-4673-8685-2. DOI: 10.1109/PlatCon.2016.7456818. URL: http://ieeexplore.ieee.org/document/7456818/.

[7] C. de Alfonso et al. “Infrastructure Deployment Over the Cloud”. In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. 2011, pp. 517–521. DOI: 10.1109/CloudCom.2011.77.

[8] C. de Alfonso et al. “Infrastructure Deployment over the Cloud”. In: *3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. IEEE, 2011, pp. 517–521. DOI: 10.1109/CloudCom.2011.77.

[9] Fernando Alvarruiz et al. “An Energy Manager for High Performance Computer Clusters”. In: *Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*. ISPA ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 231–238. ISBN: 978-0-7695-4701-5. DOI: 10.1109/ISPA.2012.38. URL: http://dx.doi.org/10.1109/ISPA.2012.38.

[10] Miguel Caballer Amanda Calatrava. *Elastic Cloud Computing Cluster tutorials available in Youtube*. URL: https://www.youtube.com/channel/UCQD6RJBs57Giz4Xm8dhDczQ.

[11] Amazon. *Amazon EC2 Spot Bid Price Limits*. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html. [Online; Accessed: 22-June-2016].

[12] Amazon. *Amazon Web Services (AWS)*. http://aws.amazon.com. [Online; Accessed: 22-June-2016].

[13] Amazon. *Spot Instances (Amazon EC2)*. https://aws.amazon.com/ec2/purchasing-options/spot-instances/. [Online; Accessed: 22-June-2016].

[14] *Amazon EC2 Container Service*. http://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html. On-line. Accessed: 04-02-2016.

[15] *Amazon Web Services, the 1K Genomes project*. http://s3.amazonaws.com/1000genomes. Online; Accessed: 22-June-2016.

[16] Jason Ansel, Kapil Arya, and Gene Cooperman. “DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop”. In: *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. IPDPS ’09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–12. ISBN: 978-1-4244-3751-1. DOI: 10.1109/IPDPS.2009.5161063. URL: http://dx.doi.org/10.1109/IPDPS.2009.5161063.

[17] Apache. *Apache CloudStack*. http://cloudstack.apache.org. [Online; Accessed: 22-June-2016].

[18] *Apache LibCloud*. https://libcloud.apache.org/. On-line. Accessed: 01-02-2016.

[19] The Grid Workloads Archive. *NorduGrid dataset*. http://gwa.ewi.tudelft.nl/datasets/gwa-t-3-nordugrid/report/. [Online; Accesed 22-June-2016].

[20] V. Ardizzone et al. "The DECIDE Science Gateway". In: *Journal of Grid Computing* 10.4 (2012), pp. 689–707. ISSN: 1572-9184. DOI: 10.1007/s10723-012-9242-3. URL: http://dx.doi.org/10.1007/s10723-012-9242-3.

[21] Ritu Arora, Purushotham Bangalore, and Marjan Mernik. "A Technique for Non-invasive Application-level Checkpointing". In: *J. Supercomput.* 57.3 (Sept. 2011), pp. 227–255. ISSN: 0920-8542. DOI: 10.1007/s11227-010-0383-5. URL: http://dx.doi.org/10.1007/s11227-010-0383-5.

[22] 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. URL: http://doi.acm.org/10.1145/1551609.1551635.

[23] Paul Barham et al. "Xen and the Art of Virtualization". In: *SIGOPS Oper. Syst. Rev.* 37.5 (Oct. 2003), pp. 164–177. ISSN: 0163-5980. DOI: 10.1145/1165389.945462. URL: http://doi.acm.org/10.1145/1165389.945462.

[24] Céline Bellard et al. "Impacts of climate change on the future of biodiversity". In: *Ecology Letters* 15.4 (2012), pp. 365–377. ISSN: 1461-0248. DOI: 10.1111/j.1461-0248.2011.01736.x. URL: http://dx.doi.org/10.1111/j.1461-0248.2011.01736.x.

[25] Anton Beloglazov and Rajkumar Buyya. "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers". In: *Concurr. Comput. : Pract. Exper.* 24.13 (Sept. 2012), pp. 1397–1420. ISSN: 1532-0626. DOI: 10.1002/cpe.1867. URL: http://dx.doi.org/10.1002/cpe.1867.

[26] Marco Bencivenni et al. “Accessing Grid and Cloud Services Through a Scientific Web Portal”. In: *Journal of Grid Computing* 13.2 (2014), pp. 159–175. ISSN: 1572-9184. DOI: 10.1007/s10723-014-9310-y. URL: http://dx.doi.org/10.1007/s10723-014-9310-y.

[27] David Bernstein. “Containers and Cloud: From LXC to Docker to Kubernetes”. In: *Cloud Computing, IEEE* 1.3 (2014), pp. 81–84. ISSN: 2325-6095. DOI: 10.1109/MCC.2014.51.

[28] Andrzej Bialecki, Christophe Taton, and Jim Kellerman. *Apache Hadoop: a framework for running applications on large clusters built of commodity hardware*. http://hadoop.apache.org/, 2010.

[29] R. Bifulco et al. “Transparent migration of virtual infrastructures in large datacenters for Cloud computing”. In: *Computers and Communications (ISCC), 2011 IEEE Symposium on*. 2011, pp. 179–184. DOI: 10.1109/ISCC.2011.5984013.

[30] *Bootstrap Framework*. http://getbootstrap.com/. On-line. Accessed: 02-02-2016.

[31] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN: 0130137847.

[32] Miguel Caballer et al. “Dynamic management of virtual infrastructures”. In: *Journal of Grid Computing* (2014). DOI: 10.1007/s10723-014-9296-5. URL: http://link.springer.com/article/10.1007/s10723-014-9296-5.

[33] Miguel Caballer et al. “Dynamic Management of Virtual Infrastructures”. In: *Journal of Grid Computing* 13.1 (2015), pp. 53–70. ISSN: 1570-7873. DOI: 10.1007/s10723-014-9296-5. URL: http://dx.doi.org/10.1007/s10723-014-9296-5.

[34] Miguel Caballer et al. “EC3: Elastic Cloud Computing Cluster”. In: *J. Comput. Syst. Sci.* 79.8 (Dec. 2013), pp. 1341–1351. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2013.06.005. URL: http://dx.doi.org/10.1016/j.jcss.2013.06.005.

[35] A. Calatrava, G. Molto, and V. Hernandez. “Combining Grid and Cloud Resources for Hybrid Scientific Computing Executions”. In: *Cloud Comput-*

ing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. 2011, pp. 494–501. DOI: 10.1109/CloudCom.2011.73.

[36] A. Calatrava et al. "Towards Migratable Elastic Virtual Clusters on Hybrid Clouds". In: *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. 2015, pp. 1013–1016. DOI: 10.1109/CLOUD.2015.139.

[37] A. Calatrava et al. "Virtual Hybrid Elastic Clusters in the Cloud". In: *8th Iberian Grid Infrastructure Conference (IberGrid 2014)*. 2014, pp. 103–114.

[38] Amanda Calatrava. *Elastic Cloud Computing Cluster documentation available in Readthedocs*. URL: http://ec3.readthedocs.org/en/devel/.

[39] Amanda Calatrava. *Elastic Cloud Computing Cluster Webpage*. URL: http://www.grycap.upv.es/ec3/.

[40] Amanda Calatrava. *Elastic Cloud Computing Cluster Webpage in GitHub*. URL: https://github.com/grycap/ec3.

[41] Amanda Calatrava et al. "Self-managed cost-efficient virtual elastic clusters on hybrid Cloud infrastructures". In: *Future Generation Computer Systems* 61 (2016), pp. 13–25. ISSN: 0167739X. DOI: 10.1016/j.future.2016.01.018. URL: http://authors.elsevier.com/sd/article/S0167739X16300024http://linkinghub.elsevier.com/retrieve/pii/S0167739X16300024.

[42] Rodrigo N. Calheiros et al. "The Aneka Platform and QoS-driven Resource Provisioning for Elastic Applications on Hybrid Clouds". In: *Future Gener. Comput. Syst.* 28.6 (June 2012), pp. 861–870. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.07.005. URL: http://dx.doi.org/10.1016/j.future.2011.07.005.

[43] S. Camarasu-Pop, T. Glatard, and H. Benoit-Cattin. "Simulating Application Workflows and Services Deployed on the European Grid Infrastructure". In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 18–25. ISBN: 978-0-7695-4996-5. DOI: 10.1109/CCGrid.2013.13. URL: http://ieeexplore.ieee.org/document/6546054/.

[44] Canonical. *LXD*. 2016. URL: https://linuxcontainers.org/lxd/introduction/.

[45] J. V. Carrión et al. “A generic catalog and repository service for virtual machine images”. In: *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*. IEEE, 2010.

[46] Jose V. Carrión et al. “A Generic Catalog and Repository Service for Virtual Machine Images”. In: *2nd International ICST Conference on Cloud Computing (CloudComp 2010)*. 2010.

[47] *Chronos*. `https://mesos.github.io/chronos/`. [Online; accessed 19-January-2016].

[48] Christopher Clark et al. “Live Migration of Virtual Machines”. In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI’05. Berkeley, CA, USA: USENIX Association, 2005, pp. 273–286. URL: `http://dl.acm.org/citation.cfm?id=1251203.1251223`.

[49] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. “Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994”. In: ed. by Karsten M. Decker and René M. Rehmann. Basel: Birkhäuser Basel, 1994. Chap. The MPI Message Passing Interface Standard, pp. 213–218. ISBN: 978-3-0348-8534-8. DOI: `10.1007/978-3-0348-8534-8_21`. URL: `http://dx.doi.org/10.1007/978-3-0348-8534-8_21`.

[50] “Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community”. In: *BMC Bioinformatics* 42.13 (2012). ISSN: 1471-2105. DOI: `10.1186/1471-2105-13-42`.

[51] *Container image of EC3 in Docker Hub*. `https://hub.docker.com/r/grycap/ec3/`. On-line. Accessed: 01-03-2016.

[52] *CORE Ranking*. `http://core.edu.au/index.php`. [Online; accessed 14-January-2016].

[53] CoreOS. *rkt*. 2016. URL: `https://coreos.com/rkt/`.

[54] *CRIU*. `http://criu.org/Main_Page`. [Online; accessed 18-January-2016].

[55] *CryoPID*. `https://code.google.com/p/cryopid/`. [Online; accessed 14-January-2016].

[56] CycleComputing. *CycleCloud*. http://www.cyclecomputing.com/cyclecloud. [Online; Accessed: 22-June-2016].

[57] Tilman Dannert and Frank Jenko. "Gyrokinetic simulation of collisionless trapped-electron mode turbulence". In: *Physics of Plasmas* 12.7, 072309 (2005). DOI: http://dx.doi.org/10.1063/1.1947447. URL: http://dx.doi.org/10.1063/1.1947447.

[58] 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. URL: http://dx.doi.org/10.1016/j.future.2012.08.014.

[59] *Deploying Cost-Efficient Virtual Elastic Clusters across Multi-Clouds*. https://indico.egi.eu/indico/event/2544/contribution/1. On-line. Accessed: 03-02-2016.

[60] *Docker*. https://www.docker.com/. [Online; accessed 18-January-2016].

[61] *Docker Engine adapted to work with CRIU*. https://github.com/boucher/docker/tree/cr-combined. [Online; accessed 22-March-2016].

[62] Frank Doelitzscher et al. "ViteraaS: Virtual Cluster As a Service". In: *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*. CLOUDCOM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 652–657. ISBN: 978-0-7695-4622-3. DOI: 10.1109/CloudCom.2011.101. URL: http://dx.doi.org/10.1109/CloudCom.2011.101.

[63] YaoZu Dong et al. "COLO: COarse-grained LOck-stepping Virtual Machines for Non-stop Service". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 3:1–3:16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523630. URL: http://doi.acm.org/10.1145/2523616.2523630.

[64] Jose Duato et al. "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters". In: *2010 International Conference on High Performance Computing & Simulation*. IEEE, 2010, pp. 224–231. ISBN: 978-1-4244-6827-0. DOI: 10.1109/HPCS.2010.5547126. URL: http://ieeexplore.ieee.org/document/5547126/.

[65] Jason Duell. *The design and implementation of Berkeley Lab's linux Checkpoint/Restart*. Tech. rep. Lawrence Berkeley National Laboratory, 2002. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.2346.

[66] Dmitry Duplyakin et al. "Rebalancing in a Multi-cloud Environment". In: *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*. Science Cloud '13. New York, New York, USA: ACM, 2013, pp. 21–28. ISBN: 978-1-4503-1979-9. DOI: 10.1145/2465848.2465854. URL: http://doi.acm.org/10.1145/2465848.2465854.

[67] Amazon EC2. *Spot Instance Termination Notices*. http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-interruptions.html. [Online; Accessed: 22-June-2016].

[68] EGI. *EGI Engage*. URL: https://www.egi.eu/about/egi-engage/.

[69] EGI. *EGI Long Tail of Science*. https://criu.org/LXC. [Online; accessed 27-Sep-2016].

[70] *EGI Federated Cloud*. https://www.egi.eu/infrastructure/cloud/. Online; Accessed: 22-June-2016.

[71] *Elastic Cluster for Docker*. https://github.com/grycap/ec4docker. [Online; accessed 9-March-2016].

[72] 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.

[73] W. Felter et al. "An updated performance comparison of virtual machines and Linux containers". In: *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. 2015, pp. 171–172. DOI: 10.1109/ISPASS.2015.7095802.

[74] Tiago C. Ferreto et al. "Server Consolidation with Migration Control for Virtualized Data Centers". In: *Future Gener. Comput. Syst.* 27.8 (Oct. 2011), pp. 1027–1034. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.04.016. URL: http://dx.doi.org/10.1016/j.future.2011.04.016.

[75] Tom Fifield et al. "Integration of cloud, grid and local cluster resources with DIRAC". In: *Journal of Physics: Conference Series*. Vol. 331. 6. IOP Publishing. 2011, p. 062009.

[76] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. ISBN: 1558609334.

[77] *Galaxy in the cloud*. `https://wiki.galaxyproject.org/Cloud`. Online; Accessed: 22-June-2016.

[78] Gartner. *Gartner Says Nearly Half of Large Enterprises Will Have Hybrid Cloud Deployments by the End of 2017*. `http://www.gartner.com/newsroom/id/2599315`. [Online; Accessed: 22-June-2016].

[79] "Genotyping in the Cloud with Crossbow". In: *Current Protocols in Bioinformatics* 3.15 (2012). ISSN: 1934-340X. DOI: `10.1002/0471250953.bi1503s39`.

[80] Wolfgang Gerlach et al. "Skyport: Container-based Execution Environment Management for Multi-cloud Scientific Workflows". In: *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. DataCloud '14. New Orleans, Louisiana: IEEE Press, 2014, pp. 25–32. ISBN: 978-1-4799-7034-6. DOI: `10.1109/DataCloud.2014.6`. URL: `http://dx.doi.org/10.1109/DataCloud.2014.6`.

[81] B. Giardine et al. "Galaxy: a platform for interactive large-scale genome analysis." In: *Genome research* 15.10 (2005), pp. 1451–1455.

[82] J. Goecks et al. "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences." In: *Genome biology* 11.8 (2010), R86.

[83] *Google Cloud Engine*. `https://cloud.google.com/compute/`. On-line. Accessed: 01-02-2016.

[84] Ricardo Graciani Diaz et al. "Belle-DIRAC Setup for Using Amazon Elastic Compute Cloud". In: *Journal of Grid Computing* 9.1 (2011), pp. 65–79. ISSN: 1572-9184. DOI: `10.1007/s10723-010-9175-7`. URL: `http://dx.doi.org/10.1007/s10723-010-9175-7`.

[85] T. Görler et al. "The global version of the gyrokinetic turbulence code GENE". In: *Journal of Computational Physics* 230.18 (2011), pp. 7053 – 7071. ISSN: 0021-9991. DOI: `http://dx.doi.org/10.1016/j.jcp.2011.`

05.034. URL: http://www.sciencedirect.com/science/article/pii/S0021999111003457.

[86] D. Gusfield. *Algorithms on strings, trees and sequences.* Cambridge University Press, 1997.

[87] Holmes V. Higgins J. and Venters C. "High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings". In: Cham: Springer International Publishing, 2015. Chap. Orchestrating Docker Containers in the HPC Environment, pp. 506–513. ISBN: 978-3-319-20119-1. DOI: 10.1007/978-3-319-20119-1_36. URL: http://dx.doi.org/10.1007/978-3-319-20119-1_36.

[88] Benjamin Hindman et al. "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation.* NSDI'11. Boston, MA: USENIX Association, 2011, pp. 295–308. URL: http://dl.acm.org/citation.cfm?id=1972457.1972488.

[89] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. "Post-copy Live Migration of Virtual Machines". In: *SIGOPS Oper. Syst. Rev.* 43.3 (July 2009), pp. 14–26. ISSN: 0163-5980. DOI: 10.1145/1618525.1618528. URL: http://doi.acm.org/10.1145/1618525.1618528.

[90] L. Hochstein, ed. *Ansible: Up and Running, Automating Configuration Management and Deployment the Easy Way.* O'Reilly Media, 2014.

[91] *HTCondor*. http://research.cs.wisc.edu/htcondor/. [Online; accessed 14-January-2016].

[92] J. Hursey et al. "The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI". In: *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International.* 2007, pp. 1–8. DOI: 10.1109/IPDPS.2007.370605.

[93] *I Enceuntro de Estudiantes de Doctorado de la Universitat Politècnica de València*. http://www.upv.es/contenidos/ENCDOC/menu_urlc.html?/contenidos/ENCDOC/info/U0657401.pdf. On-line. Accessed: 03-02-2016.

[94] IBM. *IBM Platform Dynamic Cluster*. http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=PM&subtype=SP&htmlfid=DCD12365USEN. [Online; Accessed: 22-June-2016].

[95] *Indigo Datacloud*. URL: https://www.indigo-datacloud.eu/.

[96] Alexandru Iosup et al. "The Grid Workloads Archive". In: *Future Generation Computer Systems* 24.7 (2008), pp. 672 –686. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2008.02.003. URL: http://www.sciencedirect.com/science/article/pii/S0167739X08000125.

[97] C. Isci et al. "Improving server utilization using fast virtual machine migration". In: *IBM Journal of Research and Development* 55.6 (2011), 4:1–4:12. ISSN: 0018-8646. DOI: 10.1147/JRD.2011.2167775.

[98] A. J. Izenman. *Modern Multivariate Statistical Techniques*. NY: Springer, 2008, p. 731.

[99] Bahman Javadi, Ruppa K. Thulasiram, and Rajkumar Buyya. "Characterizing Spot Price Dynamics in Public Cloud Environments". In: *Future Gener. Comput. Syst.* 29.4 (June 2013), pp. 988–999. ISSN: 0167-739X. DOI: 10.1016/j.future.2012.06.012. URL: http://dx.doi.org/10.1016/j.future.2012.06.012.

[100] Morris A. Jette, Andy B. Yoo, and Mark Grondona. "SLURM: Simple Linux Utility for Resource Management". In: *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[101] Bo Jiang, Binoy Ravindran, and Changsoo Kim. "Stabilization, Safety, and Security of Distributed Systems: 12th International Symposium, SSS 2010, New York, NY, USA, September 20-22, 2010. Proceedings". In: ed. by Shlomi Dolev et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. Chap. Lightweight Live Migration for High Availability Cluster Service, pp. 420–434. ISBN: 978-3-642-16023-3. DOI: 10.1007/978-3-642-16023-3_34. URL: http://dx.doi.org/10.1007/978-3-642-16023-3_34.

[102] Hai Jin et al. "Optimizing the Live Migration of Virtual Machine by CPU Scheduling". In: *J. Netw. Comput. Appl.* 34.4 (July 2011), pp. 1088–1096. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2010.06.013. URL: http://dx.doi.org/10.1016/j.jnca.2010.06.013.

[103] Daeyong Jung et al. "An Efficient Checkpointing Scheme Using Price History of Spot Instances in Cloud Computing Environment". English. In: *Network and Parallel Computing*. Ed. by Erik Altman and Weisong Shi. Vol. 6985. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 185–200. ISBN: 978-3-642-24402-5. DOI: 10.1007/978-3-642-

24403-2_16. URL: http://dx.doi.org/10.1007/978-3-642-24403-2_16.

[104] Gideon Juve and Ewa Deelman. "Wrangler: Virtual Cluster Provisioning for the Cloud". In: *Proceedings of the 20th international symposium on High performance distributed computing - HPDC '11*. New York, New York, USA: ACM Press, June 2011, p. 277. ISBN: 9781450305525. DOI: 10.1145/1996130.1996173. URL: http://dl.acm.org/citation.cfm?id=1996130.1996173.

[105] A. Kangarlou, P. Eugster, and D. Xu. "VNsnap: Taking Snapshots of Virtual Networked Infrastructures in the Cloud". In: *IEEE Transactions on Services Computing* 5.4 (2012), pp. 484–496. ISSN: 1939-1374. DOI: 10.1109/TSC.2011.29.

[106] Katarzyna Keahey et al. "Sky Computing". In: *IEEE Internet Computing* 13.5 (2009), pp. 43–51. ISSN: 1089-7801. DOI: http://doi.ieeecomputersociety.org/10.1109/MIC.2009.94.

[107] S. Khatua and N. Mukherjee. "A Novel Checkpointing Scheme for Amazon EC2 Spot Instances". In: *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. 2013, pp. 180–181. DOI: 10.1109/CCGrid.2013.71.

[108] Avi Kivity et al. "kvm: the Linux Virtual Machine Monitor". In: *Proceedings of the Linux Symposium*. Vol. 1. Ottawa, Ontario, Canada, June 2007, pp. 225–230. URL: http://linux-security.cn/ebooks/ols2007/OLS2007-Proceedings-V1.pdf.

[109] *Kubernetes*. http://kubernetes.io/. [Online; accessed 7-March-2016].

[110] Xiaofei Liao, Hai Jin, and Haikun Liu. "Towards a Green Cluster Through Dynamic Remapping of Virtual Machines". In: *Future Gener. Comput. Syst.* 28.2 (Feb. 2012), pp. 469–477. ISSN: 0167-739X. DOI: 10.1016/j.future.2011.04.013. URL: http://dx.doi.org/10.1016/j.future.2011.04.013.

[111] *LibVirt, virtualization API*. http://libvirt.org/. On-line. Accessed: 01-02-2016.

[112] *Linux Containers, LXC*. https://linuxcontainers.org/. [Online; accessed 18-January-2016].

[113] Feifei Liu and Xiaoshe Dong. "A Novel Elastic Resource Allocation Strategy of Virtual Cluster". In: *Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on*. 2011, pp. 168–173. DOI: 10.1109/PAAP.2011.28.

[114] H. Liu et al. "Live Virtual Machine Migration via Asynchronous Replication and State Synchronization". In: *IEEE Transactions on Parallel and Distributed Systems* 22.12 (2011), pp. 1986–1999. ISSN: 1045-9219. DOI: 10.1109/TPDS.2011.86.

[115] Haikun Liu et al. "Performance and Energy Modeling for Live Migration of Virtual Machines". In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. HPDC '11. San Jose, California, USA: ACM, 2011, pp. 171–182. ISBN: 978-1-4503-0552-5. DOI: 10.1145/1996130.1996154. URL: http://doi.acm.org/10.1145/1996130.1996154.

[116] Andrea Luzzardi and Victor Vieux. *Swarm: a Docker-native clustering system*. https://docs.docker.com/swarm/. [Online; accessed 7-March-2016].

[117] Fumio Machida, Dong Seong Kim, and Kishor S. Trivedi. "Modeling and Analysis of Software Rejuvenation in a Server Virtualized System with Live VM Migration". In: *Perform. Eval.* 70.3 (Mar. 2013), pp. 212–230. ISSN: 0166-5316. DOI: 10.1016/j.peva.2012.09.003. URL: http://dx.doi.org/10.1016/j.peva.2012.09.003.

[118] R. K. Madduri et al. "Experiences Building Globus Genomics: A Next-Generation Sequencing Analysis Service using Galaxy, Globus, and Amazon Web Services". In: *Concurrency and Computation: Practice & Experience* (2014).

[119] *Marathon*. https://mesosphere.github.io/marathon/. [Online; accessed 19-January-2016].

[120] 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.

[121] Paul Marshall et al. "Architecting a Large-scale Elastic Environment - Recontextualization and Adaptive Cloud Services for Scientific Computing." In: *ICSOFT*. Ed. by Slimane Hammoudi, Marten van Sinderen, and José Cordeiro. SciTePress, 2012, pp. 409–418. ISBN: 978-989-8565-19-8. URL:

http://dblp.uni-trier.de/db/conf/icsoft/icsoft2012.html#MarshallTKBW12.

[122] Michael Mattess, Christian Vecchiola, and Rajkumar Buyya. "Managing Peak Loads by Leasing Cloud Infrastructure Services from a Spot Market". In: *Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications*. HPCC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 180–188. ISBN: 978-0-7695-4214-0. DOI: 10.1109/HPCC.2010.77. URL: http://dx.doi.org/10.1109/HPCC.2010.77.

[123] M. Mazzucco and M. Dumas. "Achieving Performance and Availability Guarantees with Spot Instances". In: *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. 2011, pp. 296–303. DOI: 10.1109/HPCC.2011.46.

[124] Violeta Medina and Juan Manuel García. "A Survey of Migration Mechanisms of Virtual Machines". In: *ACM Comput. Surv.* 46.3 (Jan. 2014), 30:1–30:33. ISSN: 0360-0300. DOI: 10.1145/2492705. URL: http://doi.acm.org/10.1145/2492705.

[125] J. Mehta and S. Chaudhary. "Checkpointing and Recovery Mechanism in Grid". In: *Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on*. 2008, pp. 131–140. DOI: 10.1109/ADCOM.2008.4760439.

[126] I. Melia et al. *Linux Containers: Why They're in Your Future and What Has to Happen First*. Tech. rep. CISCO and Redhat, 2014.

[127] Peter Mell and Timothy Grance. *The NIST Definition of Cloud Computing*. Tech. rep. 800-145. Gaithersburg, MD: National Institute of Standards and Technology (NIST), 2011. URL: http://dx.doi.org/10.6028/NIST.SP.800-145.

[128] "Metagenomics versus Moore's law". In: *Nat Meth* 6.9 (Sept. 2009), pp. 623–623. URL: http://dx.doi.org/10.1038/nmeth0909-623.

[129] Thijs Metsch, Andy Edmonds, et al. "Open cloud computing interface-infrastructure". In: *Standards Track, no. GFD-R in The Open Grid Forum Document Series, Open Cloud Computing Interface (OCCI) Working Group, Muncie (IN)*. 2010.

[130] *Microsoft Azure for Research*. http://research.microsoft.com/en-us/projects/azure/. Online; Accessed: 22-June-2016.

[131] Mark A. Miller, Wayne Pfeiffer, and Terri Schwartz. "The CIPRES Science Gateway: A Community Resource for Phylogenetic Analyses". In: *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*. TG '11. Salt Lake City, Utah: ACM, 2011, 41:1–41:8. ISBN: 978-1-4503-0888-5. DOI: 10.1145/2016741.2016785. URL: http://doi.acm.org/10.1145/2016741.2016785.

[132] Dejan S. Milojíčić et al. "Process Migration". In: *ACM Comput. Surv.* 32.3 (Sept. 2000), pp. 241–299. ISSN: 0360-0300. DOI: 10.1145/367701.367728. URL: http://doi.acm.org/10.1145/367701.367728.

[133] A Mirkin, A. Kuznetsov, and K. Kolyshkin. "Containers checkpointing and live migration." In: 2008, pp. 85–90.

[134] MIT. *StarCluster*. http://web.mit.edu/stardev/cluster/. [Online; Accessed: 22-June-2016].

[135] MIT. *StarCluster Elastic Load Balancer*. http://web.mit.edu/stardev/cluster/docs/0.92rc2/manual/load_balancer.html. [Online; Accessed: 22-June-2016].

[136] Germán Moltó, Amanda Calatrava, and Vicente Hernández. "High Performance Computing for Computational Science - VECPAR 2012: 10th International Conference, Kope, Japan, July 17-20, 2012, Revised Selected Papers". In: ed. by Michel Daydé, Osni Marques, and Kengo Nakajima. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. Chap. A Service-Oriented Architecture for Scientific Computing on Cloud Infrastructures, pp. 163–176. ISBN: 978-3-642-38718-0. DOI: 10.1007/978-3-642-38718-0_18. URL: http://dx.doi.org/10.1007/978-3-642-38718-0_18.

[137] 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.

[138] 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: http://dx.doi.org/10.

1016/j.jpdc.2010.05.005. URL: http://www.sciencedirect.com/science/article/pii/S0743731510000985.

[139] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. "Elastic Management of Cluster-based Services in the Cloud". In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACDC '09. Barcelona, Spain: ACM, 2009, pp. 19–24. ISBN: 978-1-60558-585-7. DOI: 10.1145/1555271.1555277. URL: http://doi.acm.org/10.1145/1555271.1555277.

[140] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. "Multicloud Deployment of Computing Clusters for Loosely Coupled MTC Applications". In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (2011), pp. 924–930. ISSN: 1045-9219. DOI: http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.186.

[141] *MrBayes: Bayesian Inference of Phylogeny*. http://mrbayes.sourceforge.net/index.php. [Online; accessed 22-April-2016].

[142] D. Müllner. "fastclufast: Fast Hierarchical, Agglomerative clustering routines for R and python". In: *Journal of Statistical Software* 53.9 (2013), pp. 1–18.

[143] S. B. Needleman and C. D. Wunsch. "A general method applicable to search for similarities in the amino-acid sequence of two proteins". In: *J. Mol. Biol.* 48 (1970), pp. 443–453.

[144] Duy Nguyen and N. Thoai. "Design and implementation of the application-level migration tool on multi-site cloud". In: *Communications and Electronics (ICCE), 2012 Fourth International Conference on*. 2012, pp. 125–129. DOI: 10.1109/CCE.2012.6315883.

[145] Duy Nguyen and N. Thoai. "EBC: Application-level migration on multi-site cloud". In: *Systems and Informatics (ICSAI), 2012 International Conference on*. 2012, pp. 876–880. DOI: 10.1109/ICSAI.2012.6223147.

[146] Shuangcheng Niu et al. "Cost-effective cloud HPC resource provisioning by building semi-elastic virtual clusters". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13*. New York, New York, USA: ACM Press, Nov. 2013, pp. 1–12. ISBN: 9781450323789. DOI: 10.1145/2503210.2503236. URL: http://dl.acm.org/citation.cfm?id=2503210.2503236.

[147] Daniel Nurmi et al. "The Eucalyptus Open-Source Cloud-Computing System". 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. 124–131. ISBN: 978-0-7695-3622-4. DOI: 10.1109/CCGRID.2009.93. URL: http://dx.doi.org/10.1109/CCGRID.2009.93.

[148] *Official Docker Github issue related with checkpointing*. https://github.com/docker/docker/issues/20300. [Online; accessed 22-March-2016].

[149] A.J. Oliner et al. "Performance implications of periodic checkpointing on large-scale cluster systems". In: *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. 2005, 8 pp.–. DOI: 10.1109/IPDPS.2005.337.

[150] OpenStack. *OpenStack Cloud Software*. http://openstack.org. [Online; Accessed: 22-June-2016].

[151] Openstack. *Openstack Floating IPs*. http://docs.openstack.org/openstack-ops/content/floating_ips.html. [Online; accessed 21-May-2016].

[152] OpenVPN. *Open Source VPN*. http://openvpn.net/. [Online; Accessed: 22-June-2016].

[153] *OpenVZ*. http://openvz.org/Main_Page. [Online; accessed 14-January-2016].

[154] *ORBIT project*. http://www.orbitproject.eu/high-performance-virtual-machine-based-fault-tolerance-colo/. [Online; accessed 5-September-2016].

[155] C. Pahl and B. Lee. "Containers and Clusters for Edge Cloud Architectures – A Technology Review". In: *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. 2015, pp. 379–386. DOI: 10.1109/FiCloud.2015.35.

[156] Ju-Won Park and Jaegyoon Hahm. "Container-based Cluster Management Platform for Distributed Computing". In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*. Athens, 2015, pp. 34–40.

[157] René Peinl, Florian Holzschuher, and Florian Pfitzer. "Docker Cluster Management for the Cloud - Survey Results and Own Solution". In: *Journal of*

Grid Computing (2016), pp. 1–18. ISSN: 1572-9184. DOI: 10.1007/s10723-016-9366-y. URL: http://dx.doi.org/10.1007/s10723-016-9366-y.

[158] Jérôme Petazzoni. *Linux Containers(LXC), Docker, and Security.* http://www.slideshare.net/jpetazzo/linux-containers-lxc-docker-and-security/4-Fear_Uncertainty_and_DoubtLXC_is. [Online; accessed 18-January-2016].

[159] C.S.R. Prabhu. *GRID an Cluster Computing.* PHI Learning, 2008. ISBN: 9788120334281. URL: https://books.google.es/books?id=evcgB7Qlix4C.

[160] A. Pérez-García et al. "Architrave: Advanced Analysis of Building Structures Integrated in Computer-Aided Design". English. In: *Construction and Building Research.* Springer Netherlands, 2014, pp. 123–130. ISBN: 978-94-007-7789-7. DOI: 10.1007/978-94-007-7790-3_17.

[161] N. Regola and J. C. Ducom. "Recommendations for Virtualization Technologies in High Performance Computing". In: *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on.* 2010, pp. 409–416. DOI: 10.1109/CloudCom.2010.71.

[162] Pierre Riteau, Christine Morin, and Thierry Priol. "Shrinker: Improving Live Migration of Virtual Clusters over WANs with Distributed Data Deduplication and Content-based Addressing". In: *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I.* Euro-Par'11. Bordeaux, France: Springer-Verlag, 2011, pp. 431–442. ISBN: 978-3-642-23399-9. URL: http://dl.acm.org/citation.cfm?id=2033345.2033391.

[163] Gabriel Rodríguez et al. "CPPC: A Compiler-assisted Tool for Portable Checkpointing of Message-passing Applications". In: *Concurr. Comput. : Pract. Exper.* 22.6 (Apr. 2010), pp. 749–766. ISSN: 1532-0626. DOI: 10.1002/cpe.v22:6. URL: http://dx.doi.org/10.1002/cpe.v22:6.

[164] J. W. Sammon. "A nonlinear mapping algorithm for data structure analysis." In: *IEEE Transactions on Computers* 18(5) (1969), pp. 401–409.

[165] Michael Schatz. *Highly-sensitive short read mapping with MapReduce.* https://aws.amazon.com/articles/2272. Online; Accessed: 22-June-2016.

[166] M.J. Scheepers. "Virtualization and Containerization of Application Infrastructure: A Comparison". In: vol. 21. June 23. University of Twente, 2014.

[167] P. D. Schloss and al. "Introducing Mothur: open source, platform independent, community supported software for describing and comparing microbial communities". In: *Applied and Environmantal Microbiology* 75.23 (2009), pp. 7537–7541.

[168] *SGE, Open Grid Scheduler*. `http://sourceforge.net/projects/gridscheduler/`. On-line. Accessed: 01-02-2016.

[169] Aidan Shribman and Benoit Hudzia. "Pre-Copy and Post-Copy VM Live Migration for Memory Intensive Applications". In: *Euro-Par 2012: Parallel Processing Workshops: BDMC, CGWS, HeteroPar, HiBB, OMHI, Paraphrase, PROPER, Resilience, UCHPC, VHPC, Rhodes Islands, Greece, August 27-31, 2012. Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 539–547. ISBN: 978-3-642-36949-0. DOI: `10.1007/978-3-642-36949-0_63`. URL: `http://dx.doi.org/10.1007/978-3-642-36949-0_63`.

[170] Aleksander Slominski, Vinod Muthusamy, and Rania Khalaf. "Building a Multi-tenant Cloud Service from Legacy Code with Docker Containers." In: *IC2E*. IEEE, 2015, pp. 394–396. ISBN: 978-1-4799-8218-9.

[171] P. D. Smith and M. S. Waterman. "Identification of common molecular subsequences". In: *J. Mol. Biol.* 147 (1981), pp. 195–197.

[172] MrBayes Software. *Cynmix dataset*. `http://mrbayes.sourceforge.net/wiki/index.php/Cynmix.nex`. [Online; accessed 05-Jun-2016].

[173] Stephen Soltesz et al. "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors". In: *SIGOPS Oper. Syst. Rev.* 41.3 (Mar. 2007), pp. 275–287. ISSN: 0163-5980. DOI: `10.1145/1272998.1273025`. URL: `http://doi.acm.org/10.1145/1272998.1273025`.

[174] Yang Song, M. Zafer, and Kang-Won Lee. "Optimal bidding in spot instance market". In: *INFOCOM, 2012 Proceedings IEEE*. 2012, pp. 190–198. DOI: `10.1109/INFCOM.2012.6195567`.

[175] Borja Sotomayor et al. "Capacity Leasing in Cloud Systems using the OpenNebula Engine". In: Cloud Computing and Applications 2008 (CCA08). 2009.

[176] Petter Svärd et al. "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines". In: *Proceedings of the*

7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. VEE '11. Newport Beach, California, USA: ACM, 2011, pp. 111–120. ISBN: 978-1-4503-0687-4. DOI: 10.1145/1952682.1952698. URL: http://doi.acm.org/10.1145/1952682.1952698.

[177] Moussa Taifi, Justin Y. Shi, and Abdallah Khreishah. "SpotMPI: A Framework for Auction-based HPC Computing Using Amazon Spot Instances". In: *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part II*. ICA3PP'11. Melbourne, Australia: Springer-Verlag, 2011, pp. 109–120. ISBN: 978-3-642-24668-5. URL: http://dl.acm.org/citation.cfm?id=2075462.2075474.

[178] ShaoJie Tang, Jing Yuan, and Xiang-Yang Li. "Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance". In: *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*. CLOUD '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 91–98. ISBN: 978-0-7695-4755-8. DOI: 10.1109/CLOUD.2012.134. URL: http://dx.doi.org/10.1109/CLOUD.2012.134.

[179] Douglas Thain, Todd Tannenbaum, and Miron Livny. "Distributed computing in practice: the Condor experience." In: *Concurrency - Practice and Experience* 17.2-4 (2005), pp. 323–356.

[180] *The OpenMP API specification for parallel programming*. http://openmp.org/wp/. [Online; accessed 16-May-2016].

[181] Franco Travostino et al. "Seamless Live Migration of Virtual Machines over the MAN/WAN". In: *Future Gener. Comput. Syst.* 22.8 (Oct. 2006), pp. 901–907. ISSN: 0167-739X. DOI: 10.1016/j.future.2006.03.007. URL: http://dx.doi.org/10.1016/j.future.2006.03.007.

[182] Maurício Tsugawa, Andréa Matsunaga, and José Fortes. "User-Level Virtual Network Support for Sky Computing". In: *Proceedings of the 2009 Fifth IEEE International Conference on e-Science*. E-SCIENCE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 72–79. ISBN: 978-0-7695-3877-8. DOI: 10.1109/e-Science.2009.19. URL: http://dx.doi.org/10.1109/e-Science.2009.19.

[183] Flavio Vella et al. "GPU Computing in EGI Environment Using a Cloud Approach". In: *2011 International Conference on Computational Science and Its Applications*. IEEE, 2011, pp. 150–155. ISBN: 978-1-4577-0142-9. DOI: 10.1109/ICCSA.2011.61. URL: http://ieeexplore.ieee.org/document/5959549/.

[184] William Voorsluys, Saurabh Kumar Garg, and Rajkumar Buyya. "Provisioning Spot Market Cloud Resources to Create Cost-effective Virtual Clusters". In: *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I*. ICA3PP'11. Melbourne, Australia: Springer-Verlag, 2011, pp. 395–408. ISBN: 978-3-642-24649-4. URL: http://dl.acm.org/citation.cfm?id=2075416.2075453.

[185] William Voorsluys et al. "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation". In: *Proceedings of the 1st International Conference on Cloud Computing*. CloudCom '09. Beijing, China: Springer-Verlag, 2009, pp. 254–265. ISBN: 978-3-642-10664-4. DOI: 10.1007/978-3-642-10665-1_23. URL: http://dx.doi.org/10.1007/978-3-642-10665-1_23.

[186] Yi-Min Wang et al. "Checkpointing and its applications". In: *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*. 1995, pp. 22–31. DOI: 10.1109/FTCS.1995.466999.

[187] Xiaohui Wei et al. "Dynamic Deployment and Management of Elastic Virtual Clusters". In: *Proceedings of the 2011 Sixth Annual ChinaGrid Conference*. CHINAGRID '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 35–41. ISBN: 978-0-7695-4472-4. DOI: 10.1109/ChinaGrid.2011.31. URL: http://dx.doi.org/10.1109/ChinaGrid.2011.31.

[188] Nancy Wilkins-Diehr et al. "TeraGrid Science Gateways and Their Impact on Science". In: *Computer* 41.11 (Nov. 2008), pp. 32–41. ISSN: 0018-9162. DOI: 10.1109/MC.2008.470. URL: http://dx.doi.org/10.1109/MC.2008.470.

[189] Timothy Wood et al. "CloudNet: Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines". In: *SIGPLAN Not.* 46.7 (Mar. 2011), pp. 121–132. ISSN: 0362-1340. DOI: 10.1145/2007477.1952699. URL: http://doi.acm.org/10.1145/2007477.1952699.

[190] Ansible Works. *Ansible Software*. https://www.ansible.com/. [Online; accessed 24-May-2016].

[191] M. Gomes Xavier, M. Veiga Neves, and C. A. Fonticielha de Rose. "A Performance Comparison of Container-Based Virtualization Systems for MapReduce Clusters". In: *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*. 2014, pp. 299–306. DOI: 10.1109/PDP.2014.78.

[192] Miguel G. Xavier et al. “Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments”. In: *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. PDP ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 233–240. ISBN: 978-0-7695-4939-2. DOI: 10.1109/PDP.2013.41. URL: http://dx.doi.org/10.1109/PDP.2013.41.

[193] Xi-sheng Xiao, Ying-ping Huang, and Xi-hui Zhang. “Optimizing checkpoint for scientific simulations”. In: *Journal of Zhejiang University SCIENCE C* 13.12 (2012), pp. 891–900. ISSN: 1869-196X. DOI: 10.1631/jzus.C1200135. URL: http://dx.doi.org/10.1631/jzus.C1200135.

[194] Liao Xiaofei, Hu Liting, and Jin Hai. “Energy optimization schemes in cluster with virtual machines”. In: *Cluster Computing* 13.2 (2009), pp. 113–126. ISSN: 1573-7543. DOI: 10.1007/s10586-009-0110-2. URL: http://dx.doi.org/10.1007/s10586-009-0110-2.

[195] Xinhai Xu, Xuejun Yang, and Yufei Lin. “WBC-ALC: A Weak Blocking Coordinated Application-Level Checkpointing for MPI Programs.” In: *IEICE Transactions* 95-D.3 (2012), pp. 786–796. URL: http://dblp.uni-trier.de/db/journals/ieicet/ieicet95d.html#XuYL12.

[196] Kejiang Ye et al. “VC-Migration: Live Migration of Virtual Clusters in the Cloud”. In: *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*. 2012, pp. 209–218. DOI: 10.1109/Grid.2012.27.

[197] Katherine Yelick et al., eds. *Magellan Final Report*. Online; Accessed: 22-June-2016. 2011.

[198] Sangho Yi, A. Andrzejak, and D. Kondo. “Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances”. In: *Services Computing, IEEE Transactions on* 5.4 (2012), pp. 512–524. ISSN: 1939-1374. DOI: 10.1109/TSC.2011.44.

[199] Sangho Yi, Derrick Kondo, and Artur Andrzejak. “Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud”. In: *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*. CLOUD ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 236–243. ISBN: 978-0-7695-4130-3. DOI: 10.1109/CLOUD.2010.35. URL: http://dx.doi.org/10.1109/CLOUD.2010.35.

[200] Ping Yu et al. “Review: Application mobility in pervasive computing: A survey”. In: *Pervasive Mob. Comput.* 9.1 (Feb. 2013), pp. 2–17. ISSN: 1574-

1192. DOI: 10.1016/j.pmcj.2012.07.009. URL: http://dx.doi.org/10.1016/j.pmcj.2012.07.009.

[201] Kurkcuoglu Z. and Bonvin A. *Science in the clouds: virtualizing HADDOCK, PowerFit and DisVis using INDIGO-DataCloud solutions.* 2016. DOI: 10.7490/f1000research.1111877.1.

[202] Charles Zheng and Douglas Thain. "Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker". In: *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing*. VTDC '15. Portland, Oregon, USA: ACM, 2015, pp. 31–38. ISBN: 978-1-4503-3573-7. DOI: 10.1145/2755979.2755984. URL: http://doi.acm.org/10.1145/2755979.2755984.

[203] University of Zurich. *Elasticluster*. http://gc3-uzh-ch.github.io/elasticluster/. [Online; accessed 21-January-2016].

Acronyms

AMI Amazon Machine Image.

API Application Programming Interface.

ARP Address Resolution Protocol.

AWS Amazon Web Services.

BLCR Berkeley Lab Checkpoint/Restart.

BoT Bag of Tasks.

CEVC Container-based Elastic Virtual Cluster.

CIPRES CyberInfrastructure for Phylogenetic RESearch.

CLI Command-Line Interface.

CLUES CLUster Energy management System.

CLUVIEM Migrable Elastic Virtual Clusters on Hybrid Cloud Infrastructures.

CMFs Cloud Management Frameworks.

CMPs Cloud Management Platforms.

CPPC ComPiler for Portable Checkpointing.

CPU Central Processing Unit.

CRIU Checkpoint/Restore In Userspace.

CSGF Catania Science Gateway Framework.

CSS Cascading Style Sheets.

DCIs Distributed Computing Infrastructures.

DIRAC Distributed Infrastructure with Remote Agent Control.

DMTCP Distributed MultiThreaded CheckPointing.

DNA Deoxyribonucleic Acid.

EBC Event-Based Checkpointing tool.

EBS Amazon Elastic Block Store.

EC2 Elastic Compute Cloud.

EC3 Elastic Cloud Computing Cluster.

EC3aaS EC3 as a Service.

EC4Docker Elastic Cluster for Docker.

EGEE Enabling Grids for E-Science.

EGI European Grid Infrastructure.

GPU Graphic Processing Unit.

GRyCAP Grupo de Grid y Computación de Altas Prestaciones.

GUI Graphic User Interface.

HADDOCK High Ambiguity Driven protein-protein DOCKing.

HPC High Performance Computing.

HTC High Throughput Computing.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure as a Service.

IGI Italian Grid Infrastructure.

IM Infrastructure Manager.

INRA National Institute for Agricultural Research.

IPMI Intelligent Platform Management Interface.

JCR Journal Citation Report.

KVM Kernel-based Virtual Machine.

LAN Local Area Network.

LCG Large Hadron Collider Computing Grid.

LLM Lightweight Live Migration.

LRMS Local Resource Management System.

LTS Long Term Support.

LXC LinuX Containers.

MCMC Markov chain Monte Carlo.

MDS Multidimensional Scalling.

MPI Message Parsing Interface.

MUBDA Master's Degree in Big Data Analytics.

MUCPD Master's Degree in Parallel and Distributed Computing.

MUGI Master's Degree in Information Management.

MUIINF Master's Degree in Computer Engineering.

NAT Network Address Translation.

NFS Network File System.

NGS Next Generation Sequencing.

NIST National Institute of Standards and Technology.

OCCI Open Cloud Computing Interface.

ONE OpenNebula.

OS Operating System.

OTU Operational Taxonomic Units.

PET Positron Emission Tomography.

PGTP Plateforme Génome Transcriptome de Pierroton.

RADL Resource Application and Description Language.

RAID Redundant Array of Independent Disks.

RAM Random Access Memory.

REST Representational State Transfer.

S3 Amazon Simple Storage Service.

SaaS Software as a Service.

SAN Storage Area Network.

SGE Sun Grid Engine.

SLURM Simple Linux Utility for Resource Management.

SSH Secure Shell.

TCP Transmission Control Protocol.

UI User Interface.

VAs Virtual Appliances.

VC Virtual Cluster.

VM Virtual Machine.

VMCA Virtual Machine Consolidation Agent.

VMI Virtual Machine Image.

VMRC Virtual Machine image Repository & Catalog.

VMs Virtual Machines.

VOs Virtual Organizations.

VPN Virtual Private Network.

WAN Wide Area Network.

WN Working Node.

YAML Yet Another Markup Language.