

PROGRAMA DE DOCTORADO EN CIENCIAS, TECNOLOGÍA Y COMPUTACIÓN



CSIC

CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS



Instituto de Física de Cantabria

GRUPO DE COMPUTACIÓN AVANZADA Y E-CIENCIA
INSTITUTO DE FÍSICA DE CANTABRIA
CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS (CSIC)

**SCIENTIFIC CLOUD COMPUTING:
IMPROVED RESOURCE PROVISIONING,
INTEROPERABILITY AND FEDERATION**

—

**COMPUTACIÓN CIENTÍFICA EN LA NUBE:
MEJORAS EN LA PLANIFICACIÓN DE
RECURSOS, INTEROPERABILIDAD Y
FEDERACIÓN**

Memoria presentada por

ÁLVARO LÓPEZ GARCÍA

para optar al grado de Doctor por la Universidad de Cantabria

Santander, Febrero 2016

Agradecimientos

Quisiera mostrar mi agradecimiento a las personas que han contribuido al desarrollo de este trabajo.

En primer lugar me gustaría agradecer a mi director de tesis Jesús Marco y a mi codirector Enol Fernandez, ya que sin su dedicación, supervisión, motivación y paciencia este trabajo no habría sido posible. Gracias también a Isabel Campos, por la confianza depositada a lo largo de estos años, poniendo su experiencia y dedicación a mi disposición.

En segundo lugar, gracias a mi compañero y amigo Pablo, con el que he trabajado de forma cercana durante todo este tiempo, ya que parte del trabajo que hemos realizado juntos ha contribuido al desarrollo de esta tesis. Gracias a Miguel Ángel por toda la ayuda prestada y su disposición siempre que se lo he solicitado.

Me gustaría agradecer también mis compañeros, tanto del Grupo de Computación avanzada y e-Ciencia como del resto del Instituto de Física de Cantabria, con los que he trabajado y compartido mi tiempo durante estos años.

I would like to thank the people from the CC-IN2P3 that made possible my stage in Lyon. Thanks to Hélène Cordier for hosting me, and specially to my colleague Mattieu Puel. Special thanks to Leonello Servoli and Mirko Mariotti, from the INFN and University of Perugia, as they gave me the opportunity to start working in distributed computing ten years ago, making possible that I have arrived at this point.

Gracias a María por todos sus consejos, sus ánimos y su incansable apoyo. Gracias por estar en todo momento conmigo.

Abstract

Nowadays it is difficult to find an area in science or engineering that does not rely on computing techniques. The advancements in Scientific Computing have enabled studies in areas that were otherwise impossible. Due to the large impact of the computational science in the ongoing societal and scientific challenges, several computing research infrastructures have been developed and implemented during the last years, based on different consolidated paradigms, such as High Performance Computing, High Throughput Computing and Grids. Researchers now have access to unprecedented facilities that have revolutionized the way science is performed.

In this context, cloud computing has been embraced as a new emerging and promising paradigm by the scientific community due to the expectations generated around clouds. However, even if it is already being used by some research communities, we cannot neglect the fact that the cloud paradigm has been modelled to satisfy the industry needs for the next generation of enterprise and web applications. Scientific applications are unique on their own, therefore they have unique requirements that the cloud model is not able to satisfy due to its origins in the corporate world. This fact does not necessarily bring the feasibility of the cloud into question, but it is needed to perform an initial study so as to evaluate where the weak points are.

In this thesis I perform this initial gap analysis as a first step for collecting a set of requirements and challenges for Science Clouds. As a second step, in this work I will tackle some of the challenges yielded from this initial study. To this end, we consider two different but complimentary areas: the resource provisioning and federation and interoperability in clouds.

Regarding the first aspect, scientific workloads are different from those in the commercial world, therefore they need special attention from the Resource Provider and the open source Cloud Management Framework perspective. I have addressed how the distribution of Virtual Machine Images can influence the deployment time for the resources requested and how this time can be reduced. Secondly, I propose an implementation of preemptible instances as a way to increase the usage of clouds for opportunistic usage.

The federation and interoperability area is important due to the distributed and collaborative nature of modern science, where different researchers from different institutions join forces to reach their goals. In this context it is important to promote an open and collaborative environment, as other existing infrastructures did in the past. To this end, this work proposes the usage of Open Standards for tackling the

different federation aspects is proposed, including an initial solution for authentication and authorization based on the Virtual Organization Membership Service (VOMS).

In summary, the work and results presented here demonstrate that the cloud can be effectively use for accommodating scientific applications, but it is needed to tackle some outstanding technological and operational challenges to ensure that Science Clouds can satisfy the expectations created around them.

Resumen

Hoy en día es difícil encontrar un campo en ciencia o ingeniería que no haga uso de técnicas de cómputo para alcanzar sus objetivos. Los avances en computación científica han facilitado estudios en áreas que no hubiera sido posible realizar anteriormente. Debido al alto impacto que la computación científica tiene en los retos científicos y sociales, durante los últimos años se han desarrollado varias infraestructuras de cálculo científico, basadas en paradigmas muy consolidados, como la supercomputación, el *High Throughput Computing* o la computación *Grid*. Los investigadores tienen a su alcance infraestructuras de computación sin precedente, lo que ha revolucionado la manera en que se lleva a cabo la ciencia.

En este contexto, la comunidad científica está comenzando a aceptar la computación *cloud* como un nuevo y prometedor paradigma, debido a las altas expectativas creadas en torno al mismo. Sin embargo, y pese a que ya está siendo utilizado por algunas comunidades científicas, no se puede olvidar que este modelo ha surgido del mundo empresarial, por lo que se ha formado de acuerdo a los requerimientos de la nueva generación de aplicaciones web y empresariales. Por otro lado, las aplicaciones científicas tienen características únicas, por lo que sus requerimientos también son especiales. El modelo de computación *cloud* no satisface algunos de estos requerimientos, al haber surgido del mundo empresarial. Este hecho, de todos modos, no pone en cuestión la viabilidad del modelo para acomodar aplicaciones científicas, pero es necesario realizar un estudio inicial para evaluar cuáles son los puntos débiles del mismo.

En este trabajo de tesis realizo un análisis inicial, obteniendo un conjunto de requerimientos y desafíos para un *cloud* científico. Como segundo paso, abordo algunas de las problemáticas identificadas previamente, como son la planificación y obtención de recursos en entornos *cloud* y por otra parte la federación e interoperabilidad de dichas infraestructuras.

En lo que respecta al primer aspecto, la carga de trabajo de una aplicación científica difiere de la de una aplicación en el ámbito empresarial, por lo que necesitan una atención especial por parte de los *middleware cloud* y los proveedores de recursos. En este ámbito he abordado como la distribución de las imágenes de las máquinas virtuales pueden influenciar el tiempo de respuesta de una infraestructura y como el mismo puede ser reducido. En segundo lugar propongo la implementación de un nuevo tipo de instancias *cloud* interrumpibles por otras de mayor prioridad, como un método para proveer a los usuarios de acceso a recursos oportunistas, incrementando el uso global de la infraestructura.

La federación e interoperabilidad de recursos e infraestructuras es un área muy importante dado la naturaleza distribuida y colaborativa de la ciencia moderna, donde científicos de diferentes instituciones, distribuidos de forma global, trabajan juntos para alcanzar sus objetivos. En este contexto es necesario promover un entorno colaborativo y abierto, tal y como otras infraestructuras han hecho en el pasado. Por ello, propongo el uso de estándares abiertos para abordar los diferentes aspectos de la federación, así como una solución inicial para obtener una autenticación y autorización federadas basadas en el *Virtual Organization Membership Service (VOMS)*

En resumen, el trabajo y los resultados presentados en este trabajo demuestran que la computación *cloud* puede utilizarse de manera efectiva para acomodar aplicaciones científicas, pero es necesario abordar algunos desafíos tecnológicos y operacionales pendientes, de forma que los *cloud* científicos puedan satisfacer las expectativas creadas a su alrededor.

Contents

List of Figures	xv
List of Tables	xix
Objectives and Description of Work	1
Objetivos y descripción del trabajo	5
I Background	9
1 Cloud Computing	11
1.1. Cloud Computing Definition	13
1.2. Cloud Computing Characteristics	14
1.2.1. On-demand Self Service	14
1.2.2. Elastic Provisioning and Scalability	15

1.2.3.	Metered Usage and Billing	15
1.2.4.	Multi-tenancy and Dynamic Resource Pooling	16
1.3.	Cloud Computing Actors	16
1.4.	Key and Enabling Technologies	17
1.4.1.	Utility and Grid Computing	17
1.4.2.	Virtualization	17
1.4.3.	Web Services	19
1.5.	Cloud Taxonomy and Classification	20
1.5.1.	Cloud Service Models	20
1.5.2.	Deployment Modes	22
1.6.	Cloud Computing Challenges	23
1.6.1.	Vendor Lock-in	23
1.6.2.	Security and Privacy	24
2	Scientific Computing	25
2.1.	The Computational Problem	29
2.2.	e-Science and the Grid	32
2.3.	What Are The Requirements of Computational Science	33
2.3.1.	Large Capacity	33
2.3.2.	High-end Resources	33
2.3.3.	Availability and reliability	34
2.3.4.	Flexibility	34
2.3.5.	Security and Privacy	34
2.3.6.	Collaboration	35
3	Science Clouds	37
3.1.	Expectations from Science Clouds	40
3.1.1.	Customized Environments	40
3.1.2.	Reduced Costs	42
3.1.3.	On-demand Access	42
3.1.4.	Rapid Elasticity	43
3.1.5.	Execution of non Conventional Application Models	43
3.1.6.	Infrastructure Interoperability and Federation	43
3.2.	Selected Application Use Cases	44

3.2.1.	PROOF	45
3.2.2.	Particle Physics Phenomenology	48
3.2.3.	EGI Federated Cloud	49
3.3.	Science Clouds Open Challenges	52
3.3.1.	Usability Requirements	52
3.3.2.	Resource Allocation Problems	54
3.3.3.	Interoperability and Federation	64
3.4.	Related Work	66
3.5.	Conclusions	68

II Improved Resource Provisioning 71

4 Scheduling and Resource Provisioning in Cloud Management Frameworks 73

4.1.	Scheduling strategy	75
4.2.	Scheduling algorithms	76
4.3.	Scheduling in OpenStack	77
4.4.	Conclusions	79

5 Efficient Image Deployment 81

5.1.	Problem Statement	84
5.2.	Related Work	89
5.2.1.	Shared Storage	89
5.2.2.	Image Transfer Improvements	90
5.2.3.	Other Methods	93
5.3.	Transfer Method Evaluation	93
5.3.1.	Experimental Setup	94
5.3.2.	Test Results	95
5.3.3.	Result Comparison	98
5.4.	Efficient Image Distribution	101
5.4.1.	Evaluation	103
5.4.2.	Image pre-fetch	106
5.5.	Conclusions	107

6	Preemptible Instances Scheduling	111
6.1.	Problem Statement	113
6.2.	Related Work	116
6.3.	Preemptible Instances Design	117
6.4.	Preemptible Aware Scheduling	118
6.5.	Implementation and Evaluation	121
6.5.1.	Evaluation	122
6.6.	Conclusions	126
III	Cloud Federation and Interoperability	129
7	Open Standards for Interoperable and Federated Clouds	131
7.1.	Introduction to Federation	133
7.2.	Related work	135
7.3.	Cloud Federation Open Challenges	136
7.3.1.	On Uniform Access and Management	137
7.3.2.	On Portability	137
7.3.3.	On Authentication and Authorization	138
7.3.4.	On Information Discovery	138
7.3.5.	On Accounting and Billing	139
7.4.	Federation Enabling Standards	139
7.4.1.	Uniform Access and Management	139
7.4.2.	Portability	140
7.4.3.	Authentication and Authorization	141
7.4.4.	Information Discovery	141
7.4.5.	Accounting	142
7.5.	Conclusions	142
8	An Implementation of an Open Standard for the Cloud	145
8.1.	The Open Cloud Computing Interface (OCCI)	147
8.2.	Motivation and significance	148
8.3.	Software Description	149
8.3.1.	Foreword on WSGI	149

8.3.2.	Interacting with OpenStack	150
8.4.	ooi Functionality	154
8.5.	Performance Comparison	154
8.6.	Conclusions	156
9	Federating VO-based Cloud Infrastructures	159
9.1.	Identity Federation, Challenges and Problematic	161
9.1.1.	Current Solutions	162
9.2.	VOMS Support in OpenStack	163
9.2.1.	Keystone External Authentication	165
9.2.2.	Keystone VOMS Integration	165
9.3.	Conclusions	168
IV	Conclusions	171
10	Conclusions	173
10.1.	Contributions	175
10.2.	Publications	176
10.3.	Future Work and Perspective	180
V	Appendices	183
A	Experimental Facilities	185
A.1.	CSIC IFCA Science Cloud Production Infrastructure	185
A.2.	IFCA Batch System	187
A.3.	CSIC IFCA Cloud Test Infrastructure	188
A.4.	OCCI Testing Environment	189
A.5.	HEP Spec Tests	189
	Bibliography	191
	List of Terms and Acronyms	223

List of Figures

1.1. Layered cloud computing general architecture.	21
2.1. Scientific Computing.	27
2.2. Worldwide LHC Computing Grid (WLCG) disk requirements from 2009 to 2017 [54, 55].	30
2.3. WLCG requirements from 2009 to 2017 [54, 55].	31
2.4. Scientific computing ecosystem.	31
3.1. European Grid Infrastructure (EGI) Operating System (OS) flavors in production by number of CPUs.	41
3.2. Parallel ROOT Facility (PROOF) task duration.	46
3.3. PROOF daily request pattern along a three and a half year period.	47
3.4. Capacity evolution of EGI Federated Cloud resources.	50
3.5. Time needed to boot the number of requested instances.	58

3.6. Aggregated performance for the High Energy Physics (HEP) Spec 06 benchmark, with Virtual Machine (VM) sizes and configurations for one host. . .	59
4.1. OpenStack scheduling algorithm.	78
5.1. Boot chart for one VM on an OpenStack cloud.	86
5.2. Boot chart for one VM on an OpenStack cloud, using Copy on Write (CoW) images.	88
5.3. Image popularity based on the number of Virtual Machines spawned. . . .	92
5.4. Waiting time as a function of the number of instances requested, using Hiper Text Transfer Protocol (HTTP) as the transfer method.	96
5.5. Waiting time as a function of the number of instances requested, using File Transfer Protocol (FTP) as the transfer method.	97
5.6. Waiting time in function of the number of instances requested, using Bit-Torrent as the transfer method.	98
5.7. Seconds elapsed from request until all the machines were available.	99
5.8. Seconds elapsed from request until the first machine of the request is available.	100
5.9. CPU usage on the image catalogue.	101
5.10. Network usage on the image catalogue.	102
5.11. Seconds elapsed to boot a machine for 80 requests during 1 h, with the corresponding request trace.	104
5.12. Kernel Density Estimation (KDE) for the time elapsed to boot the requests in Figure 5.11.	105
5.13. Seconds elapsed to boot a machine for 100 requests during 1 h, with the corresponding requests trace.	106
5.14. Kernel Density Estimation (KDE) for the time elapsed to boot the requests in Figure 5.13.	107
5.15. Waiting time in function of the number of instances requested.	108
6.1. Preemptible Instances Scheduling Algorithm.	119
8.1. Proposed Open Cloud Computing Interface (OCCI)'s place in a provider's architecture according to the standard.	150
8.2. OCCI place in a provider's infrastructure, following ooi's architecture. . .	151
8.3. ooi processing pipeline.	152

8.4.	Performance comparison between the existing OpenStack OCCI implemen- tations.	156
9.1.	Authentication request in Keystone.	166
9.2.	VOMS support in Keystone.	168

List of Tables

5.1. Request characteristics.	95
6.1. Configured VM sizes.	124
6.2. Test-1, preemptible instances evaluation using the same VM size.	125
6.3. Test-2, preemptible instances evaluation using the same VM size.	125
6.4. Test-3, preemptible instances evaluation using different VM sizes.	126
6.5. Test-4, preemptible instances evaluation using different VM sizes.	127
7.1. Summary of enabling standards.	143
8.1. OCCI feature comparison of the several implementations.	155
A.1. Type-1 Virtualization Node characteristics.	186
A.2. Type-2 Virtualization Node characteristics.	187
A.3. Type-3 Virtualization Node characteristics.	187
A.4. Type-4 Virtualization Node characteristics.	187

A.5. Type-5 Virtualization Node characteristics.	187
A.6. Type-1 Worker Node characteristics.	188
A.7. Type-2 Worker Node characteristics.	188
A.8. Type-3 Worker Node characteristics.	188
A.9. Test node characteristics.	188
A.10. Test node characteristics..	189
A.11. Test node characteristics.	190

Objectives and Description of Work

Cloud computing has permeated into the IT industry in the last years as one of the most promising business models and paradigms for resource providers and infrastructure operators. The cloud enforces a new vision of compute, storage, network and software as an on demand service following the industry needs, being shaped to fulfil the requirements of the next generation of enterprise and web applications. Cloud Computing has already settled in the industry but it can be still considered in its early phase, as considerable developments and evolutions are being performed.

Nowadays, the advancement in the easy access to computational resources has made possible to develop a new way of doing science, where in many cases researchers spend as much time in front of the computer as in the laboratory. Scientific computing plays a key role in modern science as it has give way to overcoming scientific and engineering problems that could not be addressed in the past. Therefore, successful research and innovation requires access to first class computing infrastructures. The e-Science term has emerged as a way to designate a different form of collaborative research, as scientific and computing infrastructures are collaboratively utilized, sharing operational costs and avoiding resource fragmentation.

In this context, the cloud pervasiveness has led to an increase in its popularity among scientific computing users, and it is now considered a promising paradigm that has the potential to improve the e-Science panorama by giving scientists a new computational framework, filling some of the gaps and drawbacks that exist in other computing infrastructures. However, even if clouds are a good prospect they are far from being perfect for scientific applications. Clouds have been developed by the industry and for the industry, without taking into account science needs. However, Science Clouds are a reality, therefore scientific users need to be taken into account when developing and enhancing existing cloud solutions.

The main objective of this thesis is to advance in the state of the art in the cloud computing model so as to accommodate scientific users and workloads more efficiently. By carrying out an initial analysis of the existing cloud ecosystem, followed by a study of the current literature, including some selected use cases, I extract a list of challenges that Science Clouds need to address to create a difference, when compared with other existing infrastructures.

For some of the depicted challenges I have proposed and implemented a solution to tackle them. On the one hand I have focused on improving the resource provisioning strategies so as to accommodate scientific workloads more easily, proposing some solutions that are of general interest for broader usage types. On the other hand I have considered the interoperability, portability and federation aspects. Science is no longer produced in single institutions, but in global distributed collaborations. Scientific users already have access to globally distributed and federated e-Infrastructures, therefore federation and interoperability are considered a must for a scientific computing infrastructure so as to enable an effective collaboration of their users.

Outline of the Thesis

This dissertation has been split into five different parts, structured as follows:

Background Part I of this work consists of an introductory overview of the two main areas this thesis is related with: cloud computing and scientific computing. In Chapter 1 I give an introduction to the cloud computing term, outlining its definition, its key technologies and a brief classification of the cloud computing infrastructures according to its service models and deployment modes. In Chap-

ter 2 I introduce the scientific computing field. In Chapter 3 I put together the cloud and scientific computing areas, trying to answer the following question: *Is the cloud computing model a feasible paradigm to run scientific applications?* Moreover, I present a list of challenges that need to be addressed when operating and implementing a scientific cloud infrastructure.

Improved Resource Provisioning Part II addresses some of the resource provisioning challenges that are described in Chapter 3. First of all, in Chapter 4 I give an outlook of the current scheduling strategies and algorithms in the major Cloud Management Frameworks (CMFs). In Chapter 5 I cover how the Virtual Machine Image (VMI) distribution method used can influence in the users perception of the cloud reactiveness, proposing a modification on the distribution method and in the scheduling algorithms. In Chapter 6 I propose an implementation of preemptible instances as a way to increase the overall usage on cloud infrastructures without impacting on-demand access to users that require interactivity.

Cloud Federation and Interoperability Part III tackles a complimentary challenging area, introduced in Chapter 3, that is the Federation and Interoperability of cloud computing infrastructures. In Chapter 7 I give an introduction to the infrastructure federation, describing the main challenges that need to be addressed and how the existing or raising open standards can help on building such a federated infrastructure. In Chapter 8 I describe and implement one of the major cloud standards, the Open Cloud Computing Interface (OCCI) for OpenStack. In Chapter 9 I propose the usage of the Virtual Organization Membership Service (VOMS) to provide an initial identity federation on cloud infrastructures, making possible to transition from systems where the authentication is based on VOMS and X.509 certificates to the cloud.

Conclusions Part IV consists only of Chapter 10, where I summarize the main contributions of this dissertation. I also present possible future works, as long as a perspective on the technology evolution.

Appendices Part V collects the appendices to this thesis.

Objetivos y descripción del trabajo

El *cloud computing*, comúnmente traducido como computación en la nube es un nuevo modelo de computación que hace posible ofrecer recursos de computación como servicios bajo demanda. De este modo, usuarios y clientes del *cloud computing* acceden a los recursos cuando es necesario, pagando tan solo por su uso y sin necesidad de una inversión inicial. Este nuevo modelo de computación ha despertado gran interés en todos los sectores de las TIC, desde la industria al mundo académico, debido a las prometedoras características que ofrece. Aún así, hay que tener en cuenta que pese a que el *cloud computing* se ha establecido ampliamente en la industria, aún se puede considerar que está en su etapa inicial, y se esperan importantes desarrollos y evoluciones en esta tecnología.

Hoy en día, los grandes avances y la facilidad de acceso a recursos de computación han hecho posible el desarrollo de una nueva manera de llevar a cabo la investigación, donde el científico pasa tanto tiempo delante de un ordenador como en el laboratorio. La computación juega un papel fundamental en la ciencia moderna, haciendo posible abordar problemas científicos y de ingeniería que serían imposibles de abarcar o de estudiar sin su apoyo. Por lo tanto, es un hecho que el acceso a recursos de computación

de alto nivel es un requisito necesario para llevar a cabo una investigación y desarrollo fructífera y competitiva. La e-Ciencia ha surgido como nueva forma de investigación colaborativa, donde los recursos e infraestructuras son compartidos de forma colaborativa, compartiendo los recursos de operación, evitando de esta manera la fragmentación de recursos.

En este context, la ubicuidad del *cloud* ha suscitado a un creciente interés por parte de de la comunidad científica, y se le considera como un paradigma de computación muy prometedor ya que se considera que puede tener el potencial de alterar el panorama de la e-Ciencia, dando acceso a los científicos a un nuevo modelo de computación que puede satisfacer las necesidades y carencias existentes en otros modelos e infraestructuras. Aún así, y pese al prometedor potencial del *cloud*, este está lejos de ser perfecto para aplicaciones científicas. Las infraestructuras de computación en la nube han sido desarrolladas por la industria y para la industria, sin tener en cuenta las necesidades de la investigación. Sin embargo, los *cloud* científicos son una realidad por lo que las necesidades de sus usuarios han de ser tenidas en cuenta en futuros desarrollos y mejoras de las soluciones existentes.

El objetivo principal de esta tesis es avanzar en el estado del arte del modelo de computación en la nube, de forma que las aplicaciones de cálculo científico puedan hacer un mejor uso de dicho modelo. En un primer lugar se ha realizado un análisis del ecosistema *cloud* junto con la literatura existente así como una serie de casos de uso de aplicaciones científicas que han hecho uso de una infraestructura en la nube. De este modo se ha construido una lista de expectativas y desafíos tecnológicos que un *cloud* científico ha de abordar.

A continuación se han abordado algunos de los desafíos identificados previamente. En primer lugar se ha abordado la problemática de la provisión eficiente de recursos en infraestructuras *cloud* de forma que una infraestructura pueda admitir más fácilmente este tipo de aplicaciones. Se han propuesto algunas soluciones que pese a haber sido desarrolladas e implementadas teniendo en cuenta las necesidades de la computación científica son extrapolables a un uso más generalista de los recursos. Por otro lado, se han considerado los aspectos de interoperabilidad y federación de proveedores de recursos. El desarrollo de la ciencia hoy en día no está limitado a una única institución, sino que se desarrolla de forma global, en colaboraciones distribuidas a nivel mundial. En este contexto, los usuarios tienen acceso a infraestructuras distribuidas, por lo que

la interoperabilidad se considera como un requisito casi imprescindible en cualquiera infraestructura científica.

Descripción de la tesis

Este documento ha sido dividido en cinco partes diferentes, estructuradas tal y como se describe a continuación:

Background La Parte I de este trabajo consiste en una guía introductoria de las dos áreas principales que conciernen al mismo: La computación en la nube (*cloud computing*) y la computación científica. En el Capítulo 1 realizo una introducción al *cloud computing* y su definición, haciendo énfasis en las tecnologías claves que hacen posible este modelo de computación. Asimismo, doy una breve clasificación de las infraestructuras de cómputo en la nube de acuerdo a sus modelos de servicio y de despliegue. En el Capítulo 2 doy una breve descripción de la computación científica. Por último, en el Capítulo 3 reúno ambos términos tratando de responder a la pregunta «¿Es el modelo de computación en la nube un paradigma factible para ejecutar aplicaciones científicas?». Asimismo, en este capítulo presento una lista de desafíos tecnológicos que han de ser abordados cuando se trate de utilizar la computación *cloud* para ejecutar tales aplicaciones.

Improved Resource Provisioning La Parte II aborda algunos de los problemas existentes para la provisión de recursos en entornos *cloud*, tal y como se describen en el Capítulo 3. En primer lugar, en el Capítulo 4 realizo una breve introducción a los algoritmos y estrategias de planificación en los *middleware cloud* más importantes. A continuación hago frente a dos diferentes problemas de los presentados anteriormente. Por un lado, en el Capítulo 5 abordo como el método utilizado para distribuir las imágenes de disco de las máquinas virtuales a ejecutar puede influenciar negativamente en la velocidad de respuesta y la reactividad de una infraestructura *cloud*, proponiendo una modificación en el método de distribución de las imágenes, así como una mejora en el algoritmo de planificación de los *middleware cloud* existentes. Por otro lado, en el Capítulo 6 propongo la implementación y el uso de instancias de máquinas virtuales terminables (*preemptible*) como un método para incrementar el uso global de una infraestructura *cloud*, sin

producir un impacto negativo en los usuarios que requieren de interactividad y acceso bajo demanda.

Cloud Federation and Interoperability La Parte III afronta un área complementaria como es la federación e interoperabilidad de infraestructuras, introducida en el Capítulo 3. En el Capítulo 7 doy una introducción a la federación de infraestructuras de computación en la nube basadas en estándares, describiendo los mayores desafíos existentes y como ciertos estándares abiertos pueden ayudar a construir tal federación de proveedores. Asimismo, se introduce una implementación de uno de los estándares más importantes en el ámbito *cloud* como es el *Open Cloud Computing Interface (OCCI)* para OpenStack en el Capítulo 8. En el Capítulo 9 propongo el uso del *Virtual Organization Membership Service (VOMS)* para proporcionar una federación de identidad en infraestructuras *cloud*, haciendo posible la transición de otros sistemas e infraestructuras cuya autenticación está basada en certificados X.509 y VOMS.

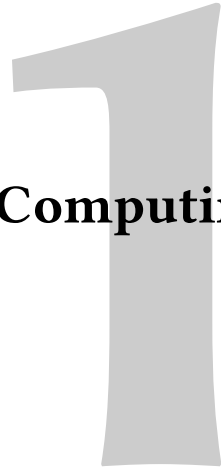
Conclusions La Parte IV consiste tan solo en el Capítulo 10, donde realizo un resumen de las principales contribuciones de este trabajo de tesis. De la misma manera, presento posibles trabajos futuros derivados de esta tesis, así como una perspectiva en la evolución de las tecnologías descritas a lo largo del trabajo presentado.

Appendices La Parte V reúne los anejos a este trabajo de tesis.

A large, light gray, serif letter 'I' is centered on the page. The letter has a classic, slightly flared top and bottom. The word 'BACKGROUND' is printed in a bold, black, serif font across the middle of the vertical stem of the letter.

BACKGROUND

Cloud Computing



Cloud computing has emerged in the Information Technology (IT) field, evolving from just a buzzword and a hype to a complete computing paradigm that has been widely adopted by the industry and community.

In this chapter I will give an overview of the existing definitions of cloud computing (Section 1.1), pointing to the main characteristics that are associated with the cloud. The cloud has its foundations in well established techniques, and it is a combination of them the fact that led to its success, as I will explain in Section 1.4. Then, in Section 1.5 I present a general taxonomy and classification of the different cloud models. Finally, I present the open challenges and risks of the cloud in Section 1.6.

1.1. Cloud Computing Definition

The cloud computing model is a new computing paradigm that enforces the view of compute, storage, network and software as an on demand service. According to the United States' National Institute of Standards and Technology (NIST), it can be defined as [1]:

“(...) 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.”

The same argumentation line—that is, on-demand access, rapid provisioning, minimal management—is shared by other authors. Vaquero et al. [2] introduced the usage of virtualization in their cloud definition:

“Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs. (...) The set of features that most closely resemble this minimum definition would be scalability, pay-per-use utility model and virtualization.”

Buyya et al. [3] defines the cloud as follows, continuing with the usage of virtualization:

“A cloud is a type of parallel and distributed system consisting of a collection of inter- connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.”

Virtualization is also mentioned by Foster et al. [4], defining a cloud as:

“A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.”

Armbrust et al. [5] define the cloud taking into account three novel aspects from the hardware provisioning and pricing point of view:

“The appearance of infinite computing resources available on demand (...). The elimination of an up-front commitment by cloud users (...). The ability to pay for use of computing resources on a short-term basis as needed (...).”

Some common key characteristics can be inferred from the multiple definitions given. These are not exclusive from the cloud, but it is the combination of all of them what makes the difference against other computing models such as the grid, High Performance Computing (HPC) and High Throughput Computing (HTC) clusters and other computing models. The two most important characteristics that have been they key for the success of the cloud computing model are the *on-demand self service* and the *elastic provisioning*, but there are other important facts that differentiate the cloud.

1.2. Cloud Computing Characteristics

1.2.1. On-demand Self Service

The users are able to provision and manage their own computing environment according to their needs, without further intervention from the provider. This character-

istic implies that he is able to request any kind of resources offered by the provider such as computing resources, storage space, network connectivity, etc.

This feature is considered one major advantage over other computing models such as the grid, where the user is tied to the environments defined by the resource provider [6]. On the other hand, this self service manageability implies a stronger awareness of the resources being used and the environment involved.

1.2.2. Elastic Provisioning and Scalability

The cloud model tries to deliver easily and rapidly the resources to the users, in a short-deadline basis. This feature, called *elasticity*, means that users are able to scale in and out their infrastructure so as to satisfy the real demand, not only by increasing their capacity, but also by shrinking it whenever it is not needed. Normally, the elasticity creates an illusion of infinite resource capacity for the users, that seem to be able to request as many resources as they want.

The elasticity, together with a self-service interface, makes possible the implementation of an automated resource management, making possible to satisfy demand surges by quickly reacting upon them.

1.2.3. Metered Usage and Billing

Although the pricing scheme may vary from one provider to another, normally a metered usage is in place [7]. This means that resources are being accounted by their usage, rather than following a subscription model. This billing scheme is an implicit requirement that follows from the *on-demand* and *elastic* self service characteristics that make a subscription billing scheme not feasible or appropriate. Users may not be aware about what their future usage will be, therefore subscribing for a service paying a fee may imply that they are over or under committing, whereas with a metered usage and billing customers are being charged only for the resources that they are actually using.

This accounting model introduces several key benefits:

- There is no upfront commitment, since the customer is only accounted for what it is being used. This lowers the barrier for accessing computing capacity to small groups and companies, leading to a faster Return of Investment (ROI) and a lower cost of ownership.

-
- Introduces an ethics of resource conservation, making users aware of what they are actually using.
 - There is a better understanding of the actual resource utilization for both the resource provider and the cloud users.

1.2.4. Multi-tenancy and Dynamic Resource Pooling

Multi-tenancy can be defined as the ability for a software or provider to deliver a service to several parties simultaneously [8] and it is a key factor in cloud computing, since services owned by several users are being co-located in the same resources [9], permeating through the several cloud layers [10] described in Section 1.5.

Users have transparent and location-independent access to a uniform pool of abstract resources. That pool is dynamically managed under the hood by the resource provider that is able to reassign resources so that the user's demands are satisfied. This capability also makes possible that a provider reassigns the resources to satisfy their capacity plans or to save costs.

1.3. Cloud Computing Actors

Following the definition from Armbrust et al. [5], we can assume three different roles in the cloud environment. However, these roles can sometimes overlap, depending on the service model and the deployment mode (Section 1.5).

Cloud Resource Provider A Resource Provider (RP) manages a set of hardware and software systems, providing access to them following the cloud computing model. The RP is responsible for allocating the resources so that they meet the Service Level Agreements (SLAs) agreed with the other cloud actors.

Cloud User The cloud users access clouds and manage directly their resources. They can build services on top of those resources —with an agreed SLA— or they can exploit them directly, acting also as the end users.

End User The end users generate the workloads that are going to be executed using the cloud resources. End users behaviour influences on how resources are managed and provisioned, even though they do not have management decisions.

1.4. Key and Enabling Technologies

Offering computing services on-demand with a metered usage is not a revolutionary model, as similar concepts have been studied and proposed during the last decades. However, as it will be explained along the following sections, it is the combination of several technologies and factors what makes the cloud computing model obtain the huge success it currently has.

1.4.1. Utility and Grid Computing

The utility computing model [11, 7] tries to deliver computing services in a pay-as-you-go basis, similar to other daily utilities that we can access (such as gas and electricity). One illustrative example of the utility computing model is the grid computing paradigm [12, 13], where computing power and storage is delivered in an infrastructure following an analogy with the electrical power grids.

Cloud computing has some similarities with the grid computing [13], and it is commonly seen as a natural evolution and continuation of it [4]. The grid changed the way of delivering resources, and it has been specially pervasive within many scientific communities. Infrastructures, such as EGI [14, 15] make possible to tackle new scientific challenges [16] that were otherwise impossible to cope with. Also, the cooperative aspects of the grid model were specially suitable for large and globally distributed collaborations.

1.4.2. Virtualization

As it has been described in Section 1.1, one of the premises of the cloud computing model is the abstraction of the resources that are delivered to the users as a kind of utility. In this context, the development and irruption of the virtualization techniques into the datacenters played a key role.

Computing virtualization refers to the abstraction and encapsulation of the hardware resources into smaller Virtual Machines (VMs) by the usage of an hypervisor or Virtual Machine Monitor (VMM). Therefore, a given physical host can then execute several VMs that are isolated –to the extent implemented by the VMM– between them.

This is not a new topic in Computer Science, since it has been present in the IT field for decades. Back in 1974, Goldberg stated that "Virtual machines have finally arrived"

[17], identifying the benefits and advantages of virtualization for its usage in mainframe computing.

However, with the adoption of the low cost personal computers the interest for the virtualization decreased, helped by the fact that modern architectures did not meet the requirements for being efficiently virtualizable, as already stated by Popek and Goldberg in their 1974 paper [18]. Their theorems states that for an efficiently virtualizable processor, its sensitive instructions must be a subset of its privileged instructions. This was not the case for the early Intel x86 architectures —one of the most popular architectures in the 1995-2005 period— so any VMM was forced to rely on software circumventions to meet those requirements. This fact produced the inefficient and heavyweight solutions that rendered virtualization unattractive, due to the performance loss that it introduced [19].

Nevertheless, virtualization became attractive again with the rebirth of the para-virtualization by the Xen project[20], making possible to achieve near to bare-metal performance on the running para-virtualized systems at the cost of porting and modifying the guest Operating System (OS). In a para-virtualized system (a new word for an old IBM concept back in 1970s [21]), the OS is modified so that it access a software interface (offered by the hypervisor) that is similar, but not identical, to the underlying hardware. This way, a simpler and more efficient VMM can be designed, reducing the performance loss introduced by the virtualization [20].

Besides, hardware improvements introduced in 2005 by means of the Intel Virtualization Technology (Intel-VT) [22] and AMD-V [23] extensions focused on making virtualization much easier, without needing to adapt the guest OS. This led to the extension of the Xen hypervisor to support this new hardware assisted virtualization [24] and the birth of the Kernel-based Virtual Machine (KVM) monitor [25].

Today virtualization is present in many datacenters, due to the important benefits that it delivers to the providers and operators, enabling to manage their infrastructure more efficiently. Virtualization is they key to carry out server consolidation [26] —that is, the execution of several VMs inside one single physical server— thus decreasing hardware and maintenance costs as the number of physical machines needed inside a datacenter is reduced. This fact leads to important energy savings [27] and makes possible to develop more advanced energy-aware placement policies [28, 29].

Virtualization makes also possible to decouple to a certain extent the OS and software

environment from the underlying hardware. Virtualization functions in hardware devices make possible to share some hardware resources (like network cards, Infiniband cards or GPUs) with several VMs. Moreover, it enables the coexistence of different OSes, avoiding incompatibilities between different softwares and libraries.

This rebirth of the virtualization has boosted the development of the cloud computing. With virtualization already in the datacenters, it is possible to abstract the physical hardware into virtual resources, and provide the capability of pooling those resources dynamically, allocating them to the users on demand.

It is worth notice that although some authors state that cloud computing can take place without virtualization, being it an implementation detail [30], it is true that the cloud could hardly exist in its current form without virtualization in place. This second virtualization trend made possible that providers could start rethinking in a way of offering their resources, leading to the cloud success.

1.4.2.1. Virtualization Performance Loss

Traditionally virtualization has been associated with a performance loss, due to the overhead introduced by the VMM. However, nowadays it is assumed that the performance penalty can be neglected [31] with a proper hypervisor tuning so that it is put on a level comparable to the bare metal hardware. This fact is specially true when using modern hardware adapted with special characteristics for virtualization [32] or when using other virtualization techniques such as OS-level virtualization [33, 34].

Studies related with efficacy of parallel communications in virtualized HPC environments have been carried out in [35, 36]. More recent efforts introduce also the characterization of low-latency communications –by means of the virtualization or passthrough of Infiniband interconnects– as described in [37, 38].

1.4.3. Web Services

With the raise of the Web 2.0, the concept of Web Services (WS) became popular as a method of communicating two different systems over the Internet. The W3C [39] defines it as follows:

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface de-

scribed in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using Hiper Text Transfer Protocol (HTTP) with an XML serialization in conjunction with other Web-related standards.”

Over the last years the application of the REST architecture to Web Services has been popularized. REST has simplified the original XML-based web services by applying the same simple concepts that led to the web success, thus the so-called RESTful Application Programming Interfaces (APIs) have emerged.

The Web Services have made possible to put together disparate systems as building blocks for more complex applications, relying on open and well established standards and protocols (such as SOAP and REST). This has contributed to the advances in software and systems integration and composition over the internet. Cloud computing APIs are offered as web services, making possible to use them as building blocks for more complex applications.

1.5. Cloud Taxonomy and Classification

There are multiple cloud taxonomies in the scientific literature [40, 41, 42]. However, the most widespread classification follows the simple taxonomy proposed by the NIST [1] classifying clouds based on the service model offered (also called by other authors business model) and the cloud deployment model.

1.5.1. Cloud Service Models

Cloud computing has been an umbrella where different service and business models have been categorized. Due to the hype caused by the cloud trend, a lot of existing computing services have been categorized under the cloud term, even if they existed well in advance.

Currently three main service models are being differentiated: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Each of these models corresponds to a different level of abstraction that is being provided. These models are combined in a layered model, as shown in Figure 1.1 where services of a

higher layer are composed by services of the underlying ones. This classification allows for the definition of additional models following this *XaaS* pattern, but normally they are a particular case, specialization or combination of these three basic models.

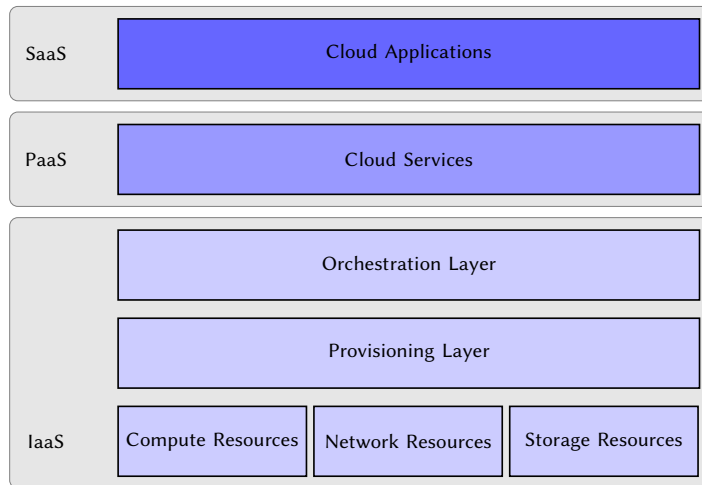


Figure 1.1: Layered cloud computing general architecture.

Infrastructure as a Service (IaaS) The IaaS model corresponds to the lowest level of abstraction and it can be considered as the foundation of the cloud model. IaaS offers its *infrastructure* resources —i.e. computing, networking, storage, etc.— to the users, that are able to manage them on their own. This way users can deploy its own OS, software, network configuration, etc.

This service model works by abstracting the underlying fabric (i.e. the physical resources) into a uniform resource layer —by virtualizing or encapsulating the raw resources— that is then exposed to the users. Users get transparent access to this layer as if they were using the bare metal resources, being able to deploy any infrastructure on top of it without the extra burden of directly managing the different physical resources.

Platform as a Service (PaaS) The PaaS layer is a second step in the abstraction level, where resources coming from an IaaS —that is, storage, network and compute— are composed so that they can be consumed by the users without requiring the

management or the knowledge of the underlying infrastructure. A PaaS should offer an environment where a user can deploy and manage its applications using the libraries, software, tools, APIs, etc. supported by the provider, but without knowing and without having control on the resources.

This service model makes possible to deliver complex applications involving different components to end-users without the need of direct managing of the machine configurations and deployment, but it also allows to define the requirements of those applications, so that the platform layer is able to orchestrate the resources.

Software as a Service (SaaS) The SaaS layer corresponds to the higher level of abstraction. This layer comprises the applications that are running on top of a cloud infrastructure. Access to SaaS applications are normally addressed using ad-hoc thin clients executed inside web browsers or applications that are executed on tablets or smartphones, directly addressing the end user.

1.5.2. Deployment Modes

Cloud computing infrastructures can be also classified according to its deployment mode, not only by its service model. Some authors [43] classify cloud infrastructures in two types: Private and Public clouds, while others extend it with more particular cases of the aforementioned: Hybrid and Community clouds [1]. This classification makes emphasis on who are the users, owners and operators of the infrastructure.

Public clouds In a public cloud, access is open to anyone willing to use the infrastructure (i.e. they imply multi-tenancy), either free or in a pay-per-use basis and are operated by third-party companies or organizations. The cloud computing commercial providers deliver this kind of service to the users.

Private clouds A private Cloud infrastructure is one that is operated exclusively for the sole usage of a given organization or its customers. However, in spite of its name, it does not need to be owned or operated by the same organization that makes use of it, that can leverage such tasks on any external party.

Community clouds A community cloud is a special case of a public cloud. In this case the access is granted only to a community of users with shared interests, whereas in a public cloud it is opened to the general public. As in the public cloud

case, it does not imply that the infrastructure is operated or owned by the user community.

Hybrid clouds An hybrid cloud is a mixed operation mode of the above (private, public and/or community clouds). A common example is a organization private cloud that is able to scale up when it has reached its complete capacity, by using resources of a public cloud provider.

1.6. Cloud Computing Challenges

The cloud still presents some challenges, risks and limitations that need to be tackled in order to take advantage of its features. Armbrust et al. [5] identified the *a*) availability/business continuity, *b*) data lock-in, *c*) data confidentiality and auditability, *d*) performance unpredictability, *e*) scalable storage, *f*) scaling quickly, *g*) reputation fair sharing, and *h*) software licensing as the main limitations and risks for cloud computing. Currently, it is widely accepted that the two main challenges for clouds are the aspects regarding the data and vendor lock-in, and all the aspects around cloud security [44].

1.6.1. Vendor Lock-in

Many cloud users consider the vendor lock-in [5] as a major concern. The cloud is still a maturing model, therefore the computing infrastructures and middleware do not offer a high degree of interoperability, making it difficult for the users to migrate their resources from one cloud to another. Commercial providers do not adopt open cloud computing interfaces, making even more difficult that interoperability.

The vendor lock-in and interoperability problem could be alleviated via the promotion and adoption of open standards [45], both for the cloud and for the underlying virtualization layers. Standardization efforts should take into account that cloud resources are being executed on top of virtualized infrastructures, therefore it is also needed to consider the compatibility and migration from one virtualization stack to another one and not stopping only at the cloud layer.

1.6.2. Security and Privacy

Every new computing paradigm introduces new security risks, and the cloud is not an exception. Security in cloud is two folded [46]. On the one hand, we should consider the security regarding the workloads being executed by the users, that are exposed to the network [5], because their cloud infrastructure is not any longer on their premises but it is outsourced. Several research works exist on this area so as to ensure enough security and privacy is delivered in a convenient way to the cloud users [47, 48].

The second part —and probably the one that more concerns raises— is due to the cloud’s multi-tenancy and the isolation between the different resources being executed in a single infrastructure or in a single virtual machine. This isolation between different computations —malicious or not— is normally fulfilled by using virtualization [49]. However, even though virtualization has widely accounted into its benefits that it is possible to securely isolate virtual machines running in a same host, this is not always true: vulnerabilities for all kinds of hypervisors do exist, therefore virtualization is clearly introducing multi-tenancy security issues that will never exist if the resources were not virtualized [50]. It must be said however that any efficient operation of the current computing resources, including multicore machines, would need to address similar problems.

2

Scientific Computing

Scientific computing –not to be confused with Computer Science– can be defined as the efficient usage of computer processing in order to solve numerical problems present in science and engineering. Heath [51] defines scientific computing as follows:

“Numerical analysis is concerned with the design and analysis of algorithms for solving mathematical problems that arise in computational science and engineering. For this reason, numerical analysis has more recently become known as scientific computing.”

Karniadakis and Kirby define it as the “intersection of numeral mathematics, computer science and modelling” [52], as shown in Figure 2.1, being at the heart of simulation science.

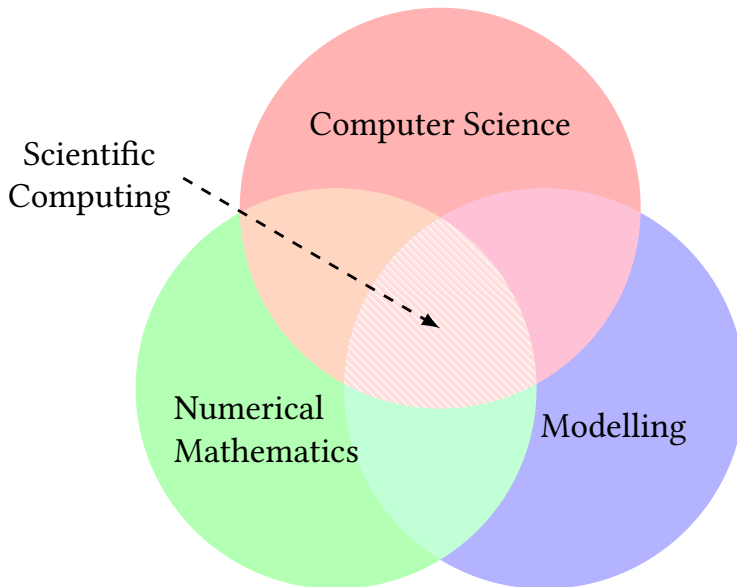


Figure 2.1: Scientific Computing.

Traditionally, the application of the computational science to tackle many scientific problems comprised the following steps [52, 53] to complete the simulation cycle:

1. Scientists elaborate a mathematical model of the system subject of study.

-
2. They develop and implement the algorithmic procedures to numerically solve the model equations.
 3. They execute this software in some kind of computing system and collect the generated results.
 4. They analyze the obtained data and visualize the results, interpreting and validating them.

The use of scientific computing has permeated so deeply in the scientific world that nowadays it is difficult to find an area in science or engineering where scientists do not rely on the simulation and modelling of the physical systems they are studying, spending as much time in front of a computer as in the laboratory. The traditional methods are more and more being substituted by the *simulate and analyze* approach. Computational science has made possible to study systems that would have been impossible or not affordable to tackle before by means of theoretical, observational or experimental methods by themselves. As a matter of fact, a large fraction of science today relies heavily on computing and it would be impossible to do research on some areas without relying on scientific computing.

A wide number of the problems that are addressed through the application of computational science are aimed to understand some natural phenomena (in the case of science) or to design some device or artifact (in the case of engineering). It is being applied in cases where it is difficult to obtain results directly in a laboratory, or in cases where it is more convenient as it economizes research costs (for instance it is more effective to simulate the air flow for a new vehicle or the fluid dynamics of a new kind of wastewater treatment reactor, compared with the traditional prototype design).

Moreover, scientific computing is not only focused just on the simulation of systems or phenomena. Some experiments produce such a vast amount of data that our capacity for assimilating it is overflowed. Scientific computations can generate data during the course of their execution, and they need from input and output data. All of these data have to be curated, stored, transferred, visualized, filtered, classified, processed and analyzed. Data management is one of the current trending topics of the Computer Science, as the literate reader will know by the BigData term.

In Summary, scientific computing is a multidisciplinary field that has its roots on numerical analysis, modelling and simulation, and Computer Science so as to get the

most convenient way of using up the current computer systems to solve scientific and engineering problems in an efficient way.

2.1. The Computational Problem

The advance in the computing power allows to simulate systems that some years ago would be impossible. Nevertheless, even with all the improvements in the field, the problems that are undertaken by the computational scientist are more and more complex, therefore they are normally not affordable by means of traditional computation (i.e. using a single computer), implying the usage of specialized resources and computing clusters.

Another important aspect is the fact that there are some disciplines that are data-driven as well. The exponential technology improvements have led to a situation where vast quantities of data are being produced and collected in fields such as astronomy, meteorology, finances and biology; just to cite some of them.

One good example of this kind of data-driven science can be the experiments taking place at the Large Hadron Collider (LHC) facilities at CERN, Geneva. The LHC experiments (ALICE, ATLAS, CMS, LHCb) operate the Worldwide LHC Computing Grid (WLCG) infrastructure, whose disk requirements from 2013 to 2016 are shown in Figure 2.2 [54, 55]. This data needs to be preprocessed, distributed, analysed and archived properly, requiring from large computing facilities that are able not only to store the data, but also to process it. The CPU requirements for the WLCG are shown in Figure 2.3 [54, 55].

Due to the large impact of the computational science in the societal and scientific challenges that exist [56, 57], several computing research infrastructures have been developed and implemented during the last years. Researchers have access to unprecedented facilities that have revolutionized the way science is performed. However, there is no rule of thumb for a computing infrastructure to be suitable for all kinds of scientific workloads. Different applications have different requirements, thus scientists need access to a large range of computing facilities and infrastructures. For example, data-bound applications will benefit from a high-speed data access, while parallel programs will need a high-speed and low latency communication channel instead.

Scientific computing traditionally spans a wide range of computing models and

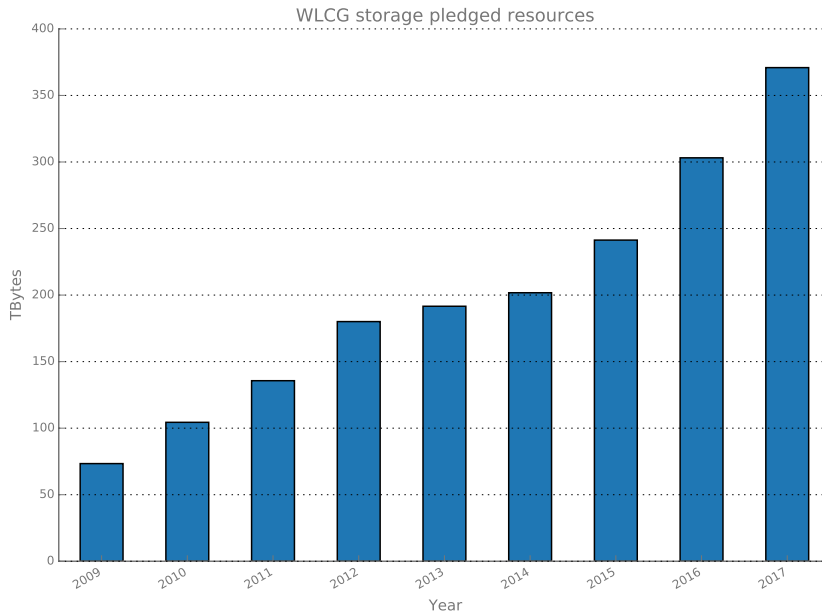


Figure 2.2: WLCG disk requirements from 2009 to 2017 [54, 55].

infrastructures as shown in Figure 2.4. From the bottom of the pyramid to the top, the first layer is composed of the personal computers and workstations that are directly used by scientists for their final analysis. Clusters are the next in size, and they span a broad range of topologies. They are typically resources owned by a single institution or research group, providing local researchers with computing power. The top of the pyramid is occupied by the High Performance Computing (HPC) resources, or supercomputing referring to the usage of large and massively parallel computers run in specialized high-end centers, whose systems are normally designed to deliver a high performance to large and tightly parallel applications.

With the improvements in network communications over the last years, and the current internet ubiquity it was possible to consider technically feasible the usage of distributed computing –not to be confused with parallel computing. Distributed computing not only makes possible to get additional computing power from systems that are geographically distributed, but it also ensures its availability and an efficient usage of the resources. A good and successful example of a distributed computing system is

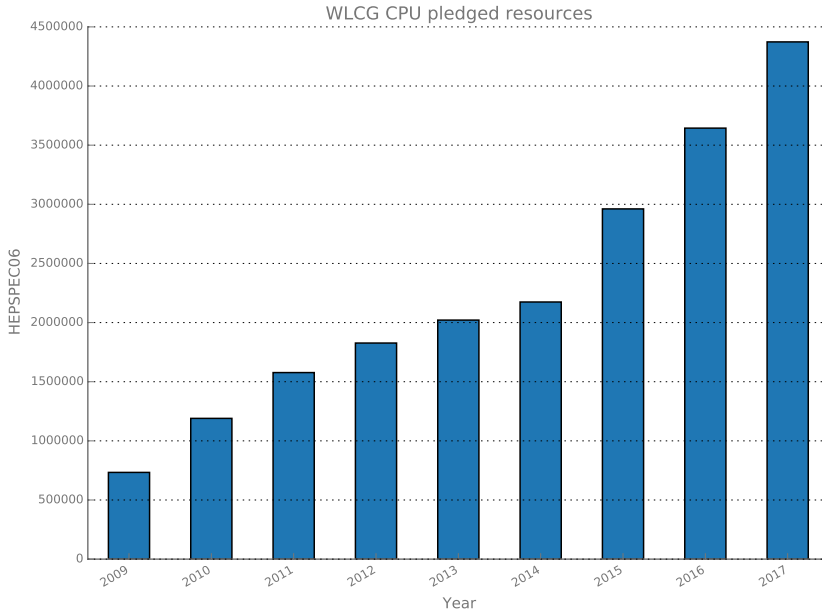


Figure 2.3: WLCG requirements from 2009 to 2017 [54, 55].

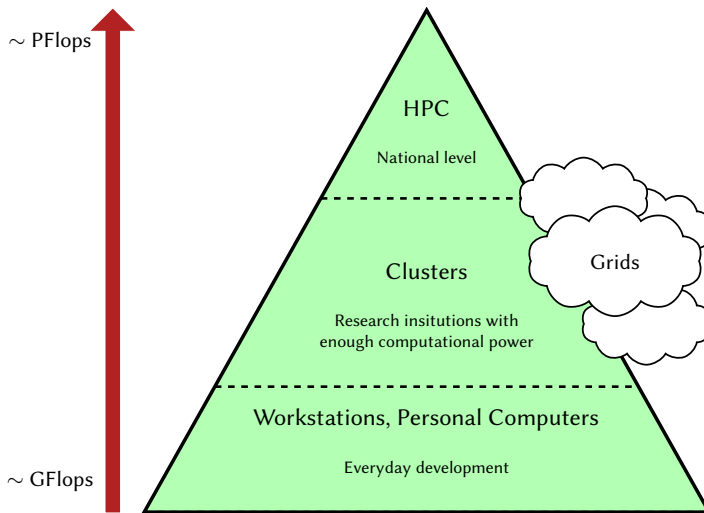


Figure 2.4: Scientific computing ecosystem.

the grid, as we will cover in section 2.2.

2.2. e-Science and the Grid

The e-Science term was originally crafted by Dr. John Taylor from the United Kingdom office of science and technology as follows: “e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it” [58]. During the last decade, the term e-Science has been used as an umbrella to refer to computationally intensive science that leverages highly distributed and disparate computing network and data environments, promoting worldwide collaborations between scientists.

Currently it is notorious how science is not any longer produced in a single research institution, therefore e-Science has become a fundamental actor in the computational science arena as it creates a suitable environment for such worldwide collaborations.

Undoubtedly, one of the key computing models that triggered the e-Science evolution was the grid. The term grid computing was initially defined as “a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities” [59] and aims to be similar in concept to the electrical power grids. The grid computing model aims to deliver access to either HPC and High Throughput Computing (HTC) resources, federating heterogeneous providers that are geographically distributed. Currently, the grid has settled as one of the most prominent distributed scientific computing infrastructures.

This computational model is sometimes referred as distributed HTC. The term HTC is used in contrast with HPC infrastructures, as they are architecturally and conceptually different, being governed by different access policies. Even though it is possible to access HPC resources using the grid, HPC resources are normally governed and driven by strict access policies, in contrast with the more loose policies that govern the grid.

The grid has made possible that scientists could access vast amounts of resources without the overhead needed for acquiring and maintaining ad-hoc computing infrastructures. In this way, its pervasiveness has played a key role in the spread of the computational science. Currently a considerable number of grid infrastructures exist. The European Grid Infrastructure (EGI) [60] operates a federation of resource providers, delivering integrated computing services —comprising more than 800 000 CPU cores

[61]— to European researchers. Similarly, the Open Science Grid Consortium (OSG) [62] facilitates a similar access to HTC resources within the United States.

2.3. What Are The Requirements of Computational Science

The complex set of computations that are usually performed within the scientific applications require unique features from the infrastructures where they are going to be executed. In the following sections we will briefly discuss the corresponding requirements.

2.3.1. Large Capacity

Researchers tend to consume vast amounts of computational resources, as it was already exposed in Section 2.1. Scientific computing facilities need to deliver enough capacity so as to satisfy the computational needs of their users. The major challenges that computational science is tackling increase year after year, spanning a large range of disciplines such as weather and climatology, astrophysics, high energy physics, materials science, life sciences, and engineering; just to cite some [56]. Infrastructure federations like EGI and OSG facilitate access to large pools of resources to scientists that are able to access resources from several institutions and providers.

2.3.2. High-end Resources

Leaving aside the obvious CPU performance, there are other aspects that may be influenced if the underlying hardware is not efficient enough and is not able to deliver the required performance.

For instance, data-intensive applications will require an efficient access to the data being analyzed and scientific applications often involve parallel calculations that can span several nodes by utilizing some kind of message passing application programming interfaces. In these cases, it is required that the resource provider is able to provide high-end network interconnects that are able to deliver high throughput and very low latency communications —such as Infiniband— so that the application can exploit its parallelism effectively.

In addition, there are some calculations that can be performed more efficiently by a Graphics Processing Unit (GPU) instead of a CPU. GPUs are high-performance many-

core processors capable of very high computation and data throughput, that can be used for general purpose computing —this technique is called General Purpose Computing on GPUs (GPGPUs)—. Some scientific applications can leverage these devices to obtain their results much faster.

2.3.3. Availability and reliability

Besides obtaining enough capacity over long periods of time, sometimes the resources need to be available when the researchers need them, not when the provider is able to deliver them. For instance, when research is bounded and driven by external factors such as disaster mitigation it is not feasible for the researchers to wait for a long period of time in order to obtain their resources.

Moreover some workloads can take weeks or months to complete. Scientific users need highly reliable infrastructures, as otherwise a failure would imply the loss of several days of work.

2.3.4. Flexibility

Scientific software evolves fast in time, with the necessity of analyzing new data, simulating new scenarios and environments or simply because of the normal improvement and bug fixing that happens in every software. Therefore, scientific applications require a certain degree of flexibility from the underlying infrastructure, posing a challenge for scientific centers to keep up to date scientific applications, that cannot be managed easily by the users without dedicated cooperation from the infrastructure operators.

2.3.5. Security and Privacy

Security is always a concern in large-scale shared facilities, being them scientific or not. Research needs to remain confidential, due to the use of sensitive and confidential data —such as medical records— or until the authors decide to publish their results whenever they decide they are camera ready. Enough privacy needs to be guaranteed, not only during the executions in multi-tenancy environments, but also for the long-term preservation of data.

2.3.6. Collaboration

Collaborations in science are essential, and distributed collaborations are nowadays more and more common. A distributed scientific computing infrastructure will enable the linking of those scientific communities, encouraging the sharing of resources and increasing the transmission of knowledge, creating interdisciplinary environments where researchers and technical people can share and improve their skills.

3

Science Clouds: Context, Definition, Expectations and Challenges

*Part of this chapter has been published as: **Á. López García** and E. Fernández-del-Castillo. "Analysis of Scientific Cloud Computing requirements". In: Proceedings of the IBERGRID 2013 Conference. 2013, p. 147 158*

As I already explained in Chapter 2, modern scientific computing takes advantage of several computing infrastructures that are grouped through the e-Science umbrella. The best example and the most concrete realization of the e-Science has been achieved by means of the Grid computing.

Cloud computing has the potential to improve the e-Science panorama [64] by giving scientists a new computational model that will fill some of the existing gaps in other infrastructures. Several studies have been performed so as to evaluate the performance and feasibility of the current cloud offerings. However, even if performance is a key factor, there are other aspects that need to be taken into account. The cloud in its current form presents several drawbacks that hinder its consolidation among the scientific users.

The success gathered by the cloud computing in the industry was not common into the scientific computing field. Virtualization has been considered as one of the main show stoppers due to the performance penalty introduced [34], but already explained in Section 1.4.2 from Chapter 1, this overhead can be neglected, and the scientific community is not considering this anymore an obstacle [31, 65]. Therefore, leaving apart the performance issue, a gap analysis for an Infrastructure as a Service (IaaS) aiming to deliver cloud resources for scientific usage—that is a Science Cloud—is needed, trying to answer the following questions taking into account the perspective of all stakeholders involved: users and resource providers.

- Is the cloud computing model a feasible paradigm to run scientific applications?
- What are the expectations for a Science Cloud from the user's standpoints?
- Does the current cloud offerings satisfy these expectations?
- What are the main challenges that an IaaS Science Cloud has to tackle?

The rest of the chapter is organized as follows. Section 3.1 outlines some of the expectations created around clouds from the scientific user perspective. In Section 3.2 I study some selected applications from several user communities that have migrated their workloads to a cloud. In Section 3.3 I elaborate the challenges for cloud infrastructures running scientific applications. Section 3.4 describes similar work in the area. Finally, the conclusions are presented in Section 3.5.

3.1. Expectations from Science Clouds

Many of the features of the cloud computing model are already present in current scientific computing infrastructures. As exposed in Chapter 2 academic researchers have used shared clusters and supercomputers since long, and they are being accounted for their usage in an utilization basis —i.e. without a fixed fee— based on their CPU-time and storage consumption. Moreover, Grid computing tried to make possible the seamless access to worldwide-distributed computing infrastructures composed by heterogeneous resources, spread across different sites and administrative domains.

However, the cloud computing model can fill some gaps that are impossible or difficult to satisfy and address with any the current computing models in place at scientific datacenters [66, 4]. In the following sections I describe the major benefits that the cloud can bring to scientific communities and the expectations created around Science Clouds.

3.1.1. Customized Environments

One of the biggest differences between the cloud model and any of the other scientific computing models (High Performance Computing (HPC), High Throughput Computing (HTC) and Grids) is the flexibility for the allocation of custom execution environments. While in conventional scientific infrastructures the execution environment is completely fixed by the providers (e.g. European Grid Infrastructure (EGI), one of the majors grid infrastructures with more than 300 resource centers providing more than 800 000 cores [61], even if it supports only a few Operating System (OS) flavors as shown in Figure 3.1, the vast majority of the resources are using the same RedHat Linux distribution family), in the cloud model the execution environments are easily adaptable or even provided by the final users. This enables the deployment of completely customized environments that perfectly fit the requirements of the final scientist's applications.

This lack of flexibility in the current computing infrastructures —where a specific (or a very limited group) OS flavor with a specific set of software and libraries is deployed across all the available computing nodes— forces most applications to go through a preparatory phase before being executed to adapt them to the execution environment idiosyncrasies, such as library and compiler versions [67]. As a consequence, once an

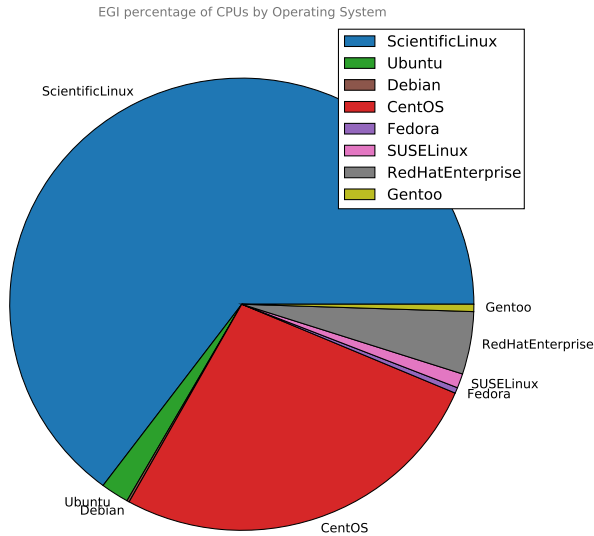


Figure 3.1: EGI OS flavors in production by number of CPUs. Data has been extracted from the EGI production information system.

application is correctly deployed, it is hard, complex and time consuming to perform any updates or apply any patches.

Moreover, some scientific applications use legacy libraries that are not compatible with the available environments, rendering this preparation step quite time-consuming or even impossible in some cases. The users could get rid of this procedure to an extent if they were able to provide its own computing environment, that will be the one used for its computations.

The requirement of a fixed environment and the absence of customization has been identified [68, 6] as one of the main show-stoppers for many scientific communities to adopt the grid computing model. Only large communities are able to tackle this issue, thanks to dedicated manpower that manage and adapt their software deployment to all the available scenarios. The *Long-Tail* of Science —the large number of laboratories and researches that do not have access to specialized Information Technology (IT) staff in contrast to *Big-Science* experiments —may find their research hindered by the lack of easy access to computational power [69]. If we take the other way around, the cloud

flexibility will allow full control over the OS and software environment, and it has been found as a substantial advantage of the cloud over grids and clusters [6].

The abstraction of the underlying resources that is being provided by the cloud (leveraging a virtualization layer), will also alleviate the problems introduced by different and heterogeneous hardware [6] present in the Grid.

Finally, this cloud characteristic makes possible for the long-term preservation of the application environment. This fact opens the possibility of running legacy software with current and future hardware, which may help in the long-term preservation of data (and analysis methods for those data) of scientific experiments [70].

3.1.2. Reduced Costs

Exploiting an in-house computing infrastructure means that research groups need to have budget for buying and maintain the hardware equipment as well as the qualified personnel that brings the expertise needed to operate the infrastructure. Research groups, specially those in the Long-Tail of Science often run in a tight budget that does not allow for large and sustained expenditures on infrastructure, therefore Science Clouds could play a key role (offering IaaS resources).

3.1.3. On-demand Access

Arguably, one of the facts that have made the cloud a success is the illusion of infinite resources that can be accessed on a pay-as-you go basis by the users. Cloud customers perceive that they can access as much resources as they can afford, without restrictions, as far as they can pay for them.

However providing an infinite resource capacity is not feasible in scientific datacenters. In these environments idle resources are not desirable, leading to situations where there are literally no available resources for satisfying a user request. Nevertheless, on-demand access to them is still a mandatory feature for users requiring interactivity, and there should be a trade-off between delivering resources following an on-demand basis (i.e. interactively) and an efficient usage of the infrastructure.

3.1.4. Rapid Elasticity

Resources in the cloud model are elastically provisioned and released, meaning that the infrastructure is able to react to the user input, both increasing and decreasing the resources allocated rapidly. This implies that the burden of providing resources to the users is done by the Cloud Management Framework (CMF) without further human interaction from the resource provider side. The elasticity opens the door to using disposable environments without the aforementioned human management overhead. These kind of disposable environments can be used for large-scale scalability tests of parallel applications, or for testing new code or library versions without disrupting production services already in place.

3.1.5. Execution of non Conventional Application Models

Most scientific computing resources (supercomputers, shared clusters and grids) are focused on processing and execution of atomic tasks, where each of these tasks may be parallel or sequential and they may have interdependencies between them or be executed concurrently. All the tasks have a common life-cycle: they are started, they process some data and eventually return a result.

However, in an IaaS cloud, this traditional task concept does not exist: instead of tasks, users manage instances of virtual machines, which are started, stopped, paused and terminated according to the users' needs. This different life-cycle makes possible to create creation of complex and dynamical long-running systems. For example this feature is used in the simulation of dynamic software agents, as in [71, 72]; the decision making process in urban management [73] or behavioral simulations using shared-nothing Map-reduce techniques [74].

3.1.6. Infrastructure Interoperability and Federation

As already explained in Chapter 2 Science today is no longer exclusively produced in single research institutions or within national boundaries. Modern scientific challenges require integrated solutions that provide computing power with flexible usage to analyse vast amounts of data. Other computational models such as the Grid have obtained great success in enabling an homogeneous access to a disparate number of resources in an interoperable way.

Science Clouds are nowadays being considered as a complementary evolution of the Grid and they have entered the e-Infrastructures ecosystem, with strong initiatives such as the Open Science Cloud [75]. Therefore the same openness, interoperability and federation level is expected by the scientific users.

Clouds will complement other existing facilities, therefore a Science Cloud should consider not only interoperability and federation in an inter-cloud scenario, but broaden it to a more general interoperability landscape, where several different and disparate providers (comprising grids, clouds, clusters and even commercial providers) are federated, building the next-generation of e-Science infrastructures.

Moreover, there are several more pragmatical motivations for moving to a resource federation, to cite some of the most relevant:

- A resource provider federation allows to cope with the limitations of a single provider which normally cannot provide all the capacity or all types of services that a certain community requires
- A federation allows communities to move computation to data when moving the data is not feasible (e.g. data with restrictive policies that cannot be moved from a given location or *too large* data sets that are not practical to move from one center to another)
- A federation creates a network of centres that both promote local economies and exploit the already existing expertise or create new knowledge hubs at multiple locations.

3.2. Selected Application Use Cases

In this section I present some preliminary use cases deployed in a cloud infrastructure. These preliminary pilot use cases have been deployed in the CSIC IFCA Science Cloud infrastructure described in Appendix A.1. For each use case we have identified what are the main benefits that the users obtain comparing to their current status and the drawbacks that should be addressed so that these scientific user could get a better experience or the infrastructure could react better to their demands.

3.2.1. PROOF

The Parallel ROOT Facility (PROOF) [76] is a commonly used tool by the High Energy Physics (HEP) community to perform interactive analysis of large datasets produced by the current HEP experiments. PROOF exploits the data-level parallelization to perform the analysis by distributing the work load (input data to process) to a set of execution hosts, following the traditional SPMD (single program, multiple data) pattern. The negligible dependencies and communication between this kind of tasks make the cloud a good candidate to accommodate embarrassingly parallel applications [77, 78, 79]

PROOF is used in the last phases of the physics analysis to produce the final plots and numbers, where the possibility of interactively change the analysis parameters to steer the intermediate results facilitates the researchers work and allows them to reach faster to better results. Data analyzed in this phase contains the relevant physics objects in set of files —produced by several previous processing and filtering steps of the original raw data collected from the detector— that may range from several GB to a few TB.

These analysis tasks are usually Input/Output (I/O) bounded [80] due to the big volume of data to process and their relatively low CPU requirements: most codes perform filtering of the data according to the relevant physics to be measured.

Running PROOF requires the pre-deployment and configuration of a master, that acts as entry point and distributes the workload, and a set of workers where the user’s analysis code is executed. There are tools that automatize the creation of such deployments, which is not trivial for most users, in batch-system environments [80, 81]. In a this case these tasks are sharing their time with other jobs.

We have collected the usage patterns for a 3.5 year period from a batch system specially configured to support this kind of tasks. The underlying infrastructure corresponds with the one described in Appendix A.2. Figure 3.2 shows the number of requests regarding the task duration. As it can be seen, all the requests can be considered short-lived, since its maximum duration is below 2 h, with the highest concentration being below 2 h.

Figure 3.3 depicts the request pattern along a three year period for the same infrastructure (Appendix A.2). As it can be seen, this kind of jobs are executed in bursts or waves, meaning that a set of users will have a high demand of resources for short periods of time —i.e. when an analysis is at a final stage.

As already stated, these traces were obtained from a batch system that is highly

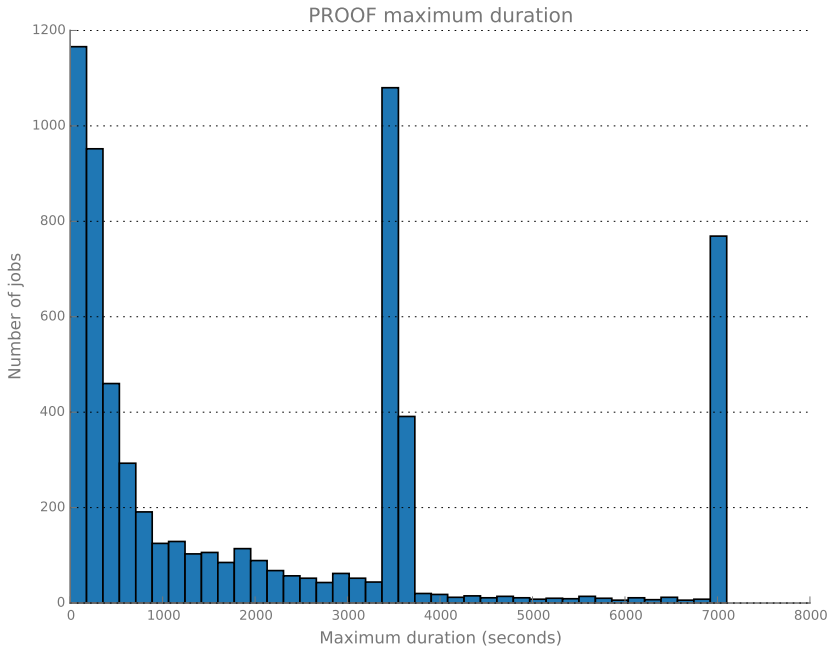


Figure 3.2: PROOF task duration.

customized and configured, involving a fine-tuning of the scheduling algorithms to make possible to get a job slot to satisfy an interactive request, even if the cluster is full. This requires close collaboration between the research community and the resource provider so as to obtain such a tuning that is able to fulfil their requirements. In some cases it is not possible to apply such a complex configuration on the batch system and the provider simply decides to reserve some nodes in advance. However, as it is shown in Figure 3.3 the usage pattern for PROOF tasks is not homogeneous, hence having nodes reserved for this kind of usage will lead to the infra-utilization of the resources in the periods of time when the workload is low.

PROOF was successfully executed on the CSIC IFCA Science Cloud infrastructure [80]. This pilot case showed that this kind of loosely coupled analysis is well suited for the cloud environment as other SPMD applications. The cloud elasticity made possible to accommodate the undetermined and high-burst patters that characterize this kind of interactive analysis (i.e short lived sessions initiated on-demand by the users). Moreover,

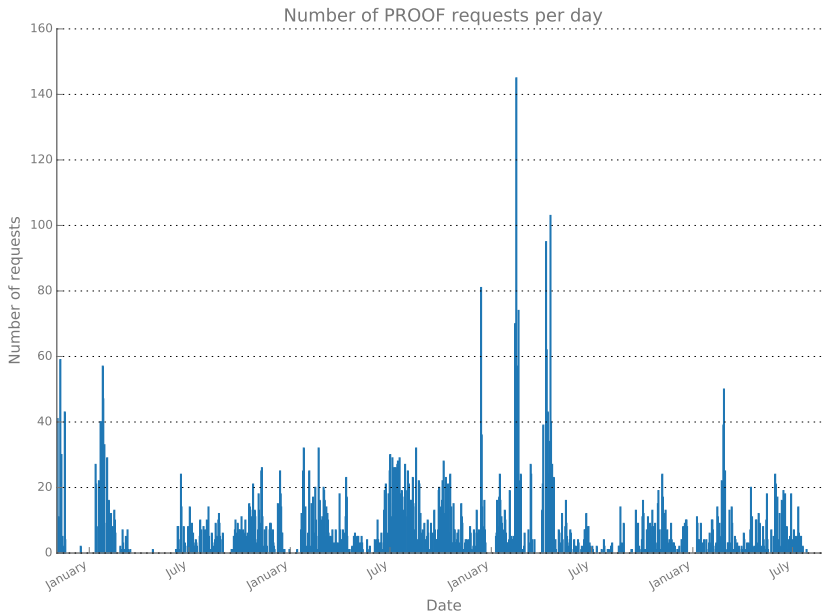


Figure 3.3: PROOF daily request pattern along a three and a half year period.

the cloud adoption enabled the deployment of a customized environments disposed when the analysis finishes, based on a specifically built OS for HEP scientists [70].

However, the cloud is far from being perfect and several obstacles were found during the deployment of the pilot case:

Node co-allocation PROOF being a parallel application needs node co-allocation for each request, meaning that it cannot start until all nodes are ready. Current cloud schedulers are not smart enough and in some cases it is not possible to make such a request. Moreover, assuming a request is successfully scheduled, an uncertainty exists on the time needed until the nodes are available [82, 83].

Elasticity not always true The illusion of the existence of infinite resources available at any time that is associated with the cloud can only be considered in commercial providers. Scientific data centers have limited resources, therefore the claimed elasticity is not always available.

Data access PROOF is a data intensive application, therefore it needs of the data being available through a corresponding high performance access [80]. Moreover, existing data in external services and storage systems needs to be made available for the users in a transparent and seamless way.

3.2.2. Particle Physics Phenomenology

As many other communities, the Particle Physics Phenomenology communities develop their own software for producing their scientific results. Software packages developed by them have evolved independently for several years, each of them with particular compiler and library dependencies. These software packages are usually combined into complex workflows, where each step requires input from previous codes execution, thus the installation and configuration of several software packages are mandatory to produce the final scientific results. Moreover, each scientific scenario to be analyzed may require different versions of the software packages, therefore the researchers need to take into account the different package versions characteristics for installing and using them.

Some of these packages also require access to proprietary software (e.g. Mathematica) that is license-restricted. Although institutional licenses may be available, these are difficult to control in shared resources (like grids or clusters) due to the lack of fine grained access control to resources.

Setting up a proper computing environment becomes an overhead for the everyday work of researchers as they have to face several problems that hinder their research:

- They must solve the potential conflicts that appear when installing multiple packages and versions on the same machine.
- The fixed execution environment supported by the resource providers forces them to deploy the tools in their own ad-hoc clusters —that are under-utilized— or even their own desktops —that cannot deliver enough computing power.
- Preservation of an execution environment at a given point in time is difficult, as the OS and its associated libraries evolve over time.

The research performed by this community was hindered due to the lack of access to more powerful computational resources, as they had to execute the simulations on

their daily desktop computers. By porting their applications to the CSIC IFCA Science Cloud, these researchers were able to deploy a stable infrastructure built with the exact requirements for their analysis where each machine is adapted to the different scientific scenarios to be evaluated, i.e. with the specific software versions needed for the analysis.

Moreover, the possibility of creating snapshots of the machines allowed the to easily recover any previous experiment without recreating the whole software setup from scratch. They benefited from contextualization tools that automatically set up and handle any dependencies of the software packages needed for the analysis upon machine creation [31].

The most important fact is that they were able to self-manage their infrastructure, with almost no intervention from the cloud operators, but some disadvantages were identified.

Software licenses The CMF should be able to enforce any usage or license restrictions for proprietary software, taking into account the current used and available license slots.

Improved contextualization Current IaaS contextualization methods need to be improved, making possible for users to easily store and retrieve contextualization scripts and recipes, easily integrated with the user interface tools.

Performance unpredictability Users perceived an uncertainty and unpredictability between different requests when requesting the same resources, due to the virtualization multi-tenancy, where different Virtual Machines (VMs) interfered the execution of others.

3.2.3. EGI Federated Cloud

The EGI community started the development of a new type of infrastructure to broaden the support for different research communities and their applications design models in 2011. The IaaS cloud service model was considered as a clear candidate to widen the usage models supported and enable higher flexibility to the final users. After an initial evaluation and collection of requirements, the EGI Federated Cloud [84] was launched into production in May 2014. It offers IaaS capabilities (VM, Block Storage, and Object Storage management) with a open library of Virtual Machine Images (VMIs)

and mechanisms to automatically replicate and distribute those images to the federation participants. The federation is enabled by the integration with a set of core services which allow the operation of a global production infrastructure. The capacity offered has notably increased from the initial tests in 2011, in contrast with the Grid resources, as shown in Figure 3.4.

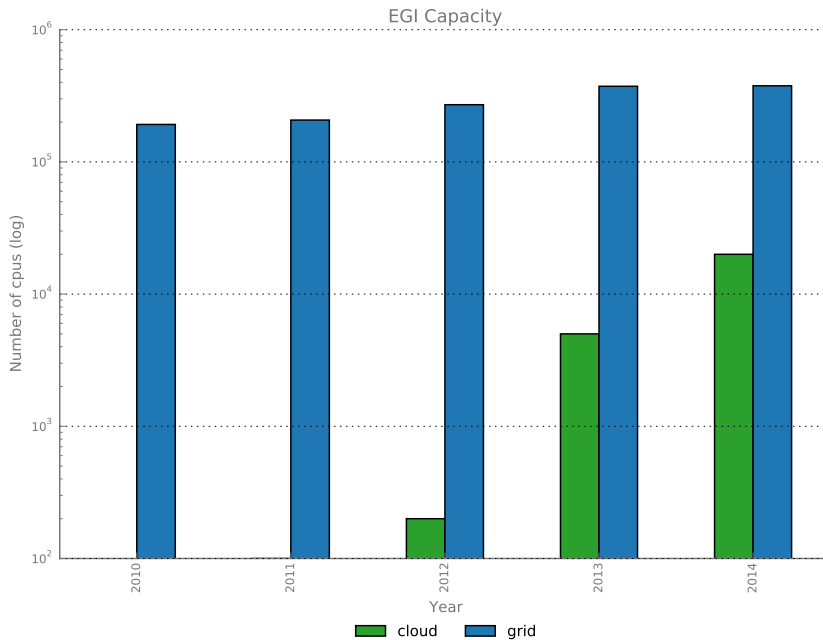


Figure 3.4: Capacity evolution of EGI Federated Cloud resources.

This Federated Cloud supports a large number of communities and applications, and we have supported the execution of several of them inside the CSIC IFCA Science Cloud infrastructure described in Appendix A.1.

The EGI Federated Cloud is not a user community *per se*, but a cloud provider federation. Its resource providers currently support more than 26 scientific communities and more than 50 use cases coming from different research disciplines: bio-informatics, physics, earth sciences, basic medicine, arts, language and architecture, mathematics, computer science, etc. In the remaining part of the section I will discuss some use cases coming from the federation, identifying the problems faced by the users regarding the

execution of their application.

3.2.3.1. Example of Use Cases

Chipster [85] is a user-friendly analysis software for next generation sequencing developed by CSC, the Finnish IT Center for Science. It provides an auto-installable Java client that connects to a set of computing servers that perform the actual analysis by combining tools in user defined workflows. Chipster is packaged as a virtual machine image, however final users struggle to set up their own server as they lack a suitable computing platform. The federation helps these users by providing an infrastructure to run the VMs whose VMIs are automatically distributed to the cloud providers. It also opens the door to European-wide horizontal scaling of the Chipster servers during load peaks although the lack of proper real-time information on the available resources and advanced brokers make this possibility hard to exploit.

VCycle [86] is a software that implements the vacuum model [87] in an IaaS cloud infrastructure, where resources appear in the vacuum, process some tasks and then disappear. For the resource providers this is an interesting computing model, since it gives them direct control over the shares that are being offered to the users, ensuring that they are being effectively used, since they will be processing tasks from a queue as long as there are pending requests. This kind of execution could potentially benefit from opportunistic usage, similarly to opportunistic backfilling in traditional batch systems. However, due to the limitations of the cloud scheduling algorithms this was not possible, and the resources had to be statically partitioned.

3.2.3.2. Problems Inherent to the Federation

The EGI Federated Cloud provides the technological building blocks that help to create cloud provider federations, however, there still are challenges that in some cases hinder the users experience. These problems, mainly related to the inherent heterogeneity of such a federation or the limitation of the current implementations, are the following:

Interoperability problems Since the federation is a collection of heterogeneous resource providers, there are different CMF being used, with different and non

compatible interfaces and virtualization technologies. This heterogeneity poses a problem for the users that find difficult to **i**) interact with different CMF and different Application Programming Interfaces (APIs) and **ii**) to migrate their executions from one provider to another due to the different underlying virtualization technologies.

Authentication and Authorization EGI provides single sign on through IGFT [88] identities based on X.509 certificates, but users expect to be able to access with their existing institutional credentials for authentication. Authorization is achieved by using Virtual Organization Membership Service (VOMS) proxies [89]. These kind of proxies are not easily usable on web-based interfaces, thus limiting the usability of the platform.

Information discovery In an heterogeneous environment such as a federation the user faces the problem of knowing exactly what are the available resources at a given point in time. The information currently available at the EGI Federated cloud offers an insufficient view of the resources and cannot be used to dynamically scale deployed applications depending on the resource usage.

Resource brokering The increased complexity of dealing with several providers can be alleviated by using resource brokering services that aid in the selection of the most appropriate resources for the execution of the user applications and hide the infrastructure heterogeneity.

3.3. Science Clouds Open Challenges

Many of the resources and services needed to stablish Science Clouds already exist. However, there are technological challenges that need to be addressed. Scientific applications have unique requirements, therefore a Science Cloud shall provide unique features and face unique challenges. In the following sections we will cover the existing gaps in the current cloud offerings regarding scientific applications.

3.3.1. Usability Requirements

The deployment of customized environments is one of the biggest advantages of the cloud computing model against any other *traditional* paradigms, but it may also

represent a drawback for users that are not familiar with systems administration. In this context, scientific application catalogs and contextualization mechanisms are needed.

3.3.1.1. Scientific Application Catalogs

The cloud flexibility to deploy customized virtual machines is one of the most prominent features of this computing model. However, great power is associated with great responsibility. In this case this fact means that the users are in charge of creating and managing their instances, not only regarding proper configuration for their image to be executed, but also taking into account security concerns. Undoubtedly this implies advanced IT systems management skills, that are normally not available for the Long-Tail of scientists, as exposed in Section 3.1.1. The management of the software in the cloud is much more flexible than other systems as the user has complete access to the machine, but the entry barrier may still be high. In this context, even if the application catalogs may be considered at a higher cloud service model (i.e. at a Platform as a Service (PaaS) layer), it is needed to offer a predefined set of standard scientific applications as a baseline for the users.

Scientific Application Catalogs Open Challenges

- How to ease the way scientists can build and deploy their applications.
- How to provide VMI catalogs in a way that scientific software is available across different resource providers.

3.3.1.2. Image Contextualization

The contextualization of images can be defined as the process of installing, configuring and preparing software upon boot time on a pre-defined virtual machine image. This way, the pre-defined images can be stored as generic and small as possible, since all the customizations will take place at boot time.

The image contextualization is tightly coupled with the scientific application catalogs described in Section 3.3.1.1. The catalogs are useful for bundling self-contained and ready to use images containing standard scientific applications that can be then customized to the user needs by means of contextualization scripts.

In those cases, instead of creating and uploading a new image for each application version and/or modification (a tedious process that is a time consuming task for the image creator), the installation and/or customization can be delayed until the machine boot time. By means of this mechanism the newest version can be automatically fetched and configured, or the defined and variable user-data provided to the image. This is done by means of ready to use and compatible image that contains all the necessary dependencies and requisites for the scientific applications to be installed. This contextualization-aware images will then be launched with some metadata associated, indicating the software to install and configure.

Nowadays powerful configuration management tools exist that can help with the implementation of the described contextualization mechanisms. Tools such as Ansible¹, Puppet², CFengine³, Chef⁴, etc. make possible to define a machine *profile* that will be then applied to a machine, so that an given machine will fit into that profile after applying it. However, these languages and tools introduce a steep learning curve, so that the CMF should provide a method to expose the defined profiles to the users easily. Therefore, Science Clouds should offer a way for users to create, deploy and shared these contextualization recipes in an easy way so as to lower the access barrier for the scientific users [31].

Image Contextualization Open Challenges

- How to ease the way scientific users can contextualize their applications without a steep learning curve.

3.3.2. Resource Allocation Problems

Current CMF is designed to satisfy the industry needs. In a commercial cloud provider users are charged (i.e. they have to pay) according to their resource consumption. Therefore a commercial resource provider might not worry about the *actual* usage of the resources, as they are getting paid by the consumed capacity, even if they are idle resources. This situation is not acceptable in scientific facilities where the maximum

¹Ansible. 2015. URL: <http://www.ansible.com/>.

²Puppet. 2015. URL: <https://puppetlabs.com/>.

³CFengine. 2015. URL: <http://www.cfengine.com>.

⁴Chef. 2015. URL: <http://www.opscode.com/chef/>.

utilization of the resources is an objective. Idle resources are an undesirable scenario as it prevents other users from accessing and using the infrastructure. Access to most scientific datacenters is not based on a pay per use basis, as user communities are granted with an average capacity over long periods of time. This capacity, even if accounted, is not paid by the users, but it is rather supported by means of long-term grants or agreements.

In traditional scientific datacenters users executed their tasks by means of traditional batch systems, where the jobs are normally time-bounded (i.e. they have a specific duration). Different policies are then applied to adjust the job priorities so that the resources are properly shared between the different users and groups. Even if the user does not specify a duration, a batch system is able to stop its execution after a given amount of time, configured by the resource provider.

However, there is no such *duration* concept in the cloud model, where a virtual machine is supposed to live as long as the user wants. Users may not stop their instances when they have finished their job (they are not getting charged for them), ignoring the fact that they are consuming resources that may be used by other groups. Therefore, resource providers have to statically partition their resources so as to ensure that all users are getting their share in the worst situation. This leads to an underutilization of the infrastructure, since a usage spike from a group cannot be satisfied by idle resources assigned to another group.

Taking as an example the PROOF use cases studied in Section 3.2, Figure 3.2 showed a typical usage pattern for interactive scientific applications, where jobs are executed in bursts or waves. This kind of usage (i.e. short lived executions that are not constant over the time) is quite common for scientific applications [31, 94, 95, 96, 97] and presents a demanding challenge for resource providers. Enough computing capacity needs to be delivered for absorbing this kind of requests, minimizing the reserved resources that will be idle for long periods of time.

Implementing an effective scheduling and resource allocation policies to ensure that the elasticity is perceived as true is a challenging task. An allocation policy that is driven by a resource provider decision can result in a low value from the user standpoint, whereas an allocation under user control may result in a high cost for the provider [98].

3.3.2.1. Instance Co-allocation

Compute and data intensive scientific workloads tend to use parallel techniques to improve their performance. Parallel executions are complex since they require intercommunication between processes, usually spread across several nodes, scenario in which resource provisioning task becomes even more challenging. Based on the assumption that a provider is capable of satisfying a request involving different instances, one has to consider the fact of managing them as an atomic workload so to assure that these instances are actually being provisioned at the same time, i.e. they are being *co-allocated*. Proper co-allocation policies should take into account network requirements, such as satisfying low latencies and appropriate bandwidths, and fulfill any constraints imposed by the parallel framework being used, as e.g. OpenMPI's intra-subnet allocation check [99].

In homogeneous and static environments, guaranteeing ordered co-allocation of resources can be easily tackled, if compared to heterogeneous scenarios. In the specific case of cloud computing, the flexibility that it introduces, makes multi-resource allocation a challenging task that must take into consideration not only the synchronized startup (see more at Section 3.3.2.2) of master and worker instances, but also how these resources are physically distributed and what are the hardware constraints (network, cpu, memory) to be considered. Only by doing this, parallel tasks provisioned in clouds would have a similar application performance to what can be obtained with homogeneous ad-hoc resources, but getting rid of the rigidity that they introduce.

Instance Co-allocation Open Challenges

- How to offer a proper Service Level Agreement (SLA) to ensure that instances need to be co-allocated.
- How to ensure that instances that need co-allocation are actually started at the same time.
- How to account (or not account) for instances that requiring co-allocation have been provisioned with an unacceptable delay. When a user is requiring this feature but the requirement cannot be fulfilled this should be taken into account.

- How to ensure that when the instances are already scheduled they are allocated within a time-frame. VM management introduces overheads and delays that should be taken into account to ensure a proper co-allocation.

3.3.2.2. Short Startup Overhead

When a request is made, the corresponding images have to be distributed from the catalog to the compute nodes that will host the virtual machines. If the catalog repository is not shared or the image is not already cached by the compute nodes, this distribution will introduce a penalty on the start time of the requested nodes. This overhead can be quite significant and has a large influence in the startup time for a request. This is specially true when large [83] requests are made by a user. Figure 3.5 shows this effect in the OpenStack test infrastructure described in Appendix A.3, using 2 GB images that were distributed using Hiper Text Transfer Protocol (HTTP). As it can be seen, the time needed to get all the machines within a single request increased with the size of the request.

Short Startup Overhead Open Challenges

- How to deploy the images into the nodes in an efficient way.
- How to deal with spikes on the requests, so that the systems are not saturated transmitting the images into a large number of nodes.
- How to implement cache mechanisms in the nodes, implementing sanity checks so that similar workloads are not constrained into a few nodes.
- How to forecast workloads, so that images can be pre-deployed, anticipating the user's requests.

3.3.2.3. Performance Aware Placement

In order to improve resource utilization, cloud schedulers can be configured to follow a fill-up strategy that might end up in multiple virtual machines running concurrently on the same physical server. This scenario leads to resource competition which surely will affect application performance. In this regard, the scheduler needs to be performance-aware (or even degradation-aware), so that e.g. two data-intensive

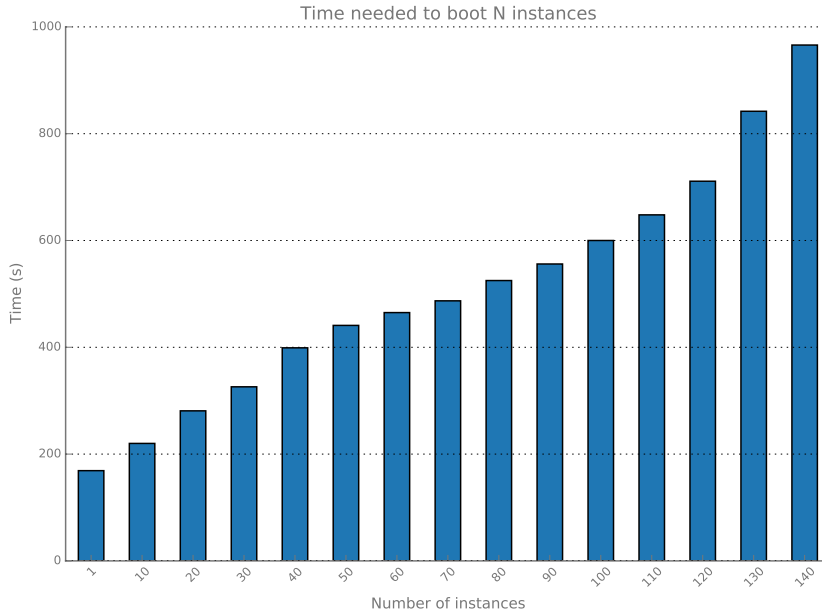


Figure 3.5: Time needed to boot the number of requested instances. Tests were performed the infrastructure described in Appendix A.3, with an image of 2 GB.

applications, executing I/O consuming tasks are not scheduled in the same physical server and, therefore, not hitting each other's performance.

Several approaches have been raised in order to diminish degradation. Some do not act directly on pro-active scheduling but instead in reactive reallocation of the affected virtual instances by using underneath hypervisor capabilities like live migration [50]. But, instead of relying in monitoring the application performance and take reallocation decisions based upon its degradation, a more pro-active scheduling is needed so to improve the suitability of the resource selection. Feeding the scheduler with more fine-grained hardware requirements, provided by the user request, such as low-latency interconnects (e.g. Infiniband, 10 GB Ethernet) or GPGPU selection, provides a better resource categorization and, consequently, will directly contribute to a more efficient execution of the application. To accomplish this, the specialized hardware must be exposed into the virtual instances, by means of PCI passthrough with IOMMU or Single Root I/O Virtualization (SR-IOV) techniques, and eventually managed by the CMF using

the underlying virtualization stack.

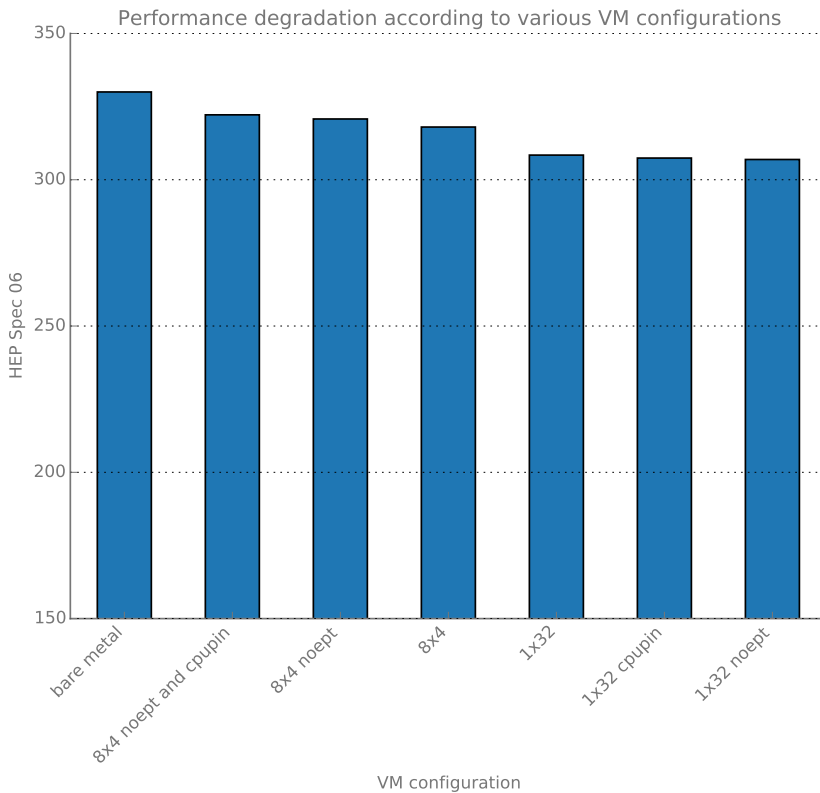


Figure 3.6: Aggregated performance regarding the HEP Spec 06 [100] benchmark, taking into account different virtual machine sizes and configurations for one host.

The hypervisor providing the virtualization appears as an important factor when measuring performance. It is widely accepted that virtualization introduces a penalty when compared with bare-metal executions. However this penalty depends on how the hypervisor is being used. Figure 3.6 shows the degradation of the aggregated performance delivered by a physical machine, using different virtual CPUs (vCPUs) sizes. The physical node where the tests were performed is described in Appendix A.5. The virtual machines were dimensioned so as to consume—in aggregate—all the resources available on the host. The label "noept" means that the Extended Page Tables (EPT) support has been disabled. The label "cpupin" means that the vCPUs have been pinned

to the physical CPUs.

Science clouds need to deliver the maximum performance possible. Therefore, the CMF should take this fact into account, by implementing scheduling policies that would help to prevent the above identified performance drops.

Performance Aware Placement Open Challenges

- How to minimize the performance loss when scheduling various virtual machines inside one host.
- How to pro-actively scheduling could be used to minimize resource competition.
- How to detect performance interferences between VMs and take appropriate actions (like live migration) to minimize them.
- How to redistribute the running instances between the resources without impacting the running applications.

3.3.2.4. Data Aware Scheduling

Several scientific disciplines —such as HEP, Astronomy or Genomics just to cite some of them— generate considerably large amounts of data (in the order of PB) that need to be analyzed. Location and access modes have clear impacts to data-intensive applications [101, 102] and any platform that supports these kinds of applications should provide data-locality and data-aware scheduling to reduce any possible bottlenecks that may even prevent the actual execution of the application.

Storage in clouds is normally decoupled from the virtual machines and attached during runtime upon user’s demand. This poses a bigger challenge to the scheduler since the location of data to be accessed is not known a priori by the system. Science clouds should be able to provide high-bandwidth access to the data, which is usually accessed over the network (e.g. block storage may use ATA over Ethernet or iSCSI; object storage usually employs HTTP). This may require enabling the access to specialized hardware from the virtual machines (e.g. Infiniband network) or re-locating the virtual machines to hosts with better connectivity to the data sources. Data-locality can also be improved by using caches at the physical nodes that host the VMs, by replicating locally popular data hosted externally to the cloud provider, or by leveraging tools like CernVMFS [103] that deliver fast access to data using HTTP proxies.

Data Aware Scheduling Open Challenges

- How to take into account cloud data management specificities when scheduling machines.
- How to ensure that the access delivers high performance for the application being executed.

3.3.2.5. Flexible Resource Allocation Policies

Long-running tasks are common in computational science. Those kind of workloads do not require from interactivity and normally are not time-bounded. Such tasks can be used as opportunistic jobs that fill the computing infrastructure usage gaps, leading to a better utilization of resources.

In traditional scientific datacenters and time-sharing facilities this is normally done in by means of several techniques, such as backfilling, priority adjustments, task preemption and checkpointing. Some of these techniques require that the tasks are time-bounded, but in the cloud a virtual machine will be executed as long as the user wants.

Commercial cloud providers have tackled this issue implementing the so called spot instances or preemptible instances. This kind of instances can be terminated without further advise by the provider if some policy is violated. For example, if the resource provider cannot satisfy a normal request—in the preemptible case— or because the user is paying a prize that is considered too low over a published price baseline—in the spot mode, where the price is governed by a stock-options like market.

The usage of this kind of instances in science clouds could make possible that the infrastructure is filled with opportunistic [104] jobs that can be stopped by higher priority tasks, such as interactive demands. The Vacuum computing model [87], where resources appear in the vacuum to process some tasks and then disappear is an ideal candidate to leverage this kind of spot instances. Tools such as VCycle [86] are already being used to profit from opportunistic usage in existing scientific infrastructures.

Flexible Resource Allocation Policies Open Challenges

- How to maximize the resource utilization without preventing interactive users from accessing the infrastructure.

-
- How to specify dependencies between virtual machines so that more complex workloads can be scheduled in a more easy way.
 - How to account for resources that are suitable for being stopped or preempted.
 - How to select the best instances that can be stopped to leave room for higher priority requests, with the compromise of reducing the revenue loss and with the smallest impact to the users.

3.3.2.6. Performance Predictability

Popular IaaS CMFs do not currently expose mechanisms for customers to define a specific set of hardware requirements that would guarantee a minimum performance when running their applications in the cloud, as performance variations exists within same instance types [105]. Real time demanding or latency sensitive applications are indeed seriously hit by this limitation, which appears as a big obstacle for integrating this type of applications into clouds [106].

Computing capabilities provide only a magnitude of multi-threading efficiency based on the number of vCPUs selected. Customers are then tied to a generic vCPU selection that may be mapped to different processors by the underlying framework, in which case different performance results could be obtained based on the same set of requirements. This unpredictability will be increased whenever resource overcommit is in place, that could lead to CPU cycle sharing among different applications.

Lack of network performance guarantees contribute also to unexpected application behavior. Enforcing network QoS to achieve customer-required network bandwidth can greatly improve application predictability, but network requirement selection are seldom offered by cloud providers [107].

Improved performance predictability is a key requirement for users [108] but also to providers. The lack of predictability leads to uncertainty [109], a fact that should be mitigated for both users and providers. An accurate provision of customer needs in terms of computing and network capabilities will not only boost user experience but also will provide a clear estimation of cost based on the different service levels that the resource provider can offer.

Performance Predictability Open Challenges

- How to expose enough granularity in the request specification without exposing the underlying abstracted resources.
- How to guarantee the performance predictability between different requests with the same hardware requests.

3.3.2.7. Licensed Software Management

One of the major barriers scientists find when moving their applications to the cloud relies in licensing troubles. Software vendors that count with policies about how to deal with licensing in virtualized environments propose the usage of Floating Network Licenses (FNL). These special licenses usually increment costs, as they can be used by different virtual instances, and require the deployment of license managers in order to be able to use the software in the cloud infrastructures. Additionally, the license managers might need to be hosted within a organization's network.

Using FNLs are the most popular solution provided by vendors, but the imposed requirements mentioned above can be difficult to satisfy in some cases: hosting a license manager is not always possible by some scientific communities and it introduces maintenance costs, whose avoidance is one of the clear benefits of moving to a cloud solution.

The need for a more straightforward way of getting licensed or proprietary software to work in virtualized environments is a must that software vendors should consider. In commercial cloud infrastructures, like Amazon Amazon Web Services (AWS), customers can make use of pre-configured images, license-granted, with the proprietary software locally available and ready to use. At the time of writing, Amazon AWS does not have agreements with all of the major software vendors, but it appears as a neat and smooth solution that requires no extra work from the end users side.

Besides the above administrative difficulties, the actual technical challenge in resource allocation for licensed software is that cloud schedulers are not license-aware [110]. This gap needs to be filled by the cloud middleware stacks, as it was solved years ago in HPC clusters.

Licensed Software Management Open Challenges

-
- Persuade commercial vendors to release more flexible licensing methods, specific for the cloud.
 - How to deal with license slots within the scheduler.

3.3.3. Interoperability and Federation

3.3.3.1. Federation

As explained in Section 3.1.6 the federation introduces some benefits in the context of e-Science infrastructures, however it also introduces some problems that are perceived not only by the users, but also by the resource providers operating the infrastructures. Apart from interoperability problems (that will be covered in Section 3.3.3.2) there are other facts that need to be taken into account when federating disparate resource providers. Initiatives such as the EGI Federated Cloud [84] have put a great effort on these areas, but there are still some missing points where a lot of improvements are needed and foreseen.

In a federated ecosystem whenever a new user group appears in the scene it is needed to procure resources that support their research. For the big-science collaborations this is done through agreed SLAs. However, a more dynamic procurement and tender process needs to be developed to support new use cases more rapidly without a bureaucratic burden.

Moreover, a federation should go one step beyond from just being an aggregation of a number of providers. Proper brokering policies should be in place so that users do not have to deal with each individual resource provider and all of its particularities. In this context, a proper information and discovery system is needed, as a mean to obtain proper information about the available resources for the users.

Federation Open Challenges

- How to publish dynamic information about the current cloud resources being offered to the users.
- How to provide brokering functionality so that the best resource provider is selected.

- How to federate the resource providers so that users can request resources directly from the federation, without knowing the particular provider procuring the infrastructure.
- How to perform intra-cloud networking in an effective way.
- How to create and maintain Science Cloud marketplaces dynamically, so that users can choose the best option for their needs.
- How to effectively support and procure resources for new use cases that could pop out.
- How to account for the resource usage across several providers.
- How to federate identities across different and disparate cloud providers.

3.3.3.2. Interoperability

Some commercial providers promote lock-ins in their infrastructures as a way of keeping their users captive on their resources, instead of trying to keeping their customers by some other added-value features. This is being considered nowadays a harmful practice and users are more and more reluctant to deploy their applications in such infrastructures.

Regarding lock-ins, two different kinds should be considered:

- Vendor lock-in, where the vendor or provider forces the user to remain with them, for instance by means of proprietary and non interoperable interfaces. In some cases, user data can even be captive on their systems by not providing an easy way of accessing it.
- Technology lock-in, where a user is induced to remain using a given technology (for example a given hypervisor), as it is not compatible with other similar technologies. A technology lock-in can be an induced case of vendor lock-in or it can be completely unintentional.

Interoperability Open Challenges

- How to ensure that proper interoperability is provided between different cloud providers.
- How to ensure that users are not locked-in in a given infrastructure or provider.
- How to ensure that data can be migrated between different providers.
- How to ensure that a running virtual machine can be migrated to a different cloud provider, taking into account the underlying virtualization technology.

3.3.3.3. Seamless Access Across e-Infrastructures

Interoperability across different infrastructures and technologies (like grids, clouds and even commercial services) is an important feature for many communities that need access to their legacy data and applications. In this context, the usage of open and established standards could alleviate these interoperability problems. However, clouds have evolved at the industry pace, not taking into account the current existing e-Science infrastructures.

Seamless Access Across e-Infrastructures Open Challenges

- How to ensure that different and disparate infrastructures are interoperable.
- How to ensure access to legacy data sets and applications from Science Clouds.
- How to homogenize identities, credentials, authentication and authorization mechanisms across infrastructures boundaries.

3.4. Related Work

As I explained in Chapter 2 scientific infrastructures give access to unprecedented computational power to scientists, and the cloud computing infrastructures are being embraced so as to enrich the scientific computing ecosystem.

Several EU actors –EUDAT, LIBER, OpenAIRE, EGI and GÉANT– have started the Open Science Cloud initiative [75], outlining the eight elements of success for such an infrastructure: **1) Open**, meaning that it should be based on open access

and open standards; **ii) Public funded and governed**, to guarantee persistence and sustainability; **iii) Research-centric**, ensuring the development of services responsive to their needs; **iv) Comprehensive**, meaning that it should be universal and specific to no particular community; **v) Diverse and distributed**, so as to leverage the richness of the EU distributed e-Infrastructures; **vi) Interoperable**, through the promotion of open standards and protocols; **vii) Service Oriented**; and **viii) Social**.

Moreover, there are some previous studies regarding the general challenges and implications of running a Science Clouds at a more technical level. The work by Blanquer et al. [111], in the scope of the VENUS-C project, evaluated the requirements of scientific applications by performing a broad survey of scientific applications within the project. Their study showed that the cloud computing model is perceived as beneficial by the users (being one of the key expectations the elasticity), although some drawbacks need to be tackled so as to improve its adoption (such as interoperability, learning curve, etc.).

Juve et al. [112] outlined what is expected from a science cloud in contrast with a commercial provider (shared memory, parallel applications, shared filesystems) so as to effectively support scientific workflows. Besides, it concluded that cloud can be beneficial for scientific users, assuming that science clouds will be build ad-hoc for its users, clearly differing from commercial offers.

Susa et al. [113] proposed a Software as a Service (SaaS) marketplace for scientific applications to be executed on top of public IaaS providers, presenting the problems faced during the deployment and testing of the first pilot version. However, they did only focus on commercial providers, without taking into consideration the usage of scientific cloud infrastructures.

The United States Department of Energy (DOE) Magellan project conducted a similar study, published by Ramakrishnan et al. [114]. This work also focused on identifying the gaps and challenges for a resource provider so as to build a science cloud. Some of the requirements identified by them are coincident with our findings, such as the **i)** access to low-latency interconnects and filesystems, **ii)** access to legacy data-sets, **iii)** availability of pre-installed and pre-tuned application software stacks; among others.

On the other hand, there is a considerable amount of research works addressing cloud resource provisioning and scheduling from the user or consumer perspective [115, 96, 116, 117]. Some authors have studied how to implement hybrid provisioning of resources between several cloud providers [118, 119], or even between different

computing infrastructures such as grids and clouds [120]. The workflow model is widely used in many scientific computing areas, and there is a vast amount of studies regarding the feasibility and challenges of executing workflows in the cloud [121, 122, 123, 124, 125, 126, 127, 128].

Regarding resource provisioning strategies from the provider standpoint. Sotomayor et al. studied how to account and manage the overheads introduced the virtual resources management [129]. Hu et al. [130] studied how to deliver a service according to several agreed SLA by using the smallest number of resources. Garg et al. [131] presented how to deal with SLAs that imply interactive and non-interactive applications. Cardonha et al. [132] proposed a patience-aware scheduling that take into account the user's level of tolerance (i.e. the patience) to define how to deliver the resources to the users.

Manvi et al. [98] performed an exhaustive review of the resource provisioning, allocation and mapping problems for a IaaS resource provider, stating some open challenges like **I**) how to design a provisioning algorithm for optimal resource utilization based on arrival data; **II**) how and when to reallocate VMs; **III**) how to minimize the cost of mapping the request into the underlying resources; **IV**) how to develop models that are able to predict applications performance; among many others.

In addition, there is a lot of research regarding resource provisioning in clouds that is only focused on energy aware aspects [29, 133, 134]. Smith et al. [135] modelled how different workloads affected energy consumption, so that an accurate proper power prediction could be made to perform an efficient scheduling. Several authors have studied how the consolidation of virtual servers in a cloud provider could lead to a reduction of the energy consumption [136, 28]. This fact can be used to increase the revenues by implementing energy-aware resource allocation policies [137].

3.5. Conclusions

In this chapter I have given a short overview —as there is a lot of literature regarding the cloud benefits— of the advantages that the cloud computing model can offer to scientific users. I have identified the expectations that the cloud adoption has created among scientific users, to afterwards evaluate several real and representative use cases obtained from our experience operating a Science Cloud. In the last part of this chapter I have elaborated a set of open challenges for Science Clouds.

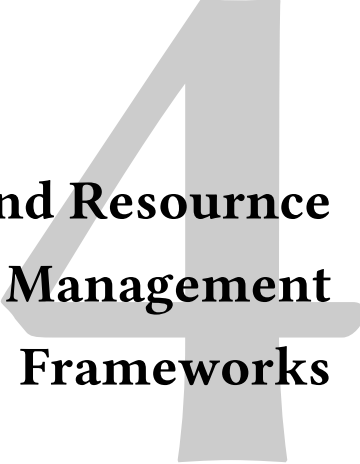
As it was noted through this chapter, Cloud Management Frameworks (CMFs) are normally being developed taking into account the point of view commercial providers and industry, focusing on satisfying their needs, but not really fulfilling science demands. scientific workloads have unique requirements that need to be specifically addressed.

From our experience, one of the main fields needing from improvement in these CMFs is the scheduling and provisioning of resources, as described in Section 3.3.2. Current scheduling strategies are too simple to accommodate the complex workflows and heavy workloads that characterize the scientific applications.

The interoperability, portability and federation aspects are important when considering that scientists will access a myriad of heterogeneous systems in a distributed way. The cloud has to learn from previous federated e-Infrastructures, and adopt the best practices and open standards and protocols that those infrastructures have promoted through the last years.

In summary, the cloud is not a silver bullet for scientific users, but rather a new paradigm that will enter the ecosystem. In the upcoming years scientific computing datacenters have to move towards a mixed and combined model, where a given user will have access to the more *traditional* computational power, but also they should provide their users with additional cloud power that will complement the former computing infrastructures. These Science Clouds should need to be tuned to accommodate the demands of the user communities supported. This way, both the users will benefit from a richer environment and resource providers can get a better utilization of their resources, since they will allow for new execution models that are currently not available.

IMPROVED RESOURCE PROVISIONING



**Scheduling and Resource
Provisioning in Cloud Management
Frameworks**

Chapter 1 exposed how the cloud computing model is aimed on delivering resources (such as virtual machines, storage and network capacity) as an on demand service. The most accepted publication defining the cloud from the United States National Institute of Standards and Technology (NIST), emphasizes the *rapid elasticity* as one of the essential characteristics of the cloud computing model: “(...) capabilities can be elastically provisioned and released, (...), to scale rapidly outward and inward (...)” [1]. Moreover, users and consumers consider them as the new key features that are more attractive [9, 5] when embracing the cloud if compared to other infrastructures.

If we take into consideration virtual machines delivered by an Infrastructure as a Service (IaaS) resource provider, these two outstanding features imply two different facts:

- **Elasticity** is the ability to start and dispose one or several Virtual Machines (VMs) almost immediately.
- **On demand** access implies that VMs are allocated whenever the user requires them, without prior advise and without human intervention from the Resource Provider (RP).

Therefore, it is the Cloud Management Framework (CMF) duty –through their scheduling component– to ensure that these perceptions are true.

In this Chapter I will give an overview of the current scheduling strategies (Section 4.1) and algorithms (Section 4.2) in the most common open source CMFs. In Section 4.3 I describe more in depth the OpenStack Compute scheduler, as it will be the chosen CMF for the implementation of the proposed solutions during the rest of this dissertation. In summary, this Chapter is a brief introduction to the CMF scheduling aspects, that will be referred later in in Chapter 5 and Chapter 6, where I will address some of the provisioning challenges presented in Chapter 3.

4.1. Scheduling strategy

Clouds do not implement queuing as other computing models do as they are more focused on the rapid scaling of the resources, rather than in batch processing, where queuing becomes useful. The default scheduling strategies in the current CMFs are based on the immediate allocation or resources. The cloud schedulers provision them when

requested, or they are not provisioned at all. Therefore, cloud requests are processed on a first-come, first-served basis, without implementing any queuing at all, as this state can be considered against the cloud model.

However, some users require for a queuing system –or some more advanced features like advance reservations– for running virtual machines. In those cases, there are some external services such as Haizea [118, 138] for OpenNebula or Blazar [139] for OpenStack. Those systems lay between the CMF and the users, intercepting their requests and interacting with the cloud system on their behalf, implementing the required functionality.

4.2. Scheduling algorithms

Besides simplistic scheduling policies like first-fit or random chance node selection, current CMFs implement a scheduling algorithm that is based on a rank selection of hosts.

OpenNebula [140] uses by default a `match making` scheduler, implementing the Rank Scheduling Policy [118]. This policy first performs a filtering of the existing hosts, excluding those that do not meet the request requirements. Afterwards, the scheduler evaluates some operator defined rank expressions against the recorded information from each of the hosts so as to obtain an ordered list of nodes. Finally, the resources with a higher rank are selected to fulfil the request.

OpenStack [141] implements a Filter Scheduler [142], based on two separated phases. The first phase consists on the filtering of hosts that will exclude the hosts that cannot satisfy the request. This filtering follows a modular design, so that it is possible to filter out nodes based on the user request (RAM, number of virtual CPUs (vCPUs)), direct user input (such as instance affinity or anti-affinity) or operator configured filtering. The second phase consists on the weighing of hosts, following the same modular approach. Once the nodes are filtered and weighed, the best candidate is selected from that ordered set.

CloudStack [143] utilizes the term *allocator* to determine which host will be selected to place the new VM requested. The nodes that are used by the allocators are the ones that are able to satisfy the request.

Eucalyptus [144] implements a greedy or round robin algorithm. The former strategy uses the first node that is identified as suitable for running the VM. This algorithm exhausts a node before moving on to the next node available. On the other hand, the later schedules each request in a cyclic manner, distributing evenly the load in the long term.

All the presented scheduling algorithms share the view that the nodes are firstly filtered out—so that only those that can run the request are considered—and then ordered or ranked according to some defined rules. Generally speaking, the scheduling algorithm can be expressed as the pseudo-code in the Algorithm 1.

Algorithm 1 Scheduling Algorithm.

```

1: function SCHEDULE REQUEST( $req, H$ )
INPUT:  $req$ : user request
INPUT:  $H$ : all host states
2:    $hosts \leftarrow []$  ▷ empty list
3:   for all  $h_i \in H$  do
4:     if  $filter(h_i, req)$  then
5:        $\Omega_i \leftarrow 0$ 
6:       for all  $r, m$  in ranks do ▷  $r$  is a rank function,  $m$  the rank multiplier
7:          $\Omega_i \leftarrow \Omega_i + m_j * r_j(h_i, req)$ 
8:       end for
9:        $hosts \leftarrow hosts + (h_i, \Omega_i)$  ▷ append to the list
10:    end if
11:  end for
12:  return  $hosts$ 
13: end function

```

4.3. Scheduling in OpenStack

OpenStack Compute includes two different schedulers: the Chance scheduler and the Filter Scheduler. The former selects hosts randomly, so it is unusable in production system, so the Filter Scheduler can be considered as the default in an OpenStack cloud.

The Filter Scheduler is a rank scheduler, implementing two different phases: filtering and weighting, as shown in Figure 4.1.

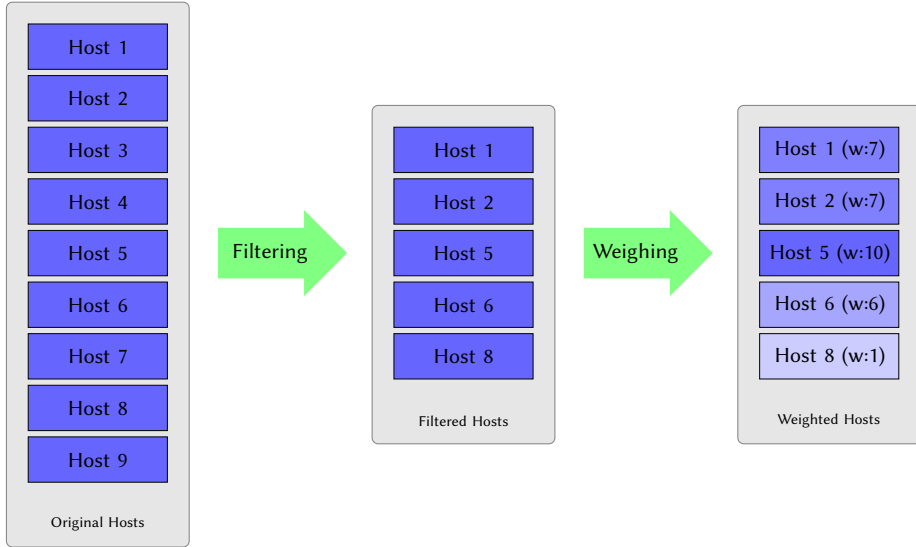


Figure 4.1: OpenStack scheduling algorithm.

Filtering The first step is the filtering phase. The scheduler applies a concatenation of filter functions to the initial set of available hosts, based on the host properties and state —e.g. free RAM or free CPU number— user input —e.g. affinity or anti-affinity with other instances— and resource provider defined configuration. When the filtering process has concluded, all the hosts in the final set are able to satisfy the user request.

Weighing If the filtering phase yields a list of suitable hosts, the weighting stage starts so that the best host —according to the defined configuration— is selected. The scheduler will apply all hosts the same set of weighters functions $w_i(h)$, taking into account each host state h . Those weighter functions will return a value considering the characteristics of the host received as input parameter, therefore, total weight Ω for a node h is calculated as follows:

$$\Omega = \sum^n m_i \cdot N(w_i(h))$$

Where m_i is the multiplier for a weighter function, $N(w_i(h))$ is the normalized

weight between $[0, 1]$ calculated via a rescaling like:

$$N(w_i(h)) = \frac{w_i(h) - \min W}{\max W - \min W}$$

where $w_i(h)$ is the weight function, and $\min W$, $\max W$ are the minimum and maximum values that the weighter has assigned for the set of weighted hosts. This way, the final weight before applying the multiplication factor will be always in the range $[0, 1]$.

Once the set of hosts have weights assigned to them, the scheduler will select the host with the maximum weight and will schedule the request into it. If several nodes have the same weight, the final host will be randomly selected from that set.

As I explained in Section 4.1, if the scheduler is not able to provision the resources when the request is made, it will return an error. However, in order to alleviate spurious problems that may occur, the scheduler implements a retry cycle, configurable by the cloud operator.

4.4. Conclusions

In this Chapter I have given a brief introductory overview to the Cloud Management Framework (CMF) scheduling strategies and algorithms. The following chapters will address some resource provisioning challenges, and they will leverage the information outlined in this Chapter.

5

Efficient Image Deployment

*Part of this chapter will be published as: **Á. López García** and E. Fernández-del-Castillo. “Efficient image deployment in Cloud environments”. In: Journal of Network and Computer Applications (2016). ISSN: 1084-8045 (accepted paper).*

Any cloud must be able to deliver rapidly the requested machines to provide a satisfactory elastic and on-demand perception according to any Service Level Agreement (SLA) [146] established with the users or customers. With this fact in mind, the Cloud Management Frameworks (CMFs) –and as a consequence the resource providers operating a cloud– face a challenge when they are requested to provision a large number of resources, specially when running large infrastructures [147] comprising more than a few nodes. These on-demand and elastic perceptions mostly depend on the time needed to serve the final resource requested, so a rapid provisioning should be one of the objectives of any cloud provider.

This fact is specially true in Science Clouds, considering that a vast number of scientific applications requiring interactive analysis need from the rapid provisioning of resources, otherwise users must wait for their resources to be allocated, losing the needed interactivity (Section 3.3.2.2).

On the other hand, some requests need co-allocation of the Virtual Machines (VMs), as explained in Section 3.3.2.1. As I will explain later in Section 5.1 once the resources are allocated for a request there is still a delay until they are actually available for the user. Therefore, CMFs must ensure proper co-allocation not only ensuring that resources are provisioned at the same time, but also assuring that they are delivered at the same time.

As it will be explained later on, besides the inherent delays introduced by the CMF there are other factors contributing to this delay from the user standpoint. Any reduction in each of these factors will yield on a better reactivity of the cloud, leading to an increase of the ability to satisfy elastic requests on-demand.

In summary, in order to deliver a rapid service, this spawning delay or penalty has to be decreased. It is the duty of the cloud provider to be able to provision efficiently the resources to the users, regardless of the size of the request, minimizing the costs of mapping the request into the underlying resources[98].

We need to study how current CMFs can reduce the start time of the virtual machines requested, therefore in this chapter:

- We will study how the deployment of Virtual Machine Images (VMIs) into the physical machines poses a problem to an Infrastructure as a Service (IaaS) resource provider and how it introduces a penalty towards the users.
- We will discuss several VMI transfer methods that alleviate this problem and

review the related work.

- We implement and evaluate some of the described methods in an existing CMF.
- We propose an improvement of the scheduling algorithm to take profit of the VMIs cached at the physical nodes.

The rest of the chapter is structured as follows: Section 5.1 presents and develops the problem statement that has been outlined in this introduction. Section 5.2 presents the related research in the area. Section 5.3 contains the evaluation of some of the methods described in the previous section. Section 5.4 contains a proposal of a modification to the scheduling algorithms, being evaluated in combination with the studied VMI transfer methods. Finally, conclusions and future works are outlined in Section 5.5.

5.1. Problem Statement

Whenever a VM is spawned in a cloud environment, its VMI disk must be available at the physical server. Obviously, if the image is not available on that host it needs to be transferred, therefore the spawning will be delayed until the transfer is finished. This delay is specially magnified if the request consists on more than a few virtual machines, as more data needs to be transferred over the network. As the underlying infrastructure increases its size the problem becomes also bigger, as the potential number of requests need to be satisfied may become larger.

Regardless of the CMF being used, the process of launching a VM in an IaaS cloud infrastructure comprises a set of common steps:

1. A VMI is created by some actor, such as a system administrator or an experienced user, containing an specific software environment.
2. This image is uploaded to the cloud infrastructure image catalog or image repository. This image is normally stored as read-only, therefore, if further modifications (for example any user customization) need to be done on a given image, a new one must be created, or an incremental difference can be stored, if the CMF allows for that functionality.
3. A user requests one or several VMs based on this image, that will be spawned into the physical machines.

4. The running VMs are customized on boot time to satisfy the user needs. This step is normally referred as contextualization and it is performed by the cloud users.

The first two steps normally happen once in the lifetime of a VMI, meaning that once the image is created and is available in the catalog, it is ready for being launched, so there is no need to recreate the image and upload it again and again.

Assuming that the IaaS provider is able to immediately satisfy the user request (i.e. there are enough available resources) whenever a user launches a VM, only the two former steps will introduce a delay in the boot time. However, the last contextualization phase is made once the virtual instance has booted, and it is normally a user's responsibility as it goes beyond the CMF control [31, 148].

Hence, the field where a IaaS resource provider can take actions to reduce the boot time of a virtual machine is the spawning phase. This stage involves several management and preparation operations that will depend on the CMF being used. Generally, these operations will consist on one or several of the following steps:

Scheduling phase where the software selects the most suitable nodes to satisfy the user's request.

Image transfer if the image data is not available in the selected physical machine, the CMF has to transfer it from the catalog into that host.

Image duplication once the image is available at the node. Some CMF duplicate the image before spawning the virtual machine. This way, the original image remains intact and it can be reused afterwards for another VM based on that same image.

Image preparation consisting in all the further image modifications prior to the virtual machine spawning, needed to satisfy the user's request. For instance, this step can comprise the image resize, image format conversion, user-data injection into the image, file system checks, etc.

Taking as an example the OpenStack cloud testbed described in the experimental setup of Section 5.3.1, Figure 5.1 shows the boot sequence for an instance once the request is scheduled into a physical machine. In this request a 10 GB image was launched with an additional local ephemeral disk of 80 GB. This ephemeral empty space is created on the fly on the local disk of the physical machine, therefore it is not transferred over

the network. In this initial setup, the images are stored in the catalog server and are transferred using Hiper Text Transfer Protocol (HTTP) when they are needed in the compute host.

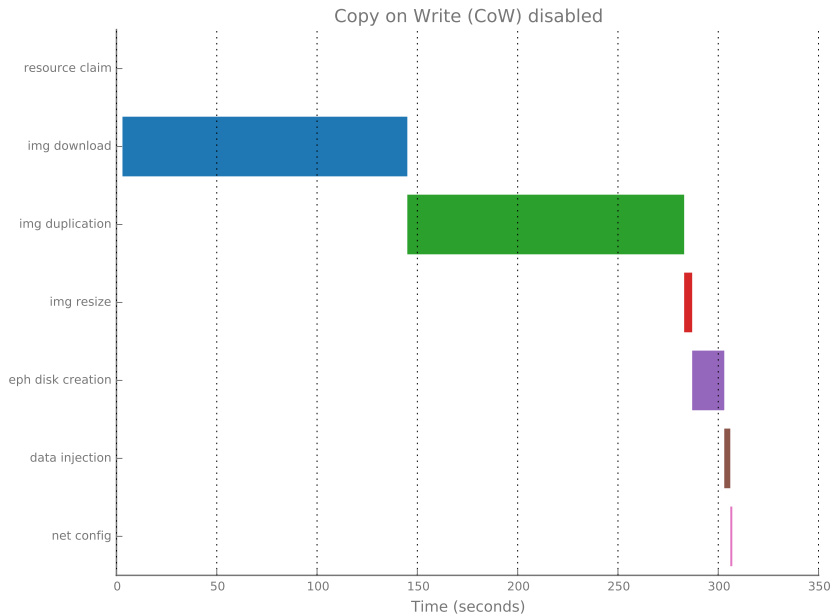


Figure 5.1: Chart of the boot process for one VM on an OpenStack cloud. The image used was 10 GB large with an 80 GB ephemeral disk.

According to Figure 5.1, the OpenStack spawning process is broken down into several sub steps:

Resource claim The compute node checks if the requested resources are available, and claims them before spawning the instance.

Image download The image is fetched from the image catalog, and it is stored in the local disk.

Image duplication An exact replica image is created from the downloaded one.

Image resize The image is resized to fit into the size request by the user. Normally minimal images are stored in order to spare disk and save transfer times, therefore these images need to be resized into the correct final size.

Ephemeral disk creation An ephemeral virtual disk is created in the local disk. This virtual disk is created on the fly and it is normally located on the local machine disk, since it is a disposable space destroyed when the instance is terminated.

Data injection Any data specified by the user is injected into the image. This step needs to figure out the image layout and try to inject the data into the correct location. This is a prone to errors step since the image structure is unknown to the middleware and therefore it can fail. It could be avoided with the usage of contextualization, assuming that the images are properly configured.

Network configuration The virtual network is configured and set up in the physical node to ensure that the instance will have connectivity.

The *Resource claim* step belongs to the *Scheduling phase*, and the steps labeled *Image resize*, *Ephemeral disk creation*, *Data injection*, *Network configuration* belong to the aforementioned *Image preparation* phase. Observing Figure 5.1 we can extract that there are three big contributors to the boot time, namely *Image download*, the *Image duplication* and the *Ephemeral disk creation* steps.

In this first test, raw images were used, meaning that the duplication involved the creation of a complete copy of the original image. This could be easily diminished by using Copy on Write (CoW) images.

The support for CoW images is implemented in all of the most common hypervisors (being the only difference the supported formats). Forcing the usage of CoW by the CMF reduces considerably the overhead, since it is not needed to duplicate the whole image container [149]. The ephemeral disk (if it exists) can be also created using CoW, so its contribution to the overhead will be diminished too. Therefore, one of the two biggest contributors to the boot time for an instance can be easily shrink with the adoption of CoW.

Figure 5.2 shows the same request, when the cloud infrastructure has been configured to use CoW images. As it is seen, two of the three biggest penalties are reduced just by switching to this method.

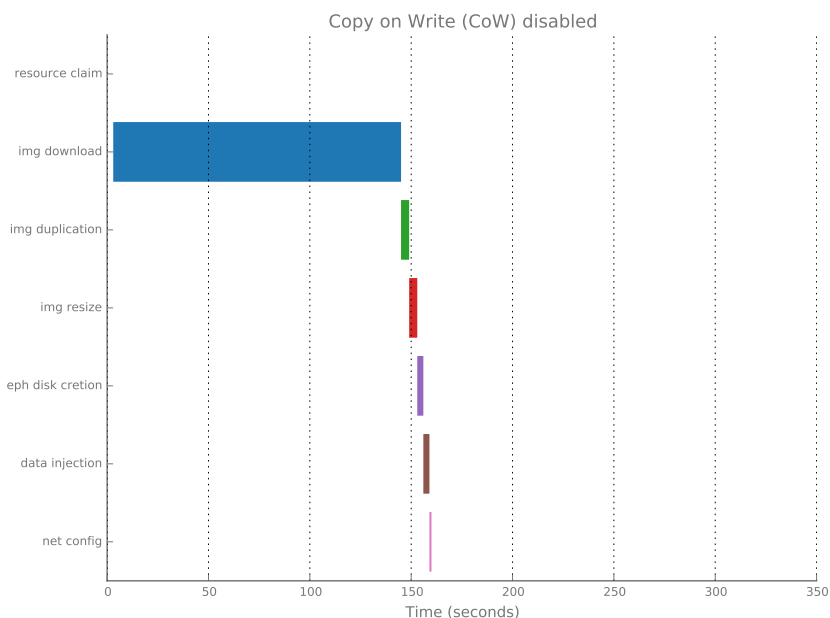


Figure 5.2: Chart of the boot process for one VM on an OpenStack cloud configured to use CoW images. The image used was 10 GB large with an 80 GB ephemeral disk.

However, the *Image download* still introduces a considerable penalty. Unfortunately, this time is dependent on several factors:

- The image delivery method used will have a large impact on the final time. It is not the same to download an image from a single central location that transfer it using peer-to-peer techniques.
- The amount of data being transferred and obviously the image size: if several hundred gigabytes need to be transferred over the network each time a machine is booted, the delay will be difficult to shrink.
- The size of the request. It is not the same to swap just a few virtual machines than spawning hundreds of VMs.
- The load on the implied systems: the network usage, catalog server and compute hosts load have an influence on the overall process.

Virtual machine images range from a few hundreds of MB to several GB [150, 151], hence an efficient image deliver method should try to tackle as much factors as possible. It should try to use a good image transfer method, should try to reduce the amount of data being transferred and thus reduce the load on the system. It should be also able to satisfy large requests, that are quite common on scientific workloads. For example, it is known that scientific communities often deploy a virtual cluster to support their users [152, 153]; sets of machines to execute a parallel application or workflow based applications [121].

5.2. Related Work

Several authors have also identified the image deployment phase as the biggest overhead to be solved when spawning VMs in a cloud infrastructure. The following subsections will describe several of the proposals that exist in the scientific literature for addressing the image distribution issue.

One of the first approaches to reduce the image distribution time is to eliminate the step itself. This could be accomplished by the usage of of a shared storage (Section 5.2.1) or by the pre-deployment or pre-fetch of images (Section 5.2.2.2 and Section 5.2.2.3 respectively). The election of a good delivery method (Section 5.2.2.1) is also crucial. Finally, some authors point towards different and novel methods requiring further developments (Section 5.2.3), that seem promising.

5.2.1. Shared Storage

This approach leverages the usage of a shared storage (such as access to a Network Attached Storage (NAS) or a Storage Area Network (SAN)) to eliminate at all any explicit image transfer. The catalog and the nodes share the same storage backend, thus once an image is uploaded to the system it will be directly available on the physical hosts. This method may seem ideal, however it has some drawbacks:

- The VMI is served over the network and nodes with an intensive Input/Output (I/O) may underperform.
- It needs a dedicated and specialized storage system and network in order to not overload the instance's network with the access to the disks. This network needs

to be properly scaled, meaning that a good performance access and acceptable reliability and availability are a must: if the shared storage does not perform as expected, it will become a bottleneck for the cloud infrastructure and will impact negatively on the virtual machines performance.

- If the system is not reliable or has a low availability, the images could not be accessed. Therefore, the IaaS resource provider needs to invest in having a good shared storage solution, with a high availability.
- The access to the shared storage by the physical machines (i.e. the hypervisor nodes) will consume resources and create undesirable Virtual Machine Monitor (VMM) Noise. This VMM Noise has been shown to have a negative impact in the virtualized guests running on those hosts and is something to avoid in scientific computing environments [154, 155, 156, 157] due to the disturbances that it may cause in the executions.

5.2.2. Image Transfer Improvements

5.2.2.1. On Demand Downloading

If no shared storage is in place, the most common approach in many CMFs is to transfer the images on-the-fly into the compute nodes when a request to launch a specific machine is made.

This is the most common and simple approach, and as already exposed in Section 5.1 the penalty introduced by this method will vary according to the size of the image, the size of the request, the network connectivity of the infrastructure, the load on the catalog servers and the transfer protocol being used.

In this case, the objective should be reducing the image transfer time. In this line there is a clear trend towards studying Peer-to-Peer (P2P) mechanisms in cloud infrastructures and data-centers. Zhang Chen et al. [147] proposed an effective approach for virtual images provisioning based on BitTorrent. Laurikainen et al. conducted a research focused on the OpenNebula CMF, taking only into account the replacement of the native image transfer method by either BitTorrent or Multicast [158]. Their conclusions showed that the existent image transfer manager (based on SSH) was rather inefficient for large requests and therefore it needed to be modified.

Wartel et al. studied BitTorrent among other solutions as the image transfer method for their initial CERN cloud infrastructure [159]. This study showed a significant performance gain when using BitTorrent over the other studied methods (that included multicasting). In the same line, Yang Chen et al. have proposed a solution based on multicasting the images instead of a direct download from the image catalog, in combination with a more efficient scheduling [160] algorithm. However, transfer an image using multicast into the nodes implies that the server is initiating the transfer (i.e. the server pushes the image into the nodes) instead of the image being pulled from the hosts. This also forces that the deployment of the images is synchronized, therefore introducing extra complexity to the scheduling algorithms that must take this synchronization into account.

Once the image is downloaded into the node, this image can be cached and reused afterwards in a subsequent request. This feature opens the door to the pre-deployment of images and the image pre-fetch, that will be discussed in Section 5.2.2.2 and Section 5.2.2.3 respectively. Multicasting is an interesting option for these two cases, since the deployment could be done in a coordinated way, without interfering with the scheduling algorithms, but when compared with multicast, using a P2P method introduces another advantage: the nodes that have an image available are part of the P2P network, participating actively in the transfer when a new request is made.

5.2.2.2. Pre-Deployment of Images

A different approach towards the elimination of the image transfer prior to the image boot consists on the pre deployment of the whole or a portion of the image catalog into the physical machines. In some environments this might be a valid solution, but it is not affordable in large setups for several reasons:

- In an infrastructure with a large catalog, storing some TB images, a considerable amount of disk space would be wasted on the nodes, not to say the penalty incurred when transmitting those images over the network. Considering that not all the images will be spawned into all the nodes at a time, this resource consumption is not affordable.
- The pre-deployment process can overload the catalog server when it is triggered, causing a denial of service if it is not carefully scheduled.

-
- A CMF using this method should also consider that a VMI that has been recently uploaded to the catalog may not be immediately available to the user, since it has to be pre-seeded into the nodes, so an alternative, on-demand method should still be available.

5.2.2.3. Smart Pre-Fetch

Another possibility, related to the previous one, is performing a selective pre-deployment of the images into the nodes (i.e. smart pre-fetch). Instead of the passive deployment of the whole catalog (or a large portion of it) into the nodes, the scheduler may chose to trigger a download of an image in advance, so that it anticipates a user request.

Image popularity (i.e. how often an image is instantiated) can be used as a parameter to decide which images to pre-fetch. A naive approach could be summing up how many virtual machines have been instantiated from a given image. Figure 5.3 shows the image *popularity* for a set of 13 500 VMs execute by 150 different users on the IFCA production infrastructure described in Appendix A.1 during one year in ascending order. The Y-axis shows the number of instances that were based on a given image.

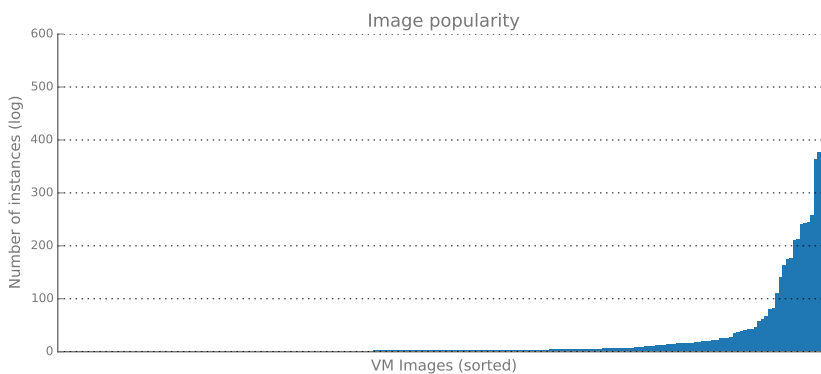


Figure 5.3: Image popularity based on the number of Virtual Machines spawned per image. Each bar represents a different image.

As it can be seen, and even if this popularity calculation is too naive and simplistic, a large proportion of the spawned instances is based on a small number of images. This

is confirmed by some other authors that have observed the same behaviour in related works, such as Peng et al. [161].

Therefore, if the CMFs could take advantage of the image popularity making those VMs available on some nodes the efficiency of the image booting process will improve.

5.2.3. Other Methods

Lagar-Cavilla et al. have developed Snowflock [162], a new model for cloud computing that introduces VM forking in a way similar to the well known and familiar concept of process forking. This method permits the cloning of an already running VM into several identical copies. However this is not transparent, and the users need to be aware of its semantics and program their application accordingly.

Some other authors have chosen a totally different approach relying on the fact that the image is not needed completely at once, therefore it can be divided into smaller chunks that will be transferred when they are needed. Peng et al. propose the usage of a collaborative network based on the sharing of similar image chunks [161]. In their studies, they found that this approach was more efficient than the usage of a P2P network, but it requires a long running preprocessing step. Moreover, this is true for the cases analyzed, where the number of different VMs requested at a time was not big but this may not apply to other cases, such as scientific cloud providers where the same image may need to be spawned into several nodes.

The work from Nicolae et al. is also based on this approach. They implemented a self adaptive mechanism, based on lazy downloads of image chunks, based on previously recorded access patterns [163].

5.3. Transfer Method Evaluation

As Section 5.2 exposed, there is not a single solution for solving the image distribution problems, as all of the presented schemes have their advantages and disadvantages. In some situations, the usage of a shared backend may be the best solution but it would not fit others. For example, sites deploying virtual machines that need high availability may already use a shared backend so that it is possible to quickly recover a running machine from a failure, whereas sites devoted to HTC and HPC computing may not find this deployment appropriate as it may impact their performance.

However, there is room for improvement in the image transfer method, taking into account that this is the default method for the most used CMF. Therefore, in this Section several image transfer methods will be evaluated and compared.

5.3.1. Experimental Setup

In order to implement the solution proposed we have used the OpenStack [141] CMF, in its Icehouse (2014.1) version, on top of the testbed described in Appendix A.3, consisting on a *head node* hosting all the required services to manage the cloud infrastructure, an *image catalog* server and 24 *compute nodes* that will eventually host the spawned virtual machines. All of them are identical machines, with two four-core Intel®Xeon®E5345 2.33 GHz processors, 16 GB of RAM and one 140 GB, 10 000 rpm hard disk.

The network setup of the testbed consists on two 10 Gbit Ethernet switches, interconnected with a 10 Gbit Ethernet link. All the hosts are evenly connected to these switches using a 1 Gbit Ethernet connection.

The operating system being used for these tests is an Ubuntu Server 14.04 LTS, running the Linux 3.8.0 Kernel.

In order to execute the same tests easily, a benchmarking as a service product was used: Rally [164]. This tool allows for the definition and repetition of benchmarks, so that the benchmarking tests can be reproduced later on.

OpenStack's default method for distributing the images into the nodes is an on-demand deployment: the images are fetched from the catalog when the new virtual machine is scheduled into a compute (physical) node and its image cannot be found on that host.

The catalog service component (whose codename is Glance) stores the images using one of the many available backends, but independently of the backend used and the default transfer method is HTTP. When Glance stores the images in a filesystem it is possible to setup a shared filesystem so that the space where the images are stored by Glance are available on the compute nodes. Other backends allow to distribute the images over the network using different protocols and methods (for example, using the Ceph Rados Block Devices (RBD)). However, since we wanted to test the influence of the transfer from the catalog to the nodes, the default method was used.

5.3.2. Test Results

In order to evaluate the effect of the image transfer method the system was stressed by making requests that involved fetching a large number of images, as described in Table 5.1, using several methods: HTTP, File Transfer Protocol (FTP) and BitTorrent. 5 GB images were used and the scheduler was configured to evenly distribute the images among the hosts in the cluster in order to maximize the effect of the image transfer on the nodes. All the tests were done by triplicate.

Name	VMs per host	Different images	Total number of VMs
1x192	8	1	192
2x96	8	2	192
4x48	8	4	192
8x24	8	8	192

Table 5.1: Request characteristics.

5.3.2.1. HTTP Transfer

In the first place the images were transferred using HTTP, since it is the default image transfer method available on OpenStack. Figure 5.4, shows the required time to boot the virtual machines for each of the requests in Table 5.1.

The best scenario in these tests is where a user requests a single image (1x192 in Figure 5.4). This is mainly because of the effect of the cache that is available in each of the nodes. Once the image is downloaded in a node, all the subsequent virtual machines can be spawned using that cached image (this fact is also true for the other studied methods). The worst scenario is when the user requested 8 groups of 24 virtual machines (8x24 in Figure 5.4), since all the 8 images had to be downloaded into each of the nodes.

5.3.2.2. FTP Transfer

As a second step, the File Transfer Protocol (FTP) was used, therefore the built-in HTTP server was substituted with a dedicated FTP. Figure 5.5 shows again the results for the requests in Table 5.1.

As it can be seen, the boot time is almost the same for both methods, being FTP more homogeneous over HTTP, resulting in a most uniform boot time for the machines.

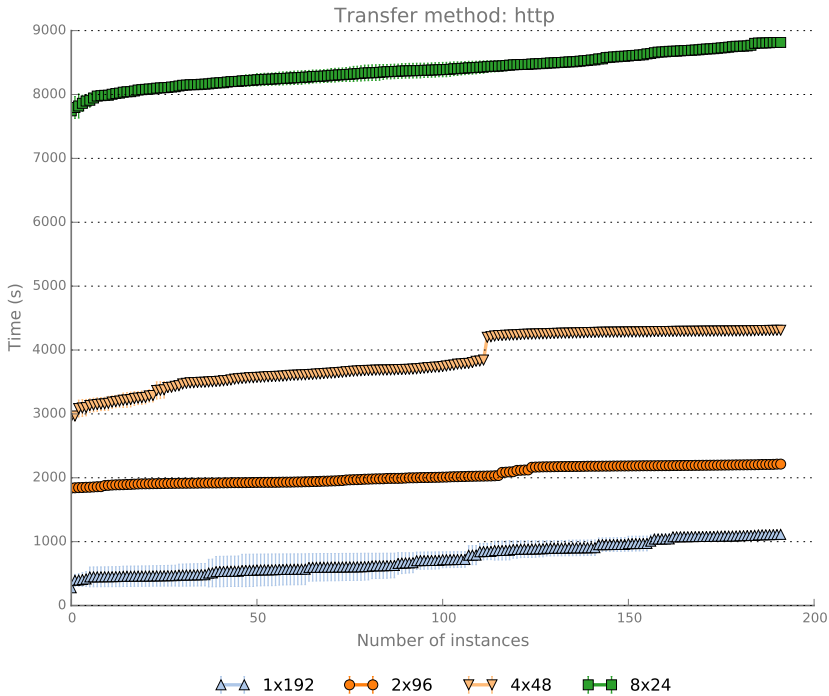


Figure 5.4: Waiting time as a function of the number of instances requested when the images are fetched using HTTP. 1x192 means 1 request of 192 machines using the same image; 2x96, 2 requests of 96 machines using two different images, 4x48, 4 requests of 48 machines with four different images; and 8x24 8 requests of 24 machines with eight different images.

5.3.2.3. BitTorrent Deployment

Both the HTTP (Section 5.3.2.1) and FTP (Section 5.3.2.2) are based on a centralized client-server model. In order to see how the system performs using a P2P model we adapted OpenStack image delivery method to use BitTorrent. It was chosen for several reasons: it is a protocol designed for to reduce the impact of transferring large amounts of data over the network [165]; it is widely used in a daily basis and there is a wide range of libraries, clients and applications available; moreover, due to this lively implementation ecosystem, we found that it could be easily integrated into OpenStack.

The chosen implementation was libtorrent [166], as it has Python bindings and OpenStack is written entirely in Python, thus it was easily integrable. Our *swarm* used

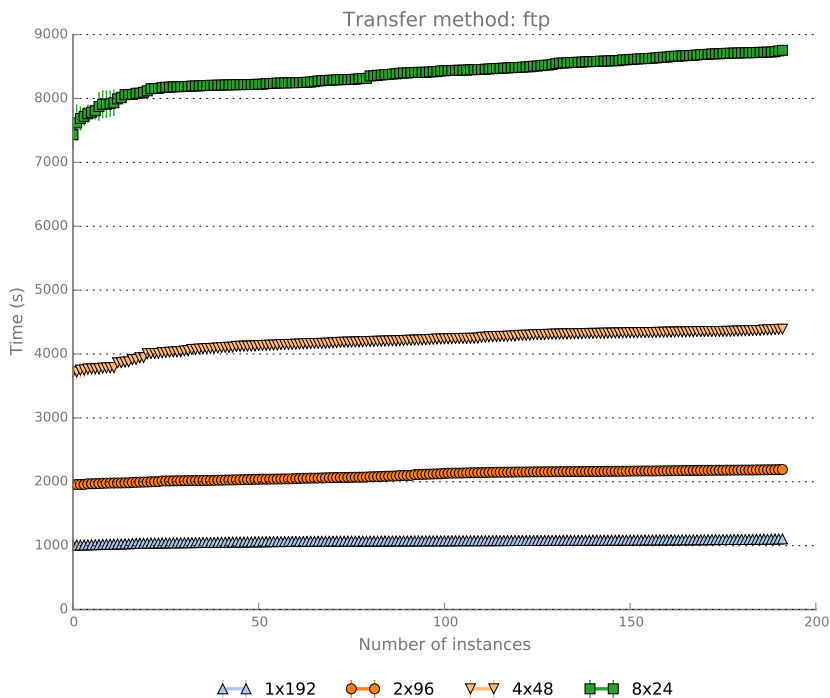


Figure 5.5: Waiting time as a function of the number of instances requested when the images are fetched using FTP. 1x192 means 1 request of 192 machines using the same image; 2x96, 2 requests of 96 machines using two different images, 4x48, 4 requests of 48 machines with four different images; and 8x24 8 requests of 24 machines with eight different images.

the BitTorrent Distributed Hash Table (DHT) extension, so that it was possible to use tracker-less torrents, although it is perfectly feasible to run a tracker. The client in each node was configured to run only three concurrent active downloads, since in preliminary tests we observed this was the best choice for our infrastructure.

The results for serving the same requests as in the HTTP and FTP cases are shown in Figure 5.6.

In our implementation a new torrent is generated whenever a new image is uploaded to the catalog. The torrent metadata is stored along with the ordinary image metadata so that whenever a download of this image is requested, both the normal HTTP and the torrent's magnet link are provided to the compute node. If the node needs to download

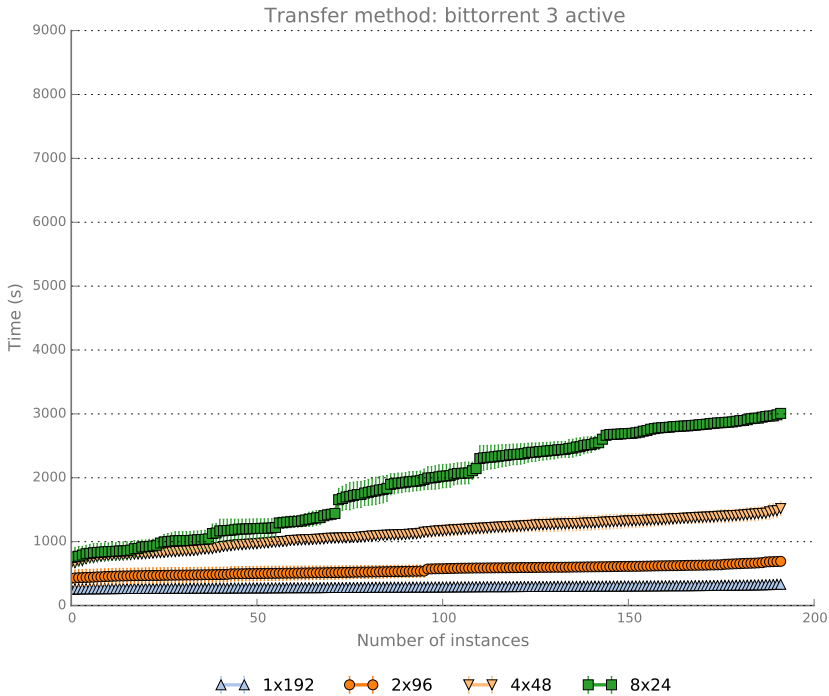


Figure 5.6: Waiting time in function of the number of instances requested when the images are fetched using BitTorrent. 1x192 means 1 request of 192 machines using the same image; 2x96, 2 requests of 96 machines using two different images, 4x48, 4 requests of 48 machines with four different images; and 8x24 8 requests of 24 machines with eight different images.

the image, and a magnet link is available, this *peer* (i.e. a BitTorrent client) will join the *swarm* (i.e. all peers sharing a torrent). Due to the segmented file transfer that BitTorrent implements, this *peer* is able to *seed* (i.e. send its available data) the received data to the other peers. This way, the original seeder of the image (i.e. the catalog server) is freed from sending that portion to every peer of the network.

5.3.3. Result Comparison

A comparison of the three methods evaluated (that is, transfer the images using HTTP, FTP and BitTorrent, and profit from the images caching) is shown in Figure 5.7.

Both FTP and HTTP threw similar results, being those limited by the bandwidth of

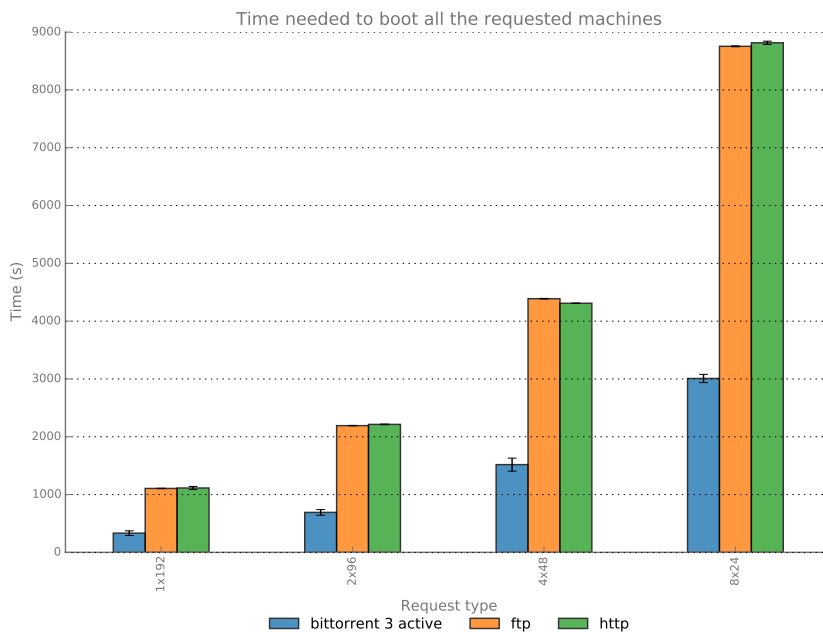


Figure 5.7: Seconds elapsed from request until all the machines were available. The VMs used was 5 GB large, and they were spawned on 24 hosts.

the server node. Using BitTorrent, there is a significant transfer time reduction. In the worst scenario (8x24: running 192 virtual machines, distributed in 8 different images in 24 nodes) it was possible to start the 192 machines at approximately one third of the time required to run those machines using HTTP or FTP.

If we take into account the boot time for the first machine of the request we can find interesting results. Figure 5.8 shows the elapsed time until the first machine is available. In this case, BitTorrent also outperforms the other transfer methods, making possible to deliver the machines earlier to the users except in the case of transferring only one image into all the nodes. In this case, HTTP and BitTorrent throw similar results.

Another important fact is that the adoption of BitTorrent not only has the effect of reducing the transfer time, but it also reduces the load of the catalog server. Since the image distribution leverages the advantages of the P2P network, where all the nodes participate in the transfer, the catalog does not need to transfer all the data to all of the nodes.

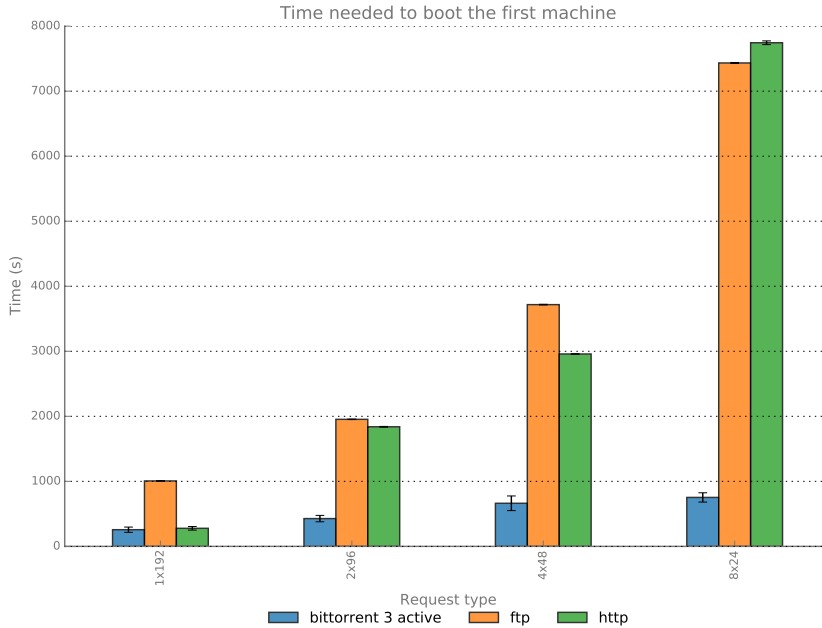
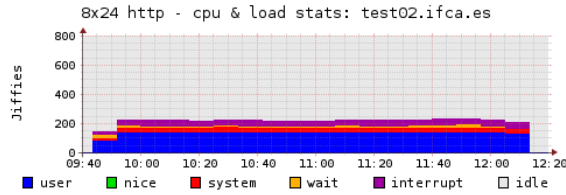


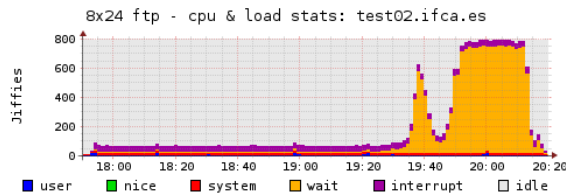
Figure 5.8: Seconds elapsed from request until the first machine of the request is available. The VMs used was 5 GB large, and they were spawned on 24 hosts.

As it can be seen in Figure 5.9 and Figure 5.10, using BitTorrent makes possible to satisfy the same request at a fraction of the CPU usage and specially network bandwidth when compared with HTTP and FTP, resulting in a better utilization of the resources.

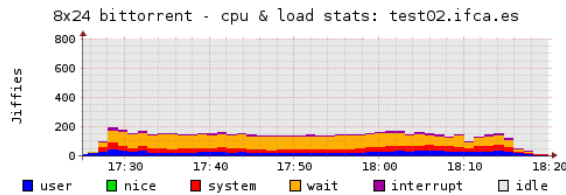
However, using BitTorrent has its drawbacks also. It needs another running service (a tracker, although it could be avoided using a Distributed Hash Table (DHT)). Moreover, the creation of a torrent file whenever a new machine image is added to the catalog takes a considerable amount of time and resources, growing with the size of the file. Therefore the torrent will not be available as soon as the image is uploaded, but a lapse of time will be introduced. Since this operation is done only once in the lifetime of a virtual machine it can be considered as part of the initial upload process.



(a) HTTP.



(b) FTP.



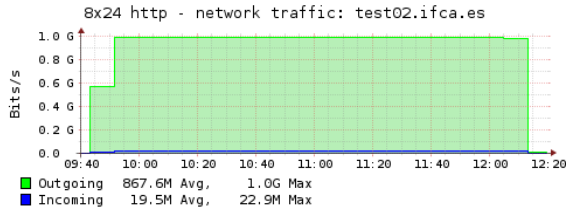
(c) BitTorrent.

Figure 5.9: CPU usage for a 192 VMs request using 8 different images (8x24) on the catalogue node.

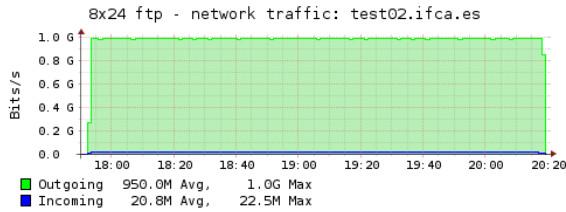
5.4. Efficient Image Distribution

The previous made emphasis in the effect of the image distribution method on the boot time for a virtual machine. All of the presented tests started from a clean environment, meaning that there were no images cached in the nodes. The tests were designed to stress the infrastructure so that the image transfer effects could be clearly noticed. In this section a different test will be performed so as to take into account the images cached in a physical node when making scheduling decisions under more realistic scenarios.

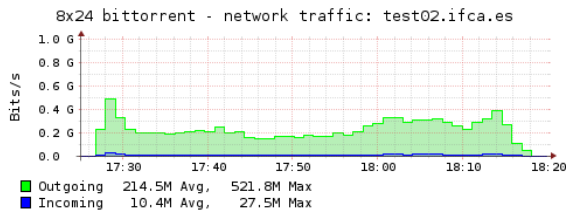
In order to evaluate the proposed solution, I have implemented the design described



(a) HTTP.



(b) FTP.



(c) BitTorrent.

Figure 5.10: Network usage for a 192 VMs request using 8 different images (8x24) on the catalogue node.

next for the OpenStack Filter Scheduler, as described in Section 4.3.

Four different scenarios were tested: using the OpenStack's unmodified scheduling algorithm and using a cache-aware scheduler; using both HTTP and BitTorrent as the transfer methods. This way the tests would assess not only the effect of the cache but also the transfer method.

In our test environment all the hosts have the same hardware characteristics, so when they are empty they are equally eligible for running a machine. As explained, the nodes will get the same weight and finally a random selection is done. Therefore it is possible that a machine is scheduled in a node that does not have the image available,

when there is another node with the same weight with the image cached. In the best case, the image is transferred only once (that, is for the first request), whereas in the worst case the image will have to be transferred every time it is used.

By default OpenStack has an image cache in each of the nodes, but the scheduler does not take it into account when selecting the host that will execute a machine. I developed several modules for OpenStack¹, allowing to weight the hosts taking into account their cached images. First of all, the nodes have report their cached images back to the scheduler. Afterwards, the cache weighter will simply weight the nodes as follows:

$$w_{\text{cache}}(h) = \begin{cases} 1 & \text{if image is cached} \\ 0 & \text{otherwise} \end{cases}$$

No other other sanity checks were applied in the weighter since this is not the purpose of our function (there are specific weighters and filters that should prevent to overload a host).

Therefore, with the above configuration in the cache-aware tests, the images were only transferred the first time they are scheduled, since all the subsequent requests will be scheduled in any of those hosts.

5.4.1. Evaluation

In order to make a realistic evaluation, I executed different simulated request traces for each of the scenarios described before: that is, an scheduler with and without cache, using HTTP and BitTorrent.

Two arrival patterns using an exponential distribution [168] were generated: one requesting at a rate of 80 machines per hour and a second one for 100 machines per hour. For each of the requests I assigned an image chosen randomly from a given set of 4 images. Finally, the two resulting traces were executed in each of the four scenarios.

Figure 5.11 shows the scatter plot of the seconds needed to boot each of the requests and its respective request pattern for 80 machines at an arrival rate of 80 machines per hour. Figure 5.12 shows the kernel density estimation of the test.

¹Á. López García. *OpenStack Compute Scheduler Cache Aware*. URL: <https://blueprints.launchpad.net/nova/+spec/cache-aware-weighter>.

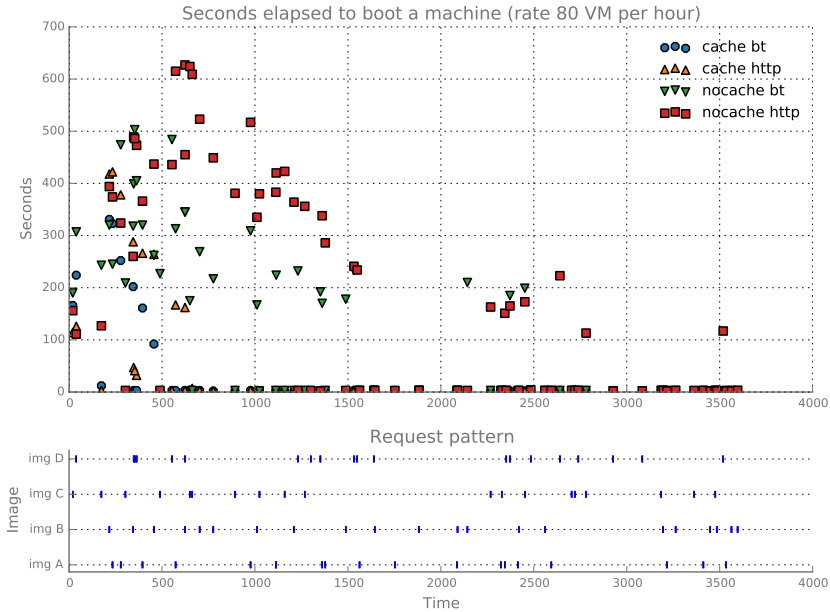


Figure 5.11: Seconds elapsed to boot a machine for 80 requests during 1 h, with the corresponding requests trace. *nocache http* and *nocache bt* refer to the default scheduling method using HTTP and BitTorrent respectively, whereas *cache http* and *cache bt* refer to the cache-aware scheduler, using HTTP and BitTorrent respectively.

Besides, Figure 5.13 contains the plot for 100 machines at an arrival rate of 100 machines per hour, with the corresponding density function shown in Figure 5.14.

As it can be seen in both Figures 5.11 and 5.13, in all evaluated scenarios the minimum values are similar and very low due to the effect of the cache. In the cases when the scheduler did not have this feature available there is still a random chance that a machine is scheduled in a node with the image cached, thus the observed results. The probability of using a node with the image already available increases with time (more nodes have been used and therefore more nodes have the image cached) and as a consequence the boot times for the last images was lower. When the cache-aware scheduler was used, only the first machines started require transfer to the nodes, hence the boot times are reduced to the minimum early in the execution of the trace.

On the other hand, Figures 5.12 and 5.14 thrown interesting results, considering the size of the requests. The best results are always obtained when using BitTorrent and a

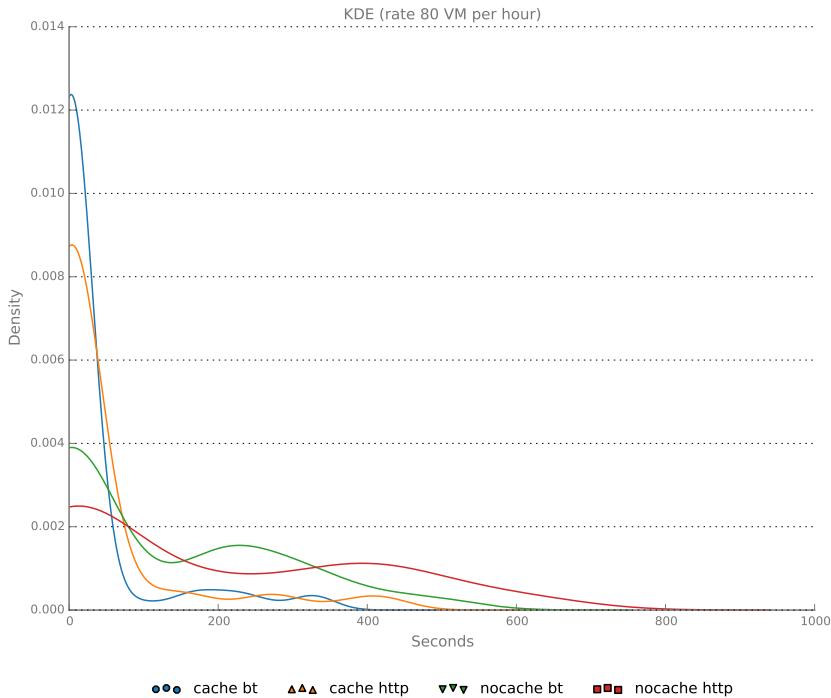


Figure 5.12: Kernel Density Estimation (KDE) for the time elapsed to boot the requests in Figure 5.11. *nocache http* and *nocache bt* refer to the default scheduling method using HTTP and BitTorrent respectively, whereas *cache http* and *cache bt* refer to the cache-aware scheduler, using HTTP and BitTorrent respectively.

cache-aware scheduler. However, the next best case depends on the request pattern. In the case of a rate request of 100 machines per hour, using BitTorrent without a cache is better than using HTTP with a cache, but in the case of a rate of 80 machines per hour it is better to use the later. This observation is due to the fact that in the 100 machines case there is a large initial portion of images that need to be transmitted if compared with the 80 machines case (as depicted by the dots between time 0 and 500 in Figures 5.11 and 5.13). Therefore BitTorrent outperforms HTTP, as already explained in Section 5.3.3. The cache does not consider the images that are being fetched, therefore the scheduler cannot take them into account. As the 100 machines case requests machines at a higher rate they are being scheduled when the images are not yet available, thus the observed

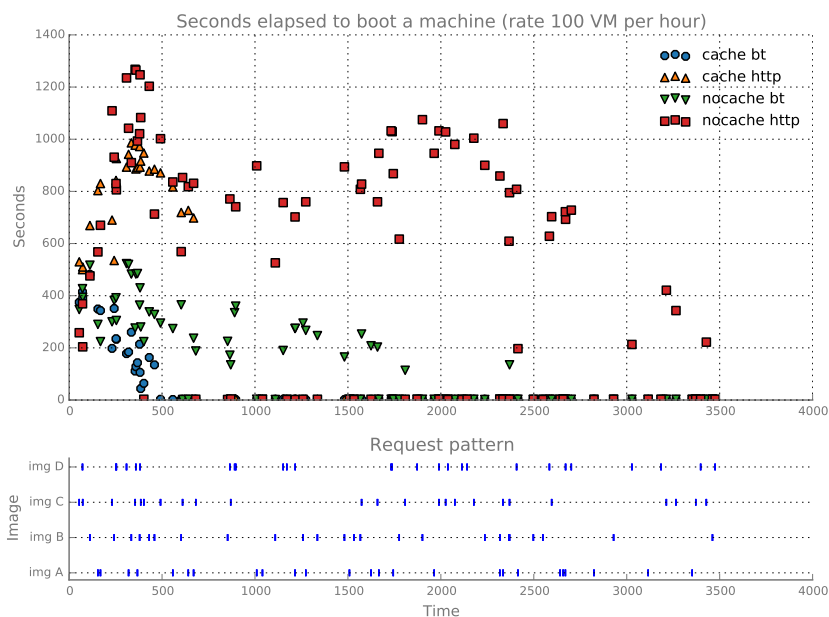


Figure 5.13: Seconds elapsed to boot a machine for 100 requests during 1 h, with the corresponding requests trace. *nocache http* and *nocache bt* refer to the default scheduling method using HTTP and BitTorrent respectively, whereas *cache http* and *cache bt* refer to the cache-aware scheduler, using HTTP and BitTorrent respectively.

results.

5.4.2. Image pre-fetch

As already explained, the usage of the cache with BitTorrent outperforms all of the other methods. In order to evaluate its effect regarding the tests shown in Section 5.3 the same requests from Table 5.1 were recreated with the images already cached on the nodes. Obviously, in this test we do not evaluate the penalty introduced by the image transfer since there is no transfer at all, but it is interesting in order to evaluate the overall performance of the system. As it can be seen in Figure 5.15, the booting time was dramatically reduced in all cases: booting all the 192 machines was done in less than 45 seconds as the only delays introduced were due to the scheduling algorithm and the different management operations.

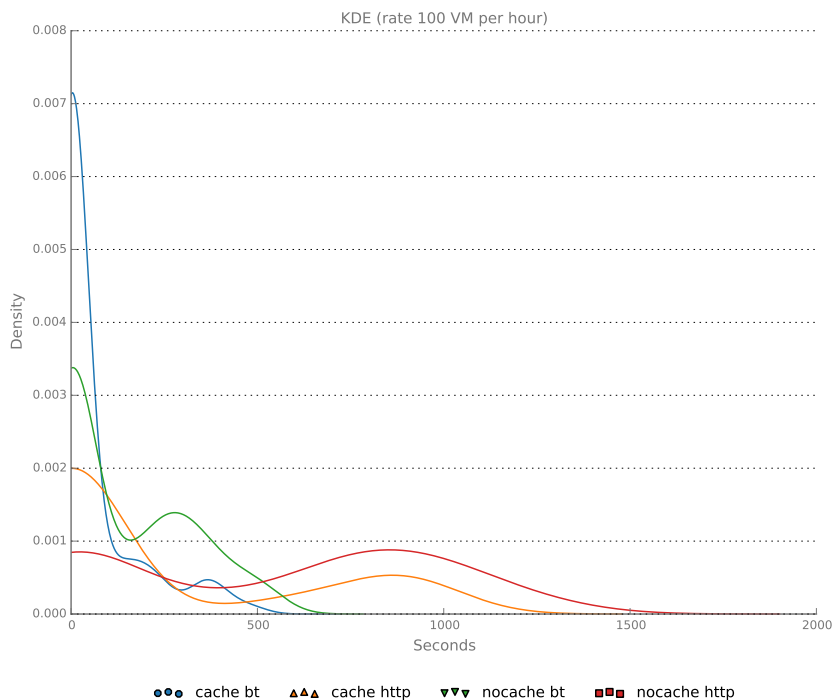


Figure 5.14: Kernel Density Estimation (KDE) for the time elapsed to boot the requests in Figure 5.13. *nocache http* and *nocache bt* refer to the default scheduling method using HTTP and BitTorrent respectively, whereas *cache http* and *cache bt* refer to the cache-aware scheduler, using HTTP and BitTorrent respectively.

5.5. Conclusions

In this chapter, I have evaluated several methods for the distribution of Virtual Machine Images (VMIs) into the compute nodes of a cloud infrastructure. Although the work was performed using the OpenStack Cloud Management Framework (CMF), the results can be extrapolated to other CMFs using similar transfer methods.

The experiments showed that composing a Peer-to-Peer (P2P) network based on a well established protocol such as BitTorrent is a simple, feasible and realistic solution to decrease the burden on the server and to reduce the transfer time to a smaller fraction of time.

Moreover, we also evaluated the usage of an image cache in each of the compute

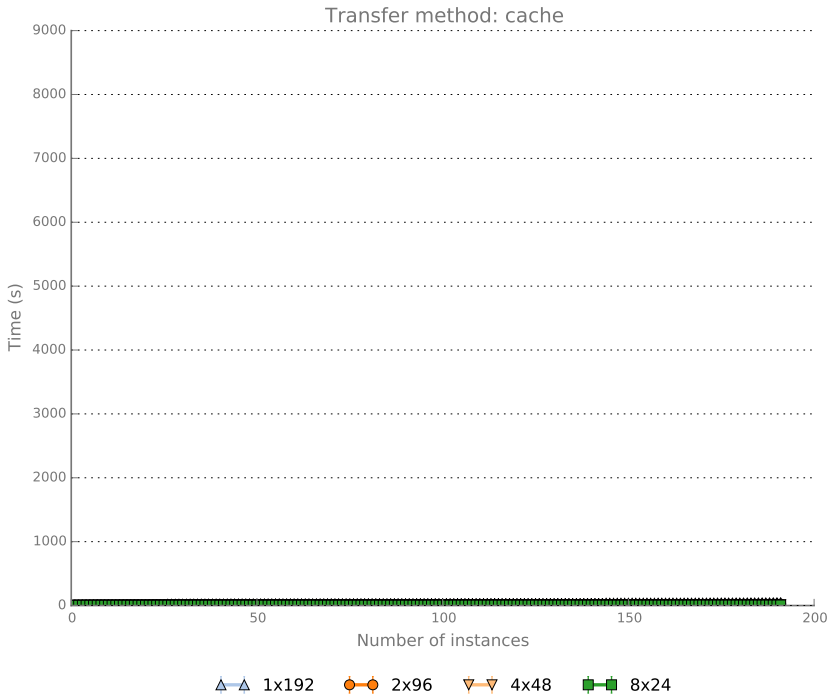


Figure 5.15: Waiting time in function of the number of instances requested when the images are cached in the nodes. 1x192 means 1 request of 192 machines using the same image; 2x96, 2 requests of 96 machines using two different images, 4x48, 4 requests of 48 machines with four different images; and 8x24 8 requests of 24 machines with eight different images.

nodes. Using an image cache obviously reduces the boot time to a minimum, since there is no transfer at all, therefore having a scheduler that takes this into account is a need. The best results were obtained when the scheduler was adapted to take into account this cache, coupled with the usage of BitTorrent as the image transfer method. Therefore, both solutions are complementary: on the one hand we reduce the image transfer time when it is needed, and on the other hand we profit from the cached images whenever possible.

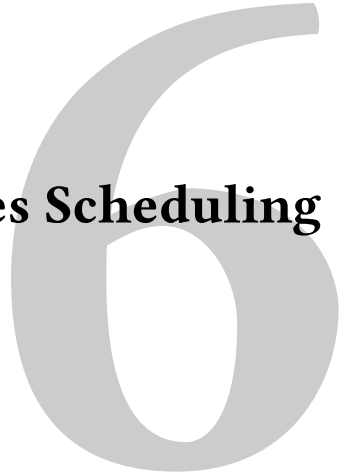
Taking into account those results, there is room for future work and improvements in the cloud scheduling algorithms so as to improve the boot time for virtual machines. Cloud schedulers should be adapted to be cache-aware, implementing at the same time

policies that would ensure a compromise between a fast boot time (i.e. the usage of a node with an image cached) and a fair utilization of the resources (i.e. not constricting all request to be scheduled only in one node). In this regard, the OpenStack Compute Scheduler has been a modified so as to take into account cached images when scheduling new instances².

On the other hand users tend to request images comprised in an small set of images (as shown in Figure 5.3 and explained in Section 5.2.2.3). Therefore, the usage of popularity based distribution algorithms (so that the most used images are available in the hosts) together with the cache aware scheduling would introduce remarkable improvements in the deployment times. In this regard, cloud monitoring [146] plays a key role, since one of the premises for doing a proper pre-fetching is proper monitoring so as to get proper metrics to evaluate if an image will be deployed or not.

²Á. López García. *OpenStack Compute Scheduler Cache Aware*. URL: <https://blueprints.launchpad.net/nova/+spec/cache-aware-weigher>.

Preemptible Instances Scheduling



As it has been already explained, using clouds for batch processing introduces resource allocation problems for resource providers if the resources are going to be shared with different workloads (for example, interactive executions) or they are requested by different users. Since virtual machines spawned in a cloud do not have a termination time, the resource providers need to manage their resources in a way that all users and kinds of workloads are able to access the infrastructure. Normally, this is accomplished by a static partitioning of the resources, that will eventually lead to an underutilization of the infrastructure or that will make impossible to react to demand peaks.

In this chapter I will tackle this allocation problem from the perspective of the resource provider, proposing a mechanism based on preemptible or terminable instances. This mechanism would make possible that resources are used together to run long running and fault tolerant tasks —such as batch processing— together with other workloads that need from interactivity or that are not fault-tolerant. The system proposed is modular enough so that more complex pricing of priorities system can be developed on top of it.

Even if the work is set out so as to satisfy the different workloads being executed in scientific datacenters, its use cases are not limited to that field, as the preemptible mechanism could be easily adopted by commercial providers so as to get a better usage of their infrastructure, offering an interesting computational option to their users.

In Section 6.1 I present the problem statement and background. I review the related work in Section 6.2. In Section 6.3 I propose a design for scheduling preemptible instances, that I evaluate in Section 6.4. Finally, the conclusions are presented in Section 6.6.

6.1. Problem Statement

Performing an efficient resource provisioning is a fundamental aspect for any resource provider. Traditionally, Local Resource Management Systems (LRMS) or batch systems have been used for decades to obtain the best usage of the resources as long as they provide a fair usage and partition of the resources for the users. In these systems, tasks (jobs) with a defined duration are executed according to their priority in the available resources. The system manages the tasks for several users, so as to ensure that all of them have their share in a given time window.

On the other hand, as I already explained, clouds are gaining interest in the scien-

tific computing. However, unlike in batch processing environments, virtual machines spawned in a cloud do not have fixed duration in time and are supposed to live forever—or until the user decides to stop them. This fact, as explained in Chapter 3, is one of the factors that make difficult for a provider to scale and partition their resources, since the requested resources can live forever if the user decides to do so.

In a commercial cloud provider this would not be a problem, since customers are being charged for their usage, so the provider would get revenues even if the machines are idle, but in the academia users do not usually pay—or at least they are not charged directly—for their resources. In a scientific computing facility it is desirable to get the maximum utilization of the resources, therefore idle nodes are not desirable as long as there is processing that needs to be done.

This issue could be solved by limiting the amount of resources that a user or group is able to consume, that is, doing a static partitioning of the resources. This kind of resource allocation, however, leads to an underutilization of the infrastructure since the partitioning needs to be conservative enough so that other users could utilize the resources.

The adoption of special instance types with a fixed duration or wall time could be considered as another possibility, so that a cloud computing infrastructure would behave like a batch system, queuing requests until there is some room for them. Due to the fact that the running virtual machines have a fixed duration, there would be a free slot sooner or later, so the request will be fulfilled after some queued time. However, this approach collides with the *rapid elasticity* and *on-demand* characteristics of the cloud model [1], expected by many users [114]. Since the requests need to be queued until they can be scheduled, users are not able to get a machine in a short period of time they lose the interactivity needed by their applications. Once again, this could be solved by a fixed resource allocation, but this will lead to a resource underutilization as well.

Another approach more would be finding a scheduling mechanism that makes possible that batch processing and interactive sessions could live together. This is an option possible in traditional batch systems using advanced techniques such as task suspension, preemption and checkpointing [169, 170, 171], where a higher priority task (for example, a short and interactive task) is able to get a slot because some other lower priority batch tasks are stopped or preempted so that their slots can be used by the new ones.

In the cloud, some commercial providers provide similar concepts in their offerings. For example, in the Amazon Elastic Compute Cloud (EC2) (*EC2 Spot Instances* [172]) users are able to select how much they are willing to pay for their resources —i.e. they put a bid on their desired price— in a market where the price fluctuates accordingly to the demand. Those requests will be executed taking into account the following points:

- The EC2 Spot Instances will run as long as the published spot price is lower than their bid.
- The EC2 Spot Instance will be terminated when the spot price is higher than the bid (out-of-bid).
- If the user terminates their spot instances, the complete usage will be accounted, but if it gets terminated by the system, the last partial hour period will not be accounted.

When an out-of-bid situation happens —meaning that there is no room in the infrastructure to satisfy a request with a higher price—, the running spot instances will be terminated without further advise. This rough explanation of the Amazon’s Spot Instances can be considered similar to the traditional job preemption based on priorities, with the difference that the priorities are being driven by an economic model instead by the usual fair-sharing or credit mechanism used in batch systems.

More recently, the Google Cloud Engine (GCE) [173] has released a new product branded as *Preemptible Virtual Machines* [174]. These new Virtual Machine (VM) types are short-lived compute instances suited for batch processing and fault-tolerant jobs, that can last for up to 24 h and that can be terminated if there is a need for more space for higher priority tasks within the GCE.

This kind of cloud usage can be leveraged by many fault-tolerant use cases not limited to scientific computing with applications ranging from Big Data to web crawling. The preemptible or spot instances can accommodate workloads designed or adapted to be interruption tolerant, so the customers running them can access computing resources at a fraction of the normal price. This concept of terminable instances can be used therefore to increase the global usage of a cloud computing infrastructure, making possible that users get computing power at a fraction of the price.

In order to be able to model any prizing, bidding or priority mechanism, an initial abstraction needs to be implemented in the underlying Cloud Management Framework

(CMF) so that an instance can be tagged as *preemptible* and terminated by the cloud scheduler if needed. Once this initial abstraction is done, it would be possible to implement a market driven by an economic model, as the Amazon EC2 Spot Instances does, but it also makes possible to develop any other model, driven by a fair-sharing mechanism or a credit system, just to cite some possibilities.

6.2. Related Work

The resource provisioning from cloud computing infrastructures using spot instances or similar mechanisms has been addressed profusely in the scientific literature in the last years. However, the vast majority of this work has been done from the users' perspective when using and consuming spot instances.

Voorsluys et al. have studied the usage of spot instances to deploy reliable virtual clusters [96, 116], managing reliably the allocated spot instances. Jain et al. have performed studies in the same line, but focused on using a batch system that leverages the spot instances [175]. In [127] the authors develop a workflow scheduling scheme that reduces the waiting time using spot instances.

Regarding Big Data analysis, several authors have studied how the usage of spot instances could be used to execute MapReduce workloads reducing the monetary costs, such as in [176, 177].

The usage of spot instances for opportunistic computing is another usage that has awakened a lot of interest, especially regarding the design of an optimal bidding algorithm that would reduce the costs for the users [178, 179]. There are already existing applications such as the vCluster framework [180] that can consume resources from heterogeneous cloud infrastructures in a fashion that could take advantage of the lower price that the spot instances should provide.

Due to the unpredictable nature of the spot instances, there are several research papers that try to improve the task completion time —making the task resilient against termination— and reduce the costs for the user. Andrzejak et al. [181] propose a probabilistic model to obtain the bid prices so that the costs and performance and reliability can be improved. In [182, 183, 184, 185] the task checkpointing is addressed so as to minimize costs and improve the whole completion time.

To the best of our knowledge, there is a lack of research in the feasibility, problematics, challenges and implementation from the perspective of the IaaS provider.

Nadjaran Toosi et al. have developed a Spot Instances as a Service (SIPaaS) framework, a set of web services that makes possible to run a spot market on an OpenStack cloud [186]. However, this framework is designed to work *on top* of a cloud, not being integrated with the rest of the system. A system such as SIPaaS or the related Ex-CORE auction algorithm could profit from my proposed architecture and design, integrating the execution of spot instances within the OpenStack scheduler.

6.3. Preemptible Instances Design

As briefly exposed in Section 6.1, the first step for making preemptible instances a reality is the initial abstraction inside the CMFs schedulers so that a user can tag an instance as being preemptible. This basic functionality should be the keystone for modeling any more complex system, such as the ones based in stock prices fluctuations.

This functionality should be designed and implemented in an agnostic and modular way, so that it is not dependant on any economic or priority model driving the price and instance selection. This way, the most basic preemptible instances mechanism—that is, without taking into account prices or priorities, just preemptible *versus* normal instances— can be offered by the resource providers.

The aim of this work is to design a modular system so that it would be possible to model any more complex model on top of this design once the initial preemptible mechanism is in place. With this vision, the scheduler should be smart enough to discriminate between two instance types: preemptible and non-preemptible or normal instances. A normal instance should be able to occupy the space utilized by a preemptible instance if the scheduler is not able to satisfy this request using free resources. Therefore, whenever a new request needs to be scheduled, the CMF should take the appropriate action, taking into account if it is a preemptible instance or not.

- If it is a normal instance and there are no free resources for it, it must check if the termination of any running preemptible instance will leave enough space for the new instance.
 - If this is true, those instances should be cleared out and the new VM should be scheduled into that freed node.

-
- If this is not possible, then the request should continue with the failure process defined in the scheduling algorithm (it can be an error, or it can be retried after some elapsed time).
 - If it is a preemptible instance, it should try to schedule it without other considerations.

The challenge here is how to perform this selection in a efficient way, ensuring that the selected preemptible instances are the less costly for the provider.

6.4. Preemptible Aware Scheduling

The proposed algorithm flow chart for scheduling preemptible instances is shown in Figure 6.1 and will be described next, taking into account that all the algorithms described in Chapter 4 are based on two complimentary phases: filtering and raking.

The filtering phase eliminates the hosts that are not able to host the new request due to its current state—for instance, because of a lack of resources or a VM anti-affinity—, whereas the raking phase is the one in charge of assigning a rank or weight to the filtered hosts so that the best candidate is selected.

Therefore, the first step is the filtering. In order to perform a filtering that is able to take into account preemptible instances, I propose to utilize two different states for the physical hosts:

h_f This state will take into account all the running VMs inside that host, that is, the preemptible and non preemptible instances.

h_n This state will not take into account all the preemptible instances inside that host. That is, the preemptible instances running into a particular physical host are not accounted in term of consumed resources.

Whenever a new request arrives, the scheduler will use the h_f or h_n host states for the filtering phase, depending on the type of the request:

- When a normal request arrives, the scheduler will use h_n .
- When a preemptible request arrives, the scheduler will use h_f .

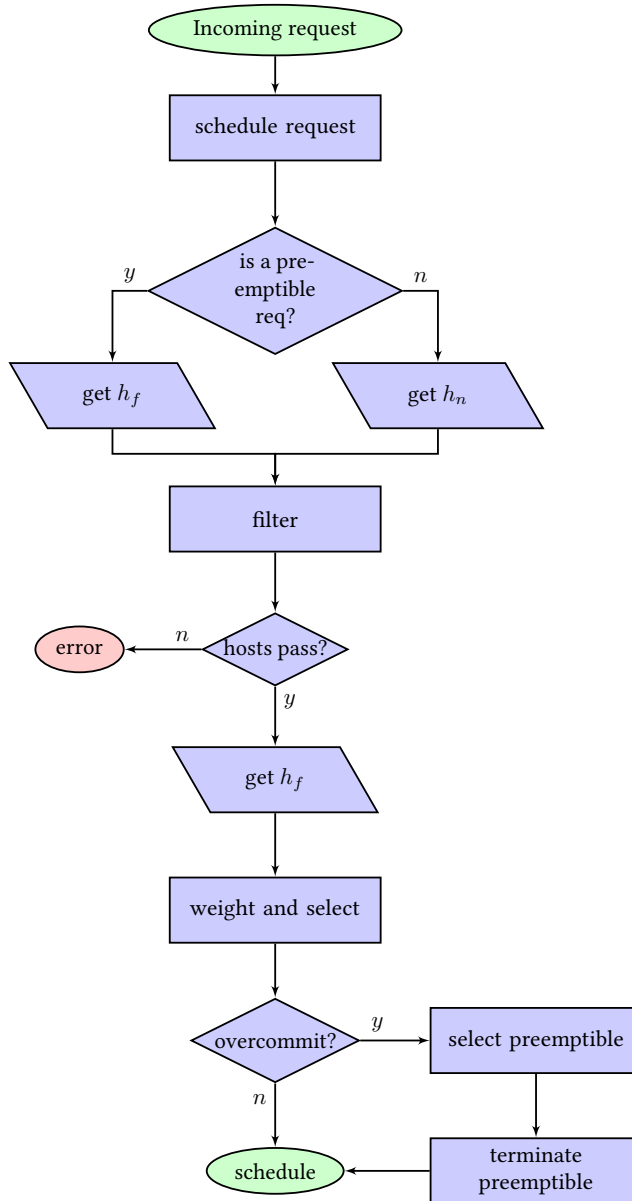


Figure 6.1: Preemptible Instances Scheduling Algorithm.

This way the scheduler ensures that a normal instance can run regardless of any preemptible instance occupying its place, so after this stage, the resulting list of hosts will contain all the hosts susceptible to host the new request, either by evacuating one or several preemptible instances or because there are enough free resources.

Once the hosts are filtered out, the ranking phase will start. However, in order to perform the correct ranking, it is needed to use the full state of the hosts, that is, h_f . This is needed as the different rank functions will require the information about the preemptible instances so as to select the best node.

The list of filtered hosts may contain hosts that are able to accept the request because they have free resources and nodes that would imply the termination of one or several instances. In order to discriminate between them the ranking function described in Algorithm 2 be implemented.

Algorithm 2 Ranking function detecting overcommit of resources.

```
1: function OVERCOMMIT RANK( $req, h_f$ )  
INPUT:  $req$ : user request  
INPUT:  $h_f$ : host state  
2:   if  $req.resources > h_f.free\_resources$  then  
3:     return  $-1$   
4:   end if  
5:   return  $0$   
6: end function
```

This function assigns a negative value if the free resources are not enough to accommodate the request, detecting an overcommit produced by the fact that it is needed to terminate one or several preemptible instances. This ranking function needs to be prioritised accordingly (via a large multiplier), so that it is taken into account against other ranks.

However, this basic function only establishes a rank between machines that can host the new request by killing a preemptible instance and machines that already have the resources available. In the case that it is needed to terminate some instances, this function does not establish any rank between them. In this case other rank functions need to be created, depending on the business model implemented by the provider. For instance, commercial providers tend to charge by complete periods of 1 h, so partial hours are not accounted. A ranking function based in this business model can be expressed as

Algorithm 3, ranking hosts according to the preemptible instances running inside them and the time needed until the next complete period.

Algorithm 3 Ranking function based on 1 h consumption periods.

```

1: function PERIOD RANK(req, hf)
INPUT: req: user request
INPUT: hf: host state
2:   weight ← 0
3:   for all instance ∈ get_instances(hf) do
4:     if (is_spot(instance) then
5:       if (instance.run_time mod 3600) > 0 then
6:         weight ← weight + instance.run_time mod 3600
7:       end if
8:     end if
9:   end for
10:  return −weight
11: end function

```

At this stage, the scheduler will have the best candidate for the new request. However, it is still needed to select those preemptible instances that need to be cleared out from that host, if any. Therefore, this design maintains the two phases, filtering and weighing with some modifications, and adds a third phase, so as to terminate the preemptible instances if needed.

This last phase will perform an additional raking and selection of the candidate preemptible instances inside the selected host, so as to select the less costly for the provider. This selection should leverage a similar ranking process, performed on the preemptible instances, considering all the preemptible instances combination and the costs for the provider, as shown in Algorithm 4.

6.5. Implementation and Evaluation

In order to evaluate the proposed solution I have implemented a modified OpenStack Filter Scheduler based on the described functionality, as shown in Algorithm 5. Moreover, I have implemented the ranking functions described in Algorithm 2 and Algorithm 3 as weighters, using the OpenStack terminology and the Scheduler design from Section 4.3. It is worth to notice that due to the modular design of the OpenStack Scheduler, it is

Algorithm 4 Preemptible instance selection and termination.

```
1: procedure SELECT AND TERMINATE(req, hf)
INPUT: req: user request
INPUT: hf: host state
2:   selected_instances  $\leftarrow$  []
3:   for all instances  $\in$  get_all_preemptible_combinations(hf) do
4:     if  $\sum(\textit{instances.resources}) > \textit{req.resources}$  then
5:       if  $\textit{cost}(\textit{instances}) < \textit{cost}(\textit{selected_instances})$  then
6:         selected_instances  $\leftarrow$  instances
7:       end if
8:     end if
9:   end for
10:  TERMINATE(selected_instances)
11: end procedure
```

possible to define any other weighting function depending on the needs of the Resource Provider (RP).

The scheduler has been also modified so as to introduce the additional select and termination phase (Algorithm 4). This phase has been implemented following the same same modular approach as the OpenStack weighting modules, allowing to define and implement additional cost modules to determine which instances are to be selected for termination.

As for the cost functions, I have implemented a module following Algorithm 3. This cost function assumes that customers are charged by periods of 1 h, therefore it prioritizes the termination of spot instances with the lower partial-hour consumption (i.e. if we consider instances with 120 min, 119 min and 61 min of duration, the instance with 120 min will be terminated).

This has been implemented for the OpenStack Kilo (2015.2) version, and this was deployed on top of the testbed described in Appendix A.3.

6.5.1. Evaluation

The purpose of this evaluation is to ensure that the proposed algorithm is working as expected, so that:

- The scheduler is able to deliver the resources for a normal request, by terminating

Algorithm 5 Preemptible Instances Scheduling Algorithm.

```

1: function SELECT DESTINATIONS(req)
INPUT: req: user request
2:   host  $\leftarrow$  SCHEDULE(req)
3:   if host is overcommitted then
4:     SELECT AND TERMINATE(req,host)
5:   end if
6:   return host
7: end function

8: function SCHEDULE(req)
INPUT: req: user request
9:    $H_f \leftarrow$  host_states(full)
10:   $H_p \leftarrow$  host_states(partial)
11:  if is_spot(req) then
12:     $H_{filtered} \leftarrow$  filter(req,  $H_f$ )
13:  else
14:     $H_{filtered} \leftarrow$  filter(req,  $H_p$ )
15:  end if
16:   $H_{weighted} \leftarrow$  weight(req,  $H_f$ )
17:  best  $\leftarrow$  select_best( $H_{weighted}$ )
18:  return best
19: end function

20: procedure SELECT AND TERMINATE(req, hf)
INPUT: req: user request
INPUT: hf: host state
21:  selected_instances  $\leftarrow$  []
22:  for all instances  $\in$  get_all_preemptible_combinations(hf) do
23:    if  $\sum$  instances.resources > req.resources then
24:      if cost(instances) < cost(selected_instances) then
25:        selected_instances  $\leftarrow$  instances
26:      end if
27:    end if
28:  end for
29:  TERMINATE(selected_instances)
30: end procedure

```

one or several preemptible instances when there are not enough free idle resources.

- The scheduler selects the best preemptible instance for termination, according to the configured policies by means of the scheduler weighters.

I have executed two different batches of tests: using the same VM size for all the requests, and using different VM sizes, according to Table 6.1. The tests were executed in the infrastructure described in Appendix A.3, using only four of the configured compute nodes for the sake of simplicity.

Name	vCPUs	RAM (MB)	Disk (GB)
small	1	2000	20
medium	2	4000	40
large	4	8000	80

Table 6.1: Configured VM sizes.

6.5.1.1. Scheduling using same Virtual Machine sizes

For the first batch of tests, I have considered same size instances, to evaluate if the proposed algorithm choses the best physical host and selects the best preemptible instance for termination. I generated requests for both preemptible and normal instances—chosen randomly—, of random duration between 10 min and 300 min, using an exponential distribution [168] until the first scheduling failure for a normal instance was detected.

The compute nodes used (Appendix A.3) have 16 GB of RAM and eight CPUs (Appendix A.3). The VM size requested was the *medium* one, according to Table 6.1, therefore each compute node could host up to four VMs.

I executed these requests and monitored the infrastructure until the first scheduling failure for a normal instance took place, thus the preemptible instance termination mechanism was triggered. At that moment I took a snapshot of the nodes statuses, as shown in Table 6.2 and Table 6.3. These tables depict the status for each of the physical hosts, as well as the running time for each of the instances that were running at that point. The shaded cells represents the preemptible instance that was terminated to free the resources for the incoming non preemptible request.

Considering that the preemptible instance selection was done according to Algorithm 4 using the cost function in Algorithm 3, the chosen instance has to be the one

Host	Instances		Preemptible Instances	
	ID	Run Time (min)	ID	Run Time (min)
host-A	A1	272	AP1	96
	A2	172	AP2	207
host-B	B1	136	BP1	71
	B2	200	BP2	91
host-C	C1	97	CP1	210
	C2	275	CP2	215
host-D	D1	16	DP1	85
			DP2	199
			DP3	152

Table 6.2: Test-1, preemptible instances evaluation using the same VM size. The highlighted cell indicates the terminated instance.

with the lowest partial-hour period. In Table 6.2 this is the instance *BP1*. By chance, it corresponds with the preemptible instance with the lowest run time.

Host	Instances		Preemptible Instances	
	ID	Run Time (min)	ID	Run Time (min)
host-A			AP1	247
			AP2	463
			AP3	403
			AP4	410
host-B	B1	388	BP1	344
	B2	103	BP2	476
host-C	C1	481	CP1	181
	C2	177	CP2	160
host-D	D1	173	DP1	384
			DP2	168
			DP3	232

Table 6.3: Test-2, preemptible instances evaluation using the same VM size. The highlighted cell indicates the terminated instance.

Table 6.3 shows a different test execution under the same conditions and constraints. Again, the selected instance has to be the one with the lowest partial-hour period. In Table 6.3 this corresponds to *CP1*, as its remainder is 1 min. In this case this is not the

preemptible instance with the lowest run time (being it *CP2*).

6.5.1.2. Scheduling using different Virtual Machine sizes

For the second batch of tests I requested instances using different sizes, always following the sizes in Table 6.1. Table 6.4 depicts the testbed status when a request for a *large* VM caused the termination of the *AP2*, *AP3* and *AP4* preemptible instances. In this case, the scheduler decided that it was terminated these three instances caused a smaller impact on the provider, as the sum of their 1 h remainders (55) was lower than any of the other possibilities (58 for *BP1*, 57 for *CP1*, 112 for *CP2* and *CP3*).

Host	Instances			Preemptible Instances		
	ID	Run Time (min)	Size	ID	Run Time (min)	Size
host-A				AP1	298	large
				AP2	278	medium
				AP3	190	small
				AP4	187	small
host-B	B1	494	large	BP1	178	large
host-C				CP1	297	large
				CP2	296	medium
				CP3	296	small
host-D	D1	176	medium			
	D2	200	medium			
	D3	116	large			

Table 6.4: Test-3, preemptible instances evaluation using different VM sizes. The highlighted cell indicates the terminated instances.

Table 6.5 shows a different test execution under the same conditions and constraints. In this case, the preemptible instance termination was triggered by a new VM request of size *medium* and the selected instance was *BP3*, as *host-B* will have enough free space just by terminating one instance.

6.6. Conclusions

I have proposed a preemptible instance design that does not modify substantially the scheduling algorithms, but that it is able to effectively terminate preemptible instances

Host	Instances			Preemptible Instances		
	ID	Run Time (min)	Size	ID	Run Time (min)	Size
host-A	A1	234	large	AP1	172	medium
	A2	122	medium			
host-B				BP1	272	large
				BP2	212	medium
				BP3	380	small
host-C	C1	182	small			
	C2	120	medium			
	C3	116	large			
host-D				DP1	232	large
				DP2	213	small
				DP3	324	medium
				DP4	314	small

Table 6.5: Test-4, preemptible instances evaluation using different VM sizes. The highlighted cell indicates the terminated instances.

whenever it is needed. The modular rank and cost mechanisms allows to define and implemented the desired Resource Provider (RP) policies. In this initial implementation and evaluation we have considered that the RP preemptible instances business model is based on charging customers by complete 1 h periods.

The presented design has been proposed for inclusion in the OpenStack Scheduler¹ and a prototype has been implemented in the IFCA Cloud Test Infrastructure (Appendix A.3) so as to be evaluated under real-world conditions by some candidate use-cases that require opportunistic usage.

¹Á. López García. *OpenStack Spot Instances Support Specification*. URL: <https://blueprints.launchpad.net/nova/+spec/spot-instances>.

III

**CLOUD FEDERATION AND
INTEROPERABILITY**



**Open Standards for Interoperable
and Federated Clouds**

This chapter will review the open challenges when building an interoperable science cloud federation and the enabling standards that can be used to leverage the construction of such a federation of Infrastructure as a Service (IaaS) resource providers. We will focus on an *horizontal federation* between different IaaS providers, therefore a *vertical federation* spanning several layers is out of the scope of this analysis.

In Section 7.1 I present an introduction to the topic and the related work in the area. In Section 7.2 I review the related work in the area. In Section 7.3 I present the biggest challenges that an interoperable cloud federation must assess. In Section 7.4 I focus on the existing and raising standards and how they can be used to tackle the problems presented in Section 7.3. Finally, the conclusions are presented in Section 7.5.

7.1. Introduction to Federation

The IT field, and in particular the cloud area is always evolving at a fast pace. Over the last years, a large number of commercial cloud providers have emerged in the market. Each of those vendors tries to differentiate their infrastructure from their competitors offering added value features on their resources. This has led to a situation where several closed and proprietary interfaces have evolved over the time, some being considered as *de-facto* standards by the industry. The resulting scenario includes infrastructures using different solutions that are actually incompatible and not interoperable. Vendor lock-ins are often considered a desirable feature by commercial providers, as a way of keeping users attached to their resources, but it is perceived negatively by cloud users and customers [188].

More recently, several open source Cloud Management Frameworks (CMFs) have appeared in the cloud ecosystem. Some of them decided to adopt those industry *de-facto* standards as their main interface, whereas others have built their own open interfaces. These decisions, instead of agreeing on a common and open interface are contributing to adding more entropy and heterogeneity. Users willing to exploit several cloud infrastructures face a discouraging panorama.

As the cloud computing paradigm is maturing and it is growing in heterogeneity, cloud interoperability and federation is becoming more important by all the involved stakeholders: that is industry, academia, providers, developers, users and customers. Chapter 3 introduced in its Section 3.1.6 the necessity for federated and interoperable

Science Clouds, but this is not a topic exclusively related to scientific environments. It is needed to provide mechanisms to provide the incorporation and interoperability of different systems [189].

In spite of introducing an additional complexity, a federated infrastructure brings some unique benefits, such as the following:

- There is a collaboration between Resource Providers, so that they can cope with limitations in their infrastructure, moving workloads to different providers when needed.
- Enables communities to access several resource providers in a seamless way, interacting with several providers with the same credentials, having uniformly accessible and structured principals.
- Makes possible to move computation near data, for cases when it is not possible to move data to computation, due to restrictive policies or due to the dataset size, making impractical to move data outside their original location.
- A Federated model, in contrast with a centralized one, creates a collaborative network of compute centers, promoting local economies, increasing and reusing expertise at multiple locations distributed geographically.

Federation and interoperability are nowadays considered as one of the main pressing issues towards cloud computing adoption [190]. The vendor lock-ins that currently exist are perceived negatively by users. Even if scientists are considering more and more Software as a Service (SaaS) solutions, the vendor lock-in still persists, as is not desirable to remain captive one vendor infrastructure. The cloud computing ecosystem, with strong industrial actors driving the developments, have promoted indirectly a panorama where de-facto, industry-driven standards have dominated the cloud landscape for years.

The cloud computing is still considered as an emerging technology, although it is now leaving its infancy phase. Standardization in the cloud was not considered an urgent matter to be addressed. An excessive focus on standardization and interoperability in the early stages of a technology can hinder its evolution, not leaving much room for the innovation needed on the early stages of the technology.

7.2. Related work

Nowadays there is a growing interest and research into cloud federation, interoperability and standardization. Building and defining frameworks for cloud interoperability is becoming therefore a topic that is gaining more and more momentum [191, 192, 193, 194, 84].

Political and government bodies such as the European Commission have stated their position towards the need for federation and interoperability of clouds [190], together with the promotion of Open Standards in in clouds for science and public administration [45, 64], with the aim of easing the aforementioned interoperability. The Open Science Cloud initiative [75] has outlined that interoperable, distributed and open principles should drive the evolution of Science Clouds as the key to success.

There are many non academic works regarding the need or lack thereof for a *cloud standard*. Authors tend to agree that there would be not such a unique standard to rule all the cloud aspects. Some preliminary work regarding the need of standards for the cloud has been done in this regard by Borenstein et al. [188] and Machado et al. [195].

G. Lewis [196] report tackles several standardization areas such as workload management, data and cloud management Application Programming Interfaces (APIs), concluding that there will be not a single standard for the cloud due to pressures and the influences of existing vendors. The author states that an agreement on a set of standards for each of the needed areas would reduce the migration efforts and enable the third generation of cloud systems.

Harsh et al. [197] work surveyed the existing standards for the management of cloud computing services and infrastructure within the Contrail project so as to avoid vendor lock-in issues and ensure interoperability. In the same line, Zhang et al. [198] performed a quite complete survey regarding Infrastructure as a Service access, management and interoperability, studying Open Virtualization Format (OVF), Cloud Data Management Interface (CDMI) and Open Cloud Computing Interface (OCCI) [198]. However, they have not entered into other details and challenges such as accounting or information discovery.

On top of those academic efforts, some open source CMFs have started to take into consideration the federation issues. There are development efforts aimed to make possible to federate different aspects of distributed cloud infrastructures to an extent:

-
- OpenStack [141] implements several levels of federation by the usage of cells and regions. The former allows to run a distributed cloud sharing the same API endpoint, whereas the latter is based on having separate API endpoints, federating some common services: OpenStack also allows the usage of a federated authentication mechanism [199], so that the identity service is able to authenticate users coming from trusted external services or from another identity service.
 - CloudStack [143] follows the same line and implements the concept of regions in their software.
 - OpenNebula [140] makes possible to configure several installations into a tightly integrated federation, sharing the same users, groups and configurations along several installations.
 - Eucalyptus [144] provides with identity and credential federation between several *regions*. Moreover, there is a partnership for offering interoperability for the interoperability between Eucalyptus private clouds and Amazon's Amazon Web Services (AWS) [200].

However, all of them rely on the fact of federating several instances of the same software stack (i.e. several OpenNebula installations, for instance), being impossible or difficult to federate disparate and heterogeneous infrastructures (e.g. an OpenStack installation together with an OpenNebula instance).

On top of that, there are a few prominent existing federated infrastructures, some of them being built on top of standards, others not. Some examples of standards-based federations are the EUBrazil Cloud Connect [201], whose middleware is being based on standards for interoperability [202]; and the European Grid Infrastructure (EGI) [60], that started as a federation of grid sites, took the strategic position of exploring and adopting a technology agnostic and based on open standards cloud [84] into their services portfolio.

7.3. Cloud Federation Open Challenges

Cloud federation goes beyond just making several clouds interoperable [203]. A federation shall enable the collaboration and cooperation of different providers in delivering resources to the users when a single resource provider is not able to satisfy

the user demands, in a collaborative way. Therefore, on top of the interoperability and portability issues, there are several challenges that any federation must tackle.

7.3.1. On Uniform Access and Management

One of the first obstacles that a heterogeneous cloud federation has to overcome is the lack of a unified cloud interface. Evolving from commercial cloud providers, each middleware implements their own —proprietary or not— interface. Some open CMFs implement an AWS Elastic Compute Cloud (EC2) [204] compatibility layer since it was considered as the *de-facto* standard for the cloud since its launch in 2006.

The adoption of the EC2 API could make two different CMFs being interoperable, but it presents several obvious drawbacks. The most prominent one is the fact that the API is proprietary and it is subject to change without prior advise by the original vendor. This will render into incompatibilities between providers and CMFs other than the original creator of the API, Amazon in this case. Implementers of proprietary interfaces need to keep aligned with the reference implementation, and are forced to invest time in following the modifications so that they ensure that its implementation remains compatible.

Secondly, its usage has as a consequence that it indirectly introduces a vendor lock-in since users may be captive into one infrastructure if the vendor decides to change its API from one day to another. Moreover, it is not a RESTful API, making difficult for developers to create applications that interact with it as they have to learn the semantics of the protocol being used instead of the well known REST architectural style.

7.3.2. On Portability

Cloud computing leverages virtualization technologies to abstract the resources being offered to the users. Several virtualization hypervisors (such as Xen, KVM, VMWare, Hyper-V) exist in the market, and each cloud provider will be using the one of its choice. Moreover, recently Operating System (OS) level virtualization (that is, container-based such as LXC, OpenVZ and Docker) have entered the game and they are being more and more adopted by the providers.

This situation renders difficult the migration of one virtual appliance prepared to be executed in one cloud provider using one virtualization technology to another provider

with a different underlying technology. Moreover, the underlying technology is hidden and abstracted from the users by the CMFs, hence even if they had the technological skills to prepare and modify a virtual appliance to be executed on another hypervisor they would have found difficulties in doing so. Porting one Virtual Machine to another hypervisor may require access to consoles and debugging output that cloud providers may be reluctant to provide.

7.3.3. On Authentication and Authorization

The delivery of an homogeneous authentication and authorization via a federated identity management system in distributed environment is a challenging topic [199, 205] not exclusive to cloud computing. As a matter of fact, cloud computing is just another player in the game. Such system should facilitate flexible authentication methods and federated authorization management.

The lack of a federated identity management systems makes difficult to manage the users globally at the federation, that is, specifying what resources a user is able to access or globally disabling a user becomes a challenging task.

Moreover, this challenge has two additional faces as it affects both the users and the resource providers.

- From the user's perspective, they are forced to cope with the burden of managing several credentials and identities. Moreover, client tools need to deal with them as well, identifying that *identity A* should be used against *provider A* but not *provider B*.
- It increases the management complexity for the resource providers. If there is not such a federated identity management system, users are to be managed manually, thus incurring in a tremendous overhead.

7.3.4. On Information Discovery

Once a federation is established, the next challenge is how users are able to discover what resources and capabilities are offered by the federation so that they can consume them.

This information may be exposed to the users via each of the middleware native APIs by each of the resource providers participating in the federation, but it is not structured

in an homogeneous way so that clients can fetch that information and help users making a decision.

7.3.5. On Accounting and Billing

In federated infrastructures it is often required to keep track of resource usage for each user and group at every individual provider, so that this information is shared and aggregated at the federation level and users are accounted properly. This aspect is tightly coupled with the federated identity management systems, as users need to be unambiguously identified throughout the infrastructure. Currently, each CMF and provider may have their own accounting method, but there is no common way for accessing and/or aggregating that information at the federation level.

7.4. Federation Enabling Standards

It may be possible to obtain interoperability without the usage of Open Standards [206], but it is arguably a more logical way to develop an interoperable federation based on them.

There is not a unique cloud standard to rule all of the aspects regarding clouds [195], neither there is such a federation standard. Nevertheless, there are several well established standards covering some of the open issues described in Section 7.3, developed prior to the raise of cloud computing and that can be simply reused, adapted or updated to fill in the needed gaps. On top of them there is a number of emerging standards being developed specifically to cover more specific cloud computing topics.

It is the combination of both —existing and emerging standards— they key to solve the federation and interoperability issues described in Section 7.3, as it will be described through the rest of the section.

7.4.1. Uniform Access and Management

Several organizations and standardization bodies have started working from the early stages of cloud computing trying to build standards for cloud management. Currently, the most prominent examples regarding IaaS computing and storage management are Open Cloud Computing Interface (OCCI), Cloud Infrastructure Management Interface

(CIMI), Topology and Orchestration Specification for Cloud Applications (TOSCA) and Cloud Data Management Interface (CDMI):

OCCI The Open Grid Forum (OGF) [207] has proposed the OCCI [208, 209, 210], focusing on facilitating an interoperable access and management of IaaS cloud resources. OCCI offers different renderings over the Hiper Text Transfer Protocol (HTTP) protocol, leading to a RESTful API implementation.

CIMI The CIMI [211] is a proposal from the Distributed Management Task Force (DMTF) [212] that has been recently registered as an ISO/IEC standard [213]. CIMI targets the management of the life-cycle of the IaaS resources, offering a RESTful API over the HTTP protocol with various renderings.

TOSCA The TOSCA standard [214] is a specification from the Organization for the Advancement of Structure Information Standards (OASIS) [215]. TOSCA provides a language to describe composite services and applications on a cloud, as well as their relationships (i.e. the topology), makes also possible to describe its operational and management aspects (i.e. its orchestration). Although TOSCA is at a higher level than simply managing the IaaS resources—it is more focused on the orchestration of the resources—it should be considered as a complementary standard for the management of the resources.

CDMI The Storage Networking Industry Association (SNIA) has proposed CDMI [216], also recently registered as an ISO/IEC standard [217], defining an interface to perform different operations (creation, retrieval, update and removal) on data stored on a cloud.

7.4.2. Portability

Open Virtualization Format (OVF) [218] is a standard developed by the DMTF for packaging and describing a Virtual Appliance (VA), comprised of an arbitrary number of Virtual Machine Images (VMIs) in a portable and vendor neutral format. An OVF package contains a XML description (e.g. hardware configuration, disks used, network configuration, contextualization information, etc.) of each component of the VA.

7.4.3. Authentication and Authorization

There is a large number of standards that can be used for authentication and authorization. The implementation and adoption of one technology or another will eventually depend on the infrastructures that are going to be federated, and there will be no silver bullet that will fit all of the existing infrastructures. Authentication and Authorization sometimes imply policy issues that are out of the scope of the standardization efforts.

The X.509 Public Key Infrastructure [219] has been used for authentication in the grid world via the Grid Security Infrastructure (GSI), based on X.509 certificate proxies [220]. Authorization is done by embedding Attribute Certificates (AC) into the proxy, containing assertions about the user. The most notable service is the Virtual Organization Membership Service (VOMS) [89], being used in several cloud infrastructures [221]. However, direct use of X.509 certificates is not considered being user-friendly—due to the management burden—and an inflexible—for instance, it is difficult to perform a delegation of trust—solution in spite of being settled on several distributed infrastructures over the years.

The OASIS Security Assertion Markup Language (SAML) [222] is built in X.509 and defines a way to define authentication and attribute assertions in XML. Shibboleth [223] is an implementation of SAML and is focused on the federation of resource providers with different authentication and authorization schemes. Several projects have started looking at SAML and Shibboleth [224, 225] as a promising way to provide access to distributed infrastructures, although they have not substituted the direct use of X.509 certificates yet.

OAuth 1.0 [226] and 2.0 [227] is an IETF open standard for authorization, providing delegated access to some resources on behalf of the resource owner. OAuth has not been designed for authentication, therefore OpenID Connect (OIDC) [228] has been developed as an authentication layer on top of OAuth 2.0. ORCID [229] is a popular author registry service in the academic and research world, can be used with OAuth2 and has on its roadmap to become an OpenID Connect provider.

7.4.4. Information Discovery

Information discovery is a problem present in other federated computing paradigms such as Grids. The Grid Laboratory Uniform Environment (GLUE) Schema—in its versions

1.x [230] and 2.0 [231]— has been designed by the OGF in order to create an information model relying on the knowledge and experience from the operations of several large Grid infrastructures.

The current GLUE 2.0 specification [231] only defines a conceptual model. It makes possible to publish, separately from the standard, concrete data model profiles that will dictate how the information is generated and used for in concrete implementation, infrastructure, etc. Therefore, the OGF GLUE 2.0 schema is a good candidate for publishing information relative to cloud infrastructures.

7.4.5. Accounting

As with the information discovery, the accounting problem is a problem that has been already tackled in the grid. The OGF Usage Record (UR) 2.0 [232] defines a common format to share and exchange basic accounting data, coming from different providers and different resources. It supersedes and integrates the different resource usage records that leveraged the previous UR 1.0 in the various infrastructures that implemented it.

The OGF UR does not specify how the records should be exploited (e.g. how they should be exported, used, aggregated, summarized, etc.) or transported. Examples on how the UR is used exist in projects such as RESERVOIR [233] and infrastructures such as EGI [60].

7.5. Conclusions

In this chapter I have presented the existing challenges that need to be tackled for building an interoperable federation of cloud providers and I have surveyed the existing and arising standards that can be used to solve those problems. Current Cloud Management Frameworks (CMFs) should adopt these existing standards for the functionality they are offering, so as to avoid vendor lock-in issues and to ensure a that proper interoperability is delivered to the users.

Cloud federation involves a lot of different areas and challenges —management, authentication, accounting, interoperability, etc—, therefore there is not a unique standard for it. However, there is a set of settled and emerging standards that can cover all the federation aspects and their problematics, as summarized in Table 7.1.

Challenge	Enabling Standard
Uniform access and management	OGF OCCI [208, 209, 210]
	DMTF CIMI [211]
	OASIS TOSCA [214]
Portability	DMTF OVF [218]
Authentication and authorization	OASIS SAML [222]
	OpenID Connect [228]
	X.509 [219]
Information discovery	OGF GLUE [230, 231]
Accounting	OGF UR [232]

Table 7.1: Summary of enabling standards.

The European Commission is encouraging the usage of Open Standards in its "European Interoperability Framework for pan-European eGovernment Services" [45]. Similarly, the United Kingdom Government provided similar set of principles, adopted in 2014 [234]. Other European initiatives, such as the Open Science Cloud [75], are also promoting the usage of Open Standards. Cloud federations must take into account these recommendations and they should promote its usage as the path to a successful federation and interoperability.

8

An Implementation of an Open Standard for the Cloud

*Part of this chapter will be published as: **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “OpenStack OCCI Interface”. In: SoftwareX (2016). ISSN: 2352-7110. DOI: 10. 1016/ j. softx. 2016. 01. 001 (accepted paper).*

In the following Sections I present an implementation of the Open Grid Forum (OGF)'s Open Cloud Computing Interface (OCCI) for OpenStack, namely `ooi` [236], promoting interoperability with other OCCI-enabled Cloud Management Frameworks (CMFs) and infrastructures. `ooi` focuses on being non-invasive with a vanilla OpenStack installation, not tied to a particular OpenStack release version.

8.1. The Open Cloud Computing Inteface (OCCI)

The OGF has proposed the OCCI [237] as an open standard defining a RESTful Application Programming Interface (API) for managing cloud resources, developed as a joint effort between industry and academia.

OCCI has been one of the first standards in the cloud ecosystem, providing the foundations for basic management tasks in Infrastructure as a Service (IaaS) providers and it can be easily extended easily so as to provide additional functionality. OCCI is a standard relevant for both cloud users and cloud providers as a way to provide an interoperable infrastructure, removing any kind of vendor lock-in.

The specification is a set of complementary documents divided into three categories: the OCCI Core, the OCCI Renderings and the OCCI Extensions. At the time of writing this document the current version of the standard is OCCI 1.1, with OCCI 1.2 version being currently under development.

OCCI Core This is a single document [209] defining the OCCI Core abstract model.

This model can be interacted with the renderings and is expanded by the OCCI extensions.

OCCI Renderings The OCCI Renderings describe how the OCCI Core model should be rendered. The current OCCI Hiper Text Transfer Protocol (HTTP) Rendering specification [208] defines how to interact with the OCCI core model and its extensions over a HTTP protocol based RESTful API. Multiple and different renderings may interact with the same instances of the OCCI Core protocol, thus not being limited to use a concrete rendering.

OCCI Extensions These specifications describe additions to the OCCI core model. The OCCI Infrastructure specification [210] contains the extension for the IaaS domain,

defining the needed resource types, attributes and actions that can be taken on each resource type.

Mixins are used to modify a particular OCCI resource, by associating one or several to a particular resource instance. Besides the ones defined within the standard, there are some Mixin extensions that have been developed so as to add additional functionality, and that are currently widely adopted.

Contextualization Extension Contextualization is the process of installing, configuring and preparing software upon boot time on a pre-defined virtual machine image. This Mixin extension allows to pass some data to the instance [238] that can be further fetched from inside the virtual machine by a software such as cloud-init [239] or Flamingo [240].

Key Pair Extension This Mixin extension allows users to inject a Secure Shell (SSH) public key for the authenticated access to the provisioned Virtual Machine (VM) [238, 241].

8.2. Motivation and significance

Currently some OCCI implementations already exist for several cloud vendors or in the form of general frameworks that can be extended with several backends. In order to get an OCCI-enabled Openstack [141] deployment, we considered two candidates: rOCCI and OCCI-OS. Other implementations exist, but either they do not have recent activity in their codebase or they are too general frameworks that needed a lot of integration efforts (for instance for the authentication and authorization parts).

rOCCI [194] is one of the most notable projects implementing OCCI. It is a framework written in Ruby that aims to improve interoperability in the cloud by delivering an OCCI implementation that can be used by both at the server and at the client side. The rOCCI-server component makes possible to add an OCCI interface to some existing cloud stacks and vendors via one of the existing configurable backends, such as OpenNebula [242], Apache CloudStack [143], VMware [243] and Amazon Elastic Compute Cloud (EC2) (EC2) [244]. It stands as a standalone server (rOCCI-server) that proxies the requests to the underlying CMF. The rOCCI-cli on the

other hand is the client component of rOCCI, making possible to interact with any OCCI-enabled framework.

OCCI-OS [245] is an implementation of OCCI for OpenStack [245], leveraging the Python Service Sharing Facility (pyssf) [246]. It consists on a new Web Server Gateway Interface (WSGI) application that uses the internal OpenStack APIs.

rOCCI-server could be adapted to be used over an OpenStack installation, but the fact of being written in Ruby is an obstacle for reusing the existing OpenStack modules (e.g. authentication) already available.

On the other hand, OCCI-OS' WSGI application speaks directly to the OpenStack internal APIs. These APIs are not versioned and can be subject to change at any point in the development, leading to incompatibilities between the OCCI modules and the different OpenStack versions. As a result, the need of several OCCI-OS releases, each one aligned with its corresponding OpenStack API version, is a must. Changes in the internal OpenStack APIs happen even between minor releases, making impractical to update the code for each new version. Making OCCI-OS use the public APIs instead involves a complete refactorization of its codebase, as it leverages all the internal backends to accomplish the desired actions.

As an aim to overcome these architectural issues, I present in this chapter `ooi`, a Python-based application designed to be easily integrated with the OpenStack core components.

8.3. Software Description

8.3.1. Foreword on WSGI

The Python Web Server Gateway Interface (WSGI) standard [247] proposes an interface between web servers and Python web applications so that it is possible for an application to handle HTTP requests using Python code. Among other things, it defines the WSGI application, server and middleware.

- The WSGI application object receives a representation of the HTTP request, processes it and returns a response that will be eventually sent back to the client.

- The WSGI server invokes the application for each request that is targeted to it. Therefore, an application receives the request from a server.
- The WSGI middleware receives a WSGI request, performs some logic on it, and sends it to the next WSGI middleware or application. Therefore, the WSGI middleware is seen as an application by a WSGI server, and as a server by a WSGI application.

It is then possible to chain several WSGI middleware together, each one adding some additional functionality before actually passing the request to the final application. This appears as an analogy with *pipes* on UNIX systems, thus often using the term *pipeline* to refer to this chain of WSGI middleware and applications.

Following this structure, the OpenStack native API is a WSGI application that leverages several of such middleware that perform additional functionalities like authentication (against the OpenStack Identity Component), and rate and size limiting, just to cite some.

8.3.2. Interacting with OpenStack

The OCCI standard defines the API as a boundary interface that acts as a frontend to the internal management APIs, as shown in Figure 8.1.

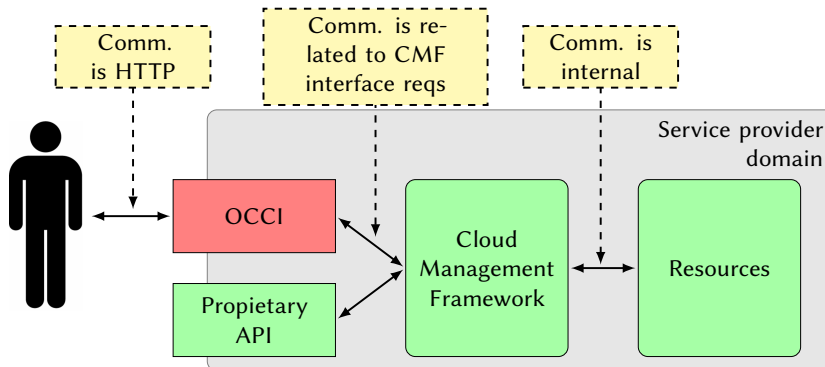


Figure 8.1: Proposed OCCI's place in a provider's architecture according to the standard. Boxes in yellow explain the type of communication being made, green depicts the CMF components, red the OCCI interface.

To interact with OpenStack, *ooi* leverages its public API interfaces [248] (Figure 8.2) rather than using the private API, as OCCI-OS [245]. This architecture decision is motivated by the fact that OpenStack public API is versioned, whereas its private interfaces are not; hence there is no contract to maintain its signature between OpenStack releases.

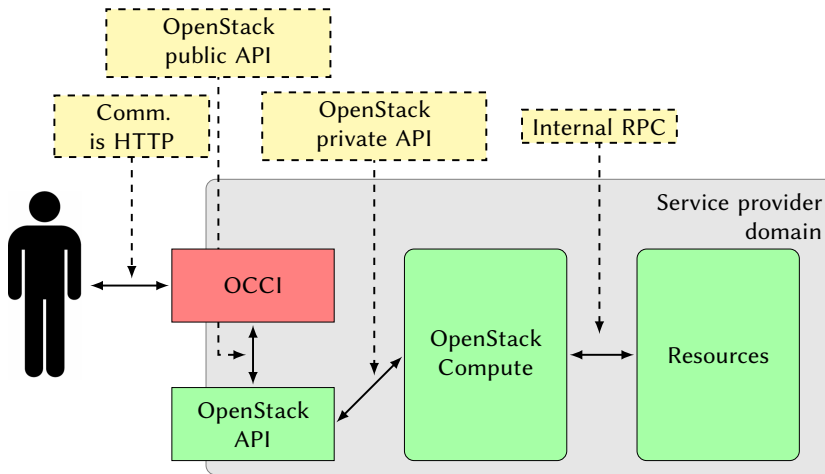


Figure 8.2: OCCI place in a provider's infrastructure, following *ooi*'s architecture. Instead of using the private APIs, OCCI requests are translated to native OpenStack requests. Boxes in yellow explain the type of communication being made, green depicts the OpenStack components, red the OCCI interface.

This fact causes that any application using the private, internal interfaces may need to be adapted throughout OpenStack releases. On the other hand, changes in the public REST API are versioned (each change increases the minor version of the API), and the same version is supported across several releases. A given version of OpenStack public API is not subject to functionality or backwards incompatible changes, since that kind of changes will increase the version number.

In this context, instead of implementing *ooi* as a WSGI application, it has been developed as a WSGI middleware that proxies the OCCI requests and translates it to an appropriate OpenStack request. This is a key aspect of *ooi*'s architecture design that, unlike other solutions (OCCI-OS [245]) does not appear as a standalone WSGI application that calls OpenStack internal interfaces but rather makes use of its public

API.

ooi's workflow is shown in Figure 8.3. The red shaded area represents the OCCI WSGI pipeline, whose components are depicted as gray boxes. As it is shown in the figure, each of the WSGI middleware process the request and perform some operation with it (for example, authentication), then they call the next application or middleware in the pipeline, until the request gets down to the final OpenStack API WSGI application. Then, the application will return a response, that will be processed back in reverse order by each of the WSGI middleware until it gets up to the WSGI server.

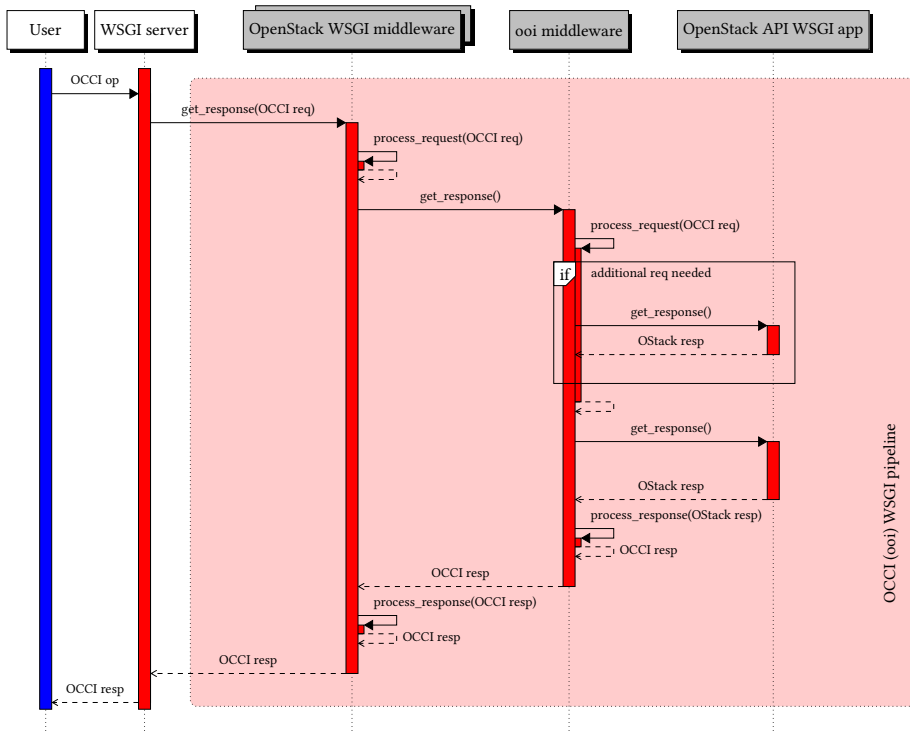


Figure 8.3: ooi processing pipeline. This figure illustrates the sequence diagram for processing an OCCI request. The red shaded area represents the WSGI pipeline, whose components are depicted with grayed boxes. Solid arrows represent operations or method calls, dashed arrows represent data types, *OCCI op* is a request for an OCCI operation, *OCCI req* represents an OCCI request type, *OStack resp* is an OpenStack response, *OCCI resp* is an OCCI response, *OpenStack WSGI middleware* are the preceding and unmodified OpenStack WSGI default middleware that are present in the pipeline.

Therefore, whenever a OCCI request arrives to the `ooi` middleware, this request is processed and translated into a new equivalent OpenStack request, based on its public API. Not only the request itself is translated, as the application Uniform Resource Locator (URL) is also modified so as to point to the current path regarding the OpenStack WSGI application route.

For instance, sending the OCCI request to create a server shown in Listing 8.1 via the POST method to the OCCI endpoint would be translated to the corresponding OpenStack V2.0 JavaScript Object Notation (JSON) request shown in Listing 8.2. If further information is needed so as to build the request, it is done transparently to the user. Whenever this transformation finishes, `ooi` passes down the corresponding OpenStack request to the OpenStack API WSGI application—the last step in the pipeline—and an OpenStack response is obtained. This response is processed again by the OCCI middleware, so that it is rendered back as a proper and valid OCCI response, and it continues its path upstream to the WSGI server.

Listing 8.1: OCCI 1.1 Server creation request.

```
1 Content-Type: text/occi
2 Category: compute; scheme="http://schemas.ogf.org/occi/infrastructure
   #"; class="kind"
3 Category: 8941da23-909d-4d7d-804f-54df629b6a86; scheme="http://
   schemas.openstack.org/template/resource#"; class="mixin"
4 Category: 4e9765ae-63ae-4b57-be86-495ce8fb9408; scheme="http://
   schemas.openstack.org/template/os#"; class="mixin"
```

Listing 8.2: OpenStack v2.0 Server creation JSON request, corresponding to Listing 8.1.

```
1 {
2   "server": {
3     "name": "server-1",
4     "imageRef": "4e9765ae-63ae-4b57-be86-495ce8fb9408",
5     "flavorRef": "8941da23-909d-4d7d-804f-54df629b6a86"
6   }
7 }
```

The development work that involves supporting new major releases of OpenStack public API is alleviated by `ooi`'s modular architecture, making possible to plug additional modules without modifying substantial parts of the code. Moreover, several OCCI endpoints, supporting different OpenStack API versions, can co-exist in a single `ooi`

installation allowing isolated environments to be used for different purposes. Thus e.g testing experimental API features can live together with the production endpoint without risks.

Currently, the supported OpenStack version is v2.1 [249]. However, it is possible to deploy `ooi` on top of the previous v2.0 API, since v2.1 is backwards compatible with the addition of strong API validation.

8.4. `ooi` Functionality

`ooi` implements the OCCI 1.1 standard as described in Section 8.2. It implements the OCCI Core Specification [209], the OCCI Infrastructure Extension [210] as well as the OCCI HTTP Rendering [208]. Additionally, two widely used extensions have been implemented: the contextualization and SSH credentials extensions.

During the development stages of `ooi` I have focused not only on following the standard, but also on remaining compatible with any other existing OCCI implementations. A comparison of the different OCCI implementations and the operations that can be performed in each of them is summarized in Table 8.1.

It is worth notice that OCCI does not mandate that all operations are actually supported by the respective backend (in this case OpenStack), therefore operations marked as N/A or marked as not implemented in Table 8.1 render the correct result as specified in the OCCI standard (that is, the HTTP 501 Not Implemented error code).

8.5. Performance Comparison

Even though it is not the original purpose of this new implementation, I found interesting to compare `ooi` performance against the existing OpenStack implementation—OCCI-OS— so as to ensure that our implementation does not impede the overall performance of the system.

For an accurate comparison to be made, I have deployed both `ooi` and OCCI-OS over the same OpenStack Infrastructure and performed some basic operations using `ooi`, OCCI-OS and the native OpenStack API. The utilized setup is described in Section A.4. In order to eliminate any potential overhead introduced by a client tool (such as authentication or data verification), the operations have been made directly to the API using the corresponding HTTP methods (i.e. GET, POST and DELETE in this case).

Query Interface			
	rOCCI	OCCI-OS	ooi
retrieve model	Y	Y	Y
filter	N	N	N
Infrastructure extension: compute			
query	Y	Y	Y
query and filter	N	N	N
create	Y	Y	Y
delete	Y	Y	Y
actions: start	Y	Y	Y
actions: stop	Y	Y	Y
actions: restart	Y	Y	Y
actions: suspend	Y	Y	Y
Infrastructure extension: network			
query	Y	N	Y
query and filter	N	N	N
create	P	N	Y
delete	Y	N	Y
attach to compute	Y	Y	Y
attach to compute	Y	Y	Y
detach from compute	Y	Y	Y
actions: up	N	N/A	N/A
actions: down	N	N/A	N/A
Infrastructure extension: storage			
query	Y	Y	Y
query and filter	N	N	N
create	Y	Y	Y
delete	Y	Y	Y
attach to compute	Y	Y	Y
detach from compute	Y	Y	Y
actions: online	Y	N/A	N/A
actions: offline	Y	N/A	N/A
actions: backup	Y	N/A	N/A
actions: snapshot	N	N	N
actions: resize	N	N/A	N/A
Contextualization extension			
contextualize compute	Y	Y	Y
SSH Key extension			
as an argument	Y	Y	Y
existing key	N	N	Y

Table 8.1: OCCI feature comparison of the several implementations. The lack of features in rOCCI is due to the backend, not rOCCI itself, since it refers to the OpenNebula backend as there is no OpenStack backend available. **N**: not implemented or available, **Y**: implemented, **P**: partially implemented, **N/A**: not applicable (backend does not support it).

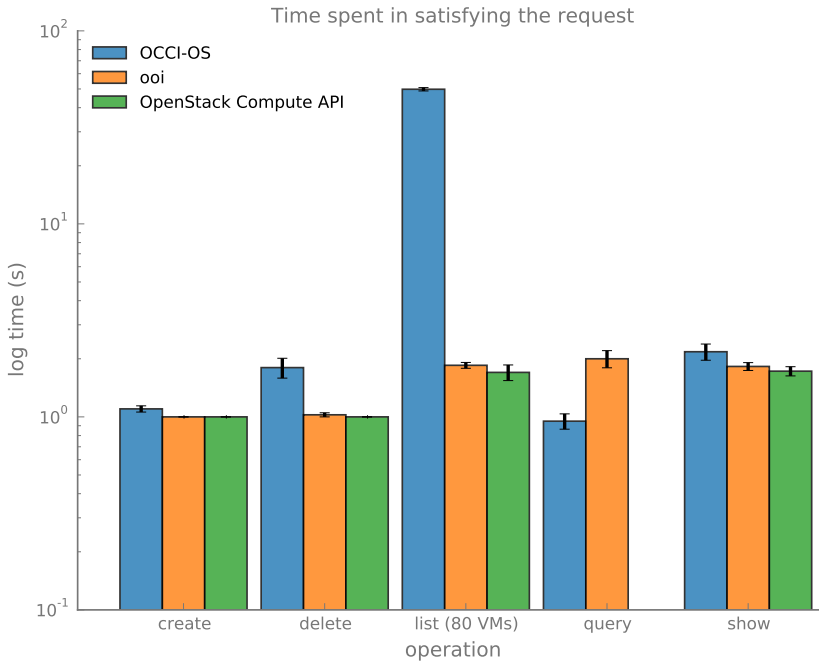


Figure 8.4: Performance comparison between the existing OpenStack implementations, using a logarithmic scale in the Y-axis. The listing of the running instances has been made against an infrastructure running 80 instances. Error bars shows statistical uncertainty. Note that the *query* operation for OpenStack is not applicable, as it is OCCI specific and there is no equivalent in OpenStack.

As it can be seen in Figure 8.4, the results for the most common operations are similar, with the exception of listing a large number of VMs (for this comparison I have deployed 80 VMs). It is worth to notice that there is no *query* operation or any equivalent in the native OpenStack Compute API, therefore it is not possible to show the results for such operation.

8.6. Conclusions

Standards in the cloud cannot evolve without a rich ecosystem of available implementations. In this context, Open Cloud Computing Interface (OCCI) is the reference standard for some federated cloud infrastructures, such as the European Grid Infras-

structure (EGI) Federated Cloud [84]. In such federated infrastructures, having a stable implementation of the OCCI interface for all of the Cloud Management Frameworks (CMFs) used —such as OpenStack— is a must. As I have stated in Section 8.2, the rOCCI framework has provided great OCCI support for several open source CMFs, there was a clear lack of a stable implementation of the OCCI standard for OpenStack.

ooi has been presented to the EGI Federated cloud as an alternative implementation for OpenStack with great acceptance, and is being adopted gradually by the OpenStack resource providers in the infrastructure.

Federating VO-based Cloud Infrastructures

*Part of this chapter has been published as: **Á. López García**, E. Fernández-del-Castillo, and M. Puel. "Identity Federation with VOMS in Cloud Infrastructures". In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science. 2013, pp. 42–48. ISBN: 978-0-7695-5095-4. DOI: 10.1109/CloudCom.2013.13. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753776>*

Chapter 7 described how the main drawbacks for a federated cloud infrastructure are a clear consequence of its heterogeneity: different resource providers, using different Cloud Management Frameworks (CMFs), that are neither designed nor adapted to interoperate and cooperate the way that any federated infrastructure is expected to do. In this context, the identity federation has been identified as one of the primary challenges to be addressed.

In this chapter I propose the usage of the Virtual Organization Membership Service (VOMS) [250] as an initial step to effectively federate the authentication and authorization in a multi-institutional cloud testbed, geographically distributed and with different CMFs. I have implemented this as a new authentication mechanism for the OpenStack CMF.

9.1. Identity Federation, Challenges and Problematic

In a heterogeneous ecosystem composed by independent Resource Providers (RPs)—with multiple CMFs and each one with multiple credential management systems—the identity and access policy management are challenging issues. Identity federation must be the cornerstone of any federation since it will create the foundations where the rest of the federated services will lean on.

From the user's perspective, identity federation is one of the core parts of any interoperable infrastructure (even when the users are not aware of that federation). If identity federation does not exist, users cannot access the infrastructure transparently, as they would need to be concerned with the specific details on how to access any given service provider, i.e. dealing with different CMFs and the tedious procedure of managing multiple identities and credentials.

Moreover, any federated infrastructure should leverage the establishment of relationships between the users and the resource providers. If a federated identity system does not exist, the resource providers need to define access policies individually for each of the users and groups willing to access the infrastructure, whereas the users need to negotiate their access policy and shares of the resources with each of the RPs. When the number of users increase this becomes an overwhelming task for the resource providers.

The problematic described above is common to any kind of federation. For example, the Shibboleth authentication system [223] has the concept of Shibboleth Federations making possible to establish agreements and trust relationships between organizations

planning to share resources. These methods are normally focused on institutional authentication, namely validating that a user is who says he is within an institution.

However, it is difficult to establish the user membership within a more abstract group —such as multi-institutional scientific collaborations— that is not bounded to any particular organization. This problem is not new, and has been already efficiently addressed in the Grid computing model by means of the Virtual Organization (VO) concept. A VO can be defined as a group of individuals and/or institutions emerged from a set of resource-sharing rules across dynamical and multi-institutional collaborations [12].

Resource providers can give access to their infrastructure on a VO-basis, meaning that a resource provider can make its resources VO-aware and create an agreement with a Virtual Organization on sharing and giving access to their resources, keeping under control what is shared and how it is shared. This way:

- A user belonging to a VO can get access to a given set of resources, using the same set of credentials, across several resource providers.
- A resource provider will be able to share its resources, with fine-grained control on what it is shared and not with a VO. The resource provider should have enough confidence in the VO users as they must sign an agreed acceptable usage policy.
- A resource provider does not need to manage individually each of the users in the VO, since this is leveraged to the VO administrators. The RP will trust the credentials endorsed by the VO.
- Once a VO —that is normally tied to a collaboration— obtains access to an infrastructure the RP can leverage to them the responsibility of tracking the internal usage of their resources, partitioning them, and prioritizing the access according to their member's needs.

9.1.1. Current Solutions

Cloud middleware developers are aware of the identity federation problematic and are tackling the problem at two different levels: establishing trust between compatible middleware and integration of external identity mechanisms.

In the first case, two sites, *site A* and *site B*, using the same or compatible cloud software can establish a trust between them, so that users from *site A* can use their credentials to access the resources of *site B*. This authentication relies on the trust of the credentials issued by one of the sites in the other site, normally by using some Public Key Infrastructure (PKI) mechanism so that a site can verify that the credentials are really issued by the other partner. This kind of authentication requires trust relationships to be established by every pair of sites in the federation and is limited to specific CMF with compatible identity services. These limitations and how some standards can alleviate them have been already discussed in Section 7.3.3 and Section 7.4.3 respectively.

Resource Providers also need to define authorization rules to access their resources once the user is authenticated. Grid infrastructures have handled this problem efficiently with the concept of VOs and the VOMS service. This service is currently the de-facto standard for VO management in EGI [60] and OSG [251] Grid infrastructures. Existing Grid middleware (such as EMI [252] and the Globus Toolkit [253]) include VOMS support as one of its core features.

VOMS allows VO managers to assign roles and groups to any given user within a VO. The VOMS server creates signed assertions with the user's VO attributes, thus RPs can trust these assertions and define the access rules for their resources according to the included attributes. The current VOMS implementation is based in X.509 Proxy Certificates [220], storing the VOMS attributes as Attribute Certificates (AC) along with the user proxy. Any VO-enabled service is therefore able to extract and verify these AC against the VOMS certificate, leveraging the VO membership management to its managers. A user that is removed from a VO, won't be able to request the VOMS attributes and won't be able to use the VO-aware services at all.

9.2. VOMS Support in OpenStack

Identity in OpenStack is managed by the OpenStack Identity Service, whose code-name is Keystone. It provides a common identity management for all the OpenStack services and serves as entry point to the other services. Keystone is organized as a group of internal modules —Identity, Token, Catalog and Policy— running as a Web Server Gateway Interface (WSGI) [247] application; each of them can be configured to use a specific back-end that adapts to different deployment scenarios.

The Identity service performs the credential validation and provides data and any associated meta-data about *users*, *tenants* and *roles*. Each user has some account credentials (consisting on a username and password) and should be associated to one or more tenants. The tenant is the unit of ownership in OpenStack, similarly to groups at the OS level. The role is a first-class piece of meta-data associated with a user-tenant pair, i.e. user *U* in tenant *T* has role *R*. Policy rules are defined using these three data types. The default Identity service back-end stores data persistently using any SQLAlchemy [254] compatible database engine.

The Token service is in charge of validating, creating and revoking the tokens that authenticate all the requests in OpenStack. The Catalog service provides an endpoint registry for discovery of other OpenStack services. Finally, the Policy service provides a rule-based authorization engine and the associated rule management tools.

The first action a user needs to perform to access the OpenStack services is to request an authentication token to Keystone. This is done by submitting a POST request to the `/tokens` with the user credentials included in the body of the request as a JavaScript Object Notation (JSON) document in the following form:

Listing 9.1: Keystone V2.0 password authentication JSON request.

```
1 {
2   "auth": {
3     "passwordCredentials": {
4       "username": "JohnDoe",
5       "password": "Top Secret"
6     }
7   }
8 }
```

If the `auth` dictionary contains a `tenantName` field a *scoped* token will be issued but, whereas this token will be *unscoped* if this field is missing. The difference with both types is that the former represents the authenticated user within a tenant. This token can be used therefore to authenticate the user with the rest of the cloud services. The latter—the unscoped token— does not have any tenant associated and can be only used against the Keystone service to **I**) enumerate the list of tenants the user has access to (useful for access discovery) or **II**) request an scoped tenant without the need to provide its user credentials again .

9.2.1. Keystone External Authentication

Keystone implements the concept of *External Authentication*. This functionality makes possible to perform authentication externally to the Keystone process and/or independently of the Identity back-end used: the Identity service considers that a user is successfully authenticated if the WSGI environment variable `REMOTE_USER` is set, otherwise the back-end serves as authentication handler. With this new architecture several authentication mechanisms can be used irrespective of the concrete back-end that still manages the users, tenants and roles data. The following two scenarios arise.

If Keystone is running as a WSGI process inside a HTTPD server with authentication capabilities, such as Apache, the web server modules can be used to authenticate any request. This allows to re-use the authentication mechanisms that are already implemented and tested in the most common web servers.

Additionally, if a developer needs to implement a particular authentication method not available in the web server, or if any extra operation before the token request should be performed, a WSGI filter middleware can be used. This filter is a module that can analyze the user request and conditionally perform the authentication if it is able to handle the credentials. The administrator can configure a set of filters that are executed before the token back-end consumes the request: if any of those correctly authenticate the user (and consequently set the `REMOTE_USER` variable), the Identity service will consider the user as valid. Figure 9.1 shows a sample call sequence for an authentication request in Keystone where there is a single WSGI filter configured.

9.2.2. Keystone VOMS Integration

The VOMS support is implemented as an external authentication handler in Keystone and executed as a WSGI of an HTTPD server enabled to use OpenSSL and configured to accept proxy certificates. The user authenticates against the HTTPD server with a VOMS proxy, and the server, after the proxy validation, includes the SSL information in the request environment. This information reaches the VOMS filter, that will try to authorize those requests that include the following JSON in the body:

Listing 9.2: Keystone V2.0 VOMS authentication JSON request.

```
1 {  
2   "auth": {
```

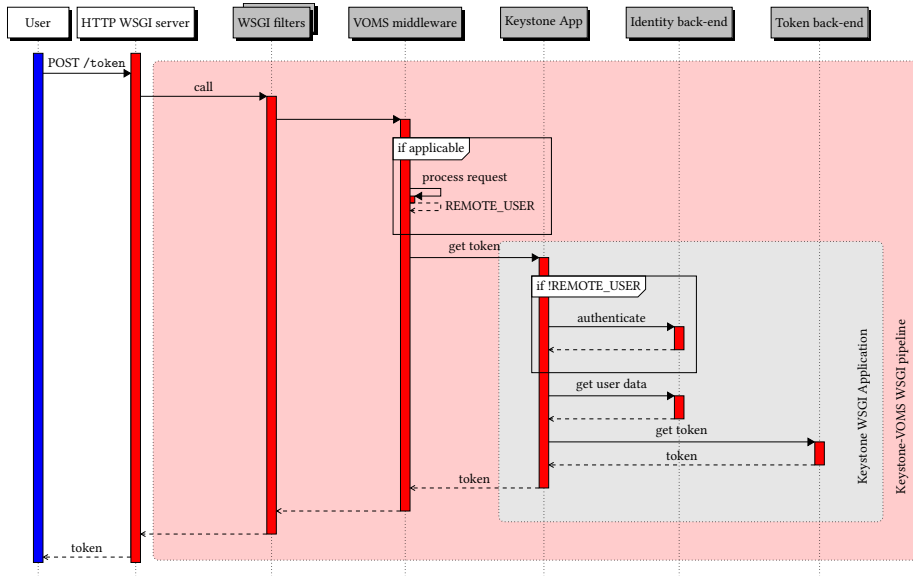


Figure 9.1: Authentication request in Keystone: **i)** The user performs a POST request to /token; **ii)** the WSGI server calls the configured WSGI pipeline; **iii)** the last WSGI middleware invokes Keystone, that uses the Identity back-end to authenticate the user if no REMOTE_USER is defined and to get user data from the back-end storage; **iv)** Token back-end issues the token; and **v)** is returned to the user.

```

3     "voms": true
4   }
5 }

```

The VOMS module invokes the VOMS library to check if the proxy is valid and if it is allowed in the server. As with any other services using VOMS, this operation uses .1sc files which contain the trust chain from the Certificate Authority (CA) to the VOMS server so no additional request is made against the VOMS servers. The server signature in clients' proxy is verified in accordance with this trust chain. Once the proxy is considered valid and allowed, the VOMS module considers the Distinguished Name (DN) of the proxy issuer as the user name. For the tenant, a JSON configuration file defines the mapping between VOMS attributes (VO name, VO groups, VO roles, etc.) and the local OpenStack tenants. Each mapping is defined as an entry in a dictionary of the following form:

Listing 9.3: VO and group JSON mapping.

```
1 "vname": {  
2   "tenant": "local_tenant"  
3 }
```

where `vname` is the Fully Qualified Attribute Names (FQANs) used by VOMS and `local_tenant` is the name of the tenant in the local OpenStack installation. For example, for the `dteam` VO, the file could be configured as:

Listing 9.4: sample VO and group JSON mapping.

```
1 {  
2   "dteam": {  
3     "tenant": "dteam"  
4   },  
5   "/dteam/NGI_IBERGRID": {  
6     "tenant": "dteam_ibergrid"  
7   }  
8 }
```

In the example above, the users of `dteam` VO in the `NGI_IBERGRID` group would be mapped to the `dteam_ibergrid` tenant, while any other `dteam` VO members would be matched to the `dteam` tenant.

It is worth to note that the mapping for a VO can vary between different providers. This can be seen as an issue since different tenant names can produce inconsistencies towards the user. Anyhow, this can be circumvented using the Keystone's ability to list the tenants that the user is able to get access to, using a *unscoped* token (as described in Section 9.2) so that the actual tenant will be selected from the obtained list.

The mapped local tenant must exist in advance for a user to be authorized. If the mapped tenant does not exist, the authorization will fail. The same applies for the user, with the particularity that the VOMS module is able to automatically create new users if enabled in the configuration. This option is disabled by default, but if the administrator wants to authorize all the users belonging to a particular VO to access the infrastructure, it should be enabled. Once a user has been granted access, the administrator can manage it as with any other user in the Keystone Identity back-end (i.e. disable/enable, grant/revoke roles, etc.). Figure 9.2 describes a token request with the VOMS module.

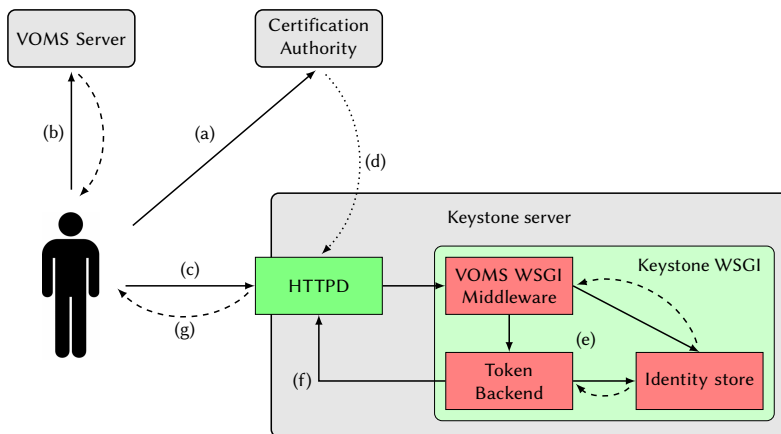


Figure 9.2: VOMS support in Keystone: **a)** The user request a certificate; **b)** creates a VOMS proxy; **c)** authenticates against Keystone using the proxy; **d)** the HTTPD server verifies the proxy against the CA and CRLs; **e)** WSGI middleware extracts VO information and maps to a tenant; **f)** token is issued; and **g)** credentials are returned to the user.

Once a user is removed from a VO, he won't be able to access the RP resources. The VOMS server will refuse to issue a valid VOMS proxy for that user, thus authentication will not take place.

9.3. Conclusions

With the introduction of the Virtual Organization Membership Service (VOMS) support in Keystone, it is possible to create a federation of OpenStack services where a single identity can be used to access the resources from different providers. Resource providers based in any other Cloud Management Frameworks (CMFs) may also join the federation if the VOMS authentication and authorization is supported. OpenNebula, through the rOCCI framework [194], also supports VOMS authentication for the Open Cloud Computing Interface (OCCI) interface. Several OpenStack and OpenNebula resource providers are integrated within the European Grid Infrastructure (EGI) Federated Cloud task force by using a common Application Programming Interface (API) (OCCI) with a single identity (based on VOMS proxies).

The identity federation should be one of the first steps when establishing a federation, since it creates the foundations where all the other services –accounting, brokering,

etc.— will rely on.

In a well established infrastructure, introducing a new authentication method is disruptive for either the resource providers —that are familiar and confident on the current system— and the final users —that are used to a set of tools. This is the case of the European Grid infrastructures, that have effectively deployed a working authentication and authorization framework via their National Research and Education Networks (NRENs), the EuGridPMA accredited Certification Authorities and the Virtual Organization (VO)-management model. Applying and adapting this same model to the CMFs that the resource providers are deploying is a step forward to the establishment of a federated European cloud.

This work is only an initial step towards a federated authentication and authorization framework, based on the VO concept. The current development is based in X.509 certificate and X.509 certificate proxies that are considered to be un-friendly by several user communities. However, the work presented in this chapter is used within the EGI Federated Cloud Task Force.

IV

CONCLUSIONS

10

Conclusions

10.1. Contributions

The work presented in this dissertation contributes to the enhancement and adaptation of the cloud computing model to accommodate scientific applications, advancing from the current state of the art of the cloud technologies. In order to reach this objective, the cloud computing ecosystem has been analysed, considering the requirements imposed by scientific computing characteristics.

In particular, in this work I have focused on two different and broad areas: the improvements in the resource allocation strategies and the need for a federated and interoperable cloud due to the way modern science is performed. Regarding the resource allocation policies, I have focused in some of the provisioning aspects that influence the elasticity perceptions for a scientific use. Considering federation and interoperability, I have analysed the main existing challenges for establishing a cloud federation based in open standards.

In summary, the major contributions of this dissertation are:

- The analysis of the cloud computing ecosystem focusing in its feasibility to execute scientific applications, taking into account their specific requirements and the challenges that a cloud provider is facing when operating a Science Cloud. With this analysis I have settled the outstanding challenges for the current technologies. Some of these challenges are addressed in this dissertation.
- The proposal of an enhancement of the Virtual Machine Image (VMI) distribution methods and the improvement of the scheduling policies within the Cloud Management Frameworks (CMFs) so that a rapid provisioning of Virtual Machines (VMs) is possible, specially when performing large requests.
- The proposal and study of preemptible instances as a way to improve the usage of the cloud infrastructures, making possible to obtain opportunistic resources for fault-tolerant and batch computing tasks, making possible that on-demand request can coexist more easily with long-running workloads.
- An implementation of the Open Cloud Computing Interface (OCCI) standard for OpenStack, making possible to integrate OpenStack-based clouds in existing federations based on open standards, such as the European Grid Infrastructure (EGI) Federated Cloud.

-
- The proposal, implementation and deployment of the Virtual Organization Membership Service (VOMS) system for obtaining a federated authentication and authorization framework in cloud systems, making easier the transition and interoperability between existing grid and cloud infrastructures.

Even if the work of this dissertation has been focused on satisfying requirements and challenges for scientific applications, the addressed topics are of general interest for the community, therefore their impact is not limited to Science Clouds.

10.2. Publications

The work described in this dissertation has produced a number of publications in scientific journals and international conferences and workshops, as it will be outlined next. Moreover, this work has produced some software products that are currently being in use in European projects and production infrastructures, such as the EGI Federated Cloud and the Spanish National Research Council (CSIC) Science Cloud infrastructure at the Instituto de Física de Cantabria (IFCA).

The following research papers have been published in scientific journals:

- **Á. López García** and E. Fernández-del-Castillo. “Efficient image deployment in Cloud environments”. In: *Journal of Network and Computer Applications* (2016). ISSN: 1084-8045 (accepted paper).
- **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “OpenStack OCCI Interface”. In: *SoftwareX* (2016). ISSN: 2352-7110. DOI: 10.1016/j.softx.2016.01.001 (accepted paper).
- I. Campos Plasencia, E. Fernández-del-Castillo, S. Heinemeyer, **Á. López García**, F. Pahlen, and G. Borges. “Phenomenology tools on cloud infrastructures using OpenStack”. In: *The European Physical Journal C* 73.4 (Apr. 2013), p. 2375. ISSN: 1434-6044. DOI: 10.1140/epjc/s10052-013-2375-0. arXiv: arXiv:1212.4784v1. URL: <http://link.springer.com/10.1140/epjc/s10052-013-2375-0>.

The following papers have been submitted to the corresponding journals:

- **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “Standards for enabling heterogeneous IaaS cloud federations”. In: *Computer Standards & Interfaces* (2016). ISSN: 0920-5489 (under review).
- **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “Resource provisioning in Science Clouds: requirements and challenges”. In: *Journal of Grid Computing* (2016). ISSN: 1572-9184 (under review).

Contributions to scientific conferences directly related to the topics of the dissertation:

- E. Fernández-del-Castillo, D. Scardaci, and **Á. López García**. “The EGI Federated Cloud e-Infrastructure”. In: *Procedia Computer Science* 68 (2015), pp. 196–205. ISSN: 18770509. DOI: 10.1016/j.procs.2015.09.235. URL: <http://linkinghub.elsevier.com/retrieve/pii/S187705091503080X>.
- I. Blanquer, G. Donvito, P. Fuhrmann, **Á. López García**, and G. Molto. *An integrated IaaS and PaaS architecture for scientific computing*. Oral Contribution. EGI Community Forum: Bari (Italy), Nov. 10–13, 2015
- **Á. López García**, P. Fuhrmann, G. Donvito, and A. Chierici. *Improving IaaS resources to accommodate scientific applications*. Oral Contribution. HEPiX Fall 2015 Workshop: Brookhaven National Laboratory, New York (USA), Oct. 12–16, 2015
- **Á. López García**, P. Orviz Fernández, F. Aguilar, E. Fernández-del-Castillo, I. Campos Plasencia, and J. Marco de Lucas. “The role of IBERGRID in the Federated Cloud of EGI”. in: *Proceedings of the IBERGRID 2014 Conference*. Editorial Universidad Politecnica de Valencia, 2014, pp. 3–14. ISBN: 978-84-9048-246-9.
- **Á. López García**. *OpenStack Cloud Workshop*. Workshop. 8th Iberian Grid Computing Conference – IBERGRID 2014: University of Aveiro, Aveiro (Portugal), Sept. 8–10, 2014
- **Á. López García**. *OpenStack hands on*. Workshop. EGI Community Forum: Helsinki (Finland), May 19–23, 2014

-
- **Á. López García**, E. Fernández-del-Castillo, and M. Puel. “Identity Federation with VOMS in Cloud Infrastructures”. In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. 2013, pp. 42–48. ISBN: 978-0-7695-5095-4. DOI: 10.1109/CloudCom.2013.13. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753776>.
 - **Á. López García** and E. Fernández-del-Castillo. “Analysis of Scientific Cloud Computing requirements”. In: *Proceedings of the IBERGRID 2013 Conference*. 2013, p. 147–158.
 - **Á. López García** and P. Orviz Fernández. *Integrating cloud computing within an existing infrastructure*. Poster. EGI Technical Forum: Madrid (Spain), Sept. 16–20, 2013
 - **Á. López García** and E. Fernández-del-Castillo. *Analysis of Scientific Cloud Computing requirements*. Oral Contribution. EGI Technical Forum: Madrid (Spain), Sept. 16–20, 2013
 - **Á. López García** and E. Fernández-del-Castillo. *Analysis of Scientific Cloud Computing requirements*. Oral Contribution. 7th Iberian Grid Computing Conference – IBERGRID 2013: Madrid (Spain), Sept. 19–20, 2013
 - E. Fernández-del-Castillo, I. Campos, S. Heinemeyer, **Á. López García**, and F. Pahlen. *Phenomenology Tools on a OpenStack Cloud Infrastructure*. Oral Contribution. EGI Community Forum 2013: The University of Manchester, Manchester (UK), Apr. 8–12, 2012
 - M. Airaj, C. Cavet, V. Hamar, M. Jouvin, C. Loomis, et al. “Vers une fédération de Cloud Académique dans France Grilles”. In: *Journées SUCCES 2013*. Paris, France, Nov. 2013. URL: <https://hal.archives-ouvertes.fr/hal-00927506>.
 - A. Y. Rodríguez-Marrero, I. González Caballero, A. Cuesta Noriega, E. Fernández-del-Castillo, **Á. López García**, et al. “Integrating PROOF Analysis in Cloud and Batch Clusters”. In: *Journal of Physics: Conference Series* 396.3 (Dec. 2012), p. 32091. ISSN: 1742-6588. DOI: 10.1088/1742-6596/396/3/032091. URL: <http://stacks.iop.org/1742-6596/396/i=3/a=032091?key=crossref.a4219812d32b8f4fb9c750e66cfcde37>.

- E. Fernández-del-Castillo, **Á. López García**, I. Campos Plasencia, M. A. Nuñez Vega, J. Marco de Lucas, et al. “IberCloud: federated access to virtualized resources”. In: *Proceedings of the IBERGRID 2012 Conference*. Lisbon, Nov. 2012, pp. 195–205.
- E. Fernández-del-Castillo, **Á. López García**, I. Campos, M. Á. Nuñez, J. Marco, et al. *IberCloud: federated access to virtualized resources*. Oral Contribution. EGI Technical Forum 2012: Prague (Czech Republic), Sept. 17–21, 2012
- E. Fernández-del-Castillo, **Á. López García**, I. Campos, M. Á. Nuñez, J. Marco, et al. *IberCloud: federated access to virtualized resources*. Oral Contribution. 6th Iberian Grid Computing Conference – IBERGRID 2012: Lisbon (Portugal), Sept. 7–9, 2012
- **Á. López García** and M. Puel. *France-Grilles dans la Federation Cloud Task-Force d’EGI*. Oral Contribution. Atelier Operations France-Grilles: INRA, Villenave d’Ornon (France), Nov. 29–30, 2012
- **Á. López García** and M. Puel. *Cloud Computing at CC-IN2P3*. Invited Talk. Rencontre LCG-France: SUBATECH, Nantes (France), Sept. 18–19, 2012

The work performed in this dissertation has produced additional software products that enable the integration of OpenStack clouds in the EGI Federated Cloud infrastructure:

- Keystone-VOMS¹, a module for the OpenStack Identity service (Keystone) that provides VOMS-based authentication and authorization.
- ooi² [235], an implementation of the Open Grid Forum (OGF) OCCI standard for the OpenStack CMF.
- cASO³, an accounting extractor, based on the OGF cloud Usage Record (UR) standard.

¹**Á. López García** and E. Fernández-del-Castillo. *Keystone-VOMS*. URL: <https://github.com/IFCA/keystone-voms>.

²**Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. *OpenStack OCCI Interface*. 2016. URL: <https://github.com/openstack/ooi>.

³**Á. López García** and E. Fernández-del-Castillo. *cASO: OpenStack Accounting Extractor*. URL: <https://github.com/IFCA/caso>.

-
- The cloud information system provider⁴, a module for extracting information from the underlying CMF and publishing it using the Grid Laboratory Uniform Environment (GLUE) 2.0 schema.

In addition, besides the standalone software products outlined above, I have contributed several improvements to the upstream OpenStack project:

- I have implemented the current weight normalization (as described in Section 4.3 mechanism in the OpenStack Compute Scheduler⁵. Some of the work performed in this dissertation relies in how the scheduler weights the nodes so as to select the best suited host for scheduling a request. The scheduler applied several weights to each of the hosts according to the weighed characteristics (for instance free RAM or number of Input/Output (I/O) operations) without normalizing them (i.e. the values were in a free range), hence it was impossible to predict the scheduling behaviour or to apply a higher priority to one weight against others.
- I have proposed and implemented a cache-aware weigher for the OpenStack scheduler⁶, according to the functionality described in Chapter 5
- I have submitted an initial proposal for the implementation of the preemptible instances mechanism described in this dissertation⁷, following the design described in Chapter 6.

10.3. Future Work and Perspective

Although the proposed solutions in this dissertation are functional and provide and advancement in the state of the art on their own, there are research areas that can be explored in the future so as to enhance and complete the proposed solutions.

⁴Á. López García and E. Fernández-del-Castillo. *cloud-bdii-provider*. URL: <https://github.com/EGI-FCTF/cloud-bdii-provider>.

⁵Á. López García. *OpenStack Compute Scheduler weight normalization*. URL: <https://blueprints.launchpad.net/nova/+spec/normalize-scheduler-weights>.

⁶Á. López García. *OpenStack Compute Scheduler Cache Aware*. URL: <https://blueprints.launchpad.net/nova/+spec/cache-aware-weigher>.

⁷Á. López García. *OpenStack Spot Instances Support Specification*. URL: <https://blueprints.launchpad.net/nova/+spec/spot-instances>.

- In Chapter 5 we have demonstrated how an image cache aware scheduler reduces the boot time when large requests are made. However, the cache mechanism still relies in the fact that the VMI is available in the physical node, otherwise a cache miss will exist. I think that it is worth exploring a method for a smart image preseed into the nodes based on a calculated usage popularity. This design would detect if there are nodes able to satisfy a popular request at a given moment and preseed the popular images to the free nodes if it is not possible.
- Regarding the preemptible instances support exposed in Section 6, this functionality can be exploited so as to implement more complex policies on top of it. A bidding system with a price fluctuation depending on the past and current usage of the infrastructure (similar to a stock market) could be interesting for commercial providers willing to increase their revenues. The preemptible instances makes possible to implement other policies, such as credit systems or fair-sharing.
- The AAI and identity federation is evolving at a fast pace, moving to more integrated solutions where users can use several identities (for instance X.509, institutional identity providers using OpenID Connect, etc.) being mapped to the same account within one realm, regardless of the authentication method being used.
- Standards need to evolve to satisfy the needs of the actors using them and become useful instruments. The OCCI 1.2 specification that is under development has addressed some of these requirements after collecting feedback from all the involved parties (that is, resource providers and users). Contributions to these standards and the relevant standardization bodies is a must, so as to enrich and enhance the open standards ecosystem.
- Besides, some of the proposed challenges in Chapter 3 still remain open, mainly those related with the resource provisioning and scheduling.
- Part II of this dissertation was focused on VMs as the resources that are managed by the CMF. Nowadays Operating System (OS)-level virtualization is becoming very popular. This virtualization technique, in contrast with the Virtual Machine Monitor (VMM)-based, leverages the OS kernel to isolate several user-space instances instead of just one, called *containers* —such as Docker, LXC, etc. OS-based

virtualization imposes little no overhead, therefore it is gradually becoming more popular among users, such as the INDIGO-Datacloud EU project⁸. From the CMF point of view, containers can be treated as lightweight virtual machines, therefore all the proposed improvements in Part II still apply.

⁸*INDIGO-Datacloud*. 2015. URL: <http://indigo-datacloud.eu>.



APPENDICES



Experimental Facilities

Unless stated otherwise, each of the tests performed in this dissertation have been performed in one of the following testing infrastructures or environments.

A.1. CSIC IFCA Science Cloud Production Infrastructure

The Spanish National Research Council (CSIC) Science Cloud infrastructure at the Instituto de Física de Cantabria (IFCA) consists on the following services:

- A Head node hosting:
 - The OpenStack Compute Application Programming Interfaces (APIs).
 - The OpenStack OCCI Interface.
 - The OpenStack Compute Scheduler service.
- An OpenStack Identity Service (Keystone)
- An OpenStack Volume Service (Cinder)

-
- A MariaDB 10.1.0 server.
 - Two nodes hosting the RabbitMQ 3.2.4 servers and the OpenStack Compute Conductor service.
 - An Image Catalog running the OpenStack Image Service (Glance) serving images from its local disk.
 - Several OpenStack Compute nodes, as explained below.

These servers —except the compute nodes— are deployed as virtual machines on a dedicated infrastructure that is devoted to run services in High Availability.

The OpenStack Compute nodes consist on a variety of servers, utilizing Xen version 4.X as the virtualization technology.

- 36 nodes with the characteristics described in Table A.1.
- 34 nodes with the characteristics described in Table A.2.
- 18 nodes with the characteristics described in Table A.3.
- 8 nodes with the characteristics described in Table A.4.
- 32 nodes with the characteristics described in Table A.5.

The Operating System (OS) being installed in all of these nodes is Ubuntu Server 14.04 LTS, running the Linux 3.8.0 Kernel. The OpenStack version deployed has evolved during the writing of this thesis, starting with the 2012.1 (Essex) version up to 2015.2 (Liberty).

CPU	2 x Intel®Xeon®CPU X5670 @ 2.93 GHz
RAM	48 GB
Disk	250 GB
Network	4 x 1 Gbit Ethernet

Table A.1: Type-1 Virtualization Node characteristics.

CPU	2 x Intel®Xeon®CPU E5-2670 0 @ 2.60 GHz
RAM	128 GB
Disk	500 GB
Network	2 x 10 Gbit Ethernet

Table A.2: Type-2 Virtualization Node characteristics.

CPU	2 x Intel®Xeon®CPU E5-2697 v2 @ 2.70 GHz
RAM	96 GB
Disk	400 GB
Network	2 x 10 Gbit Ethernet

Table A.3: Type-3 Virtualization Node characteristics.

CPU	2 x Amd Opteron 6176 SE
RAM	256 GB
Disk	860 GB
Network	2 x 10 Gbit Ethernet

Table A.4: Type-4 Virtualization Node characteristics.

CPU	2 x Intel®Xeon®CPU E31260L @ 2.40 GHz
RAM	16 GB
Disk	1 TB
Network	2 x 1 Gbit Ethernet

Table A.5: Type-5 Virtualization Node characteristics.

A.2. IFCA Batch System

The IFCA Batch System runs on top of the cloud resources described in A.1. All the nodes run the same Scientific Linux 6 OS and is orchestrated using the Son of Grid Engine [276] batch system. Therefore, the worker nodes are virtualized machines with following the configuration described below:

- 36 nodes with the characteristics described in Table A.6.
- 34 nodes with the characteristics described in Table A.7.
- 18 nodes with the characteristics described in Table A.8.

CPU	2 x Intel®Xeon®CPU X5670 @ 2.93 GHz
RAM	45 GB
Disk	220 GB
Network	1 x 1 Gbit Ethernet

Table A.6: Type-1 Worker Node characteristics.

CPU	2 x Intel®Xeon®CPU E5-2670 0 @ 2.60 GHz
RAM	124 GB
Disk	320 GB
Network	1 x 10 Gbit Ethernet

Table A.7: Type-2 Worker Node characteristics.

CPU	2 x Intel®Xeon®CPU E5-2697 v2 @ 2.70 GHz
RAM	92 GB
Disk	290 GB
Network	1 x 10 Gbit Ethernet

Table A.8: Type-3 Worker Node characteristics.

A.3. CSIC IFCA Cloud Test Infrastructure

The CSIC IFCA Cloud test infrastructure consists on a dedicated cloud testbed that comprises a set of 26 identical IBM HS21 blade servers, with the characteristics shown in Table A.9

CPU	2 x Intel®Xeon®Quad Core E5345 2.33 GHz
RAM	16 GB
Disk	140 GB, 10 000 rpm hard disk
Network	1 Gbit Ethernet

Table A.9: Test node characteristics.

The network setup of the testbed consists on two 10 Gbit Ethernet switches, interconnected with a 10 Gbit Ethernet link. All the hosts are evenly connected to these switches using a 1 Gbit Ethernet connection.

The system architecture is as follows:

- A Head node hosting all the required services to manage the cloud test infrastruc-

ture, that is:

- The OpenStack Compute APIs.
 - The OpenStack Compute Scheduler service.
 - The OpenStack Compute Conductor service.
 - The OpenStack Identity Service (Keystone)
 - A MariaDB 10.1.0 server.
 - A RabbitMQ 3.2.4 server.
- An Image Catalog running the OpenStack Image Service (Glance) serving images from its local disk.
 - 24 Compute Nodes running OpenStack Compute, hosting the spawned instances.

The OS being used for these tests is an Ubuntu Server 14.04 LTS, running the Linux 3.8.0 Kernel. The OpenStack version deployed depends on the test performed, therefore it has been stated in each individual chapter.

A.4. OCCI Testing Environment

In order to perform the performance comparison of the different Open Cloud Computing Interface (OCCI) implementations, we deployed it on a virtual machine with the characteristics depicted in Table A.10

CPU	1 x eight-core Intel®Xeon®E5-2640 2.00 GHz equivalent.
RAM	16 GB
Network	1 Gbit Ethernet

Table A.10: Test node characteristics..

This machine only executed the required APIs (that is, the OpenStack native API, `oai`, and OCCI-OS) making use of the rest of the services described in Section A.3. The OpenStack version used for these tests was the OpenStack 2015.2 (Kilo) release.

A.5. HEP Spec Tests

CPU	2 x Intel®Xeon®E5-2670 2.60 GHz.
RAM	128 GB

Table A.11: Test node characteristics.

Bibliography

- [1] P. Mell, T. Grance, and P. Mell. *The NIST definition of cloud computing*. Tech. rep. Special Publication 800-145. National Institute of Standards and Technology ({NIST}), Sept. 2011.
- [2] L. M. L. L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. “A break in the clouds: towards a cloud definition”. In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 50–55. URL: <http://dl.acm.org/citation.cfm?id=1496100>.
- [3] R. Buyya, C. S. Yeo, and S. Venugopal. “Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities”. In: *2008 10th IEEE International Conference on High Performance Computing and Communications*. 2008, pp. 5–13. ISBN: 978-0-7695-3352-0. DOI: 10.1109/HPCC.2008.172. arXiv: 0808.3558. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637675>.
- [4] I. Foster, Y. Zhao, I. Raicu, and S. Lu. “Cloud computing and grid computing 360-degree compared”. In: *Grid Computing Environments Workshop, 2008. GCE'08*.

-
- 2008, pp. 1–10. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=4738445.
- [5] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, et al. “A view of cloud computing”. In: *Communications of the ACM* 53.4 (Apr. 2010), p. 50. ISSN: 00010782. DOI: 10.1145/1721654.1721672. URL: <http://portal.acm.org/citation.cfm?doid=1721654.1721672>.
- [6] F. Oesterle, S. Ostermann, R. Prodan, and G. J. Mayr. “Experiences with distributed computing for meteorological applications: grid computing and cloud computing”. In: *Geoscientific Model Development* 8.7 (2015), pp. 2067–2078. ISSN: 1991-9603. DOI: 10.5194/gmd-8-2067-2015.
- [7] M. a. Rappa. “The utility business model and the future of computing services”. In: *IBM Systems Journal* 43.1 (2004), pp. 32–42. ISSN: 0018-8670. DOI: 10.1147/sj.431.0032.
- [8] J. M. A. Calero, N. Edwards, J. Kirschnick, L. Wilcock, M. Wray, J. M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. “Toward a multi-tenancy authorization system for cloud services”. In: *IEEE Security and Privacy* 8.6 (2010), pp. 48–55. ISSN: 15407993. DOI: 10.1109/MSP.2010.194.
- [9] Q. Zhang, L. Cheng, and R. Boutaba. “Cloud computing: state-of-the-art and research challenges”. In: *Journal of Internet Services and Applications* 1.1 (Apr. 2010), pp. 7–18. ISSN: 1867-4828. DOI: 10.1007/s13174-010-0007-6. URL: <http://www.springerlink.com/index/10.1007/s13174-010-0007-6>.
- [10] Juniper Networks. *Securing the multitenancy and cloud computing*. Tech. rep. 2012, pp. 1–5.
- [11] G. J. Feeney, R. D. Hilton, R. L. Johnson, T. J. O’Rourke, and T. E. Kurtz. “Utility Computing: A Superior Alternative?” In: *Proceedings of the May 6-10, 1974, National Computer Conference and Exposition*. AFIPS ’74. New York, NY, USA: ACM, 1974, p. 1003. DOI: 10.1145/1500175.1500370. URL: <http://doi.acm.org/10.1145/1500175.1500370>.
- [12] I. Foster, C. Kesselman, and S. Tuecke. “The anatomy of the grid”. In: *International Journal of Supercomputer Applications* 15.3 (Aug. 2001), pp. 200–222. ISSN: 1094-3420. DOI: 10.1177/109434200101500302. arXiv: 0103025 [cs]. URL: <http://hpc.sagepub.com/content/15/3/200.abstract>

- <http://hpc.sagepub.com/content/15/3/200.full.pdf>
<http://hpc.sagepub.com/content/15/3/200.short>
<http://213.55.83.52/ebooks/Electrical%20and%20computer%20engineering/501571.pdf>.
- [13] I. Foster. “What is the grid? A three point checklist”. In: *GRIDtoday* (2002). URL: <http://www.citeulike.org/group/1880/article/798241>.
- [14] D. Kranzlmüller, J. M. de Lucas, and P. Öster. “The european grid initiative (EGI)”. In: *Remote Instrumentation and Virtual Laboratories*. Springer, 2010, pp. 61–66. URL: <http://www.egi.eu>.
- [15] T. Ferrari and L. Gaido. “Resources and services of the EGEE production infrastructure”. In: *Journal of Grid computing* 9.2 (2011), pp. 119–133.
- [16] EGI.eu. *EGI Case Studies*. Tech. rep. EGI.eu. URL: <https://www.egi.eu/case-studies/>.
- [17] R. P. Goldberg. “Survey of virtual machine research”. In: *Computer* 7.6 (1974), pp. 34–45.
- [18] G. J. Popek and R. P. Goldberg. “Formal requirements for virtualizable third generation architectures”. In: *Communications of the ACM* (1974). URL: <http://dl.acm.org/citation.cfm?id=361073>.
- [19] R. Figueiredo, P. a. Dinda, and J. Fortes. “Resource virtualization renaissance”. In: *Computer* 38.5 (2005), pp. 28–31. ISSN: 00189162. DOI: 10.1109/MC.2005.159.
- [20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. “Xen and the art of virtualization”. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles SE - SOSP '03*. New York, NY, USA: ACM, 2003, pp. 164–177. ISBN: 1-58113-757-5. URL: [citeulike-article-id:168648%20http://dx.doi.org/10.1145/945445.945462](http://dx.doi.org/10.1145/945445.945462).
- [21] K. Milberg. *IBM and HP virtualization: A comparative study of UNIX virtualization on both platforms*. Tech. rep. 2009, pp. 1–11. URL: <http://www.ibm.com/developerworks/aix/library/au-aixhpvirtualization/index.html>.

-
- [22] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. Martins, A. V. Anderson, S. M. Bennett, A. Kägi, F. H. Leung, and L. Smith. *Intel virtualization technology*. Tech. rep. 03. 2005, pp. 48–56. DOI: 10.1535/itj.1003. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Da11.jsp?arnumber=1430631.
- [23] AMD. “Putting Server Virtualization to Work”. In: *AMD White Paper (2007)*, pp. 1–4.
- [24] Y. Dong, S. Li, A. Mallick, J. Nakajima, K. Tian, X. Xu, F. Yang, W. Yu, and Yaozu DongShaofan LiAsit MallickJun NakajimaKun TianXuefei XuFred YangWilfred Yu. “Extending Xen with Intel Virtualization Technology.” In: *Intel Technology Journal* 10.3 (2006), p. 193. ISSN: 1535864X. DOI: 10.1535/itj.1003.
- [25] A. Kivity, U. Lublin, A. Liguori, Y. Kamay, and D. Laor. “kvm: the Linux virtual machine monitor”. In: *Proceedings of the Linux Symposium*. Vol. 1. 2007, pp. 225–230.
- [26] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. *Performance Evaluation of Virtualization Technologies for Server Consolidation*. Tech. rep. HPL-2007-59. HP Laboratories, 2007, p. 15. DOI: 10.1.1.70.4605. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.70.4605%7B%5C%7D&rep=rep1%7B%5C%7D&type=pdf>.
- [27] M. Guazzone. “Power and Performance Management in Cloud Computing Systems”. PhD thesis. University of Torino.
- [28] S. Srikantaiah, A. Kansal, and F. Zhao. “Energy Aware Consolidation for Cloud Computing”. In: *Proceedings of HotPower ’08 Workshop on Power Aware Computing and Systems*. San Diego, CA, USA: USENIX, 2008. URL: [citeulike-article-id:4234554%20http://www.usenix.org/events/hotpower08/tech/full%7B%5C_%7Dpapers/srikantaiah/srikantaiah.pdf](http://www.usenix.org/events/hotpower08/tech/full%7B%5C_%7Dpapers/srikantaiah/srikantaiah.pdf).
- [29] R. Buyya, A. Beloglazov, and J. Abawajy. “Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges”. In: *PDPTA 2010: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, 2010, pp. 6–17. URL: [citeulike-article-id:7356645%20http://arxiv.org/abs/1006.0308](http://arxiv.org/abs/1006.0308).

- [30] E. Keller, J. Szefer, J. Rexford, and R. Lee. “NoHype: virtualized cloud infrastructure without the virtualization”. In: *Computer Architecture News* (2010). URL: <http://dl.acm.org/citation.cfm?id=1816010>.
- [31] I. Campos Plasencia, E. Fernández-del-Castillo, S. Heinemeyer, **Á. López García**, F. Pahlen, and G. Borges. “Phenomenology tools on cloud infrastructures using OpenStack”. In: *The European Physical Journal C* 73.4 (Apr. 2013), p. 2375. ISSN: 1434-6044. DOI: 10.1140/epjc/s10052-013-2375-0. arXiv: arXiv:1212.4784v1. URL: <http://link.springer.com/10.1140/epjc/s10052-013-2375-0>.
- [32] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan. “Performance implications of virtualizing multicore cluster machines”. In: *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. New York, NY, USA: ACM, 2008, pp. 1–8. ISBN: 978-1-60558-120-0. URL: [citeulike-article-id:4504070%20http://dx.doi.org/10.1145/1435452.1435453](http://dx.doi.org/10.1145/1435452.1435453).
- [33] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo. “A Comparison of Virtualization Technologies for HPC”. In: *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*. Ieee, 2008, pp. 861–868. ISBN: 978-0-7695-3095-6. DOI: 10.1109/AINA.2008.45. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4482796>.
- [34] N. Regola and J.-C. Ducom. “Recommendations for Virtualization Technologies in High Performance Computing”. In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, Nov. 2010, pp. 409–416. ISBN: 978-1-4244-9405-7. DOI: 10.1109/CloudCom.2010.71. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708479>.
- [35] L. Youseff, R. Wolski, B. Gorda, and R. Krintz. “Paravirtualization for HPC Systems”. In: *IN PROC. WORKSHOP ON XEN IN HIGH-PERFORMANCE CLUSTER AND GRID COMPUTING* (2006), pp. 474–486. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.3446>.
- [36] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. “Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems”. In: *Virtualization Technology in Distributed Computing, 2006. VTDC 2006. First International*

-
- Workshop on*. 2006, p. 1. URL: citeulike-article-id:2902268%20http://dx.doi.org/10.1109/VTDC.2006.4.
- [37] B. Li, Z. Huo, P. Zhang, and D. Meng. “Virtualizing Modern High-Speed Interconnection Networks with Performance and Scalability”. In: *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*. 2010, pp. 107–115. URL: citeulike-article-id:10418487%20http://dx.doi.org/10.1109/CLUSTER.2010.19.
- [38] W. Huang. “High Performance Network I/O in Virtual Machines over Modern Interconnects”. In: *Engineering (2008)*. URL: http://nowlab.cse.ohio-state.edu/NOW/dissertations/huang.pdf.
- [39] W3C Working Group. *Web Services Glossary*. 2015. URL: http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/ (visited on 01/01/2015).
- [40] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Stöber. “Cloud Computing – A Classification, Business Models, and Research Directions”. In: *Business {&} Information Systems Engineering 1.5 (2009)*, pp. 391–399. ISSN: 1867-0202. DOI: 10.1007/s12599-009-0071-2.
- [41] C. N. Höfer and G. Karagiannis. “Cloud computing services: Taxonomy and comparison”. In: *Journal of Internet Services and Applications 2.2 (2011)*, pp. 81–94. ISSN: 18674828. DOI: 10.1007/s13174-011-0027-x.
- [42] D. de Oliveira, F. A. Baião, and M. Mattoso. “Towards a taxonomy for cloud computing from an e-science perspective”. In: *Cloud Computing*. Springer, 2010, pp. 47–62.
- [43] A. Fox and R. Griffith. *Above the clouds: A Berkeley view of cloud computing*. Tech. rep. 2009, pp. 7–13. URL: http://www-inst.cs.berkeley.edu/%7B-%7Dcs10/sp11/lec/20/2010Fa/2010-11-10-CS10-L20-AF-Cloud-Computing.pdf.
- [44] T. Dillon, C. W. C. Wu, and E. Chang. “Cloud Computing: Issues and Challenges”. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. 2010, pp. 27–33. ISBN: 978-1-4244-6695-5. DOI: 10.1109/AINA.2010.187.
- [45] Idabc. *European Interoperability Framework for pan-European eGovernment Services*. Tech. rep. 2004, pp. 1–25. DOI: 10.1109/HICSS.2007.68.

- [46] M. Christodorescu and R. Sailer. "Cloud security is not (just) virtualization security: a short paper". In: *Proceedings of the 2009 ACM workshop on Cloud computing security*. 2009, pp. 97–102. ISBN: 9781605587844. URL: <http://dl.acm.org/citation.cfm?id=1655022>.
- [47] D. Zissis and D. Lekkas. "Addressing cloud computing security issues". In: *Future Generation Computer Systems* 28.3 (2012), pp. 583–592. ISSN: 0167739X. DOI: 10.1016/j.future.2010.12.006. arXiv: S0167739X10002554. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X10002554>.
- [48] M. Zhou. "Data security and integrity in cloud computing". PhD thesis. University of Wollongong, 2013.
- [49] F. Lombardi and R. Di Pietro. "Secure virtualization for cloud computing". In: *Journal of Network and Computer Applications* 34.4 (July 2011), pp. 1113–1122. ISSN: 10848045. DOI: 10.1016/j.jnca.2010.06.008. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1084804510001062>.
- [50] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. "Cross-VM side channels and their use to extract private keys". In: *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. New York, New York, USA: ACM Press, 2012, p. 305. ISBN: 9781450316514. DOI: 10.1145/2382196.2382230. URL: <http://dl.acm.org/citation.cfm?doid=2382196.2382230>.
- [51] M. T. Heath. *Scientific computing*. McGraw-Hill, 2001.
- [52] G. E. Karniadakis, R. M. Kirby, and I. I. Kirby. *Parallel Scientific Computing in C++ and MPI*. Cambridge University Press, 2003. ISBN: 978-0521520805. URL: <http://scholar.google.com/scholar?hl=en%7B%5C%7DbtnG=Search%7B%5C%7Dq=intitle:Parallel+Scientific+Computing+in+C+++and+MPI%7B%5C%7D2%20http://scholar.google.com/scholar?hl=en%7B%5C%7DbtnG=Search%7B%5C%7Dq=intitle:RM:+Parallel+Scientific+Computing+in+C+++and+MPI%7B%5C%7D0>.
- [53] Z. Constantinescu. "A Desktop Grid Computing Approach for Scientific Computing and Visualization". PhD thesis. Norwegian University of Science and Technology, 2008, p. 246. ISBN: 9788247191583.

-
- [54] V. Breton, T. Cass, J. Flynn, F. Gaede, J. Kleist, et al. *Computing Resources Scrutiny Group Report*. Tech. rep. CERN-RRB-2015-014. Geneva: CERN, Mar. 2015. URL: <http://cds.cern.ch/record/2002240>.
- [55] I. Bird, P. Buncic, F. Carminati, M. Cattaneo, P. Clarke, et al. *Update of the Computing Models of the WLCG and the LHC Experiments*. Tech. rep. CERN-LHCC-2014-014. LCG-TDR-002. Geneva: CERN, Apr. 2014. URL: <https://cds.cern.ch/record/1695401>.
- [56] M. Guest. *PRACE – The Scientific Case for HPC in Europe*. Tech. rep. PRACE, 2012.
- [57] European Commission. *High-Performance Computing: Europe’s place in a Global Race*. Tech. rep. Brussels: European Commission, 2012.
- [58] a. J. G. Hey and a. E. Trefethen. “The UK e-Science Core Programme and the Grid”. In: *Future Generation Computer Systems* 18.8 (2002), pp. 1017–1031. ISSN: 0167739X. DOI: 10.1016/S0167-739X(02)00082-1. URL: <http://eprints.ecs.soton.ac.uk/7644/>.
- [59] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Ed. by I. Foster and C. Kesselman. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 13–24. ISBN: 1558604758. DOI: citeulike-article-id:340626. URL: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%7B%5C%7Dpath=ASIN/1558604758>.
- [60] European Grid Initiative. *EGI*. 2015. URL: <http://www.egi.eu/%20http://egi.eu>.
- [61] *EGI in numbers*. 2015. URL: https://www.egi.eu/infrastructure/operations/egi%7B%5C_%7Din%7B%5C_%7Dnumbers/index.html.
- [62] *Open Science Grid (OSG)*. 2014.
- [63] **Á. López García** and E. Fernández-del-Castillo. “Analysis of Scientific Cloud Computing requirements”. In: *Proceedings of the IBERGRID 2013 Conference*. 2013, p. 147–158.
- [64] G. Cattaneo, M. Claps, S. Conway, M. (Bardellini, S. Muscella, S. Parker, and N. (I. Ferguson. *Cloud for science and public authorities*. Tech. rep. European Commission, 2013, p. 104. DOI: 10.2759/25446.

- [65] G. Birkenheuer, A. Brinkmann, J. Kaiser, A. Keller, M. Keller, et al. “Virtualized HPC: a contradiction in terms?” In: *Software: Practice and Experience* 42.4 (Apr. 2012), pp. 485–500. ISSN: 00380644. DOI: 10.1002/spe.1055. URL: <http://doi.wiley.com/10.1002/spe.1055>.
- [66] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. “Scientific Cloud Computing: Early Definition and Experience”. In: *2008 10th IEEE International Conference on High Performance Computing and Communications*. Ieee, Sept. 2008, pp. 825–830. ISBN: 978-0-7695-3352-0. DOI: 10.1109/HPCC.2008.38. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637787>.
- [67] M. Iliáš and M. Dobrucký. “Grid Computing with Relativistic Quantum Chemistry Software”. In: *Journal of Grid Computing* 12.4 (2014), pp. 681–690. ISSN: 1570-7873. DOI: 10.1007/s10723-014-9309-4. URL: <http://link.springer.com/10.1007/s10723-014-9309-4>.
- [68] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. “On the Use of Cloud Computing for Scientific Workflows”. In: *2008 IEEE Fourth International Conference on eScience*. Ieee, Dec. 2008, pp. 640–645. ISBN: 978-1-4244-3380-3. DOI: 10.1109/eScience.2008.167. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4736878>.
- [69] R. Mitchum. *Unwinding the ‘Long Tail’ of Science*. URL: <https://www.ci.uchicago.edu/blog/unwinding-long-tail-science>.
- [70] P. Buncic, C. Aguado Sánchez, J. Blomer, a. Harutyunyan, and M. Mudrinic. “A practical approach to virtualization in HEP”. In: *The European Physical Journal Plus* 126.1 (2011), p. 13. ISSN: 2190-5444. DOI: 10.1140/epjp/i2011-11013-1. URL: <http://www.springerlink.com/index/10.1140/epjp/i2011-11013-1>.
- [71] P. Sethia and K. Karlapalem. “A multi-agent simulation framework on small Hadoop cluster”. In: *Engineering Applications of Artificial Intelligence* 24.7 (Oct. 2011), pp. 1120–1127. ISSN: 09521976. DOI: 10.1016/j.engappai.2011.06.009. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0952197611001096>.

-
- [72] D. Talia. “Cloud Computing and Software Agents: Towards Cloud Intelligent Services”. In: WOA. Ed. by G. Fortino, A. Garro, L. Palopoli, W. Russo, and G. Spezzano. Vol. 741. CEUR Workshop Proceedings. CEUR-WS.org, 2011, pp. 2–6. URL: http://dblp.uni-trier.de/db/conf/woa/woa2011.html%7B%5C#%7DTalia11%20http://ceur-ws.org/Vol-741/INV02%7B%5C_%7DTalia.pdf.
- [73] Z. Khan, D. Ludlow, R. McClatchey, and A. Anjum. “An architecture for integrated intelligence in urban management using cloud computing”. In: *Journal of Cloud Computing: Advances, Systems and Applications* 1.1 (2012), p. 1. ISSN: 2192-113X. DOI: 10.1186/2192-113X-1-1. URL: <http://www.cloud-casa.com/content/1/1/1>.
- [74] G. Wang, M. Salles, B. Sowell, and X. Wang. “Behavioral simulations in mapreduce”. In: *Proceedings of the VLDB Endowment*. VLDB Endowment, 2010, pp. 952–963. arXiv: arXiv:1005.3773v1. URL: <http://dl.acm.org/citation.cfm?id=1920962>.
- [75] K. Koski, K. Hormia-Poutanen, M. Chatzopoulos, Y. Legré, and B. Day. *Position Paper: European Open Science Cloud for Research*. Tech. rep. October. Bari: EUDAT, LIBER, OpenAIRE, EGI, GÉANT, 2015.
- [76] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, et al. “ROOT – A C++ framework for petabyte data storage, statistical analysis and visualization”. In: *Computer Physics Communications* 180.12 (2009), pp. 2499–2512. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2009.08.005.
- [77] T. Gunarathne, T. L. Wu, J. Y. Choi, S. H. Bae, and J. Qiu. “Cloud computing paradigms for pleasingly parallel biomedical applications”. In: *Concurrency Computation Practice and Experience* 23.17 (2011), pp. 2338–2354. ISSN: 15320626. DOI: 10.1002/cpe.1780.
- [78] A. Gupta and D. Milojicic. “Evaluation of HPC applications on cloud”. In: *Proceedings - 2011 6th Open Cirrus Summit, OCS 2011*. IEEE, 2012, pp. 22–26. ISBN: 9780769546506. DOI: 10.1109/OCS.2011.10.
- [79] S. N. Srirama, P. Jakovits, and E. Vainikko. “Adapting scientific computing problems to clouds using MapReduce”. In: *Future Generation Computer Systems* 28.1 (Jan. 2012), pp. 184–192. ISSN: 0167739X. DOI: 10.1016/j.future.2011.

- 05.025. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X11001075>.
- [80] A. Y. Rodríguez-Marrero, I. González Caballero, A. Cuesta Noriega, E. Fernández-del-Castillo, **Á. López García**, et al. “Integrating PROOF Analysis in Cloud and Batch Clusters”. In: *Journal of Physics: Conference Series* 396.3 (Dec. 2012), p. 32091. ISSN: 1742-6588. DOI: 10.1088/1742-6596/396/3/032091. URL: <http://stacks.iop.org/1742-6596/396/i=3/a=032091?key=crossref.a4219812d32b8f4fb9c750e66cfcde37>.
- [81] P. Malzacher and A. Manafov. “PROOF on Demand”. In: *Journal of Physics: Conference Series* 219.7 (2010), p. 72009. DOI: 10.1088/1742-6596/219/7/072009.
- [82] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. “A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing”. In: *Cloud computing*. Springer, 2010, pp. 115–131.
- [83] M. Mao and M. Humphrey. “A Performance Study on the VM Startup Time in the Cloud”. In: *2012 IEEE Fifth International Conference on Cloud Computing*. Ieee, June 2012, pp. 423–430. ISBN: 978-1-4673-2892-0. DOI: 10.1109/CLOUD.2012.103. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6253534>.
- [84] E. Fernández-del-Castillo, D. Scardaci, and **Á. López García**. “The EGI Federated Cloud e-Infrastructure”. In: *Procedia Computer Science* 68 (2015), pp. 196–205. ISSN: 18770509. DOI: 10.1016/j.procs.2015.09.235. URL: <http://linkinghub.elsevier.com/retrieve/pii/S187705091503080X>.
- [85] M. A. Kallio, J. T. Tuimala, T. Hupponen, P. Klemelä, M. Gentile, I. Scheinin, M. Koski, J. Käki, and E. I. Korpelainen. “Chipster: user-friendly analysis software for microarray and other high-throughput data.” In: *BMC genomics* 12.1 (2011), p. 507. ISSN: 1471-2164. DOI: 10.1186/1471-2164-12-507. URL: <http://www.biomedcentral.com/1471-2164/12/507>.
- [86] *VCycle: VM lifecycle management*. 2015. URL: <http://www.gridpp.ac.uk/vcycle/>.

-
- [87] a. McNab, F. Stagni, and M. U. Garcia. "Running Jobs in the Vacuum". In: *Journal of Physics: Conference Series* 513.3 (2014), p. 32065. ISSN: 1742-6588. DOI: 10.1088/1742-6596/513/3/032065. URL: <http://stacks.iop.org/1742-6596/513/i=3/a=032065?key=crossref.9c1d2ef7e37d7f8ebe4db79276add6d4>.
- [88] *Interoperable Global Trust Federation (IGTF)*. 2015. URL: <http://www.igtf.eu/>.
- [89] V. Venturi and F. Stagni. "Virtual organization management across middleware boundaries". In: *IEEE International Conference on e-Science and Grid Computing*. 2007, pp. 545–552. ISBN: 0769530648. DOI: 10.1109/e-Science.2007.19. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=4426930.
- [90] *Ansible*. 2015. URL: <http://www.ansible.com/>.
- [91] *Puppet*. 2015. URL: <https://puppetlabs.com/>.
- [92] *CFEngine*. 2015. URL: <http://www.cfengine.com>.
- [93] *Chef*. 2015. URL: <http://www.opscode.com/chef/>.
- [94] P. Nilsson, K. De, A. Filipcic, A. Klimentov, T. Maeno, D. Oleynik, S. Panitkin, T. Wenaus, and W. Wu. "Extending ATLAS Computing to Commercial Clouds and Supercomputers". In: *The International Symposium on Grids and Clouds (ISGC)*. Vol. 2014. 2014, pp. 1–11.
- [95] V. Fernández Albor, M. Seco, V. Méndez Muñoz, T. Fernández, J. S. Silva Pena, and R. Graciani Diaz. "Multivariate Analysis of Variance for High Energy Physics Software in Virtualized Environments". In: *International Symposium on Grids and Clouds, Academia Sinica, Taipei, Taiwan*. Vol. 160. 2015, pp. 1–15.
- [96] W. Voorsluys, S. K. Garg, and R. Buyya. "Provisioning spot market cloud resources to create cost-effective virtual clusters". In: *Lecture Notes in Computer Science* 7016 LNCS.PART 1 (2011), pp. 395–408. ISSN: 03029743. DOI: 10.1007/978-3-642-24650-0_{_}34. arXiv: 1110.5972.
- [97] G. Juve and E. Deelman. "Resource Provisioning Options for Large-Scale Scientific Workflows". In: *2008 IEEE Fourth International Conference on eScience*. Ieee, Dec. 2008, pp. 608–613. ISBN: 978-1-4244-3380-3. DOI: 10.1109/eScience.2008.160.

- [98] S. S. S. S. S. S. S. S. Manvi, G. K. Shyam, and G. Krishna Shyam. "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey". In: *Journal of Network and Computer Applications* 41 (May 2014), pp. 424–440. ISSN: 10958592. DOI: 10.1016/j.jnca.2013.10.004. URL: <http://www.sciencedirect.com/science/article/pii/S1084804513002099>.
- [99] C. Evangelinos and C. N. Hill. "Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2". In: *The 1st Workshop on Cloud Computing and its Applications (CCA)*. Vol. 2. 2.40. 2008, pp. 2–34. ISBN: 1047-6210. DOI: 10.1136/emj.2010.096966. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.296.3779%7B%5C%7Drep=rep1%7B%5C%7Dtype=pdf%20http://www.cca08.org/speakers/evangelinos.php>.
- [100] M. Michelotto, M. Alef, A. Iribarren, H. Meinhard, P. Wegner, et al. "A comparison of HEP code with SPEC ¹ benchmarks on multi-core worker nodes". In: *Journal of Physics: Conference Series* 219.5 (2010), p. 052009. ISSN: 1742-6596. DOI: 10.1088/1742-6596/219/5/052009. URL: <http://stacks.iop.org/1742-6596/219/i=5/a=052009?key=crossref.1b7af551798d30461b7c4603051d4e3e>.
- [101] J. Shamsi, M. A. Khojaye, and M. A. Qasmi. "Data-Intensive Cloud Computing: Requirements, Expectations, Challenges, and Solutions". In: *Journal of Grid Computing* 11.2 (2013), pp. 281–310. ISSN: 1570-7873. DOI: 10.1007/s10723-013-9255-6. URL: <http://link.springer.com/10.1007/s10723-013-9255-6>.
- [102] T. Kosar. "A new paradigm in data intensive computing: Stork and the data-aware schedulers". In: *Challenges of Large Applications in Distributed Environments, 2006 IEEE*. 2006, pp. 5–12. ISBN: 1-4244-0420-7. DOI: 10.1109/CLADE.2006.1652048. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1652048>.
- [103] J. Blomer, P. Buncic, I. Charalampidis, a. Harutyunyan, D. Larsen, and R. Meusel. "Status and future perspectives of CernVM-FS". In: *Journal of Physics: Conference Series* 396.5 (2012), p. 052013. ISSN: 1742-6588. DOI: 10.1088/1742-6596/

396/5/052013. URL: <http://stacks.iop.org/1742-6596/396/i=5/a=052013?key=crossref.a5336f7112426095113f65d2f9576b27>.

- [104] M. Hategan, J. Wozniak, and K. Maheshwari. “Coasters: Uniform Resource Provisioning and Access for Clouds and Grids”. In: *Fourth IEEE International Conference on Utility and Cloud Computing*. 2011, pp. 114–121. ISBN: 978-1-4577-2116-8. DOI: 10.1109/UCC.2011.25. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6123488>.
- [105] O. Adam, Y. C. Lee, and A. Y. Zomaya. “Constructing Performance-Predictable Clusters with Performance-Varying Resources of Clouds”. In: *Computers, IEEE Transactions on PP.99* (2015), p. 1. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2510648.
- [106] J. A. Stankovic and K. Ramamritham. “What is predictability for real-time systems?” In: *Real-Time Systems 2.4* (1990), pp. 247–254.
- [107] J. C. Mogul and L. Popa. “What we talk about when we talk about cloud network performance”. In: *ACM SIGCOMM Computer Communication Review 42.5* (2012), pp. 44–48.
- [108] F. Fakhfakh, H. H. Kacem, and A. H. Kacem. “Workflow Scheduling in Cloud Computing: A Survey”. In: *IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014*. Vol. 71. 9. Springer US, 2014, pp. 372–378. ISBN: 978-1-4799-5467-4. DOI: 10.1109/EDOCW.2014.61. URL: <http://ieeexplore.ieee.org/ielx7/6971861/6975303/06975385.pdf?tp=%7B%5C%7Darnumber=6975385%7B%5C%7Disnumber=6975303>.
- [109] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E.-g. Talbi. “Towards Understanding Uncertainty in Cloud Computing Resource Provisioning”. In: *Procedia Computer Science 51* (2015), pp. 1772–1781. ISSN: 18770509. DOI: 10.1016/j.procs.2015.05.387. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1877050915011953>.
- [110] M. a. Vouk. “Cloud computing – Issues, research and implementations”. In: *ITI 2008 - 30th International Conference on Information Technology Interfaces* (June 2008), pp. 31–40. DOI: 10.1109/ITI.2008.4588381. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4588381>.

- [111] I. Blanquer, G. Brasche, and D. Lezzi. “Requirements of Scientific Applications in Cloud Offerings”. In: *Proceedings of the 2012 Sixth Iberian Grid Infrastructure Conference*. 2012, pp. 173–182. ISBN: 978-989-98265-0-2.
- [112] G. Juve and E. Deelman. “Scientific workflows and clouds”. In: *Crossroads* 16.3 (Mar. 2010), pp. 14–18. ISSN: 15284972. DOI: 10.1145/1734160.1734166. URL: <http://portal.acm.org/citation.cfm?doid=1734160.1734166>.
- [113] D. Susa, H. Castro, and M. Villamizar. “Closing the Gap between Cloud Providers and Scientific Users”. In: *Cloud Computing with e-Science Applications*. CRC Press, Jan. 2015, pp. 115–140. ISBN: 978-1-4665-9115-8. DOI: doi:10.1201/b18021-7. URL: <http://dx.doi.org/10.1201/b18021-7>.
- [114] L. Ramakrishnan and P. T. T. Zbiegel. “Magellan: experiences from a science cloud”. In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. 2011, pp. 49–58. ISBN: 9781450306997. URL: <http://dl.acm.org/citation.cfm?id=1996119>.
- [115] S. Chaisiri, R. Kaewpuang, B. S. Lee, and D. Niyato. “Cost minimization for provisioning virtual servers in amazon elastic compute cloud”. In: *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings*. 2011, pp. 85–95. ISBN: 9780769544304. DOI: 10.1109/MASCOTS.2011.30.
- [116] W. Voorsluys and R. Buyya. “Reliable provisioning of spot instances for compute-intensive applications”. In: *Proceedings - International Conference on Advanced Information Networking and Applications, AINA (2012)*, pp. 542–549. ISSN: 1550445X. DOI: 10.1109/AINA.2012.106. arXiv: 1110.5969.
- [117] H. Huang, L. Wang, B. C. Tak, and C. Tang. “CAP 3: A Cloud Auto-Provisioning Framework for Parallel Processing Using On-demand and Spot Instances”. In: *IEEE Sixth International Conference on Cloud Computing (CLOUD), 2013*. 2013, pp. 228–235. URL: <http://www.cs.uwo.edu/~7B-%7Dlwang7/papers/Cloud-2013.pdf>.
- [118] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. “Virtual infrastructure management in private and hybrid clouds”. In: *IEEE Internet Computing* 13 (2009), pp. 14–22. ISSN: 10897801. DOI: 10.1109/MIC.2009.119.

-
- [119] R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. “An elasticity model for High Throughput Computing clusters”. In: *Journal of Parallel and Distributed Computing* 71.6 (June 2011), pp. 750–757. ISSN: 07437315. DOI: 10.1016/j.jpdc.2010.05.005. URL: <http://www.sciencedirect.com/science/article/pii/S0743731510000985>.
- [120] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya. “The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds”. In: *Future Generation Computer Systems* 28.6 (2012), pp. 861–870. ISSN: 0167739X. DOI: 10.1016/j.future.2011.07.005. URL: <http://dx.doi.org/10.1016/j.future.2011.07.005>.
- [121] M. Hardt, T. Jejkal, I. Campos Plasencia, E. Fernández-del-Castillo, A. Jackson, M. Weiland, B. Palak, M. Plociennik, and D. Nielsson. “Transparent Access to Scientific and Commercial Clouds from the Kepler Workflow Engine”. In: *Computing and Informatics* 31.1 (2012), p. 119. URL: <http://www.ipe.fzk.de/%7B~%7Dstotzka/publications/publications/Hardt2012.1.pdf>.
- [122] Y. C. Lee, H. Han, A. Y. Zomaya, and M. Yousif. “Resource-efficient workflow scheduling in clouds”. In: *Knowledge-Based Systems* 80 (2015), pp. 153–162. ISSN: 09507051. DOI: 10.1016/j.knosys.2015.02.012. URL: <http://www.sciencedirect.com/science/article/pii/S0950705115000556>.
- [123] Q. Jiang. “Executing Large Scale Scientific Workflows in Public Clouds”. PhD thesis. University of Sidney, 2015.
- [124] S. Smachat and K. Viriyapant. “Taxonomies of workflow scheduling problem and techniques in the cloud”. In: *Future Generation Computer Systems* 52 (2015), pp. 1–12. ISSN: 0167739X. DOI: 10.1016/j.future.2015.04.019. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0167739X15001776>.
- [125] M. A. Rodriguez and R. Buyya. “Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds”. In: *IEEE Transactions on Cloud Computing* 2.2 (2014), pp. 222–235.
- [126] X. Lin and C. Q. Wu. “On scientific workflow scheduling in clouds under budget constraint”. In: *Proceedings of the International Conference on Parallel Processing*. 2013, pp. 90–99. ISBN: 9780769551173. DOI: 10.1109/ICPP.2013.18.

- [127] D. Jung, J. Lim, H. Yu, J. Gil, and E. Lee. “A workflow scheduling technique for task distribution in spot instance-based cloud”. In: *Ubiquitous Information Technologies and Applications*. Springer, 2014, pp. 409–416.
- [128] C. Szabo, Q. Z. Sheng, T. Kroeger, Y. Zhang, and J. Yu. “Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows”. In: *Journal of Grid Computing* 12.2 (2014), pp. 245–264. ISSN: 15707873. DOI: 10.1007/s10723-013-9282-3.
- [129] B. Sotomayor, K. Keahey, and I. Foster. “Overhead Matters: A Model for Virtual Resource Management”. In: *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing SE - VTDC '06*. Washington, DC, USA: IEEE Computer Society, 2006, p. 5. ISBN: 0-7695-2873-1. URL: citeulike-article-id:2902283%20http://dx.doi.org/10.1109/VTDC.2006.9.
- [130] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu. “Resource provisioning for cloud computing”. In: *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research - CASCON '09* (2009), p. 101. DOI: 10.1145/1723028.1723041. URL: http://portal.acm.org/citation.cfm?doid=1723028.1723041.
- [131] S. K. Garg, S. K. Gopalaiyengar, and R. Buyya. “SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7016 LNCS.PART 1 (2011), pp. 371–384. ISSN: 03029743. DOI: 10.1007/978-3-642-24650-0{_}32.
- [132] C. Cardonha, M. D. Assunção, M. A. S. Netto, R. L. F. Cunha, and C. Queiroz. *Patience-aware scheduling for cloud services: Freeing users from the chains of boredom*. Ed. by S. Basu, C. Pautasso, L. Zhang, and X. Fu. Springer Berlin Heidelberg, 2013. ISBN: 9783642450044. DOI: 10.1007/978-3-642-45005-1{_}45. arXiv: arXiv:1308.4166v1.
- [133] A.-C. Orgerie, M. Assunção, and L. Lefèvre. “Energy Aware Clouds”. In: *Computer Communications and Networks*. Ed. by M. Cafaro and G. Aloisio. London: Springer London, 2011, pp. 143–166. ISBN: 978-0-85729-048-9. URL: citeulike-article-id:9050498%20http://dx.doi.org/10.1007/978-0-85729-049-6%7B%5C_%7D7.

-
- [134] A. Beloglazov, J. Abawajy, and R. Buyya. "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing". In: *Future Generation Computer Systems* 28.5 (2012), pp. 755–768. ISSN: 0167739X. DOI: 10.1016/j.future.2011.04.017. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0167739X11000689>.
- [135] J. W. Smith and I. Sommerville. "Workload Classification & Software Energy Measurement for Efficient Scheduling on Private Cloud Platforms". In: *ACM SOCC 2011*. Vol. abs/1105.2. 2011, p. 10. arXiv: 1105.2584. URL: <http://arxiv.org/abs/1105.2584>.
- [136] A. Corradi, M. Fanelli, and L. Foschini. "VM consolidation: A real case based on OpenStack Cloud". In: *Future Generation Computer Systems* 32 (Mar. 2014), pp. 118–127. ISSN: 0167739X. DOI: 10.1016/j.future.2012.05.012. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X12001082>.
- [137] M. Mazzucco, D. Dyachuk, and R. Deters. "Maximizing Cloud Providers Revenues via Energy Aware Allocation Policies". In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2011. URL: [citeulike-article-id: 8825819%20http://arxiv.org/abs/1102.3058](http://arxiv.org/abs/1102.3058).
- [138] B. Sotomayor. *Haizea*. 2009. URL: <http://haizea.cs.uchicago.edu/>.
- [139] *OpenStack Bazaar*. 2015. URL: <https://launchpad.net/blazar>.
- [140] OpenNebula Project. *OpenNebula*. 2015. URL: <http://openebula.org/>.
- [141] OpenStack Foundation. *OpenStack*. 2015. URL: <http://www.openstack.org%20http://openstack.org>.
- [142] O. Litvinski and A. Gherbi. "Experimental evaluation of OpenStack compute scheduler". In: *Procedia Computer Science* 19.Ant (2013), pp. 116–123. ISSN: 18770509. DOI: 10.1016/j.procs.2013.06.020. URL: <http://dx.doi.org/10.1016/j.procs.2013.06.020>.
- [143] Apache Foundation. *Apache CloudStack*. 2015. URL: <https://cloudstack.apache.org%20http://incubator.apache.org/cloudstack/%20http://cloudstack.apache.org/>.
- [144] Eucalyptus Systems. *Eucalyptus*. 2015. URL: <https://www.eucalyptus.com/>.

- [145] **Á. López García** and E. Fernández-del-Castillo. “Efficient image deployment in Cloud environments”. In: *Journal of Network and Computer Applications* (2016). ISSN: 1084-8045.
- [146] G. Aceto, A. Botta, W. De Donato, and A. Pescape. “Cloud monitoring: Definitions, issues and future directions”. In: *2012 1st IEEE International Conference on Cloud Networking, CLOUDNET 2012 - Proceedings*. 2012, pp. 63–67. ISBN: 9781467327985. DOI: 10.1109/CloudNet.2012.6483656.
- [147] Z. C. Z. Chen, Y. Z. Y. Zhao, X. M. X. Miao, Y. C. Y. Chen, and Q. W. Q. Wang. “Rapid Provisioning of Cloud Infrastructure Leveraging Peer-to-Peer Networks”. In: *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*. 2009, pp. 324–329. ISBN: 1545-0678. DOI: 10.1109/ICDCSW.2009.35. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5158873>.
- [148] W. Li, P. Svard, J. Tordsson, and E. Elmroth. “A General Approach to Service Deployment in Cloud Environments”. In: *Cloud and Green Computing (CGC), 2012 Second International Conference on* (2012), pp. 17–24. DOI: 10.1109/CGC.2012.90.
- [149] M. McLoughlin. *The QCOW2 image format*. 2008. URL: <https://people.gnome.org/~7B~%7Dmarkmc/qcow-image-format.html>.
- [150] B. Segal, P. Buncic, C. Aguado Sanchez, J. Blomer, D. Garcia Quintas, a. Harutyunyan, P. Mato, J. Rantala, D. Weir, and Y. Yao. “LHC Cloud Computing with CernVM”. In: *Proceedings of the 13th International Workshop on Advanced Computing and Analysis Techniques in Physics Research. February 22-27, 2010, Jaipur, India*. <http://acat2010.cern.ch/>. Published online at <http://pos.sissa.it/cgi-bin/reader/conf.cgi?confid=>. Vol. 1. 2010, p. 4. URL: http://pos.sissa.it/archive/conferences/093/004/ACAT2010%7B%5C_%7D004.pdf.
- [151] M. Femminella, E. Nunzi, G. Reali, and D. Valocchi. “Networking issues related to delivering and processing genomic big data”. In: *International Journal of Parallel, Emergent and Distributed Systems* 30.1 (2014), pp. 46–64. ISSN: 1744-5760. DOI: 10.1080/17445760.2014.929685. URL: <http://www.tandfonline.com/doi/abs/10.1080/17445760.2014.929685>.

-
- [152] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. "Science clouds: Early experiences in cloud computing for scientific applications". In: *Cloud computing and applications 2008* (2008), pp. 825–830. URL: <http://www.chinacloud.cn/download/research/ScienceClouds.pdf>.
- [153] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor. "Galaxy CloudMan: delivering cloud compute clusters". In: *BMC bioinformatics* 11 Suppl 1.Suppl 12 (Jan. 2010), S4. ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-S12-S4. URL: <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3040530%7B%5C%7Dttool=pmcentrez%7B%5C%7Drendertype=abstract>.
- [154] A. Menon, J. R. Santos, Y. Turner, J. Janakiraman, W. Zwaenepoel, G. J. Janakiraman, W. Zwaenepoel, J. Janakiraman, and W. Zwaenepoel. "Machine Environment Diagnosing Performance Overheads in the Xen Virtual Machine Environment". In: *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*. VEE '05 June. 2005, pp. 11–12. ISBN: 1-59593-047-7. DOI: 10.1145/1064979.1064984.
- [155] K. B. Ferreira, P. Bridges, and R. Brightwell. "Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection". In: *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press. 2008, pp. 1–20.
- [156] F. Petrini, D. J. D. Kerbyson, and S. Pakin. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q". In: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. SC '03 September 2001. New York, NY, USA: ACM, 2003, pp. 55–. ISBN: 1-58113-695-1. DOI: 10.1145/1048935.1050204. URL: <http://doi.acm.org/10.1145/1048935.1050204%20http://stumptown.cc.gt.atl.ga.us:8080/cse6230-hpcta-fa09/paper-pres-Black-Petrini-2003jb.pdf>.
- [157] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, V. Gupta, R. Nathuji, R. Niranjana, A. Ranadive, and P. Saraiya. "High-performance hypervisor architectures: Virtualization in hpc systems". In: *Workshop on System-level Virtualization for HPC (HPCVirt)*. 1. Citeseer. 2007. ISBN: 1595930906. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.1105%7B%5C%7Drep=rep1%7B%5C%7Dtype=pdf>.

- [158] R. Laurikainen, J. Laitinen, P. Lehtovuori, and J. K. Nurminen. “Improving the Efficiency of Deploying Virtual Machines in a Cloud Environment”. In: *2012 International Conference on Cloud and Service Computing*. Ieee, Nov. 2012, pp. 232–239. ISBN: 978-1-4673-4724-2. DOI: 10.1109/CSC.2012.43. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6414505>.
- [159] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, and U. Schwick-erath. “Image Distribution Mechanisms in Large Scale Cloud Providers”. In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science* (Nov. 2010), pp. 112–117. DOI: 10.1109/CloudCom.2010.73.
- [160] Y. Chen, T. Wo, and J. Li. “An Efficient Resource Management System for On-Line Virtual Cluster Provision”. In: *2009 IEEE International Conference on Cloud Computing* (2009), pp. 72–79. DOI: 10.1109/CLOUD.2009.64. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5284146>.
- [161] C. Peng, M. Kim, Z. Zhang, and H. Lei. “VDN: Virtual machine image distribution network for cloud data centers”. In: *2012 Proceedings IEEE INFOCOM* (Mar. 2012), pp. 181–189. DOI: 10.1109/INFCOM.2012.6195556. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6195556>.
- [162] H. A. Lagar-Cavilla, J. a. Whitney, R. Bryant, P. Patchin, M. Brudno, E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell. “SnowFlock: Virtual Machine Cloning as a First-Class Cloud Primitive”. In: *ACM Transactions on Computer Systems* 29.1 (Feb. 2011), pp. 1–45. ISSN: 07342071. DOI: 10.1145/1925109.1925111. URL: <http://portal.acm.org/citation.cfm?doid=1925109.1925111>.
- [163] B. Nicolae, F. Cappello, and G. Antoniu. “Optimizing multi-deployment on clouds by means of self-adaptive prefetching”. In: *Euro-Par 2011 Parallel Processing*. Vol. 6852 LNCS. 2011, pp. 503–513. ISBN: 9783642233999. DOI: 10.1007/978-3-642-23400-2_{_}46.
- [164] OpenStack Foundation. *OpenStack Rally*. 2015. URL: <https://wiki.openstack.org/wiki/Rally>.

-
- [165] B. Cohen. *BEP 3: The Bittorrent Protocol Specification*. Tech. rep. BitTorrent.org, 2008.
- [166] A. Norberg. *libtorrent*. 2015. URL: <http://www.libtorrent.org/>.
- [167] **Á. López García**. *OpenStack Compute Scheduler Cache Aware*. URL: <https://blueprints.launchpad.net/nova/+spec/cache-aware-weigher>.
- [168] D. E. Knuth. *The Art of Computer Programming (Volume 2)*. Addison-Wesley, 1981. ISBN: 0201038226. URL: <http://profs.scienze.univr.it/%7B-%7Dmanca/storia-informatica/mmix.pdf>.
- [169] J. Ansel, K. Aryay, and G. Coopermany. “DMTCP: Transparent checkpointing for cluster computations and the desktop”. In: *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–12.
- [170] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. *Checkpoint and migration of UNIX processes in the Condor distributed processing system*. Computer Sciences Department, University of Wisconsin, 1997.
- [171] P. H. Hargrove and J. C. Duell. “Berkeley lab checkpoint/restart (blcr) for linux clusters”. In: *Journal of Physics: Conference Series*. Vol. 46. 1. IOP Publishing, 2006, p. 494.
- [172] Amazon. *Amazon EC2 Spot Instances*. 2015. URL: <http://aws.amazon.com/ec2/purchasing-options/spot-instances/>.
- [173] *Google Compute Engine*. 2015. URL: <https://cloud.google.com/products/compute-engine>.
- [174] *Google Compute Engine Preemptible Virtual Machines*. 2015. URL: <https://cloud.google.com/preemptible-vms/>.
- [175] I. Menache and O. Shamir. “On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud”. In: *11th International Conference on Autonomic Computing (ICAC 14) (2014)*, pp. 177–187. URL: <https://www.usenix.org/conference/icac14/technical-sessions/presentation/menache>.

- [176] N. Chohan, C. Castillo, M. Spreitzer, and M. Steinder. "See Spot Run: Using Spot Instances for MapReduce Workflows". In: *HotCloud 2010* (2012), pp. 1–7. URL: papers2://publication/uuid/F72C8382-8639-45E6-8B12-957738AA3973.
- [177] H. Liu. "Cutting MapReduce Cost with Spot Market". In: *USENIX HotCloud'11* (2011), p. 5.
- [178] Y. Song, M. Zafer, and K.-W. Lee. "Optimal bidding in spot instance market". In: *INFOCOM, 2012 Proceedings IEEE*. Mar. 2012, pp. 190–198. DOI: 10.1109/INFCOM.2012.6195567.
- [179] K. Sowmya and R. P. Sundarraj. "Strategic Bidding for Cloud Resources under Dynamic Pricing Schemes". In: *Cloud and Services Computing (ISCOS), 2012 International Symposium on*. Dec. 2012, pp. 25–30. DOI: 10.1109/ISCOS.2012.28.
- [180] S.-Y. Noh, S. C. Timm, and H. Jang. "Vcluster: a Framework for Auto Scalable Virtual Cluster System in Heterogeneous Clouds". In: *Cluster Computing* 17.3 (2013), pp. 741–749. ISSN: 1386-7857. DOI: 10.1007/s10586-013-0292-5. URL: <http://link.springer.com/10.1007/s10586-013-0292-5>.
- [181] a. Andrzejak, D. Kondo, and S. Y. S. Yi. "Decision Model for Cloud Computing under SLA Constraints". In: *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. 2010. ISBN: 978-1-4244-8181-1. DOI: 10.1109/MASCOTS.2010.34.
- [182] S. Yi, D. Kondo, and A. Andrzejak. "Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud". In: *Proceedings - 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD 2010* (2010), pp. 236–243. DOI: 10.1109/CLOUD.2010.35.
- [183] S. Yi, A. Andrzejak, and D. Kondo. "Monetary cost-aware checkpointing and migration on amazon cloud spot instances". In: *IEEE Transactions on Services Computing* 5.4 (2012), pp. 512–524. ISSN: 19391374. DOI: 10.1109/TSC.2011.44.
- [184] S. Khatua and N. Mukherjee. "Application-centric resource provisioning for Amazon EC2 spot instances". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*

-
- 8097 LNCS (2013), pp. 267–278. ISSN: 03029743. DOI: 10.1007/978-3-642-40047-6_{_}_{}29. arXiv: arXiv:1211.1279v1.
- [185] D. Jung, S. Chin, K. Chung, H. Yu, and J. Gil. “An efficient checkpointing scheme using price history of spot instances in cloud computing environment”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6985 LNCS (2011), pp. 185–200. ISSN: 03029743. DOI: 10.1007/978-3-642-24403-2_{_}_{}16.
- [186] A. Nadjaran Toosi, F. Khodadadi, and R. Buyya. “SipaaS: Spot instance pricing as a Service framework and its implementation in OpenStack”. In: *Concurrency and Computation: Practice and Experience* (2015), n/a–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.3749. URL: <http://dx.doi.org/10.1002/cpe.3749>.
- [187] **Á. López García**. *OpenStack Spot Instances Support Specification*. URL: <https://blueprints.launchpad.net/nova/+spec/spot-instances>.
- [188] N. Borenstein and J. Blake. “Cloud computing standards: Where’s the beef?” In: *IEEE Internet Computing* 15.3 (2011), pp. 74–78. ISSN: 10897801. DOI: 10.1109/MIC.2011.58.
- [189] L. Schubert, K. Jeffery, and B. Neidecker-Lutz. *A Roadmap for Advanced Cloud Technologies under H2020*. Tech. rep. December. European Commission, 2012, p. 30.
- [190] L. Schubert, K. Jeffery, and B. Neidecker-Lutz. *The Future of Cloud Computing. Opportunities for European Cloud Computing Beyond 2010*. Tech. rep. European Commission, 2010, p. 66. DOI: 10.1016/B978-1-59749-537-0.00012-0.
- [191] E. Fernández-del-Castillo, **Á. López García**, I. Campos Plasencia, M. A. Nuñez Vega, J. Marco de Lucas, et al. “IberCloud: federated access to virtualized resources”. In: *Proceedings of the IBERGRID 2012 Conference*. Lisbon, Nov. 2012, pp. 195–205.
- [192] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. “Blueprint for the intercloud - Protocols and formats for cloud computing interoperability”. In: *Proceedings of the 2009 4th International Conference on Internet and Web Applications and Services, ICIW 2009* (2009), pp. 328–336. DOI: 10.1109/ICIW.2009.55.

- [193] B. Parák and Z. Sustr. “Challenges in Achieving IaaS Cloud Interoperability across Multiple Cloud Management Frameworks”. In: *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*. IEEE, 2014, pp. 404–411.
- [194] B. Parák, Z. Sustr, F. Feldhaus, P. Kasprzak, and M. Srba. “The rOCCI Project – Providing Cloud Interoperability with OCCI 1.1”. In: (2014), pp. 1–15.
- [195] G. S. Machado, D. Hausheer, and B. Stiller. “Considerations on the Interoperability of and between Cloud Computing Standards”. In: *Scenario Section 4 (2009)*, pp. 1–4. URL: <http://www.csg.uzh.ch/publications/ogf27-g2cnet-discussion-cc-standards-finalversion.pdf>.
- [196] G. a. Lewis. *The Role of Standards in Cloud- Computing Interoperability*. Tech. rep. CMU/SEI-2012-TN-012. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2012, p. 39. URL: <http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=28017%20http://www.sei.cmu.edu/reports/12tn012.pdf>.
- [197] P. Harsh, F. Dudouet, R. G. Cascella, Y. Jégou, and C. Morin. “Using Open Standards for Interoperability - Issues, Solutions, and Challenges facing Cloud Computing”. In: *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualization management (svm)*. 2012, pp. 435–440. ISBN: 9783901882487. arXiv: 1207.5949. URL: <http://arxiv.org/abs/1207.5949>.
- [198] Z. Zhang, C. Wu, and D. W. L. Cheung. “A Survey on Cloud Interoperability: Taxonomies, Standards, and Practice”. In: *SIGMETRICS Perform. Eval. Rev.* 40.4 (2013), pp. 13–22. ISSN: 0163-5999. DOI: 10.1145/2479942.2479945. URL: <http://doi.acm.org/10.1145/2479942.2479945>.
- [199] D. W. Chadwick, K. Siu, C. Lee, Y. Fouillat, and D. Germonville. “Adding Federated Identity Management to OpenStack”. In: *Journal of Grid Computing* 12.1 (2014), pp. 3–27. ISSN: 1570-7873. DOI: 10.1007/s10723-013-9283-2. URL: <http://link.springer.com/10.1007/s10723-013-9283-2>.
- [200] Hewlett Packard Enterprise. *HPE Helion Eucalyptus - Extend your private cloud to work with AWS*. Tech. rep., p. 2015.
- [201] EUBrazil Cloud Connect. *EUBrazil Cloud Connect*. 2015. URL: <http://www.eubrazilcloudconnect.eu/>.

-
- [202] M. J. Dias de Lima and G. Baptista. *D3.1 Adaptation Requirements for CSGrid Middleware*. Tech. rep. EU-Brazil Cloud Connect, 2014, pp. 1–20.
- [203] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze. “Cloud Federation”. In: *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. 2011. ISBN: 9781612081533.
- [204] Amazon. *Amazon Web Services*. 2015. URL: <http://aws.amazon.com>.
- [205] S. S. Y. Shim, G. Bhalla, and V. Pendyala. “Federated Identity Management”. In: *Computer* 38.September (2001), pp. 120–122. ISSN: 00189162. DOI: 10.1109/MC.2005.408.
- [206] International Telecommunication Union (ITU). *Definition of "Open Standards"*. 2015. URL: <http://www.itu.int/en/ITU-T/ipr/Pages/open.aspx>.
- [207] *Open Grid Forum (OGF)*. 2015. URL: <https://www.ogf.org%20http://www.ogf.org/>.
- [208] T. Metsch and A. Edmonds. *Open cloud computing interface-RESTful HTTP rendering*. Tech. rep. Open Grid Forum, 2011.
- [209] R. Nyrén, T. Metsch, A. Edmonds, and A. Papaspyrou. *Open cloud computing interface-core*. Tech. rep. Open Grid Forum, 2010. URL: https://ogf.org/Public%7B%5C_%7DComment%7B%5C_%7DDocs/Documents/2010-12/ogf%7B%5C_%7Ddraft%7B%5C_%7Docci%7B%5C_%7Dcore.pdf.
- [210] T. Metsch and A. Edmonds. “Open Cloud Computing Interface – infrastructure”. In: *Open Grid Forum-OCCI Working group (2010)*. URL: http://ogfweb.ogf.org/Public%7B%5C_%7DComment%7B%5C_%7DDocs/Documents/2010-12/ogf%7B%5C_%7Ddraft%7B%5C_%7Docci%7B%5C_%7Dinfrastructure.pdf.
- [211] DMTF Cloud Management Working Group. *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol*. ISO ISO/IEC 19831:2015. Technical report, Distributed Management Work Force (DMTF), 2012.
- [212] *Distributed Management Task Force (DMTF)*. 2015. URL: <https://www.dmtf.org>.
- [213] DMTF Cloud Management Working Group. *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol – An Interface for Managing Cloud Infrastructure*. ISO ISO/IEC 19831:2015. ISO, 2015.

- [214] P. (T. Lipton, S. (Moser, D. (Palma, and T. (Spatzier. *Topology and Orchestration Specification for Cloud Applications*. Tech. rep. March. OASIS Standard, 2013, pp. 1–114. URL: <http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html>.
- [215] *Organization for the Advancement of Structured Information Standards (OASIS)*. 2015. URL: <https://www.oasis-open.org>.
- [216] Storage Networking Industry Association. *Cloud data management interface (CDMI)*. Tech. rep. Storage Networking Industry Association, 2015, p. 276.
- [217] Storage Networking Industry Association (SNIA). *Cloud Data Management Interface (CDMI)*. ISO ISO/IEC 17826:2012. ISO, 2015.
- [218] DMTF OVF Workgroup. *Open Virtualization Format Specification (OVF)*. Tech. rep. DMTF Standard, 2014.
- [219] R. Housley, W. F. T. Polk, and D. Solo. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. 1999. URL: <http://www.ietf.org/rfc/rfc2459.txt>.
- [220] S. Tuecke, V. Welch, and D. Engert. *Internet X.509 public key infrastructure (PKI) proxy certificate profile*. Tech. rep. Internet Engineering Task Force, 2004, pp. 1–37.
- [221] **Á. López García**, E. Fernández-del-Castillo, and M. Puel. “Identity Federation with VOMS in Cloud Infrastructures”. In: *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. 2013, pp. 42–48. ISBN: 978-0-7695-5095-4. DOI: 10.1109/CloudCom.2013.13. URL: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6753776>.
- [222] OASIS. *Assertions and Protocols for the OASIS security assertion markup language (SAML) v2.0*. Tech. rep. March. 2005, pp. 1–86. URL: <https://svn.softwareborsen.dk/sosi-gw/tags/release-1.1.4/vendor/doc/saml-core-2.0-os.pdf>.
- [223] R. L. B. Morgan, S. Cantor, S. Carmody, W. Hoehn, and K. Klingenstein. “Federated Security: The Shibboleth Approach”. In: *EDUCAUSE quarterly* 27.4 (2004), pp. 12–17.

-
- [224] W. Qiang, A. Konstantinov, D. Zou, and L. T. Yang. “A standards-based interoperable single sign-on framework in ARC Grid middleware”. In: *Journal of Network and Computer Applications* 35.3 (2012), pp. 892–904.
- [225] R. Murri, P. Z. Kunszt, S. Maffioletti, and V. Tschopp. “GridCertLib: a single sign-on solution for Grid web applications and portals”. In: *Journal of Grid Computing* 9.4 (2011), pp. 441–453.
- [226] D. Hardt. *The OAuth 2.0 authorization framework*. Tech. rep. Internet Engineering Task Force, 2012.
- [227] B. Leiba. “OAuth web authorization protocol”. In: *IEEE Internet Computing* 16.1 (2012), pp. 74–77. ISSN: 10897801. DOI: 10.1109/MIC.2012.11.
- [228] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. “Openid connect core 1.0”. In: *The OpenID Foundation* (2014).
- [229] *Open Researcher and Contributor ID (ORCID)*. 2015. URL: <http://orcid.org/>.
- [230] S. Andreozzi, B. Stephen, L. Field, S. Fisher, J. Jensen, K. Balazs, M. Viljoen, A. Wilson, and R. Zappi. “GLUE Schema Specification version 1.3”. In: *Open Grid Forum-Glue Schema Working group* (2007), pp. 1–26.
- [231] S. Andreozzi, S. Burke, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. P. Navarro. *GLUE Specification v. 2.0*. Tech. rep. 2009, p. 76. URL: <http://www.ogf.org/documents/GFD.147.pdf>.
- [232] A. Cristofori, J. K. Nilsen, J. Gordon, M. Jones, J. A. Kennedy, and R. Müller-Pfefferkorn. “Usage Record – Format Recommendation”. In: *Open Grid Forum* (2013), pp. 1–62.
- [233] E. Elmroth, F. G. Márquez, D. Henriksson, and D. P. Ferrera. “Accounting and billing for federated cloud infrastructures”. In: *8th International Conference on Grid and Cooperative Computing, GCC 2009* (2009), pp. 268–275. DOI: 10.1109/GCC.2009.37.
- [234] UK Government Cabinet Office. *Open Standards Principles*. 2015. URL: <https://www.gov.uk/government/publications/open-standards-principles/open-standards-principles>.

- [235] **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “OpenStack OCCI Interface”. In: *SoftwareX* (2016). ISSN: 2352-7110. DOI: 10.1016/j.softx.2016.01.001.
- [236] **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. *OpenStack OCCI Interface*. 2016. URL: <https://github.com/openstack/ooi>.
- [237] Open Grid Forum (OGF). *OCCI Working Group*. 2015. URL: <https://www.ogf.org>.
- [238] E. Fernandez. *OCCI Contextualization Extension*. 2015. URL: <https://wiki.egei.eu/wiki/Fedcloud-tf:WorkGroups:Contextualisation%7B%5C#%7DContextualization>.
- [239] *cloud-init*. 2015. URL: <https://launchpad.net/cloud-init>.
- [240] *Flamingo*. 2015. URL: <https://github.com/tmrts/flamingo>.
- [241] F-W. project. *OCCI Key Pair extension*. 2015. URL: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/DCRM%7B%5C_%7D0CCI%7B%5C_%7D0pen%7B%5C_%7DRESTful%7B%5C_%7DAPI%7B%5C_%7DSpecification%7B%5C#%7DKey%7B%5C_%7DPair%7B%5C_%7DExtension.
- [242] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. “IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures”. In: *Computer* 45.12 (2012), pp. 65–72. ISSN: 0018-9162. DOI: <http://doi.ieeecomputersociety.org/10.1109/MC.2012.76>.
- [243] *VMWare*. 2015. URL: <http://www.vmware.com>.
- [244] *Amazon Elastic Compute Cloud (EC2)*. 2015. URL: <http://aws.amazon.com/ec2/%20https://aws.amazon.com/ec2/>.
- [245] *OCCI-OS*. 2015. URL: <https://wiki.openstack.org/wiki/Occi>.
- [246] T. Metsch and C. Smith. *Service Sharing Facility*. 2015. URL: <http://pyssf.sourceforge.net>.
- [247] P. J. Eby. *Python Web Server Gateway Interface v1.0.1*. Sept. 2010. URL: <http://legacy.python.org/dev/peps/pep-3333/>.
- [248] OpenStack Foundation. *OpenStack APIs*. 2015. URL: <http://developer.openstack.org/>.

-
- [249] OpenStack Foundation. *OpenStack Compute API v2.1*. 2015. URL: <http://developer.openstack.org/api-ref-compute-v2.1.html>.
- [250] R. Alfieri, R. Cecchini, V. Ciaschini, L. Dell’Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. “VOMS, an authorization system for virtual organizations”. In: *Proceedings of the 1st European Across Grids Conference*. Springer. 2004, pp. 33–40.
- [251] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, et al. “The open science grid”. In: *Journal of Physics: Conference Series*. Vol. 78. 1. IOP Publishing. 2007, p. 012057. doi: 10.1088/1742-6596/78/1/012057.
- [252] *European Middleware Initiative (EMI)*. 2015. URL: <http://www.eu-emi.eu/>.
- [253] *The Globus Toolkit*. 2015. URL: <http://www.globus.org/toolkit/>.
- [254] *SQLAlchemy*. 2015. URL: <http://www.sqlalchemy.org/>.
- [255] **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “Standards for enabling heterogeneous IaaS cloud federations”. In: *Computer Standards & Interfaces* (2016). ISSN: 0920-5489.
- [256] **Á. López García**, E. Fernández-del-Castillo, and P. Orviz Fernández. “Resource provisioning in Science Clouds: requirements and challenges”. In: *Journal of Grid Computing* (2016). ISSN: 1572-9184.
- [257] I. Blanquer, G. Donvito, P. Fuhrmann, **Á. López García**, and G. Molto. *An integrated IaaS and PaaS architecture for scientific computing*. Oral Contribution. EGI Community Forum: Bari (Italy), Nov. 10–13, 2015.
- [258] **Á. López García**, P. Fuhrmann, G. Donvito, and A. Chierici. *Improving IaaS resources to accommodate scientific applications*. Oral Contribution. HEPiX Fall 2015 Workshop: Brookhaven National Laboratory, New York (USA), Oct. 12–16, 2015.
- [259] **Á. López García**, P. Orviz Fernández, F. Aguilar, E. Fernández-del-Castillo, I. Campos Plasencia, and J. Marco de Lucas. “The role of IBERGRID in the Federated Cloud of EGI”. In: *Proceedings of the IBERGRID 2014 Conference*. Editorial Universidad Politecnica de Valencia, 2014, pp. 3–14. ISBN: 978-84-9048-246-9.

- [260] **Á. López García**. *OpenStack Cloud Workshop*. Workshop. 8th Iberian Grid Computing Conference – IBERGRID 2014: University of Aveiro, Aveiro (Portugal), Sept. 8–10, 2014.
- [261] **Á. López García**. *OpenStack hands on*. Workshop. EGI Community Forum: Helsinki (Finland), May 19–23, 2014.
- [262] **Á. López García** and P. Orviz Fernández. *Integrating cloud computing within an existing infrastructure*. Poster. EGI Technical Forum: Madrid (Spain), Sept. 16–20, 2013.
- [263] **Á. López García** and E. Fernández-del-Castillo. *Analysis of Scientific Cloud Computing requirements*. Oral Contribution. EGI Technical Forum: Madrid (Spain), Sept. 16–20, 2013.
- [264] **Á. López García** and E. Fernández-del-Castillo. *Analysis of Scientific Cloud Computing requirements*. Oral Contribution. 7th Iberian Grid Computing Conference – IBERGRID 2013: Madrid (Spain), Sept. 19–20, 2013.
- [265] E. Fernández-del-Castillo, I. Campos, S. Heinemeyer, **Á. López García**, and F. Pahlen. *Phenomenology Tools on a OpenStack Cloud Infrastructure*. Oral Contribution. EGI Community Forum 2013: The University of Manchester, Manchester (UK), Apr. 8–12, 2012.
- [266] M. Airaj, C. Cavet, V. Hamar, M. Jouvin, C. Loomis, et al. “Vers une fédération de Cloud Académique dans France Grilles”. In: *Journées SUCCES 2013*. Paris, France, Nov. 2013. URL: <https://hal.archives-ouvertes.fr/hal-00927506>.
- [267] E. Fernández-del-Castillo, **Á. López García**, I. Campos, M. Á. Nuñez, J. Marco, et al. *IberCloud: federated access to virtualized resources*. Oral Contribution. EGI Technical Forum 2012: Prague (Czech Republic), Sept. 17–21, 2012.
- [268] E. Fernández-del-Castillo, **Á. López García**, I. Campos, M. Á. Nuñez, J. Marco, et al. *IberCloud: federated access to virtualized resources*. Oral Contribution. 6th Iberian Grid Computing Conference – IBERGRID 2012: Lisbon (Portugal), Sept. 7–9, 2012.
- [269] **Á. López García** and M. Puel. *France-Grilles dans la Federation Cloud Task-Force d’EGI*. Oral Contribution. Atelier Operations France-Grilles: INRA, Villenave d’Ornon (France), Nov. 29–30, 2012.

-
- [270] **Á. López García** and M. Puel. *Cloud Computing at CC-IN2P3*. Invited Talk. Rencontre LCG-France: SUBATECH, Nantes (France), Sept. 18–19, 2012.
- [271] **Á. López García** and E. Fernández-del-Castillo. *Keystone-VOMS*. URL: <https://github.com/IFCA/keystone-voms>.
- [272] **Á. López García** and E. Fernández-del-Castillo. *cASO: OpenStack Accounting Extractor*. URL: <https://github.com/IFCA/caso>.
- [273] **Á. López García** and E. Fernández-del-Castillo. *cloud-bdii-provider*. URL: <https://github.com/EGI-FCTF/cloud-bdii-provider>.
- [274] **Á. López García**. *OpenStack Compute Scheduler weight normalization*. URL: <https://blueprints.launchpad.net/nova/+spec/normalize-scheduler-weights>.
- [275] *INDIGO-Datacloud*. 2015. URL: <http://indigo-datacloud.eu>.
- [276] *Son of Grid Engine*. 2015. URL: <https://arc.liv.ac.uk/trac/SGE>.
- [277] E. Parzen. “On estimation of a probability density function and mode”. In: *The annals of mathematical statistics* (1962), pp. 1065–1076.
- [278] R. T. Fielding. “Architectural styles and the design of network-based software architectures”. PhD thesis. University of California, Irvine, 2000.

List of Terms and Acronyms

A

API

Application Programming Interface. 20, 22, 52, 135–138, 140, 147, 149–154, 156, 168, 185, 189

AWS

Amazon Web Services. 63, 136, 137

C

CDMI

Cloud Data Management Interface. 135, 140

CIMI

Cloud Infrastructure Management Interface. 139, 140, 143

CMF

Cloud Management Framework. 3, 43, 49, 51, 52, 54, 58, 60, 62, 69, 75, 76, 79, 83–85, 87, 90, 92–94, 107, 115, 117, 133, 135, 137–139, 142, 147, 148, 150, 157, 161, 163, 168, 169, 175, 179–182

CoW

Copy on Write. 87, 88

D**DMTF**

Distributed Management Task Force. 140, 143

E**EC2**

Elastic Compute Cloud (EC2). 115, 116, 137, 148

EGI

European Grid Infrastructure. 32, 33, 40, 41, 49–52, 64, 66, 136, 142, 156, 157, 168, 169, 175, 176, 179

F**FTP**

File Transfer Protocol. 95–102

G**GLUE**

Grid Laboratory Uniform Environment. 141–143, 180

H**HEP**

High Energy Physics. 45, 47, 59, 60

HPC

High Performance Computing. 14, 19, 30, 32, 40, 63

HTC

High Throughput Computing. 14, 32, 33, 40

HTTP

Hiper Text Transfer Protocol. 20, 57, 60, 86, 94–107, 140, 147, 149, 154

I

I/O

Input/Output. 45, 58, 89, 180

IaaS

Infrastructure as a Service. 20, 21, 39, 42, 43, 49, 51, 62, 67, 68, 75, 83–85, 90, 133, 139, 140

IT

Information Technology. 13, 17, 41, 53

J

JSON

JavaScript Object Notation. 153, 164–167

K

Kernel Density Estimation (KDE)

KDE is a non-parametric way of estimating the probability density function population [277]. 105, 107

L

LHC

Large Hadron Collider. 29

N

NAS

Network Attached Storage. 89

NIST

National Institute of Standards and Technology. 13

O

OASIS

Organization for the Advancement of Structure Information Standards. 140, 141, 143

OCCI

Open Cloud Computing Interface. 3, 8, 135, 139, 140, 143, 147–157, 168, 175, 179, 181, 189

OGF

Open Grid Forum. 140, 142, 143, 147, 179

OS

Operating System. 18, 19, 21, 40–42, 47, 48, 137, 181, 186, 187, 189

OSG

Open Science Grid Consortium. 33

OVF

Open Virtualization Format. 135, 140, 143

P

P2P

Peer-to-Peer. 90, 91, 93, 96, 99, 107

PaaS

Platform as a Service. 20–22, 53

PROOF

Parallel ROOT Facility. 45–48, 55

R

REST

Representational State Transfer (REST) is the software architectural style of the World Wide Web. REST was defined by Roy Thomas Fielding [278]. RESTful systems typically, but not always, communicate over HTTP using the HTTP verbs GET, POST, PUT, DELETE, etc. REST systems interface with external systems as web resources identified by URI. 20, 137, 140, 147, 151

RP

Resource Provider. 16, 75, 122, 127, 161–163, 168

S

SaaS

Software as a Service. 20, 22, 67, 134

SAN

Storage Area Network. 89

SLA

Service Level Agreement. 16, 56, 64, 68, 83

SNIA

Storage Networking Industry Association. 140

SR-IOV

Single Root I/O Virtualization. 58

SSH

Secure Shell. 148

swarm

All the peers sharing a torrent. 96

T

TOSCA

Topology and Orchestration Specification for Cloud Applications. 140, 143

U

UR

Usage Record. 142, 143, 179

URL

Uniform Resource Locator. 153

V

vCPU

virtual CPU. 59, 62, 76

VM

Virtual Machine. 17–19, 49, 51, 57, 60, 68, 75–77, 83–86, 88, 89, 92, 93, 95, 101, 102, 115, 117, 118, 124–127, 148, 156, 175, 181

VMI

Virtual Machine Image. 3, 49, 51, 53, 83–85, 89, 92, 99, 100, 107, 140, 175, 181

VMM

Virtual Machine Monitor. 17–19, 90, 181

VO

Virtual Organization. 162, 163, 166–169

VOMS

Virtual Organization Membership Service. 3, 8, 52, 141, 161, 163, 165–168, 176

W

WLCG

Worldwide LHC Computing Grid. 29–31

WSGI

Web Server Gateway Interface. 149–153, 163, 165, 166, 168

X

X.509

X.509 is a standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI), specifying standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. 3, 52, 141, 143

