Tesis Doctoral

Agregación de infraestructuras computacionales usando técnicas de meta-planificación centradas en el usuario

Aggregation of computing infrastructures by using user-centered meta-scheduling techniques

Doctorado en Matemáticas y Computación

Autor: Jose Carlos Blanco
Director: Antonio S. Cofiño

Abril - 2017

Agradecimientos

Resumen

La comunidad científica en la actualidad se caracteriza por la gran demanda de capacidad de cálculo y almacenamiento. Tradicionalmente, las comunidades científicas han resuelto este tipo de necesidades mediante sus propios sistemas de cálculo y almacenamiento (ordenadores personales, *workstations, clusters…*), con el consiguiente esfuerzo adicional y el coste añadido de recursos materiales y humanos. Esta coyuntura además puede suponer una limitación para la realización de tareas y proyectos que demandan gran capacidad de cálculo. Por ejemplo, la ejecución de modelos numéricos de predicción meteorológica requiere una infraestructura de supercomputación, sólo al alcance de muy pocas instituciones.

Una alternativa a los recursos privados, son el uso de e-Infraestructuras distribuidas geográficamente a nivel nacional e internacional como RES (Red Española de Supercomputación), EGI (European Grid Initiative) o PRACE (Partnership for Advanced Computing in Europe). Estas infraestructuras no sólo proveen a los investigadores de recursos HPC (computación de altas prestaciones) y HTC (computación de alta productividad), si no que también ofrecen sistemas de almacenamiento de datos, instrumentación avanzada y repositorios, todos ellos unidos entre sí mediante servicios y redes de alto rendimiento con el objetivo de mejorar la productividad en la investigación y permitir avances que de otro modo no serían posibles. Sin embargo, el acceso, gestión y uso de estos recursos suele suponer un esfuerzo añadido a la propia investigación, propiciando que muchos investigadores no usen nuevas infraestructuras. De hecho, otra demanda actual de la comunidad investigadora es la gestión eficiente e integradora de estos recursos para centrar sus esfuerzos en la investigación, y no en los diversos requisitos para acceder a los mismos.

En esta Tesis se aplican técnicas para la agregación de recursos computacionales con un diseño centrado en el usuario, mediante herramientas de meta-planificación aplicadas a la ejecución de tareas computacionales. Estas herramientas se caracterizan por ofrecer al usuario final un acceso único y transparente a diferentes infraestructuras computacionales. El objetivo es que el usuario pueda disponer de un conjunto agregado de recursos que abarque desde los suyos personales hasta las grandes infraestructuras a las que pueda tener acceso.

La comunidad científica actual necesita disponer de una herramienta capaz de proveer de un entorno que harmonice tareas tales como: el descubrimiento de recursos, ejecución y supervisión de

trabajos, transferencia y gestión de datos remotos sin necesidad de modificar la infraestructura computacional existente. Para este fin, se plantearán los requerimientos y las especificaciones de diseño que intenten cubrir las necesidades de la comunidad científica. Estas técnicas además han de garantizar una implementación estable, escalable y robusta que cumpla los requerimientos a establecer. El resultado que se pretende obtener es dotar a la comunidad científica de un entorno con un alto grado de accesibilidad y usabilidad, para la ejecución de trabajos en todos los recursos de computación disponibles, sin importar su tipo, simplificando el acceso a los mismos para llevar a cabo sus trabajos de forma eficiente y productiva.

Abstract

The scientific community is currently characterized by the high demand for calculation and storage capacity. It has addressed these kind of needs by the use of private calculation and storage systems (PCs, workstations, clusters...), involving considerable effort in terms of computing resources and manpower costs, hindering certain tasks and projects that require high capacity calculation resources. For instance, the execution of numerical weather prediction models requires supercomputing infrastructures available to very few institutions.

e-Infrastructures geographically distributed at national and international level such as RES (Red Española de Supercomputación), EGI (European Grid Initiative) or PRACE (Partnership for Advanced Computing in Europe) are alternatives to private resources. These infrastructures not only provide researchers with HPC (High Performance Computing) and HTC (High Throughput Computing) resources, but also offer data storage systems, advanced instrumentation, repositories, and high performance networks in order to improve scientific advances. However, the access and the management of these resources usually involve an additional effort, which in practice means that many researchers do not use them. Therefore, another current demand in research communities is for an efficient and integrated management of resources that allows researchers to concentrate on their research and not on access to resources.

This thesis aims to apply techniques to aggregate computational resources with a user-centered design, employing meta-scheduling tools to execute computational tasks. These tools offer end users unique and transparent access to different computer infrastructures, with the goal of providing the user with the capacity to aggregate resources such as private computing resources or large infrastructures.

The contemporary scientific community needs a tool capable of offering an environment that harmonizes tasks such as resource discovery, job execution and supervision, and remote data management, without the need for modifying existing computational infrastructures. For this purpose, this thesis considers the necessary set of requirements and design specifications, and provides the scientific community with an accessible and usable environment to run jobs efficiently on all available computing resources, regardless of type, simplifying access to them.

1 Índice de contenido

2 Índice de figuras

3 Índice de tablas

Capítulo 1. Introduction

1.1 Motivation

Modern research requires integrated services in order to manage scientific infrastructures. In the last years, there are more and more infrastructure services available, in some form, all over the world. These infrastructures have facilities for high-performance computing, high-throughput computing and storage, advanced networks for research and education, etc. However, the resources available vary considerably in terms of access, type, functionality, capacity and governance model among different research groups, institutes, universities and countries. These differences are natural, as infrastructures are organised by different types of resources, institutes, or countries, which often restrict or hamper the access to research communities.

As a result, there are several projects and institutions devoted to the development of e-Infrastructures. In fact, there is already an extensive ecosystem of e-Infrastructures with the goal of increasing international research collaboration [1]. But among them at present there is a clear lack of cohesion. There are often separate e-Infrastructures for high-performance computing, high-throughput computing, storage, and other services, generally with very different interfaces. The result is that users, researchers and research communities must devote time and effort to

gaining access to these services instead doing research. They all need infrastructure services that are well-integrated from a users' standpoint; otherwise, the multiple e-Infrastructures are more of a hindrance than a help.

The need to process an ever-increasing amount of calculation is more and more challenging. This study arises from the need to provide a user-centered application for most types of existing computing infrastructures, and not only e-Infrastructures, where the concepts of functionality and usability have been considered from the beginning of the development process. It is important to keep in mind that even PCs are used for computing purpose on a daily basis.

After analysing the usability requirements, scheduling capabilities, target users and identified functionalities, the meta-scheduling technique was selected as a very promising approach to exploit efficiently existing computing infrastructures. This decision was based on the proven capacity of meta-scheduling for managing scalability, interoperability and flexibility in large infrastructures [2] [3].

That said, the manner by which all types of infrastructures can function together in a combined system requires both interoperability and scalability among them. This combination is necessary because interoperability without scalability no longer makes sense. For instance, [4] shows an example with more than 7,000 computational jobs, kind of experiments now quite common; and [5] shows that in *Compact Muon Solenoid* (CMS) [6] 1,000 to 250,000 jobs are submitted daily and up to 30,000 jobs are running in parallel. Our goal goes further; it is that only one researcher can manage this number of jobs with a single user-centered framework. To do this, a framework has been developed called *Distributed Resource Management for Grid* (DRM4G), which will be analysed in depth in this thesis. Furthermore, a set of production applications will also be studied in order to compare this application with the solutions in current use.

To this end, this thesis will examine how to simplify the scheduling problem across computing infrastructures in a simple and easy way, by proposing a new architecture that covers all users' demands. The objective will not be to offer a new centralized application or a new scientific portal to gain access to resources requiring user login. Rather the idea is to go further and focus on those users who want to take advantage of their current computing resources, but occasionally use other infrastructures for peak workloads [7] [8], or take advantage of some hours they may have in an HPC infrastructures.

1.2 Objectives

The main objective of this thesis is the study of the integration of existing local, national and international distributed computing infrastructures. This integration will be carried out with a framework with several benefits, in terms of resource access simplification, robustness, flexibility, scalability and efficiency, using existing infrastructure services, as well as being fully transparent to end users. The challenge will be to offer this type of experience for the full set of infrastructure components. This framework will be the tool to federate distributed resources and the services for high-level jobs and data management. In consequence, DRM4G will help to improve the usability of distributed computing infrastructures, reducing the gap between science user communities and computing infrastructures.

More specifically, the key objectives addressed by this thesis are the following:

- Study the main features of today's computing infrastructures.
- Study the problem of job scheduling in distributed computing infrastructures.
- Study existing production applications for scheduling jobs.
- Implement scheduling strategies keeping in mind computing resource features by designing task scheduling capable of efficient use of hybrid infrastructures.
- Determine current users' demands in terms of job computing executions.
- Provide uniform access to high-performance computing and high-throughput computing, including different type of infrastructures.
- Provide a uniform access to high-end storage for large data sets.
- Develop advanced services to connect computing and storage resources to job execution.
- Develop components to enable the seamless user computing services, including authentication, authorisation and accounting.
- Propose an advanced scheduling system that enables the use of Cloud Computing infrastructures.
- Provide high-quality support for legacy applications and tools with scheduling capacities.
- Implement a framework that satisfies users' needs under user-centered requirements.

1.3 Thesis organization

The remainder of this thesis is structured as follows. Chapter 2 gives an overview on state-of-the-art research infrastructures focusing on the previous work that takes advantage of distributed computing. Then, Chapter 3 introduces the DRM4G framework, its design parameters, and a detailed comparison with other current solutions in production. Chapter 4 evaluates the main features of DRM4G: scalability, robustness and interoperability. Chapter 5 presents two scientific applications that utilize DRM4G. Finally, Chapter 6 summarizes the main conclusions of this thesis and future work.

Capítulo 2. Estado del arte

2.1 Introducción

Hoy en día, los servicios que ofrecen las *e-Infraestructuras* [9] son esenciales para llevar a cabo cualquier tipo de investigación, aportando servicios como: *High Performance Computing* (HPC) [10], *High Throughput Computing* (HTC) [11], almacenamiento para grandes volúmenes de datos, redes de altas prestaciones que conectan recursos y usuarios, servicios de *autenticación*, autorización y *accounting*, etc. Estos servicios y las condiciones para el acceso a los mismos varían considerablemente dependiendo del tipo de infraestructura, recurso computacional o de almacenamiento, institución o instituciones que los conforman, o incluso del país donde se encuentran desplegados. Estas diferencias son normales, teniendo en cuenta que las *e-Infraestructuras* suelen estar organizadas por país, y por tipo de recurso. Por lo tanto, cada *e-Infraestructura* tiene sus propias reglas, políticas, y procedimientos de acceso. Sin embargo, la investigación actual es internacional por naturaleza y, por lo tanto, requiere de servicios integrados y de infraestructuras interoperables sin importar su origen y condición.

2.1 Introducción

Usuarios, investigadores y comunidades de investigación demandan cada vez más servicios de alta calidad de las *e-Infraestructuras* que estén bien gestionados e integrados desde el punto de vista del usuario, para que estos puedan focalizar sus esfuerzos en investigación, y no en los diversos requisitos para acceder a los mismos. Además, con la creciente importancia de la colaboración internacional en investigación, y el uso de grandes volúmenes de datos, también conocido como *Big Data* [12], esta necesidad se hace aun más importante. El *Big Data* es percibido ampliamente como uno de los temas más difíciles de abordar en los próximos años, desafiando las limitaciones de las infraestructuras actuales en términos de volumen, variedad, velocidad y acceso.

Sin embargo no debe olvidarse que aunque las *e-Infraestructuras* son una pieza fundamental en la investigación actual debido a su volumen y capacidad, el investigador actual en su día a día trata también con otro tipo de infraestructuras menos complejas quizás, pero de igual o mayor relevancia para sus investigaciones. Nos estamos refiriendo a infraestructuras de cálculo privadas como son: *workstations*, computadores departamentales o simples PCs.

Por consiguiente, a modo de resumen en este capítulo se presentarán los conceptos básicos asociados a infraestructuras computacionales actuales dedicadas a la investigación como: tipos, acceso, gestión, interfaces, uso distribuido, usuarios, paradigmas, etc. Conceptos que serán utilizados a lo largo de esta Tesis y que proporcionarán una visión general del estado del arte en este campo.

2.2 e-Infraestructuras

Las necesidades tecnológicas de los investigadores a menudo van por delante de las soluciones disponibles en el mercado comercial y, por lo general, no pueden cumplirse con el uso de las soluciones estándar disponibles. En el pasado, los investigadores han estimulado el desarrollo de nuevos componentes o soluciones específicas (por ejemplo, la red Internet), de los cuales también han sido los primeros clientes. Un ejemplo es la comunidad de la física de alta energías, que construyó una computación distribuida basada en una *e-Infraestructura* innovadora para hacer frente a las necesidades de procesamiento de datos del *Large Hadron Collider* (LHC) [13]. Una característica distintiva de este tipo de investigación dirigida a través de una *e-Infraestructura* es su capacidad para ofrecer servicios innovadores a las comunidades de

usuarios. Estos servicios no suelen estar disponibles en el mercado y, por lo tanto, puede anticipase a las tendencias del mismo.

Una *Electronic Infractucture* (*e-Infractucture*) o *Cyberinfrastructure* puede definirse según [14] como: "*it consists of computing systems, data storage systems, advanced instruments and data repositories, visualization environments, and people, all linked together by software and high performance networks to improve research productivity and enable breakthroughs not otherwise possible*". Además, según la *European Research Area* (ERA) [15] estos tipos de infraestructuras aportan los siguientes beneficios a la investigación:

- Evita el desvío de recursos para la investigación en servicios *ad-hoc*.
- Evita la duplicación innecesaria de recursos, al aprovechar las experiencias existentes.
- Facilita la integración y la *interoperabilidad* de las diferentes comunidades e infraestructuras de investigación.
- Amplía los compromisos a nivel europeo e internacional.
- Fomenta y apoya la investigación y la innovación abierta.

2.2.1 Elementos de una e-Infraestructura

Tomando como fuente de partida la definición de *e-Infraestructura* de la sección anterior, una e-*Infraestructura* (término europeo) o *Cyberinfraestructura* (término americano) debe ofrecer a los usuarios una serie de servicios y elementos que favorezcan la investigación, como son: computación, almacenamiento de datos, redes de interconexión, grupos de usuarios organizados en comunidades virtuales, instrumentación remota, *autenticación*, autorización, *accounting* y sistemas de información.

2.2.1.1 Computación

Ya sea un PC, un sistema de cálculo distribuido, o un supercomputador, los investigadores actuales necesitan capacidad de cómputo. Hoy en día, los investigadores pueden elegir entre una gran variedad de infraestructuras que van desde de su propio portátil, pasando por un *cluster* [16] departamental, hasta infraestructuras de cómputo a nivel nacional e internacional.

2.2.1.1 Computación

Mientras que los servicios de cálculo local (un PC por ejemplo) son cada vez más importantes y se encuentran en mejor sintonía con las necesidades de los usuarios, en cambio, las infraestructuras de cómputo nacionales e internacionales se centran más en ofrecer acceso a recursos de altas prestaciones para ejecutar grandes proyectos que de otro modo serían irrealizables. Cada vez más, las distancias entre los recursos locales, nacionales e internacionales son mayores en este sentido, dificultando a los usuarios alcanzar los recursos computacionales necesarios para su investigación [17]. De esta forma, existe una creciente necesidad de mejorar la accesibilidad a los recursos demandada por el usuario final.

Ejemplos de infraestructuras de cómputo destacables serían:

- A nivel nacional, la Red Española de Supercomputación (RES) [18], creada en 2007 por el Ministerio de Educación y Ciencia (MEC), consiste en una estructura distribuida de supercomputadores que da soporte a las necesidades de los grupos de investigación en España, como respuesta a la creciente demanda de supercomputación de la comunidad científica. La red consta de 12 nodos que están coordinados por el *Barcelona Supercomputing Centre* (BSC).
- A nivel europeo, las infraestructuras mejor consolidadas son la infraestructuras HTC *Grid* [19] y *Cloud* [20] integradas en la *European Grid Infrastructure* (EGI) [21] y la infraestructura de HPC representada por *Partnership for Advanced Computing in Europe* (PRACE) [22], operadas, respectivamente, por las organizaciones EGI Fountation o EGI.eu y PRACE AISBL. EGI y PRACE ofrecen modos de acceso complementarios. En PRACE los recursos se asignan dos veces al año en base a un proceso de revisión por pares, mientras EGI ofrece soluciones tanto para servicios de datos como de cálculo basándose en federaciones, usando diversos modelos para el acceso.
- A nivel internacional, se pueden destacar grandes infraestructuras de como la *Open Science Grid* (OSG) [23] y la *Extreme Science and Engineering Discovery Environmen* (XSEDE) [24] en USA, la *National Computational Infrastructure* (NCI) [25] en Austalia, o la *Global Access to Resource Using Distributed Architecture* (GARUDA) [26] en India.

2.2.1.2 Infraestructuras de Datos

Las infraestructuras de datos a día de hoy aún no están tan consolidadas como las infraestructuras computacionales, ni a nivel europeo ni a nivel internacional. Sin embargo, se han

dado pasos significativos en áreas de servicios de almacenamiento y replicación de datos, a través de proyectos como la *European Data Infrastructure* (EUDAT) [27]; y en el acceso a publicaciones y resultados de investigación con el proyecto *Open Access Infrastructure for Research in Europe* (OpenAIRE) [28]. Además, existe un esfuerzo global para la definición y desarrollo común de formatos, metadatos y servicios de gestión para permitir la *interoperabilidad* y el intercambio de datos. Un ejemplo claro es la *e-Infraestructura Earth System Grid Federation* (ESGF) [29], que desde 2004 provee un servicio común para la publicación y mantenimiento de datos climáticos a nivel mundial.

Los datos y la información están creciendo no sólo en volumen sino también en complejidad, ya que los investigadores requieren cada vez más la información derivada de múltiples fuentes y formatos. Además, las comunidades de usuarios son cada vez más diversas, y generan nuevas demandas de almacenamiento, publicación y citación de datos. A medida que se crean datos, almacenarlos y utilizarlos de una manera nueva y con una frecuencia no sistemática también genera incompatibilidades, que inevitablemente vienen de la creación y organización de los mismos.

Para hacer frente a grandes volúmenes de datos emergentes, se están desarrollando nuevas herramientas y métodos. Hadoop [30], es un ejemplo de ello. Hadoop es un *framework*, específicamente diseñado para trabajar con datos de manera intensiva; que proporciona las herramientas necesarias para métodos *Big Data* como reconocimiento de patrones o modelos de predicción.

Otras herramientas relevantes para el análisis de *Big Data* son las bases de datos NoSQL (HBase [31]) y *data warehouse* (Hive [32]). Estas permiten que los datos sean divididos en diferentes máquinas, ya que no hay necesidad de seguir un esquema fijo relacional. Este enfoque proporciona una mayor *escalabilidad* y capacidad de recuperación de fallos, en comparación con los sistemas de bases de datos relacionales clásicos.

2.2.1.3 Redes

Estas infraestructuras forman una base sólida para muchas colaboraciones de investigación nacionales e internacionales. Las redes, y entre ellas Internet, son un elemento indispensable para el desarrollo y uso de las *e-Infraestructuras*. Estas han evolucionando en las últimas décadas

proporcionando altas calidades en el servicio, y conexiones directas de punto a punto, permitiendo transferencias de datos dedicadas entre lugares específicos (por ejemplo, entre usuarios e instrumentaciones remotas). Esta capacidad de dedicar una ruta de datos permite a los proveedores garantizar la calidad del servicio, incluso para grandes volúmenes de datos.

Sin embargo, hay espacio para mejora en la funcionalidad, rendimiento y facilidad de uso, siendo la innovación continua una de las claves características de las redes de investigación. Desde la perspectiva del usuario, las redes deben proporcionar una conexión sin fisuras de extremo a extremo para su uso, en cualquier momento y lugar. Pero en la práctica, este camino debe cruzar diferentes dominios de varios proveedores de red. En consecuencia, para los investigadores estas redes pueden incluir redes de campus universitarios, redes nacionales de investigación, e infraestructuras de interconexión internacionales; por lo que, no siempre el servido ofrecido es el deseado.

En Europa, la red de altas prestaciones que aporta la conectividad necesaria para compartir, acceder y procesar grandes volúmenes de datos se denomina GEANT [33]. Esta red conecta más de 50 millones de usuarios en 10.000 instituciones a través de Europa, operando a velocidades superiores a los 500 Gbps sobre más de 100 redes nacionales [33].

2.2.1.4 Virtual Research Communities

El desarrollo de comunidades virtuales de usuarios o *Virtual Research Communities* (VRC) permite a los grupos de investigadores dispersos geográficamente trabajar juntos a través del uso de tecnologías de la información. Las VRCs son también conocidas como *Virtual Organizations* (VO) [34], término que proviene de la tecnología *Grid* [35], y facilitan la integración de las capacidades de investigación, la movilidad virtual de investigadores físicamente distantes, el acceso a los resultados de investigación, y la colaboración a nivel regional y global.

En la actualidad, existen múltiples VRCs en distintos campos de la investigación como pueden ser: *West-Life* [36] en la biología, *Multi-scale complex Genomics* (MuG) [37] en genómica o *European Virtual Environment for Research – Earth Science Themes* (EVER-EST) [38] en ciencias de la tierra.

2.2.1.5 Instrumentación remota

El acceso a equipos remotos de laboratorio es muchas veces un elemento esencial en el ámbito investigador. Un equipo único, o casi único, y costoso es a menudo una condición previa para llevar a cabo una investigación; sin embargo, este tipo de equipos usualmente son sólo accesibles a nivel local. Por lo tanto, el acceso compartido, independientemente del investigador y de su localización deber ser un elemento esencial de las *e-Infraestructuras* [39] [40].

El uso compartido de equipamiento científico remoto permite reducir sustancialmente el coste de las investigaciones. Además, ciertos instrumentos se encuentran localizados en zonal remotas por necesidad, como es el caso de observatorios astrofísicos. En estos casos, la capacidad de que los instrumentos sean accesibles de forma remota mejora su uso eficiente, tanto del tiempo de los investigadores como el de los equipos; y aumenta el rendimiento de las inversiones en grandes instalaciones.

El desarrollo y la difusión de las técnicas de instrumentación remota es una gran oportunidad para las comunidades científicas. Sin embargo, para aprovechar al máximo esta oportunidad se necesita una integración eficaz de las infraestructuras de asimilación de datos con las infraestructuras de procesamiento de datos, a través de interfaces estándar apropiadas para la instrumentación remota [41].

2.2.1.6 Autenticación, autorizacion, accounting y sistemas de información

Mientras la *autenticación* permite a los usuarios crear una identidad dentro de una infraestructura especifica. La autorización establece los derechos de los usuarios a realizar ciertas las operaciones dentro de esa infraestructura, en función de las políticas de acceso establecidas.

El desarrollo y despliegue de los servicios de *autenticación* y autorización, también denominados *Authentication and Authorisation Infrastructure* (AAI), han sido y son diferentes dependiendo del tipo de infraestructura. Por ejemplo, en infraestructuras de tipo *Grid* el control de acceso se basa en certificados X.509 [42], en HPC los usuarios suelen disponen de un nombre de usuario y una clave, y en infraestructuras como ESFG se utiliza OpenIDs [43], un sistema de *autenticación* digital descentralizado.

Por su parte, los sistemas de *accounting* y monitorización ofrecen una medición detallada de la capacidad y calidad del servicio actual y pasado de la infraestructura. Datos que son publicados a través de sistemas de información o *Information Systems* (IS) (por ejemplo

servidores LDAP [44]), de forma que muestran en todo momento al usuario información actualizada de los recursos disponibles en la infraestructura. Estos, a su vez, pueden resultar interesantes desde el punto de vista de los usuarios o aplicaciones para la gestión propia de los recursos mediante técnicas de meta-planificación [45] [46] [47] [48] [49].

2.2.2 e-Infraestructuras federadas

Según [50] se puede definir la federación como: “la unión comprendida de un número de entidades parcialmente auto gobernadas, unidas por medio de un gobierno central o federal”. Este concepto trasladado a *e-Infraestructuras* implica la asociación de varios proveedores de infraestructuras que comparten y ofrecen servicios siguiendo modelos estandarizados (por ejemplo FitSM [51] o SOA [52]) de forma voluntaria. Estos servicios tienen como objetivo lograr una visibilidad común de las infraestructuras, sin limitarse a ofrecer interfaces comunes para intelectualizar con ellas. Y es que un interfaz común, por si solo, no garantiza la federación [53]. Así, en [50] se indica que la federación surge de la conjunción de varios factores clave como: la *interoperabilidad*, la *escalabilidad*, el acceso transparente y los interfaces comunes. Existe, por lo tanto, una diferencia conceptual entre una *e-Infraestructura* federada y un simple grupo de proveedores de infraestructuras [54] [55].

Si un usuario quiere utilizar múltiples infraestructuras independientes, no federadas entre sí, entre las que no existe una relación directa. Es el usuario quien debe gestionar por completo la compatibilidad entre los interfaces de cada recurso, supervisar cada infraestructura y gestionar sus cuentas y roles, ya que está operando con diferentes dominios de configuración y seguridad. La distribución de cálculo entre entornos multi-infraestructura puede llegar ha ser interoperable, pero si no existe *escalabilidad*, una gran carga de trabajo que puede generar un cuello de botella en el accesos a los recursos.

Esta situación supone limitaciones en tanto en cuanto es el usuario quien debe explotar las distintas capacidades de las infraestructuras; ya sea de manera individual, usando sólo una infraestructura, como en entornos distribuidos, en los que se trata de repartir la carga de trabajo existente entre los recursos disponibles de manera concurrente. Por consiguiente, el uso de sistemas federados facilita la gestión de recursos, y es un elemento esencial para hacer frente a los problemas de gestión de recursos computacionales entre infraestructuras distribuidas [56] [57] .

Pero el hecho de que varias infraestructuras formen una federación no implica necesariamente que múltiples federaciones a su vez sean federables directamente [58]. Lo cierto es que la realidad muestra todo lo contrario, el resultado suele ser la existencia de infraestructuras federadas independientes unas de otras que no son fácilmente interoperables entre ellas, como serían los casos de las *e-Infraestructuras* EGI y PRACE. Además, debe tenerse en cuenta que si bien el uso de la federación viene siendo empleado de manera habitual en infraestructuras *Grid*, no es algo tan común en infraestructuras de tipo *Cloud* [59] o HPC. Así, en la actualidad, existen un gran número de proveedores de recursos, como por ejemplo *Amazon Web Servces* (AWS) [60], que son capaces de ofrecer una gran cantidad de recursos hasta cierto punto ilimitados y que no pertenecen a ningún tipo de federación.

Para solucionar o al menos mitigar esta problemática, en la literatura se recogen varias propuestas como por ejemplo: InterGrid [58], Multi-Grid [61], GridWay [53] , Multi-Cloud [62] o Inter-Cloud [63], donde se trata de adaptar las infraestructuras existentes en producción mediante el uso de técnicas de *interoperabilidad*. Nótese que las soluciones propuestas hacen referencia sólo a la *interoperabilidad* entre infraestructuras del mismo tipo, y no de tipos diferentes, lo que supondría si cabe un mayor reto.

Por lo tanto, uno de los principales objetivos de estudio de esta Tesis es poder ofrecer al usuario final un acceso transparente y sencillo a diferentes tipos de infraestructuras, sean federadas o no, y que además sea escalable dadas la necesidades actuales de cómputo. Debe considerarse que muchos usuarios en la actualidad sólo utilizan determinados recursos (por ejemplo, PCs o *workstations*), por las dificultades que a veces el uso de determinadas infraestructuras les supone; perdiendo así, la posibilidad de acceso a múltiples recursos de forma distribuida.

2.2.2.1 Ejemplos de e-Infraestructuras federadas

Las principales *e-Infraestructuras* europeas proporcionan las herramientas necesarias para gestionar grandes infraestructuras de forma distribuida, garantizando así un funcionamiento estándar a través de recursos heterogéneos que provienen de distintos proveedores independientes. Esto incluye, por ejemplo, AAI, *accounting*, IS y otros servicios necesarios para federar el acceso de múltiples VRCs. El entorno federado de las *e-Infraestructuras* resulta un factor clave para la gestión y tratamiento de datos de manera distribuida, permitiendo a los

investigadores centrarse en sus aplicaciones; despreocupándose así de la complejidad de la infraestructura subyacente.

A continuación se presentan ejemplos de servicios federados que han sido adoptados por las principales *e-Infraestructuras* a nivel europeo:

- **EGI** Core es la capa de la infraestructura de EGI que proporciona todas las herramientas y los servicios de acuerdo con la norma de gestión de servicios FitSM. La función de catálogo de servicios es ofrecida por GOCDB [64], un registro central para proveer información sobre la topología de la *e-Infraestructura* que en la actualidad es utilizado por EGI, EUDAT y WLCG [65]. La información incluye entidades como: los centros de operaciones, centros de recursos, tiempos de parada de los mismos, información de contacto y los roles de los usuarios responsables de las operaciones en los diferentes niveles de la estructura. El servicio de registro AAI EGI [66] permite a las VRCs acceder a los servicios de EGI mediante certificados X.509 y servicios de autorización VOMS [67]. El sistema de *accounting* de EGI se realiza con la herramienta APEL [68], la cual recopila los datos de uso de los *sites* que forman parte de EGI y de las infraestructuras WLCG, así como de los *sites* pertenecientes a otras organizaciones que colaboran con EGI, como OSG o NorduGrid [69]. Finalmente, la monitorización de la infraestructura se lleva a cabo con la plataforma ARGO [70], mediante un seguimiento de los servicios de disponibilidad, fiabilidad y métricas de *Service Level Agreements* (SLA) [71] de todos los servicios de EGI.
- **PRACE** dispone de un catálogo de servicios: *PRACE Service Catalog*, donde se define un conjunto común de servicios que los proveedores de las infraestructuras deben ofrecer. Por ejemplo, a nivel AAI: GSISSH [72], SSH [73] y MyProxies [74] basados en certificados X.509; y a nivel de IS y *accounting* LDAP. El funcionamiento de PRACE como infraestructura HPC distribuida implica la coordinación de estos servicios que se integran en sistemas Tier-0 y Tier-1 [75]. Para asegurar un uso continuo de los mismos estos se encuentra disponibles en todos los Tier, permitiendo así la *interoperabiliad* entre todos los ellos.
- **EUDAT** opera como una federación de los proveedores de servicios de datos genérica mediante herramientas basadas en el estándar FitSM. Estas han sido desarrolladas o adaptadas de soluciones existentes usadas en otras infraestructuras. Así, por ejemplo, la

gestión de identidades y acceso se realiza con B2ACCESS [76], y permite gestionar: *Security Assertion Markup Language* (SAML) [77], OAuth2 [78], OpenIDs, certificados X.509, LDAP y *autenticaciones* usuario/clave; el sistema de motorización utiliza ARGO; la herramienta *Data Project Management Tool* (DPMT), basada en PLONE [79], facilita a los proveedores de recursos registrar sus servicios; y el IS está actualmente basado en el GOCDB.

2.3 Paradigmas de computación

Las *e-Infraestructuras*, federadas o no, y las infraestructuras en general permiten a los usuarios acceder y compartir recursos computacionales, y recursos de almacenamiento de altas prestaciones. Pudiéndose utilizar para diversas aplicaciones, desde ejecutar complejos modelos computacionales (por ejemplo el modelo climático WRF [80]), hasta llevar a cabo análisis de datos a grandes escalas. Estas infraestructuras ofrecen el uso compartido y rentable de la tecnología y una accesibilidad generalizada, poniéndola a disposición de prácticamente todos los investigadores.

Pero hasta el momento, no existe un sistema de computación unificada única como un *World Wide World* (WWW); más bien hay varios paradigmas cada uno con características e interfaces diferentes que son utilizados por las diferentes infraestructuras. Esta clasificación en paradigmas nos va a ayudar a la abstracción de las diferentes infraestructuras (que son muchas) en paradigmas genéricos. Un posible conjunto de paradigmas que englobe la mayor parte de las infraestructuras actuales existentes según [81] [82] [83] [84] serían: HPC o *Cluster Computing*, *Volunteer Computing* [82], *Grid* y *Cloud* [84].

En esta Tesis nos centraremos solamente en tres tipos de paradigmas: HPC, *Grid* y *Cloud*, sobre los cuales ya han sido testados exitosamente diferentes técnicas de meta-planificación [85] [3] [86] [87]. Además, estos paradigmas tienen una estrecha relación entre ellos, pudiendo llegar a ser utilizados de manera conjunta. Por ejemplo, desplegando *clusters* elásticos sobre infraestructuras *Cloud* [88] [89], e incluso infraestructuras *Grid* elásticas [90] [91]. *Volunteer Computing* por el contrario, ha sido descartada de este estudio debido a los requerimientos que sobre los *jobs* se imponen para ser ejecutados en este paradigma, ya que limitan su ejecución en entornos distribuidos multi-paradigma [92].

2.3.1 Características

Según [93] los paradigmas HPC, *Grid* y *Cloud* se pueden caracterizar de acuerdo a una serie de parámetros como: la propiedad del recurso: local o externo; accesibilidad: privada o pública; el tamaño del recurso: estático o dinámico; la política de acceso: exclusiva o compartida; y la portabilidad de aplicaciones: específicas para la plataforma o independientes. Siguiendo estos criterios:

- En HPC o *Cluster Computing* los recursos suelen ser de propiedad local(por ejemplo, de un grupo de investigación o una institución local o nacional) donde sus miembros y colaboradores disponen de un acceso privado. Los recursos suelen tienen un tamaño estático que sólo cambia mediante la compra de nuevos recursos, en consecuencia, suelen ser recursos cuasi-homogéneos. La asignación de recursos es típicamente mediante la asignación de cuotas de cálculo (horas de cálculo) y almacenamiento con respecto a un usuario, un grupo o un proyecto. Respecto a las aplicaciones a ejecutar, estas han evolucionado desde el desarrollo para una plataforma específica, hasta la actualidad en la que son portales entre infraestructuras; usualmente basadas en entornos de programación paralelos MPI [94], OpenMP [95] o PVM [96].
- En computación *Grid*, los recursos tienden a tener una propiedad tanto local como externa. Como parte del acceso a los recursos es público, esto significa que algunos de los recursos locales se ponen a disposición de usuarios externos que son miembros de una determinada VO. Los recursos son heterogéneos (*hardware* y *software*) y el tamaño de los mismos es dinámico, pudiendo llegar a entrar y salir de la infraestructura, situación que altera la capacidad y el rendimiento del sistema. Esto hace que el manejo de recursos, su administración y planificación sea todo un reto. Las aplicaciones en *Grid* son una mezcla de portátiles y específicas de la plataforma, consecuencia de la adaptación que han sufrido algunas de ellas para ejecutarse en un entrono tan complejo [97] [98]. Las infraestructuras *Grid* actuales favorecen la ejecución de aplicaciones que se pueden resolver medicate la división de tareas en *jobs* independientes, y cuyo resultado final se obtiene con la combinación de los resultados parciales de cada *job*. Un tipo de estas aplicaciones son las conocidas como aplicaciones *sweep* [99], como por ejemplo las simulaciones de Monte-Carlo [100].

- En computación *Cloud*, los recursos pueden ser de propiedad privada o pública. Los *Cloud* públicos (por ejemplo, AWS) ofrecen un acceso libre a los usuarios, que normalmente se basa en el formato *pay-as-you-use* [101], mientras que los *Cloud* privados sólo son accedidos por usuarios pertenecientes a un determinando grupo o a una institución. Ambos tipos de *Cloud* permiten a los usuarios acceder a recursos cuyo tamaño es dinámico, este crece o disminuye por medio de la demanda (por ejemplo, en el número de maquinas virtuales) [59]; esta creación dinámica, la migración o la destrucción de recursos está soportada por tecnologías de *virtualización* [102]. Las aplicaciones en *Cloud* pueden ser ofrecidas como servicio *Software-as-a-Services* (SaaS) [59] que proveen de aplicaciones específicas optimizadas para el entrono *Cloud*; o los propios usuarios haciendo uso de la *Infraestructura-as-a-Service* (IaaS) [59] pueden llegar a desplegar sus propias aplicaciones como [103] [104] [105].

Para concluir, la tabla 2.1 recoge un resumen de los principales atributos de cada paradigma extraídos de [93] [106]. Como se puede observar, ninguno de los tres es considerado como la mejor solución en cada una de las características. Aparentemente, el mejor paradigma sería HPC, al contar con una capacidad de cómputo alta y estática, y por lo tanto más estable. Sin embargo, *Grid* y *Cloud* con capaces de ofrecer un mayor número de recursos con capacidad de computo media, además de soportar *interoperabilidad*.

2.3.1 Características

Característica	HPC	Grid	Cloud
Capacidad	Fija	Media alta	Alta
Capacidad de cómputo	Muy alta	Media	Media
Capacidad para compartir recursos	Limitada	Alta	Limitada
Soporte para *virtualización*	Muy poco frecuente	A veces	Siempre
Seguridad	Alta	Media	Baja media
Interoperabilidad	No aplicable	Media	Limitada
Heterogeneidad de los recursos	Baja	Alta	Media baja
Distribución geográfica	Limitada	Alta	Media
Distribución de carga de trabajo	No aplicable	Alta	Limitada

Tabla 2.1: Comparación de los atributos principales entre HPC, Grid y Cloud.

2.3.2 Interfaces

Un elemento indispensable en el acceso a las infraestructuras son los diferentes interfaces de los servicios que disponen y ofrecen a usuarios y aplicaciones, y que varían dependiendo del tipo de paradigma, recurso y *software*. La interacción con los recursos además suele disponer de uno o varios tipos de interfaces: *Web Services* (WS), linea de comandos o *Command Line Interface* (CLI) y *Application Programming Interface* (API). En este sentido, existen múltiples esfuerzos para proveer de interfaces estándar para gestión de recursos como: SAGA [107] , DRMAA [108] y OGSA-BES [109], pero ninguno de ellos ofrece un interfaz definitivo aplicable a todos los paradigmas tratados en esta Tesis.

2.3.2.1 HPC

Los recursos HPC son accedidos tradicionalmente de manera centralizada a través de un *User Interface* (UI) o *frontend*, encargado de gestionar las credenciales de autorización de los usuarios. El UI posibilita el acceso mediante sesiones remotas SSH o GSISSH (si los usuarios disponen de certificados X.509) para la configuración y ejecución de *jobs* en linea de comandos. UNICORE ofrece de manera excepcional interfaces gráficos [110]. La interacción con la

infraestructura se realiza mediante sistemas de colas, también conocidos como *Local Resource Management Systems* (LRMS) [16] o *Distributed Resource management Systems* (DRMS) [111]. Ejemplos de los más utilizados son: SLURM [112], PBS [113], SGE [114], LSF [115] o LoadLeveler [116]. Mientras que el almacenamiento que se proporciona al usuario recae sobre sistemas de ficheros es en red como: NFS [117], GPFS [118] o Lustre [119].

2.3.2.1 Grid

Globus Toolkit [120] fue el primer *middleware Grid* [34] implementado, convirtiéndose en la referencia y en un estándar *de facto* en el acceso a recursos *Grid*. Los servicios que Globus Toolkit ofrecía incluían entre otros: monitorización y descubrimiento de servicios (servidores LDAP con esquemas GLUE [121]), gestión de *jobs* (GRAM [122]), seguridad (basada en certificados X.509), y servicios de trasferencia de ficheros (GridFTP [123]).

Debido al éxito de las versiones iniciales de Globus Toolkit otros *middlewares* fueron apareciendo (gLite [124], UNICORE o ARC [125] por ejemplo) con el objetivo de mejorar sus funcionalidades y su eficiencia; implementando siempre sus operaciones básicas en los servicios iniciados por Globus. En la actualidad, la computación *Grid* está basada en servicios de computación o *Compute Elements* (CE) como: Globus GRAM [126], CREAM [127], HTCondor-CE [128] o ARC-CE [129]; y en servicios de almacenamiento o *Storage Elements* (SE) accedidos mediante protocolos como: SRM [130] o GridFTP.

2.3.2.2 Cloud

Los usuarios con acceso a recursos IaaS *Cloud* pueden crear infraestructuras capaces de manejar recursos físicos y virtuales de una manera integral. Esta orquestación que incluye la gestión de máquinas virtuales o *Virtual Machines* (VMs), servicios de almacenamiento o redes; se realiza mediante un *software* denominado *Virtual Infrastructure Manager* (VIM) [131]. Este tipo de *software* recuerda a un sistema operativo tradicional, pero en este caso encargado de manejar no una única máquina sino una agregación de múltiples de ellas, mostrando una vista uniforme al usuario y a las aplicaciones. Ejemplos de los VIM más populares son: AWS (considerado como *de facto* estándar), OpenNebula [131], OpenStack [132], Nimbus [133] o Eucalypus [134].

2.3.2.2 Cloud

Estos VIMs pueden ser accesibles mediante varios estándares: AWS que incluye *Amazon Elastic Compute Cloud* (EC2) [135] y *Amazon Simple Storage Service* (S3) [136], *Cloud Data Management Interface* (CDMI) [137] propuesto por la *Storage Networking Industry Association* (SNIA) [138], *Open Cloud Computing Interface* (OCCI) [139] propuesto por el *Open Grid Forum*(OGF) [140] y *Cloud Infrastructure Management Interface* (CIMI) [141] propuesto por la *Distributed Management Task Force* (DMTF) [142]. Pero, a pesar de estas estandarizaciones la realidad es que los proveedores de infraestructuras *Cloud* suelen ofrecer sus propios interfaces, pudiendo llegar a existir tantos como número de proveedores.

De acuerdo con [63] existen dos tipos de soluciones para acceder de una manera uniforme a los interfaces *Cloud*: una basada en servicios ofrecidos por terceros y otra basada en librerías específicas. En el primer caso, se usa un intermediario que es el encargado de la negociación directa con cada proveedor. De esta forma, el usuario dispone de un mismo interfaz para el despliegue, la ejecución y la supervisión de VMs. Además, suelen disponer de la capacidad de agregar *Cloud* privados a los que el usuario pueda tener acceso. Ejemplos de plataformas de este tipo serían: RigthScale [143], Scalr [144] o Kaavo [145].

En el segundo caso, una librería es la encargada de facilitar de una manera uniforme el acceso a múltiples servicios y recursos de múltiples *Clouds*. Varias librerías han sido desarrolladas en los últimos años, siendo Apache Libcloud [146] desarrollada en Python y JClouds [147] desarrollada en Java las más exitosas. Estas librerías diseñadas para abstraer a usuarios y desarrolladores de las diferencias entre los distintos interfaces *Cloud* ofrecen servicios de integración con más de 50 proveedores [148] en el caso de Apache Libcloud.

Mención aparte requiere la configuración de las instancias de las VMs. Esta se puede realizar de dos formas: la primera, generando el propio usuario una imagen personalizada de la VM, que se deberá incluir en el repositorio de imagines del proveedor o *marketplace* (por ejemplo EGI AppDB [149]); la segunda, mediante *contextualización* [150], proceso que permite la configuración de las instancias de VMs durante su *inicialización*. Esta segunda forma, más eficiente, permite la personalización de plantillas genéricas ofrecidas por el proveedor, en lugar de crear una propia. El problema reside en que cada VIM suele disponer de un proceso de *contextualización* propio [151] [152]. Sin embargo, existen aplicaciones genéricas como *cloud-init* [153] que pueden utilizarse en la mayoría de proveedores IaaS disponibles en la actualidad.

2.4 Ejecución de aplicaciones en entornos multi-infraestructura

El uso de diferentes infraestructuras para la ejecución de aplicaciones suele implicar que los experimentos compuestos por múltiples *jobs* se ejecuten en entornos heterogéneos, ya sea por el sistema operativo, el tipo de procesador, el tipo y versión del compilador, la versión de la aplicación, el propio *hardware*, etc. Esto supone un reto añadido a cada usuario que quiera ejecutar cualquier aplicación en un entono distribuido. Además, debe tenerse en cuenta que la paquetera de *software* que en algunos casos demandan los usuarios tiene de licencia de pago, como por ejemplo es el caso de aplicaciones como Matlab [154].

En infraestructuras HPC, el usuario suele disponer de una serie de compiladores, librerías y aplicaciones normalmente ya optimizadas y adaptadas por los administradores de la infraestructura al entorno de ejecución con el objetivo de obtener el mayor rendimiento posible. No siendo excluyente que el usuario pueda desplegar sus propias aplicaciones y/o librerías de manera personal o con el apoyo de los administradores.

En infraestructuras *Grid* por el contrario, los administradores de los *sites* sólo proveen de un *software* mínimo, y es el usuario quien normalmente ha de desplegar sus propias aplicaciones. En consecuencia, el usuario debe diseñar sus *jobs* con una etapa previa a la ejecución de la aplicación propiamente dicha, donde se configure el entorno de ejecución. En ésta, se verifican los recursos *software* disponibles y se compilan aplicaciones y/o las librerías, si es necesario. En algunos casos, si la configuración requiere un complejo proceso de compilación, como es el caso de los modelos climáticos, se importa una versión ya precompilada de la aplicación [155]. También existe la posibilidad de instalar aplicaciones y librerías *open source* concretas mediante solicitudes a los administradores de las VOs, como por ejemplo en EGI.

En *Cloud*, la situación es completamente diferente a las anteriores infraestructuras. En este caso, el usuario es a la vez administrador de la infraestructura desplegada y, por lo tanto, dispone de total libertad para la instalación de software. Además, este puede incluso seleccionar el tipo y la versión sistema operativo que considere, y en ciertos caso el *hardware*. También ha de tenerse en cuenta que esta configuración personalizada puede ser almacenada para ser reutilizable para futuras ejecuciones, suponiendo un ahorro de tiempo considerable.

Por consiguiente, es el usuario quien normalmente ha de adaptar las necesidades de sus aplicaciones a los diferentes recursos, y lo al revés. De ahí, que exista una demanda de *frameworks* que faciliten la ejecución de aplicaciones en entornos multi-infraestructura.

Por otro lado, el uso de multi-infraestructuras en procesos de ejecución distribuida están expuestos a problemas de reproducibilidad, es decir, ante los mismos datos de entrada no se generan los mismos resultados. Un ejemplo de esta problemática se puede encontrar en las simulaciones climáticas. Donde una pequeña diferencia en el cálculo en un error de redondeo puede conducir a una evolución completamente diferente de los patrones climáticos [156], llegando a obtener resultados incoherentes. Ser capaz de reproducir una simulación en un entorno de computación heterogéneo debe de ser necesario para poder emitir un conjunto consistente de hipótesis al analizar la simulación.

Pero la reproducibilidad puede ser incluso afectada por el número de procesos con que una aplicación es ejecutada [157], situación que extiende esta problemática a cualquier tipo de infraestructura. Por lo tanto, la reproducibilidad también ha se ser tenida muy en cuenta por todos los usuarios de aplicaciones computacionales, y no sólo por aquellos usuarios que ejecutan sus aplicaciones entornos multi-infraestructura. Además, la reproducibilidad ofrece ventajas evidentes para la depuración de errores y los test de prueba. Ser capaz de reproducir el comportamiento exacto de una aplicación con una arquitectura *hardware* o un entorno *software* diferente es a menudo clave para localizar y aislar posibles errores.

A pesar de la complejidad de esta problemática, la reproducibilidad es un problema ya estudiado y que puede ser minimizado a distintos niveles, como por ejemplo, a nivel algorítmico y de programación [157] [158] [159]. Aunque asumiendo posibles costes en *overheads* en la ejecución [157].

2.5 Planificación en el acceso a infraestructuras

Durante años, la planificación en el acceso a entornos distribuidos ha sido estudiada ampliamente. De hecho, la planificación es un problema que aparece de forma recurrente desde hace tiempo en la literatura [160]. Como resultado una gran variedad de teorías y herramientas han sido propuestas, como por ejemplo [161] [162] [163] [85] [164] [165] entre otras, con el objetivo de mejorar la eficiencia en el uso de recursos. Sin embargo, como se indica en [166] llevar a escenarios reales este tipo de herramientas es ciertamente complejo.

En el caso de sistemas distribuidos tradicionales como HPC, este problema resulta más sencillo que los sistemas *Grid* y *Cloud*, los cuales poseen una naturaleza dinámica y una mayor

heterogeneidad de recursos como se vio en la sección 2.3.1. Además, cabe destacar la complejidad extra que suponen los recursos *Cloud*, donde intervienen más parámetros que en la planificación de sistemas *Grid* [167] [168].

El término meta-planificador es un término utilizado con frecuencia en computación *Grid*, y cuya finalidad es la de establecer una política de control sobre los recursos distribuidos. Su misión es clave para distribuir las tareas entre los planificadores que existen a nivel local dirigiendo o redirigiendo las tareas definidas por el usuario hacia el mejor recurso o recursos. El "recurso optimo" obviamente puede ser considerado por varios criterios, teniendo siempre en cuenta como objetivo el mejor rendimiento en términos de cálculo y tiempo de ejecución. En consecuencia, los meta-planificadores se enfocan hacia a un entorno interoperable, eficiente y flexible. Así, en trabajos como [169] [46] se presentan estudios comparativos de planificadores, donde se discute sobre cómo explotar los paradigmas de computación haciendo uso de la meta-planificación. Las conclusiones estos estudios determinan que la planificación ofrece la flexibilidad necesaria para manejar los requerimientos de los *jobs* en situaciones de multi-paradigmas y multi-infrastructuras y, por lo tanto, situaciones reales en las que existen fallos o modificaciones del entorno pueden ser gestionadas de manera dinámica.

2.5.1 Tipos de planificadores

En el campo de la planificación existen varios términos que pueden resultar confusos y similares, especialmente relacionados con los tipos de planificadores existentes. En esta sección se revisarán algunos de los más destacados:

- Planificador *Grid*: Debido a la complejidad de los sistemas *Grid* múltiples niveles de planificación pueden ser identificados [170]. En consecuencia, un planificador *Grid* es el *software* encargado de mapear *jobs* y/o aplicaciones sobre recursos *Grid* en función de los criterios impuestos por los usuarios y la propia infraestructura. Por ejemplo, Condor-G [171] y gLite-WMS [172] son actualmente los planificadores *Grid* existentes para utilizar las infraestructuras OSG y EGI respectivamente.
- Planificador local: Estos planificadores se encargan de la asignación de *jobs* a recursos dentro de una misma red de área local (por ejemplo Maui [173] o Moab [174]). El

planificador gestiona los recursos locales mediante sistemas de colas también denominados LRMS o DRMS.

- Super-planificador: Este tipo de planificadores, también denominados en la literatura *resource-brokers* [175], corresponden a una planificación centralizada en la que se utilizan los planificadores locales para reservar y asignar los recursos. Los planificadores locales gestionan su cola de *jobs*, mientras el super-planificador es el encargado de gestionar la reserva de recursos con cada planificador local. Debe tenerse en cuenta que los *jobs* son siempre completados en un recurso único. Según [176] pueden considerarse como super-planificadores NASA-super scheduler [177] o Nimrod-G [178].
- Meta-planificador: Surgidos con la idea de permitir la ejecución distribuida de *jobs* en diferentes recursos. Los meta-planificadores coordinan diferentes planificadores locales para distribuir la carga de trabajo. Como en el caso de los super-planificadores, los meta-planificadores utilizan los planificadores locales para interactuar con los recursos. Pero en este caso, la carga de trabajo pude ser repartida entre distintos sistemas o infraestructuras. GridWay [179] es uno de los referentes en este campo.
- *Cloud-broker*: Entendemos un *cloud-broker* como servicios de alto nivel destinados al usuario final para facilitar la *interoperabilidad Cloud*, permitiendo a los usuarios indicar lo que necesitan y cuando lo necesitan(VMs o almacenamiento por ejemplo). El *cloud-broker* se encargará de desplegar la infraestructura necesaria y ejecutar las posibles cargas de trabajo o *jobs*, además de recolectar los resultados. Ejemplos de este tipo son: CompatibleOne [180], Slipstream [181] o DIRAC [182] con su extensión VMDIRAC [183].
- *Meta-broker*: El concepto de *meta-brokering* [87] fue introducido inicialmente por [184] con el objetivo de interaccionar con múltiples *resource-brokers*. En [185] este concepto fue extendido con con la idea de además soportar interoperabilidad entre los recursos. Concretamente el *meta-broker* se dispone encima de los *resource-brokers* utilizando los meta-datos proporcionados por ellos para la planificación de *jobs*.
- Planificador *enterprise*: Este tipo de planificador surge en las grandes empresas que tienen recursos computacionales distribuidos en distintos departamentos. En estos casos, el planificador utiliza los planificadores locales desplegados a lo largo de los departamentos pertenecientes a la misma empresa.

- *Workflow manager* o WfM: Es el responsable de la ejecución y planificación de cada componente de un *workflow* [186] en diferentes recursos (HPC, *Grid*, *Cloud*, etc). Los WfMs son utilizados por los investigadores en la actualidad por las capacidades de interoperatividad entre infraestructuras y los servicios que proveen para el acceso a instrumentación, fuentes específicas de datos, etc. Algunos ejemplos son: Pegasus [187], Swift [188], Kepler [189], Taverna [190] o PGRADE [191].

Para finalizar, cabe indicar que otros autores presentan algunos de los planificadores referidos anteriormente en otras categorías. Esto se debe fundamentalmente a que varios de ellos han evolucionado de un tipo a otro, o que simplemente pueden considerarse dentro de varias categorías a la vez. Un ejemplo claro es Condor-G, considerado como planificador *Grid*, meta-planificador, super-planificador e incluso un como un WfM.

2.5.2 Fases en el proceso de planificación

El proceso de planificación llevado a cabo por un planificador, independientemente del tipo que sea (ver sección 2.5.1), puede clasificarse en los siguientes pasos:

- Recopilación de información de la infraestructura: Los planificadores tienen acceso a los IS de las infraestructuras (si es que existen) por lo que disponen de información actualizada de los recursos disponibles y sus características. Esta información es esencial para la planificación de *jobs*. En consecuencia, el uso de *e-Infraestructuras* federadas suelen facilitar su propio uso, siempre y cunado se encuentren adecuadamente configurados.
- Selección del recurso: No todos los recursos de una infraestructura, ni todas las infraestructuras son susceptibles de ser candidatas para la ejecución de una o un grupo de tareas. Y por lo tanto, es necesario un proceso de selección entre las características de los recursos disponibles y los requerimientos de las tareas a ejecutar. Este proceso también dependerá de los distintos algoritmos de planificación de que disponga el planificador. Por ejemplo, un algoritmo Round-Robin [192] simple, no seleccionará los mismos recursos que un planificador que basa sus decisiones en el estado actual de los recursos y en el histórico ocurrido en pasadas ejecuciones.

2.5.2 Fases en el proceso de planificación

- Cálculo de la planificación de tareas: En este paso se planifican las tareas o *jobs* en función de distintos parámetros como su prioridad, tiempo de ejecución, tiempo de espera, etc. El tiempo total de cálculo variará dependiendo del tipo de algoritmo o de la topología del planificador [193].
- Asignación de tareas: Cuando se ha realizado la planificación de tareas estas son asignadas a los recursos previamente seleccionados.
- Monitorización de tareas: Una vez se ha producido la asignación, es necesario disponer de un sistema de seguimiento para informar del progreso y de posibles fallos del *job*. Dependiendo del tipo de políticas de planificación el *job* en caso de fallo podría ser re-planificado o migrado a otro recurso [194].

2.5.3 Planificación en paradigmas de computación

En esta sección nos centraremos en cómo se realiza la planificación de recursos y tareas en los paradigmas HPC, *Grid* y *Cloud*; con el objetivo de mostrar una comparativa de cómo se gestionan las tareas en cada uno de los ellos, así como su problemática asociada.

2.5.3.1 HPC

En HPC la planificación de *jobs* y recursos se presenta como el caso más sencillo de los tres, debido a su infraestructura estática. Los planificadores están basados en descripciones estáticas de los recursos configurados normalmente por los administradores de los sistemas. Un planificador local (MAUI por ejemplo) gestiona las políticas de ejecución de las tareas, mientras un LRMS (PBS por ejemplo) es el encargado de encolar los *jobs* en las colas del sistema, previo paso a su ejecución en los nodos de trabajo o *Worker Nodes* (WNs) [10]. [195] refleja una visión comparativa y detallada de los distintos modos de planificación aplicados al HPC.

Un LRMS puede disponer de varias colas con distintas características y/o limitaciones como *cputime* o *walltime*, que ayudan a usuarios y aplicaciones a seleccionar los recursos del sistema en función las características de sus *jobs*. Pero no todos los sistemas aún disponiendo de los mismos sistemas de colas y planificadores se comportan de la misma forma. Esta puede variar dependiendo de la gestión del administrador y de las políticas de la institución a la que pertenece

la infraestructura. De esta forma, pueden existir sistemas en los que el usuario desconozca el estado de los recursos; se desconoce la carga de tareas totales ejecutándose y en espera, así como la cantidad y número de recursos disponibles. En estos casos, el usuario o aplicación simplemente se limita a describir las características de los *jobs* que desea ejecutar (por ejemplo, tiempo de ejecución, número de procesadores, memora RAM, etc), y es el sistema el que se encarga del resto.

2.5.3.1 Grid

En el paradigma *Grid* por el contrario, debido a su dinamismo y heterogeneidad, seleccionar los recursos óptimos para los *jobs* de una manera eficiente es más difícil, y hace de él un sistema complejo [196]. Pero para reducir esta complejidad inicial, existen los planificadores *Grid*, diseñados para hacer frente a este dinamismo, y proveer de un acceso desatendido y automatizado a la gran variedad de recursos existentes. Estos planificadores no manejan directamente los recursos, como en el caso de los planificadores locales, básicamente reúnen información del sistema para a continuación utilizarla en la planificación de tareas. En la planificación *Grid*, la diversidad de sistemas y aplicaciones y las diferentes versiones y los modos de planificación han de ser consideradas.

Una gran diversidad de implementaciones de planificadores han sido propuestos [197] [198], pero sólo unos pocos han sobrevivido con el paso del tiempo, bebido quizás a la complejidad del sistema, su *escalabilidad*, dificultad mantenimiento o simplemente su dificultad de uso. De hecho, en la actualidad son sólo dos los planificadores en producción que predominan en la dos mayores *e-Infraestructuras Grid* en el mundo: gLite-WMS en EGI y Condor-G en OSG.

El mayor problema en la planificación de recursos *Grid* es que los planificadores utilizan la información publicada por los IS [199], y esta no siempre es correcta o está actualizada, ni es fiable [200]. Los IS de manera habitual suelen estar mal configurados o disponen de información incompleta o con errores que no son subsanados, manteniéndose por largos periodos de tiempo. Además, carecen de datos que ayudarían a una mejor planificación, como el número máximo de *jobs* que un cierto usuario puede encolar y ejecutar en un CE, la cuota de almacenamiento de un usuario en un SE, o estadísticas a cerca de la calidad y SLA de los *sites* (datos que son obtenidos normalmente mediante prueba y error). Sin embargo, estos comportamientos pueden ser mitigados con diversas técnicas; por ejemplo utilizando estadísticas pasadas de las ejecuciones

realizadas [201] (tiempos de espera y ejecución, número de ejecuciones satisfactorias en un *site*, etc).

2.5.3.2 Cloud

A la hora de aplicar planificación a recursos *Cloud* nos centraremos exclusivamente en la capa o tipo de servicio IaaS [202], por ser la capa *Cloud* a través de la que se ofrecen los servicios computacionales. La IaaS es gestionada mediante diversos tipos de VIMs [203], lo que se dificulta la provisión de recursos debido a que cada VIM suele proporcionar servicios a través de APIs de gestión propietarias, que están basados en formatos de datos personalizados, que utilizan diferentes tecnologías [204]. Además, debe tenerse en cuenta que existen distintos modelos uso de los recursos: *reserved*, *on-demand* y *on-spot* [205]; haciendo que la planificación no sea trivial. Mientras las instancias *reserved* y *on-demand* tienen un precio fijo por hora, la instancias *on-spot* proporcionan una capacidad a de cálculo sin compromiso inicial y con un precio variable por hora, con un límite superior definido por el usuario [206]. Todo esto unido a los errores potenciales que según [207] existen a la hora de ejecutar aplicaciones hace del paradigma *Cloud* igual o más complejo de tratar que el *Grid*.

Para facilitar o al menos reducir la complejidad subyacente a usuarios y aplicaciones los *cloud-brokes* hacen de intermediarios entre los proveedores *Cloud* y estos; de esta forma, los *cloud-brokes* permiten elegir los mejores proveedores en función de diferentes parámetros (precio, rendimiento de cálculo, etc.) o desplegar servicios en múltiples *Clouds* pudiendo utilizar varios de manera concurrente. Así, se pueden encontrar *cloud-brokes* como CloudBroker Platform [208], CompatibleOne o Slipstream orientados al despliegue y planificación de aplicaciones específicas; y otros más generales como VMDIRAC, que proveen de una completa gestión para la ejecución de *jobs* en entornos *Cloud*. Pero a pesar de esta gestión de recursos avanzada, ninguno de ellos expone sus servicios mediante un estándar de conexión como OCCI; que facilite su integración con planificadores para una posible interconexión por ejemplo.

Junto a los *cloud-brokes* también ha aparecido el concepto *cloud orchestration* [209], los cuales proveen y configuran de manera dinámica recursos *Cloud*. Pero a diferencia de los *cloud-brokers* estos se encuentran más orientados al despliegue de entornos virtuales y no al de aplicaciones. Ejemplos de *cloud orchestrators* serían EC3 [210] o OCCO [211].

Otra posibilidad es utilizar los estándares de conexión *Cloud* junto a procesos de *contextualizacion* para gestionar directamente la infraestructura a través de un *service manager* [212]. En este caso de las VMs son accedidas directamente mediante SSH. Esta estrategia es extensible a cualquier proveedor, aunque puede resultar difícil de manejar de una manera eficiente si el número de instancias es elevado.

En cualquier caso, ya sea a través de un *cloud-broker*, de un *cloud orchestrator* o accediendo directamente, al igual que en las infraestructuras *Grid*, los IS en *Cloud* también adolecen de información; como por ejemplo, las cuotas establecidas por los proveedores de servicios a los usuarios, la capacidad del proveedor, o estadísticas de uso.

2.5.4 Topologías de meta-planificación

Los problemas de meta-planificación se pueden abordar desde diferentes perspectivas y topologías que influyen de manera notable en su eficiencia. En [193] se clasifican las topologías de meta-planificaciones en: centralizada, *hierarchic* y descentralizada (ver Figura 2.1), aunque el esquema *hierarchic* suele caracterizarse simplemente como una extensión de las centraliza. Esencialmente, la diferencia entre ellas radica en el control de los recursos, además del conocimiento global de la infraestructura. En el caso de la meta-planificación centralizada, existe un mayor control sobre los recursos, el planificador tiene un conocimiento amplio del estado de la infraestructura(s) a través de la monitorización, y por lo tanto, los planificadores tienen a ser más eficientes. Sin embargo, este tipo de topología suele adolecer de problemas de escalabilidad y tolerancia a fallos, debido a su propia arquitectura la cual genera cuellos de botella.

La topología *hierarchic* presenta una tolerancia a fallos y *escalabilidad* mejor que la centralizada, aunque mantienen los mismos problemas. En este caso, existen varios planificadores en distintos niveles inter-conectados, donde los planificadores en los niveles bajos disponen de un conocimiento de los recursos.

En la planificación descentralizada o distribuida no existe un elemento central encargado de la gestión y coordinación. En consecuencia, resulta mas complicado obtener un planificador tan eficiente como en el esquema centralizado, a cambio ofrecen una mayor *escalabilidad*. En este caso, los usuarios interaccionan con los meta-planificadores y estos a su vez interacciones entre sí y con los planificadores locales de cada recurso.

2.5.4 Topologías de meta-planificación

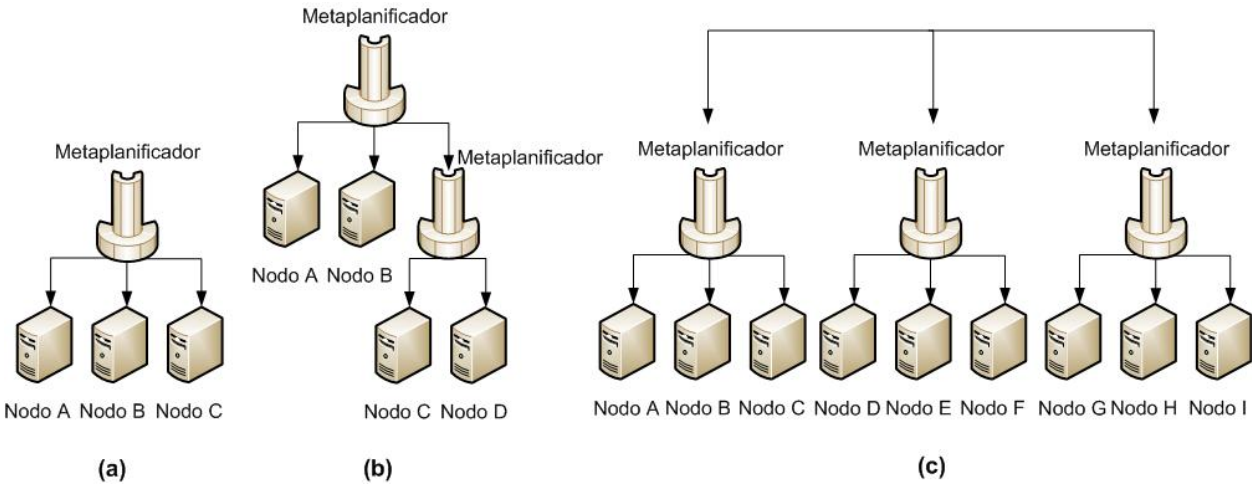

Figura 2.1: Esquemas de meta-planificación (a) centralizada, (b) híbrida y (c) distribuida.

2.5.5 Requerimientos para la meta-planificación

Una vez realizado el análisis de las características fundamentales de los paradigmas HPC, *Grid* y *Cloud* presentados en la sección 2.3.1 y de las técnicas y herramientas utilizadas en su planificación en la sección 2.5.3 así como de las posibles topologías de meta-planificación en la sección 2.5.4, es necesario elaborar una lista con los requisitos o requerimientos mínimos necesarios para aplicar técnicas de meta-planificación sobre multi-infraestructuras y/o multi-paradigmas. Debe recordarse que entre los objetivos de esta Tesis está proveer de una herramienta uniforme para el uso no sólo de múltiples infraestructuras sino también de múltiples paradigmas. Por lo tanto, se trata de identificar las características clave en el desarrollo de la planificación de estos escenarios.

Según se muestra en la sección 2.2.2, el que una infraestructura sea federada, independientemente de su tipo, facilita su explotación. Pero esta condición por sí sola no es suficiente. Además existen también otro tipo de infraestructuras que actualmente son utilizadas por la comunidad científica para el desarrollo de sus investigaciones, como son los *Cloud* públicos y cuyo máximo exponente es AWS [213] [214]. Tampoco debe olvidarse de elementos de cálculo que los usuarios utilizan de manera habitual, y que normalmente no son considerados debido a su capacidad como el propio PC, una *workstation* o un *cluster* departamental.

2.5.5 Requerimientos para la meta-planificación

Por lo tanto, con el objetivo de agregar distintos tipos de paradigmas e infraestructuras para su uso distribuido por parte de un usuario final, a continuación se indican los requerimientos suficientes que a nuestro entender deben de cumplirse. Esta lista ha sido confeccionada a partir de los requerimientos previamente enumeraros en [215]:

- Acceso a IS de las infraestructuras para el descubrimiento de recursos disponibles y sus características principales si existen.
- *Interoperabilidad* y flexibilidad para lograr una gestión eficiente de los recursos y tareas computacionales.
- *Escalabilidad* tanto en número de tareas o *jobs* como en número de recursos o infraestructuras. Este es uno de los requerimientos más críticos en la actualidad con el desarrollo del *Big Data*. Y a su vez es uno de los menos testeados, puesto que que normalmente se realizan estudios con cientos [216], y a veces miles de *jobs* [217]; pero en periodos relativamente amplios de tiempos como meses o años.
- Gestión del dinamismo de los entornos *Grid* y *Cloud* que genera impredecibilidad en la planificación.
- Gestión de la distribución geográfica entre recursos de distintas infraestructuras o dentro de la misma infraestructura.
- Abstracción del tipo recurso que permita desplegar *jobs* en diferentes entornos.
- Provisión de recursos en función de la carga de trabajo disponible (por ejemplo VMs para picos de carga de trabajo).
- Re-planificación tareas en caso de fallo de un recurso.
- Migración oportunista de tareas. Evaluación de los beneficios de la migración de los *jobs* enviados a nuevos recursos disponibles mediante la comparación del *ranking* de recursos.
- Reserva avanzada para la mejora de la eficiencia en la planificación.
- Políticas de planificación para definir el balance de la carga de los *jobs* entre los recursos disponibles.
- Capacidad de *checkpointing* necesaria para apoyar la migración bajo demanda, la migración oportunista y la tolerancia a fallos.

2.5.5 Requerimientos para la meta-planificación

- Planificación adaptativa que considere las características particulares de cada infraestructura para el uso eficiente de *jobs* forma distribuida.
- Uso de experiencias previas de los recursos como fuente de información para la planificación futura.
- *Accounting* de los recursos utilizados así como su influencia en la planificación. Por ejemplo, elementos de coste (*Cloud*) y horas de cálculo (HPC).
- Seguridad en el acceso a paradigmas e infraestructuras. Normalmente, este problema está fuera del alcance de la meta-planificación y los meta-planificadores suelen ignorarlos.
- Gestión de identidades que incluye su almacenamiento, delegación, cifrado, tiempo de uso, etc.
- Desacoplamiento o acoplamiento débil entre *jobs* y recursos.
- Gestión del acceso a recursos de almacenamiento dependiendo del recurso y/o la infraestructura.
- Capacidad de descripción de los requerimientos de las tareas: número de procesos, memoria RAM, tiempo mínimo necesario para su ejecución, etc.
- La capacidad de dependencia entre tareas para el envío de *jobs* dependiendo de la finalización de otros.

Esta lista pretende ser la guía a seguir para el desarrollo de un *framework* que gestione de manera eficiente distintos tipos de *jobs* y recursos computacionales a través de distintos tipos de infraestructuras. Según se indica en [218] existe un número reducido de herramientas ampliamente testadas que podrían cumplir si no todas, al menos una parte de esta lista como: GridWay, DIRAC, gUSE/WS-PGRADE [219] o Condor-G. Sin embargo, como también indica [218] sólo uno de ellos, GridWay, dispone de la tecnología de planificación necesaria para realizar una planificación adaptativa, tolerante fallos y con migración oportunista; además de poseer una infraestructura modular que facilita su integración con cualquier tipo de infraestructura y/o paradigma. Todo ello será analizado en detalle en el siguiente capítulo de esta Tesis.

2.6 Conclusiones

En este capítulo se ha realizado un repaso general de las infraestructuras actuales disponibles para los investigadores y sus características principales. De igual modo, se han revisado algunos de los principales paradigmas de computación existentes en la actualidad (HPC, *Grid* y *Cloud*) como forma de catalogar las infraestructuras. Para finalizar, se ha presentado el uso de los planificadores como un elemento clave desde el que proporcionar una solución eficiente para explotar los recursos de las infraestructuras.

Como se ha detallado en el capítulo, existe en la actualidad una extensa y variada cantidad de recursos disponibles para los investigadores. Esto hace que el uso de los mismos sea complejo, incluso si estos se encuentran agrupados bajo *e-Infraestructuras* federadas. Como respuesta a esta complejidad, numerosas herramientas, entre ellas los planificadores, tratan de facilitar la explotación de los mismos de una manera eficiente. Pero, esta planificación por lo general se suele focalizar de manera independiente a cada paradigma de computación. Y son pocas las opciones ofrecidas (en producción) para explotar los paradigmas HPC, *Grid* y *Cloud* de una manera combinada y sencilla. Esta situación se debe en gran parte, a las diferencias existentes entre cada paradigma; para los que no existe un estándar de acceso único a los recursos.

Para concluir, en la sección 2.5.5 hemos indicado aquellos requerimientos mínimos que un (meta-)planificador debería de disponer para cubrir esta problemática, así como una lista con algunas de las soluciones actuales que son utilizadas y que serán analizadas en el próximo capítulo. También se discutirá sobre la necesidad de un nuevo *framework* que trate las posibles carencias de los mismos.

Capítulo 3. DRM4G

3.1 Introducción

Distribited Resouce Management for Grid (DRM4G) [220] [221] es un *framework* diseñado desde el punto de vista del usuario final, para facilitar la explotación de *e-Infraestructuras*, federadas o no, e infraestructuras de computación en general que se circunscriben a paradigmas tales como: HPC, G*rid* y *Cloud*. En esta Tesis el concepto *Grid*, del término DRM4G, no se refiere una implementación concreta del paradigma, sino como la definición propuesta por Ian Foster en [35]; donde *Grid* es definido como un sistema de recursos distribuidos, en general, y que no está caracterizado por un *software* o *middleware* concreto.

El objetivo de este *framework* no es sólo extender los numerosos avances que se han implementado en meta-planificación sobre entornos *Grid* [194] a otros paradigmas como HPC y *Cloud*, puesto que DIRAC, gUSE/WS-PGRADE y Condor ya ofrecen esta posibilidad; el objetivo es desarrollar un *framework* que permita satisfacer las requisitos planteados en la sección 2.5.5. Esto en la práctica supone realizar una serie de mejoras sobre la implementación de GridWay actual, y la extensión del mismo a través de nuevos servicios. Proporcionado así una

manejabilidad, *interoperabilidad* y *escalabilidad* no satisfecha en la actualidad por ninguna herramienta en producción como: DIRAC, gUSE/WS-PGRADE, Condor o GridWay, y que los usuarios actuales demandan.

Para el diseño del DRM4G se ha optado por un desarrollo basado en el meta-planificador GridWay debido a que proporciona una planificación adaptativa avanzada que incluye tolerancia a fallos, con migración oportunista [222]. Aunque estas características también son ofrecidas por otros *frameworks* como Condor [223] [224], sólo GridWay a través de su estructura modular permite una fácil adaptación a distintos tipos recursos e infraestructuras.

Además se ha prestado especial atención a la sostenibilidad futura del *software* en lo que a su evolución en el tiempo concierne. El código empleado ha sido liberado bajo una licencia *open source* [225] y se encuentra disponible en GitHub [221]. De tal forma que cualquier usuario pueda colaborar libremente aportando sus modificaciones y mejoras al desarrollo de la aplicación, o simplemente descargarla e instalarla.

Este capítulo se centrará en los desarrollos llevados a cabo, así como las modificaciones realizadas sobre GridWay, para crear DRM4G. Además, se realizara una comparativa con las soluciones implementadas y las disponibles en los otros *frameworks* seleccionados, con el objetivo de mostrar como DRM4G es una herramienta que cubre las necesidades demandadas por los usuarios actuales.

3.2 Arquitectura

Para comprender el funcionamiento de DRM4G es necesario primeramente conocer el de GridWay. Por lo tanto, en esta sección se realizará un resumen de los principales elementos que componen la arquitectura de GridWay, seguida de su posterior comparación con la existente en DRM4G.

3.2.1 GridWay

GridWay es un meta-planificador desarrollado inicialmente para utilizar los servicios básicos de proveía Globus Toolkit ofreciendo una funcionalidad superior tanto a usuarios como aplicaciones, que simplificaba el uso el paradigma G*rid* [226]. GridWay fue diseñado con una

arquitectura modular, como se aprecia en la figura 3.1, que permite una adaptación sencilla e independiente del meta-planificador a distintos tipos de infraestructuras [227]. Dentro de esa arquitectura diferenciamos distintos componentes o elementos como *User Interface*, *GridWay Core* (GWC), *Scheduler* y *Middleware Access Drivers* (MADs).

El **User Interface** consiste en una serie de comandos o CLI, similares a los ofrecidos por sistemas LRMS, y un interfaz DRMAA [228] [227] que habilitan al usuario para el envío, monitorización y manejo de *jobs*. *Jobs* que son descritos de acuerdo a plantillas, donde el usuarios pueden especificar las características y requerimientos de los mismos. El *user interface* también provee de un sistema de monitorización de los recursos computacionales disponibles y consumidos de las infraestructuras. El *user interface* resulta independiente del tipo de recurso y/o infraestructura.

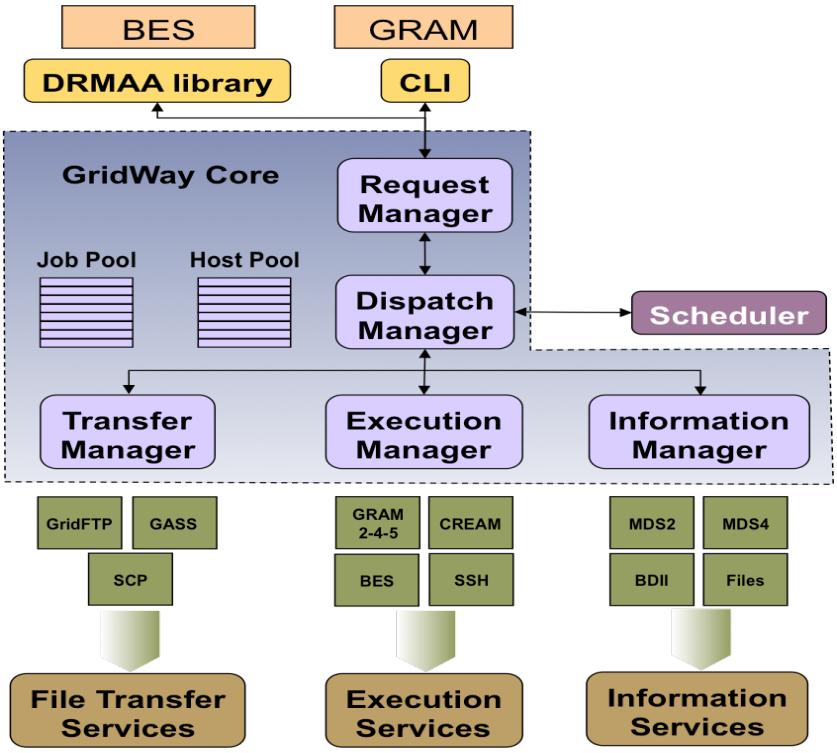

Figura 3.1: La arquitectura de GridWay está compuesta por: un User Interface, GridWay Core, Scheduler y Middleware Access Drivers. Imagen obtenida de [227].

GWC es el encargado de la gestión y ejecución de *jobs* y dispone de:

- *Request Manger* (RM) que maneja las peticiones del usuario.
- *Dispatch Manager* (DM) encargado de la planificación y monitorización los *jobs* a lo largo de sus distintas etapas.

- *Information Manager* (IM) para descubrir y monitorizar los recursos disponibles.
- *Execution Manager* (EM) responsable de la ejecución y monitorización de *jobs* en los recursos
- *Transfer Manager* (TM) encargado de la copia de ficheros y manejo de los directorios remotos.

Los **MADs** son componentes pueden ser configurados como *plugins*. Estos *plugins* interactúan directamente con las infraestructuras para: obtener información de las mismas (IM), la gestión de *jobs* (EM), y la transferencia de archivos y gestión de directorios de *setup* (TM). Los MADs implementan una serie de operaciones básicas que pueden ser adaptadas a cualquier tipo de infraestructura [228].

El **Scheduler** la planificación se realiza considerando la lista de los *jobs* pendientes de planificar, los recursos disponibles que cumplen los requisitos de los *jobs* y el pasado de estos. En consecuencia, el algoritmo de planificación hace uso del sistema configurable de políticas y condiciones del usuario por cada *job* y recurso. El proceso de planificación en GridWay no se encuentra integrado en el GWC, de hecho, este proceso es implementado en un programa que se ejecuta de manera independiente. Este dispone de un interfaz propio que facilita la implementación de nuevas políticas de planificación [229] o mejoras en su planificación [230].

3.2.2 DRM4G

La arquitectura de DRM4G ha sido diseñada teniendo en cuenta los requerimientos marcados en la sección 2.5.5. La figura 3.2 muestra un diagrama con la arquitectura donde se muestran todos los componentes del *framework* DRM4G: *DRM4G User Interface*, *GridWay Core*, *DRM4G Scheduler* y *DRM4G MAD*. La figura utiliza la arquitectura de GridWay, descrita en la sección anterior, como base para reflejar los cambios realizados sobre la misma.

El ***User Interface*** dispone de un CLI [231], que es similar a los comandos disponibles en sistema LRMS, y de una API DRMAA para Python [232]; ambos similares a sus homólogos existentes en GridWay, pero con sutiles diferencias. Estas diferencias radican en la necesidad de atender las nuevas características que los paradigmas HPC y *Cloud* plantean, y que no son atendidas por el interfaz actual de GridWay. En esta lista de características encontramos entre otros, elementos tales como: configuración y gestión de las identidades de los usuarios en el

acceso a infraestructuras, variables para la definición de entornos de ejecución paralelos como *Process Per Node* (PPN), descripción de entornos de almacenamiento distribuido o el control del gasto de ejecución de instancias en infraestructuras de tipo *Cloud* (ver sección 3.3.2).

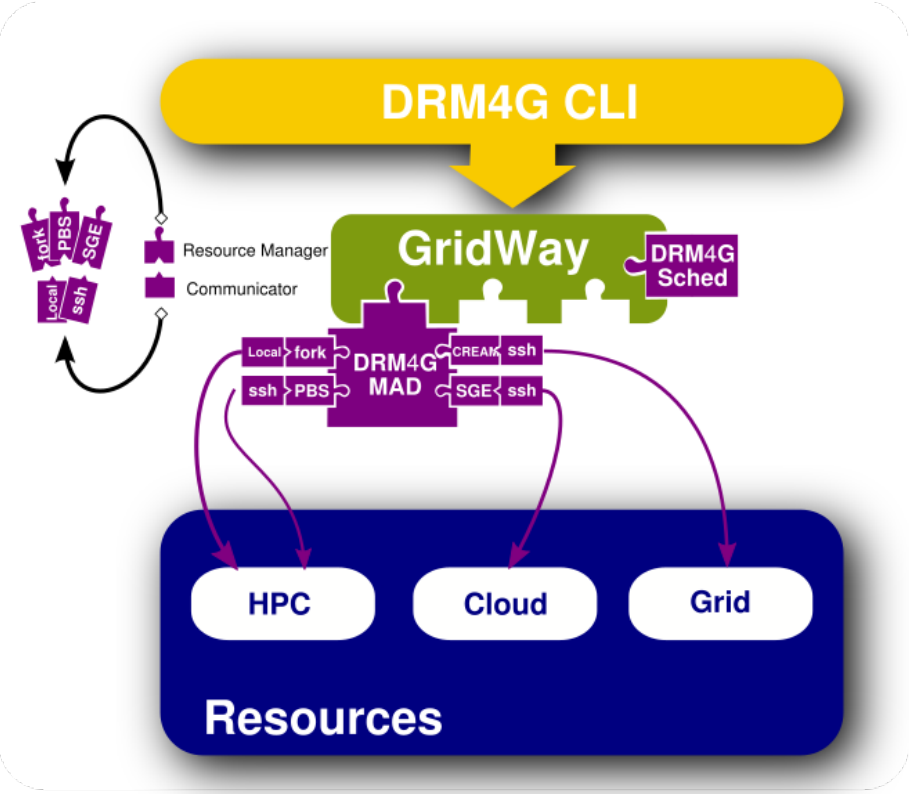

Figura 3.2: La arquitectura del DRM4G basada en la arquitectura de GridWay.

Una de las grandes aportaciones de GridWay es su *modularidad* [233], que permite su extensión en múltiples sentidos, sin necesidad de modificar todos sus elementos. Esto ha permito que el ***GridWay Core*** sólo haya sido modificado levemente para añadir el *accounting* de *jobs* a nivel de las colas de cada recurso de computación o *host* (nomenclatura usada en GridWay), el cual, no era considero por GridWay; y para la consideración de nuevas variables de configuración en el proceso de planificación (ver sección 3.4.1).

El ***Scheduler*** es uno de los elementos sobre el que se ha producido una mayor actuación. Esta se ha llevado a cabo en dos sentidos: por un lado, mediante la adaptación del planificador a las peculiaridades que cada paradigma plantea (ver sección 3.4.1), por otro, mejorando la *escalabilidad* mediante la combinación de un sistema de colas FIFO gestionada concurrentemente por *threads* (ver sección 3.3.5.2).

3.2.2 DRM4G

Para finalizar, el **DRM4G MAD** se ha diseñado desde cero siguiendo una estructura modular como la planteada por GridWay. En consecuencia, existe una sería de elementos denominados *Resource Managers* (RM) y *Comunicators* (ver figura 3.2) empleados para gestionar los distintos tipos de recursos y el accesos a los mismos respectivamente. Estos pueden ser combinados en el modo en el que el usuario indique. Si ninguno de ellos ofrece parcial o totalmente las funcionalidades que el usuario demanda, estos pueden ser extendidos como una clase de Python [234]. El catalogo actual de clases cubre todos los interfaces de recursos planteados en la sección 2.3.2 y la combinación de estos con los accesos mediante SSH, GSISSH y acceso local donde se haya desplegado la aplicación. Por otro lado, el diseño realizado también tiene en consideración la *escalabilidad* mediante la gestión y uso de sistemas de colas y *pools* de *threads* para gestionar de manera concurrente cientos de miles de *jobs*. En este sentido, los accesos remotos (SSH y GSISSH) se realizan utilizando *multiplexación* de manera que sólo una conexión es abierta por recurso y/o infraestructura (ver sección 3.3.5.1).

3.3 Características principales

Esta sección pretende a modo de resumen mostrar las principales características que DRM4G aporta y/o le diferencia respecto a los actuales *frameworks* de acceso a recursos distribuidos que fueron seleccionados en capítulo anterior, y que son utilizados actualmente por diferentes VRCs en la actualidad: DIRAC, GridWay, gUSE/WS-PGRADE y Condor.

3.3.1 Despliegue

DRM4G es una aplicación *open source* desarrollada en Python, compatible con Python 2.6+ y Python 3.3+, y que está disponible a través del repositorio de paquetes Python PyPI [235]. Por lo tanto, DRM4G se puede instalar mediante gestores de paquetes de Python como *easy_install* [236] o *pip* [237]. Esto gestores simplifican la instalación de paquetes Python gestionando posibles dependencias, como en este caso la instalación del propio GridWay (debe recordarse que DRM4G utiliza el *core* de GridWay). Esto en la práctica se traduce que cualquier usuario puede instalar DRM4G en cualquier entorno Linux (estas limitaciones están impuestas por GridWay que no es soportado en entornos Windows o Mac OS X) mediante la ejecución de un simple comando y sin necesidad de ser administrador del sistema como se indica en [238]. Al

tratase de una aplicación destinada al usuario final, uno de los requerimientos indispensables impuesto al inicio de su desarrollo fue que el usuario no necesitara permisos de administrador o *root* en el entorno de instalación (ver sección 2.5.5).

Por el contrario, la instalación de aplicaciones como DIRAC requiere de una versión especifica de Python y de múltiples dependencias de binarios, en gUSE/WS-PGRADE se recomienda versiones específicas de Java OpenJDK(1.7.x) y MySQL(5.1) para desplegar Liferay [239], y Condor se recomienda ser instalado mediante repositorios de paquetes [240].

3.3.2 Acceso distribuido a datos

La gestión del acceso a *datasets* desde cada *job*, tanto para datos de entrada como de salida, suele ser una tarea que recae sobre el propio usuario que define el *job*. Especialmente si los datos se encuentran en distintos sistemas de almacenamiento que a su vez disponen de diferentes sistemas de acceso. Esta situación se trata de paliar ofreciendo variables como *input* o *output sandbox*, que permiten agregar directamente ficheros a los *jobs*. Estos viajan como ficheros adjuntos a los *jobs* independiente de donde se ejecuten. Pero el tamaño de los ficheros a adjuntar suele estar limitado, bien por el *framework* (por ejemplo, 100MB en gUSE/WS-PGRADE y DIRAC) o por la propia infraestructura (por ejemplo, en *Grid* es típicamente 100MB). En consecuencia, y para los volúmenes actuales de *datasets* esta aproximación no resulta viable. Estas variables de descripción de ficheros de entrada y salida también permiten indicar ficheros utilizando diferentes tipos de URL/URI, pero estos valores suelen mantenerse de forma estática para cada *job*; no siendo dependientes por ejemplo del recurso en el que se ejecuta el *job*.

En DRM4G se facilita la descripción de múltiples repositorios de entrada y salida de manera independiente, adecuados a cada tipo recurso y/o infraestructura. Por lo tanto, un mismo *dataset* puede tener varios *inputs*, y estos pueden estar distribuidos en distintos tipos de repositorios de acuerdo al tipo de recursos donde un *job* se vaya a ejecutar. Esta configuración se realiza mediante la inclusión de secciones independientes en la descripción de los *jobs* como muestra el ejemplo de la figura 3.3, y usando un *schema* de tipo URL/URI. La variedad de protocolos soportados para estas interacciones son: s3/s3n/s3a [241], hdfs [242], swift [243], http/https, rsync, sftp/ftp, file (ficheros locales a la ejecución), gsiftp y lfn(gLite LFC [244]).

```
EXECUTABLE   = mpirun_script.sh

STDOUT_FILE  = stdout.${JOB_ID}
STDERR_FILE  = stderr.${JOB_ID}

NP           = 32

[Altamira]
INPUT_FILES  = file:///gpfs/proyects/group/user/input_file input_file
OUTPUT_FILES = output_file file:///gpfs/proyects/group/user/output_file

[EGIFedCloud]
INPUT_FILES  = swift://user:key@bucket/input_file input_file
OUTPUT_FILES = output_file swift://user:key@bucket/output_file

[AWS]
INPUT_FILES  = s3n://user:key@bucket/input_file input_file
OUTPUT_FILES = output_file s3n://user:key@bucket/output_file
```

Figura 3.3: Descripción de un job paralelo en DRM4G con datos de entrada y salida dependientes de la infraestructura donde de ejecute. Los infraestructuras indicados son el supercomputador Altamira perteneciente a la RES, EGI Federated Cloud y AWS.

3.3.3 Gestión de identidades

La gestión de las identidades es una problemática ampliamente conocida en el acceso a recursos computaciones independientemente de su tipo [245] [246]. En general, esta gestión suele recaer de manera frecuente sobre el usuario, que es quien debe gestionar típicamente sus certificados X.509, sus claves públicas y privadas, sus *passwords* y los posibles *tokens*. Esta situación plantea a menudo un escenario ciertamente problemático, a la vez que complejo, para los usuarios que quieren configurar un recurso [247] [248], incluso si se trata de una *e-Infraestructura* federada. De hecho, la federación como tal no implica necesariamente una facilidad de uso.

En consecuencia, si este planteamiento se extiende de una identidad en una infraestructura a varias dentro de la misma infraestructura, o si se plantean varias infraestructuras con distintos tipos de identidades, la situación es compleja para el usuario. Debe recordarse también que las identidades en ciertos tipos de infraestructuras son temporales y han de renovarse periódicamente, como por ejemplo en los casos de *tokens* y *proxies*.

Los *frameworks* de forma general suelen proveer de herramientas para la gestión de identidades para infraestructuras de tipo G*rid* y *Cloud* basadas en certificados X.509, donde el usuario crea *proxies* y los asocia a VOs para su uso. Estas herramientas contemplan la

posibilidad de utilizar certificados de tipo *robot* [249] a modo de alternativa en vez de certificados X.509 propios, como en gUSE/WS-PGRADE o DIRAC por ejemplo. De esta forma, el usuario a través del certificado robot es abstraído completamente de la gestión de identidades. En cambio, en otras ocasiones como en recursos HPC, estas herramientas simplemente se limitan a indicar al usuario los paramentos que se han de completar para la configuración de un recurso: *username*, *password* si se requiere, *frontend*, directorio de la clave privada y pública, etc.

Estas soluciones disponibles resultan orientadas a tratar infraestructuras concretas, como por ejemplo *e-Infraestructuras* G*rid* y *Cloud* federadas, y de manera independiente, es decir, existe normalmente un interfaz o comando especifico para cada tipo de infraestructura. Esta situación obviamente no beneficia al usuario final en la gestión de sus identidades.

Por lo tanto, en DRM4G se ha tratado de reunir la configuración de diferentes tipos de identidades de diferentes tipos de recursos de forma que el usuario pueda: ver, configurar, renovar y revocar todas las identidades de cada recurso (pueden existir varias por recurso) o de todos los recursos con un sólo comando [231]. DRM4G ofrece soporte para recursos cuyas AAI requieran: certificados X.509, *proxies*, clave y usuario, clave privada y clave pública y *tokens*.

```
[user@local~]$ drm4g id gisela init --lifetime 168 --cert my_certificate.p12
--> Starting ssh-agent ...
      Adding '/home/user/.ssh/id_rsa' into ssh-agent for 168 hours
--> Copying '~/.ssh/id_rsa' to ~/.ssh/authorized_keys file on 'frontend'
--> Converting my_certificate.p12 key to pem format …
      Insert your certificate password:
--> Creating '~/.globus' on 'frontend'
--> Create a local proxy credential ...
      Insert your Grid password:
      Your identity: /DC=es/DC=irisgrid/O=unican/CN=user
      Creating proxy ..................... Done
      Proxy Verify OK
Lifetime set to 7 days, 0:00:00
```

Figura 3.4: Comando de configuración de identidades en DRM4G. Este ejemplo muestra el proceso de configuración para un recurso de tipo Grid que previamente a sido detallado por el usuario indicando sus parámetros GUI (frontend, username, clave privada, servidor myproxy, VO, etc).

A modo de ejemplo, la figura 3.4 muestra el comando necesario a ejecutar para que un usuario de DRM4G pueda configurar una identidad en la VO GISELA [250] perteneciente en la *e-Infraestructura* EGI. Para esta configuración, el usuario ha de disponer de un certificado X.509 en formato *p12* [251] y de acceso a un *Grid User Interface* (GUI) (el formato p12 es opcional si el usuario ya dispone de un certificado previo configurado bajo el directorio *~/.globus* en el GUI). Como se aprecia en la figura 3.4 DRM4G no sólo configura el certificado bajo el

directorio *~/.globus* en el GUI y crea un *proxy* de una semana de duración, sino que también configura el acceso al GUI si este no existiera, delegando la gestión de la clave privada sobre un *ssh-agent* [252].

3.3.4 Adaptabilidad

El uso de una infraestructura, aún siendo federada y aún perteneciendo aún mismo tipo de paradigma, puede presentar o presenta determinadas particularidades que hacen que no se accesible a través de un acceso estándar (entendemos por acceso estándar aquel que es comúnmente aceptado por la comunidad de administradores, desarrolladores y usuarios). Estas particularidades pueden estar impuestas por una política en concreto de la infraestructura (por ejemplo, comandos especiales para la gestión de *jobs*), por cuestiones de seguridad (por ejemplo, el puesto de acceso a la infraestructura no es el utilizado con normalidad) o simplemente por los gustos, arbitrarios o no, del conjunto de administradores de la misma (por ejemplo, directivas especiales en las plantillas y directivas de los *jobs*).

Para tratar este tipo de particularidades que con frecuencia un usuario final se encuentra, los *frameworks* ofrecen configuraciones *ad-hoc* para infraestructuras concretas. Así por ejemplo, gUSE/WS-PGRADE o Condor sólo permiten acceso a recursos computaciones previamente configurados por los administradores de los propios *frameworks* (debe recordarse que este tipo de aplicaciones multiusuario no son administradas por los propios usuarios). De forma que recae sobre los administradores la configuración o personalización de posibles nuevos recursos que un usuario demande, con el consiguiere tiempo de espera que esta situación suele conllevar. DIRAC por su parte, además de ofrecer configuraciones específicas para infraestructuras concretas, también ofrece la posibilidad al usuario de configurar sus propios recursos.

En DRM4G al tratarse de una herramienta centrada en el usuario, esta facilita un control total para la configuración. Los recursos se encuentran agrupados por tipos de RM y no por tipos de infraestructuras y, por lo tanto, el usuario los puede adaptar a cada infraestructura de la que disponga acceso. Pero como se ha indicado anteriormente, existen situaciones particularidades en las cuales las infraestructuras no pueden ser empleadas a través de una configuración genérica. Para estos casos, DRM4G facilita una serie de opciones de personalización que se indican a continuación:

- Plantillas de *jobs* configurales para cada RM con objeto de actualizar y/o modificar posibles directivas.
- Lista de los comandos utilizados para interaccionar con cada RM para facilitar su actualización. En ciertas ocasiones por ejemplo ha de indicarse un *path* concreto para el uso un comando o un grupo de comandos.
- Lista de los puertos utilizados por defecto en las conexiones en los *Communicators* con los *frontends* de las infraestructuras.
- Lista de los directorios locales y remotos empleados por la aplicación para ficheros temporales. El uso de cuotas de almacenamiento en las infraestructuras suele requerir la actualización de los mismos.

3.3.5 Escalabilidad

La *escalabilidad* no es normalmente estudiada en profundidad en el ámbito del acceso a recursos distribuidos. Por lo general, los autores suelen simplemente hacer hincapié en la *interoperabilidad* de los *frameworks,* y citan levemente el número de recursos y *jobs* que estos pueden asimilar [253].

En DRM4G en cambio la *escalabilidad* es considerada como un elemento fundamental y se tratada a dos niveles: a nivel acceso y gestión a las infraestructuras a través del DRM4G MAD, y a nivel de planificación de *jobs* y recursos mediante el DRM4G Scheduler. La combinación de ambos posibilita como se muestra en la figura 4.6 la planificación y gestión de cientos de miles de *jobs* de manera concurrente ejecutándose en infraestructuras de manera distribuida.

3.3.5.1 DRM4G MAD

El DRM4G MAD presenta un sistema escalable a través de *pools* de *threads* encargados de atender las distintas peticiones que el GWC plantea para gestionar los *jobs* enviados por los usuarios los recursos configurados para ellos. El uso de *pools*, mediante la determinación de valores mínimos y máximos de *threads*, permite un escalado dinámico de la aplicación que reduce o aumenta el consumo de recursos computaciones (CPU y RAM) en función de las necesidades del sistema. La consideración de un número máximo de *threads*, configurable por el

usuario, supone en la práctica que a partir de ese límite todas las posibles peticiones con encoladas y tratadas a medidas que los recursos disponibles son liberados. Esto hace que posibles situaciones de saturación (se considera una situación de saturación la gestión concurrente de cifras superiores a 100K *jobs*), puntuales o no, puedan tratase con la única desventaja del aumento de los tiempos de respuesta.

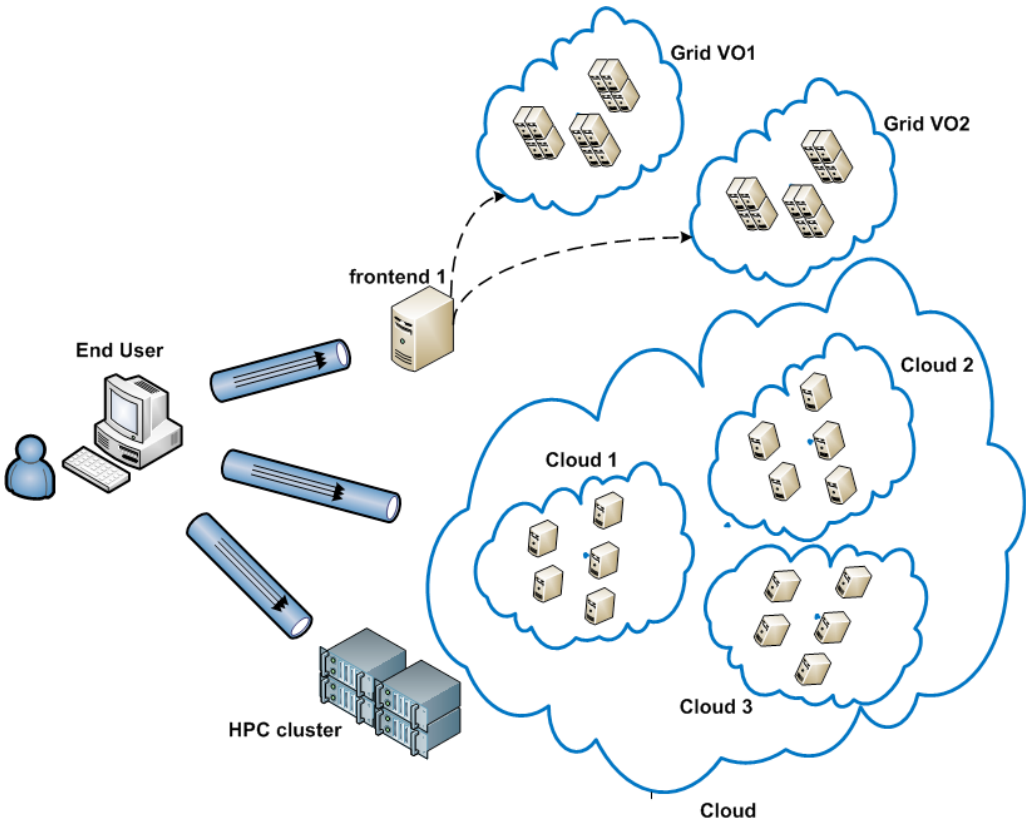

Figura 3.5: Acceso a recursos distribuidos desde el PC de un usuario. En este caso el usuario ha desplegado DRM4G en su PC y desde el accede a un cluster HPC a recursos Grid a través del frontend1 y a recursos Cloud. Las conexiones desde el PC están multiplexadas evitado así cientos de conexiones concurrentes para el envío y gestión de jobs.

Por otro lado, se encuentra la *multiplexación* en las comunicaciones (no aportada por ninguno de los *frameworks* citados hasta el momento). Esta permite cientos de consultas sobre los *frontends* en infraestructuras HPC o sobre los GUIs sobre una única conexión por recurso. Esto garantiza el envió de *jobs* y su monitorización de forma similar a la que un usuario realizaría para el manejo de un simple *job*. También a de tenerse en cuenta que situaciones de múltiples conexiones en las que se accede a servidores que facilitan el acceso a las infraestructuras pueden considerarse como potenciales ataques por parte del sistema y del administrador del mismo. En

DRM4G, de manera independiente al número de *jobs*, sólo una conexión es abierta y se mantiene abierta por recurso y/o infraestructura durante el uso de la infraestructura, en caso se un corte repentino se la conexión esta es re-conectada automáticamente sin que el usuario lo perciba quedando simplemente reflejada en los *logs* de la aplicación. La *multiplexación* a su vez, facilita que desde un PC se puedan manejar cientos de miles de *jobs* que pueden estar a su vez distribuidos por cientos de recursos. Situación fácilmente probable, si el usuario configura un *cluster* HPC, varias *Grid* VOs o infraestructuras *Cloud* con múltiples VMs como representa la figura 3.5. Las estrategias anteriormente indicadas son aplicables a cualquier tipo de RM y de manera extensible a sus nuevas versiones implementadas por los potenciales usuarios y/o desarrolladores, los cuales, sólo han de configuras aquellas características particulares de sus infraestructuras, despreocupase de este tipo de situaciones.

3.3.5.2 Scheduler

Como se muestra en [53] la *escalabilidad* de GridWay en términos de número de *jobs* de manera concurrente puede igualar la cifra de 100K *jobs*. Pero esta ficha hoy en día cuando se relaciona con las fichas que gestionan paradigmas como *Big Data* [254] puede resultar menor. Para superar esta barrera, pues según nuestros experimentos por encima de esta cifra GridWay es fácilmente susceptible de auto-bloquease, se han realizado varias modificaciones en el código de GridWay. Estas permiten como muestra la figura superar ampliamente esta barrera, pudiendo llegar a cifras en torno al 1M de *jobs*. Nótese, que en ningún momento se hace referencia al número de recursos. Esto se debe fundamentalmente a los órdenes de magnitud de recursos y *jobs*; los recursos computaciones suelen ser cientos y los *jobs* en cantidades superiores a los miles.

Las desventajas detectadas en GridWay que limitaban su *escalabilidad* en términos de planificación eran:

- El número de conexiones realizas desde el *scheduler* al GWC. La planificación de *jobs* en GridWay se realiza cada cierto intervalo de tiempo que es configurable por el usuario. Y cada planificación implica la comunicación mediante el uso de *sockets* [255] del planificador y el GWC, y, por lo tanto, al menos una conexión es abierta y cerrada por cada *job* y planificación (ver figura 3.6). En consecuencia, cuando el número de *jobs* a planificar es elevado, miles de conexiones pueden acabar

bloqueando mutuamente a ambos procesos. Siendo la única alternativa, el parar y arrancar de nuevo todo el sistema. Situación que supone una recuperación del estado de todos los *jobs* existentes, que se puede traducir a su vez en otro bloqueo al no poder GridWay gestionar todas la recuperaciones de forma consecutiva.

- El envío de un *job* por parte de los usuarios a un recurso a través de GridWay implica que ese *job* sea previamente planificado. En GridWay, esta petición es enviada directamente desde el GWC al *scheduler* en el momento en que el *job* es *enviado*, el cual las atiende de manera *síncrona*. Cuando el envió de una tarea implica más de un *job*, por ejemplo, cientos o miles de *jobs*, todos los ellos han de ser atendidos secuencialmente por el *scheduler*. Esta situación, cuando el número es superior a 10K *jobs* genera un bloqueo del *scheduler* que se propaga al GWC; y como resultado el proceso de envío de *jobs* se bloquea de manera permanente. Y de nuevo, la única alternativa es volver reiniciar todo el sistema.

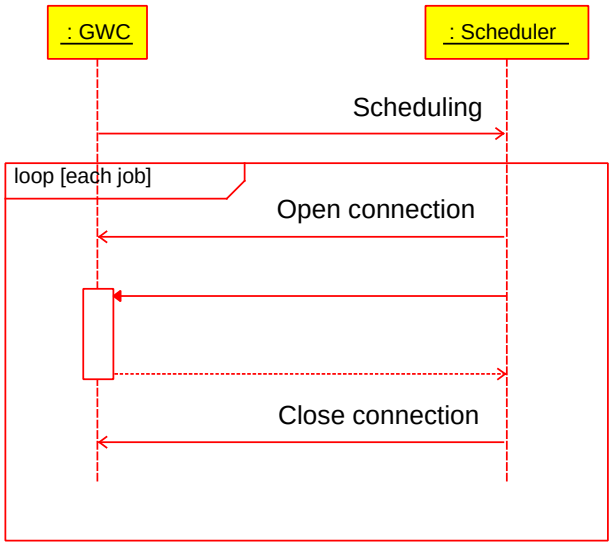

Figura 3.6: Secuencia actual de los accesos del planificador al GWC en GridWay en cada planificación.

Para mitigar y/o eliminar estas desventajas citados con anterioridad, las soluciones adoptadas han sido:

- En el caso de las conexiones, se ha optado la reutilización de *sockets*, que permite un número estable y continuo de conexiones independiente del número de *jobs*. Así, la conexión entre el *scheduler* y el GWC es única y continua. Situación que a su vez reduce

el tiempo de planificación, al evitar la apertura y cierre de las conexiona de manera continuada. La figura 3.7 muestra la secuencia de la solución adoptada.

- Para mejorar la gestión sincronía de peticiones por parte del planificador se ha optado por una gestión asíncrona mediante una cola FIFO operada por *threads*. De esta forma, todas las peticiones recibidas por el *scheduler* son encoladas mediante un *thread*, mientras otro es encargado de procesarlas una a una. El resultado es una asimilación más rápida de *jobs* por parte de GridWay (alrededor de un 75% más rápido, ver sección 4.2.1.1), a la vez que ha aumentado el número de *jobs* que de manera concurrente puede procesar. La tabla 50 muestra los tiempos medios necesarios para aceptar de manera secuencial tareas compuestas por diferente numero de *jobs*. Nótese que los cambios realizados permiten incluso la asimilación de tareas compuestas por un 1M de *jobs*.

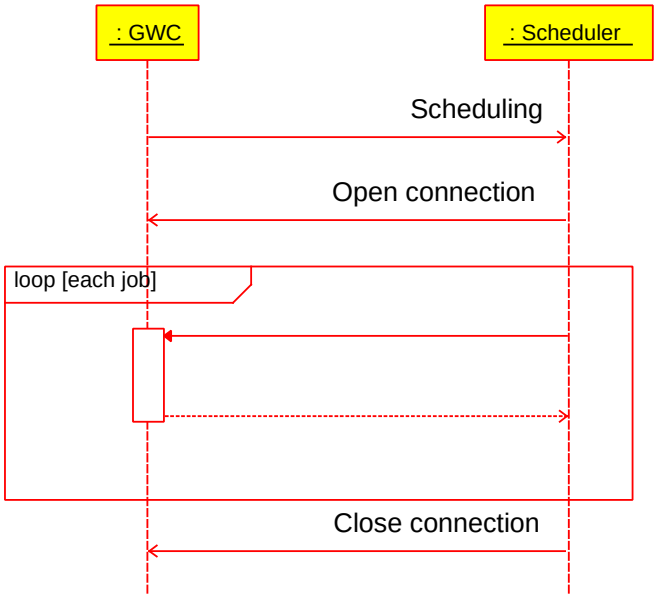

Figura 3.7: Secuencia de la solución propuesta para reducir el número de conexiones entre el scheduler y el GWC.

3.3.5.2 Scheduler

Numero de *jobs*	Tiempo
1K	3 segundos
10K	34 segundos
100K	9.76 minutos
1M	161.83 minutos

Tabla 3.1: Tiempos de asimilación de tares en función del número de jobs tras la mejoras..

3.3.6 Interoperabilidad

El acceso simple o combinado a múltiples infraestructuras supone un reto tanto desde el punto de vista de su desarrollo técnico, como de la propia definición de los mismos. Esta sección trata de recalcar una de las mayores problemáticas de los meta-planificadores que es la complejidad en el acceso y configuración de los recursos. A continuación, se indicará a través de cada paradigma estudiado en esta Tesis los distintos tipos de recursos e infraestructuras que DRM4G permite configurar. Además, se mostrarán ejemplos de configuración de alguna de las infraestructuras más representativas a nivel nacional e internacional (ver sección 2.2.2.1).

3.3.6.1 HPC

Los tipos de LRMS y, por tanto, los RM existentes en DRM4G son: TORQUE/PBS (*pbs)*, Grid Engine (*sge)*, LoadLeveler (*loadleveler)*, LSF (*lsf)*, FORK (*fork)*, SLURM (s*lurm)* y RES (*slurm_res)*. Los cuales, como se indicó en la sección 3.3.4 son configurables por el usuario para atender a posibles necesidades particulares de una infraestructura. De hecho, el LRMS RES es una extensión de SLURM, que ha sido modificado para atender las directivas existentes en la RES. Todos ellos además son combinables y accesibles con a través de accesos SSH y GSISSH. La figura 3.8 muestra una posible configuración para realizar *tests* en el supercomputador MareNostrum3 [256], que está presente en las *e-Infraestructuras* PRACE y RES.

El acceso a infraestructuras HPC implica a veces el desconocimiento de los recursos que realmente se encuentran disponibles en cada momento como número el número de WNs disponibles, incluso si la infraestructura dispone de IS, como PRACE o RES. Para solventar esta carencia, el usuario dispone de dos variables de configuración denominadas *max_jobs_running* y

max_jobs_in_queue (DIRAC, permite una configuración similar), que indican respectivamente el número máximo de *jobs* que el usuario estima que serán susceptibles de ejecutarse de manera concurrente y el número de *jobs* que serán encolados en el sistema LRMS. Esta definición suele resultar crítica, puesto que para que un *job* sea ejecutado en un sistema LRMS suele depender de su prioridad, que a su vez depende en gran medida del tiempo de espera del *job* en el LRMS.

```
[MN3]
communicator      = ssh
username          = user
frontend          = mn1.bsc.es
private_key       = ~/.ssh/id_rsa
lrms              = lsf
queue             = debug
max_jobs_running  = 1
```

Figura 3.8: Descripción de la configuración para el uso del supercomputador MareNostrum3. En este ejemplo el meta-planificador se encuentra desplegado en un PC local y accederá al los recursos del MareNostrum3 a través del nodo de login mn1.bsc.es (frontend) mediante SSH (communicator) usando la clave privada (private_key). La cola a usar es debug (queue) que según el guía de usuario del MareNostrum3 [257] permite 1 job como máximo en ejecución (max_jobs_running).

Sin embargo, existen otros casos en los que el usuario dispone de una información detallada de la ocupación del *cluster* de computación y puede ajustar estos valores de forma más precisa. Pero, independientemente del conocimiento o desconocimiento de la infraestructura, el usuario puede modificar estos valores en cualquier momento para realizar de manera dinámica los reajustes que considere, que se traducirán se manera inmediata sobre los *jobs* existentes en el LRMS. Especialmente, en el número máximo de *jobs* que desea encolar, puesto que el número de *jobs* en ejecución depende normalmente de factores ajenos al propio usuario.

3.3.6.2 Grid

DRM4G dispone de varios RMs dependiendo si el tipo de infraestructura *Grid* utiliza un *middleware* u otro, existido RMs para CREAM (*cream*) y Globus Toolkit 2/4/5 (*globus*). El hecho de existir RMs para *middlewares* que ya soportaba GridWay reside en la falta de *escalabilidad* de los mismos. Su uso se sustentaba en la creación de *threads* independientes sin un número límite de creación para atender los *jobs* existentes en el sistema, situación que podría llegar a provocar un *overflow* si se produce un envío concurrente de miles de *jobs*. DRM4G permite además el acceso a recursos *Grid* de manera remota. En otras palabras, el usuario puede

configurar un acceso remoto (de tipo SSH o GSISSH) al GUI que tenga acceso para enviar *jobs* a infraestructuras *Grid*. De esta forma, múltiples infraestructuras pueden ser utilizadas a través de uno o varios GUIs sin necesidad de desplegar el meta-planificador en ninguno de ellos, también puede ser utilizada sobre la mis infraestructura pero con diferentes VOs. Esta posibilidad no ha sido implementada en ninguno de los *frameworks* que se han analizado hasta el momento.

A modo de ejemplo la figura 3.9 muestra la configuración de la VO EGI ESR [258] a través de un GUI remoto.

```
[VO_ESR]
communicator   = ssh
username       = user
frontend       = ui.meteo.unican.es
private_key    = ~/.ssh/id_rsa
lrms           = cream
vo             = esr
bdii           = bdii.grid.sara.nl:2170
myproxy_server = px.grid.sara.nl
```

Figura 3.9: Descripción de la configuración de la VO ESR EGI. En este ejemplo el meta-planificador se encuentra desplegado en un PC y accederá al los recursos Grid expuestos por la VO ESR por medio de un GUI. Los parámetros necesarios para la configuración serán, por un lado, los necesarios para conectarse al GUI: tipo de conexión (communicator), usuario en GUI (username), hostname del GUI (frontend) y la clave privada a utilizar para logearse (private_key); por otro lado los necesarios para describir la VO: nombre de la VO (vo), tipo de RM (lrms), BDII para obtener la información (bdii) y el servidor de proxies X509 para la autenticación (myproxy_server).

3.3.6.3 Cloud

El acceso a recursos IaaS *Cloud*, para la ejecución de *jobs* en la actualidad, se plantea de dos formas esencialmente: los recursos pueden ser creados de manera especifica para la ejecución de *jobs* concretos, y por lo tanto, cuando el *job* termina y no hay otro *job* a ejecutar el recurso es liberado y por lo tanto la VM es destruida, son los casos de DIRAC, gUSE/WS-PGRADE y Condor; o los recursos se encuentran disponibles *a priori* antes de la planificación de *jobs* (GridWay).

En DRM4G se ha optado por la segunda opción, extendiendo así las experiencias previas ya desarrolladas en GridWay como [91] [212] [259], que se basaban en el despliegue de entornos *Cloud* a través de *Grid middlewares*. Aunque a diferencia de estos desarrollos, DRM4G utiliza el interfaz que ofrece la librería Apache Libcloud para acceder de manera transparente y directa a

múltiples proveedores de infraestructuras *cloud*. Pero, como Apache Libcloud no dispone de un interfaz para la infraestructura EGI FedCloud, ha sido necesario diseñar otro conector *Cloud* adicional basado en rOCCI [260]. Mientras que para la *contextualización* de instancias se ha optado por *cloud-init*, evitando así la necesidad de que el usuario cree imágenes personalizadas de VMs para sus ejecuciones (ver sección 2.3.2.2).

El uso de instancias previamente disponibles a la planificación de *jobs*, facilita el control por parte del usuario del número y tipo de instancias creadas en todo momento en la infraestructura. Esta *instanciación* previa, facilita además reducir tanto el tiempo de ejecución de *jobs*, las VMs ya están *instanciadas* antes del proceso de planificación de *jobs*; como la incertidumbre y los errores suscitan el proceso de *instanciación* de una VM. La contraprestación de esta técnica suele ser la falta de adaptabilidad a las cargas dinámicas de trabajo. Aunque como se incida en la sección 3.4.1.3, DRM4G proporciona una planificación multinivel, donde el número de instancias también puede ser dinámico sin la necesidad de intervención del propio usuario. Este puede configurar expresiones de dependencia (ver figura 3.14) para que el número de instancias sera proporcional, por ejemplo, al número de *jobs* existentes en el sistema.

Una vez los recursos (instancias) se encuentran disponibles, estos son accedidos de igual forma que un recurso remoto, en el cual se ejecutan los *jobs* de acuerdo al RM configurado por el usuario (recordemos que en el proceso de *contextualización* el usuario puede configurar la VM e instalar aplicaciones). Existe también la posibilidad, como indica la sección 3.4.4, de utilizar de manera dinámica una estructura multi-planificador, donde otro DRM4G sea desplegado en cada instancia en el proceso de *contextualización*, siendo éste el encargado de gestionar a modo de LRMS local los *jobs* a ejecutar en la instancia. Situación que a su vez permite encolar *jobs* de manera local dotando de un mayor grado de autonomía a la planificación al no tratarse se una estructura centralizada.

Para configurar un recurso *Cloud* en DRM4G es necesario no sólo indicar cómo es el acceso proveedor de la infraestructura *cloud_connector* (por ejemplo *ec2*, *fedcloud*, *openstack* o *opennebula*), sino también los paramentos de configuración de las instancias a crear como: el identificador de la imagen en el repositorio, el tamaño o características de la imagen a utilizar (CPU, RAM, HD, etc) y el número de instancias que se quieren. En el caso de infraestructuras federadas, habría que identificar además los servicios IS, AAI y la VO a la que el usuario pertenece. Mientras en el caso proveedores de recursos, como AWS, sería necesario añadir la

región donde se desplegarán las instancias y las características de gasto a aplicar (ver sección 3.4.1.3) sobre las instancias.

```
[FedCloud_BIFI]
cloud_connector   = fedcloud
vo                = fedcloud.egi.eu
myproxy_server    = myproxy1.egee.cesnet.cz
endpoint          = http://server4-eupt.unizar.es:8787
image             = os_tpl#2ea8cce2-d13c-48af-b436-051b5a6366b1
size              = resource_tpl#m1-medium
instancies        = 10
extra_volume      = 60
public_key        = ~/.ssh/id_rsa.pub
lrms              = fork
max_jobs_running  = 1
```

Figura 3.10: Descripción de la configuración para el acceso al site Cloud BIFI perteneciente al la VO fedcloud.egi.eu. En este ejemplo el meta-planificador se encuentra desplegado en un PC local que dispone del interfaz rOCCI client configurado de acuerdo con [261]. Los parámetros los necesarios para describir la infraestructura son: nombre de la VO (vo), el servidor para la autenticación (myproxy_server) y el contact endpoint del site (endpoint) y el conector para la infraestructura (cloud_connector); los necesarios para describir las VMs: identificador de la imagen el el repositorio EGI Appdb (image), las características físicas de la VM (size), el número de instancias a desplegar (instancies), el almacenamiento extra que dispondrá la instancia (extra_volume), el acceso a la instancia mediante la clave pública (public_key); y los necesarios para la ejecución de jobs en las instancias desplegadas: el RM en las instancias y el número de jobs máximo a ejecutar en cada una de ellas (max_jobs_running).

La configuración de recursos *Cloud* en DRM4G incluye también la posibilidad de añadir almacenamiento extra a cada instancia. Esta característica no es usualmente tenida en cuenta entre los actuales *frameworks* (como solución se utilizan otras instancias que dispongan de más almacenamiento), y sin embargo resulta de vital importancia en la ejecución de determinadas aplicaciones. Debe tenerse en cuenta que las instancias por defecto suelen disponer de unos pocos GB de almacenamiento por defecto (8 GB típicamente) que no resultan suficientes muchas aplicaciones actuales. Por lo tanto, DRM4G ofrece una variable, denominada *extra_volume*, para indicar el volumen extra en GBs que la instancia dispondrá. Por ejemplo, en EGI FedCloud se realiza mediante *Block Storage* [262] y en AWS con *Elastic Block Store Block* (EBS) [263]. Nótese que no todos los proveedores disponibles a través de Apache Libcloud disponen de esta propiedad.

Finalmente, la figura 3.10 muestran el ejemplo de la configuración del *site* BIFI [264] perteneciente a la *e-Infraestructura* federada EGI FedCloud.

3.3.7 Acceso estándar mediante API DRMAA

Un elemento esencial para favorecer la interacción con un meta-planificador es proporcionar una API bajo un determinado estándar. Por consiguiente, en DRM4G se ha optado por un estándar ampliamente consolidado y ofrecido por sistemas como: PBS, SGE, SLURM, Condor o GridWay como es DRMAA. Esta API diseñada por el OGF para la configuración, supervisión y control de *jobs* proporciona en la actualidad un interfaz de alto nivel sobre diversos tipos de DRMS. Como se ha indicado previamente, Condor también ofrece dispone de esta API, mientras que en los casos de gUSE/WS-PGRAGE y DIRAC estos ofrecen una APIs para accesos remotos denominadas Remote API [265] y DIRAC RESTful API [266] respectivamente, pero ninguna de ellas bajo algún estándar.

La especificación DRMAA consiste en un conjunto de funciones, que se agrupan en rutinas de inicialización y finalización, rutinas de *templates* de *jobs*, rutinas de envío y monitorización de *jobs*, rutinas de control de *jobs* y rutinas auxiliares. La API se basa en el concepto de sesión, en el que todos los *jobs* enviados se agrupan en una sesión específica de una instancia. Esto permite al desarrollador de aplicaciones realizar la sincronización y supervisión en todos los *jobs* enviados por la instancia de una aplicación [267]. Siguiendo estas directrices, DRM4G implementa una API en Python con las siguientes rutinas que a continuación se detallan:

- Inicialización y finalización de sesiones: las sesiones se gestionan mediante el clase *Session*, que es la encargada de proporcionar todos los métodos para la descripción y manejo de *jobs*. Las sesiones se *inicializan* con el método *initialize*, que genera un objeto sesión para la gestión de los *jobs* que se envíen a través de la API. La finalización de la sesión se realiza con el método *exit*.
- Definición de *jobs*: los *templates* de los *jobs* a enviar se crean con el método *createJobTemplate*, que devuelve un objeto con los atributos configurales para el *jobs* como: *jobName* (nombre del *job*), *remoteCommand* (ejecutable), *args* (argumentos), etc.
- Envío de *jobs:* se realiza mediante los métodos *runJob* para *jobs* simples o *runBulkJob* para el envío de *arrays* de *jobs.*
- Control de *jobs*: el método *control* modifica el estado de los *jobs.*
- Sincronización de *jobs*: el método *wait* espera hasta la finalización de un *job,* mientras que *synchronize* permite esperar a una lista de *jobs.*

3.3.7 Acceso estándar mediante API DRMAA

- Monitorización de *jobs*: el método *jobStatus* devuelve el estado de los *jobs* que puede ser:
  - UNDETERMINED: el estado del *job* no puede ser determinado.
  - QUEUED_ACTIVE: el *job* está encolado en el recurso.
  - USER_ON_HOLD: el *job* ha sido detenido por el usuario.
  - RUNNING: el *job* está ejecutándose.
  - SYSTEM_SUSPENDED: el *job* ha sido suspendido por el recurso.
  - USER_SUSPENDED: el *job* ha sido suspendido por el usuario.
  - DONE: el *job* ha finalizado.
  - FAILED: el *job* ha finalizado, pero con un fallo.

3.4 Mejoras en la planificación

Los meta-planificadores actuales como GridWay o DIRAC se diseñaron originalmente para satisfacer las necesidades y la complejidad que los recursos de computación *Grid* planteaban [226] [268]. Y, por lo tanto, a medida que otros paradigmas han aparecido, como el paradigma *Cloud* entre otros, estos han evolucionado mediante extensiones como VMDIRAC en el caso de DIRAC, o añadiendo *service managers* [269] [270] para gestionar recursos *Cloud* en GridWay. Sin embargo, estas adaptaciones no se han trasladado a los planificadores de los *frameworks*; los cuales, han ido evolucionando a lo largo de las diferentes versiones, pero sin dar respuesta a las problemáticas específicas que cada paradigma suscitada. En esta sección veremos los cambios realizados sobre el planificador de GridWay, para adaptarlo a recursos HPC y C*loud*; y cómo se mejorado su adaptación al *Grid*.

Otro tema a tratar en esta sección será la meta-planificación multinivel como solución distribuida para el acceso a infraestructuras. En concreto, se mostrará cómo DRM4G facilita esta configuración sin necesidad de elementos intermedios como *middlewares*.

3.4.1 Planificación adaptativa

Algo común a todos los *frameworks* citados, son sus propuestas relativas a la abstracción de los diferentes tipos de recursos y/o paradigmas a los usuarios finales. Situación que suele conllevar que la planificación de *jobs* también sea abstraída del tipo de recurso. En otras palabras, todos los recursos son tratos por igual independientemente de su origen. Pero como se vio en la sección 2.3.1, existen particularidades de los paradigmas y/o de las infraestructuras que deben ser trasladadas a la planificación para una mejora de la explotación de los mismos. Concretamente, nos estamos refiriendo al consumo de horas de cálculo en HPC, al dinamismo y la fiabilidad en *Grid* y el coste económico (*billing*) que supone calcular en algunos entornos *Cloud*.

3.4.1.1 **HPC**

El uso de *e-Infraestructuras* HPC, como PRACE, implica de manera frecuente el uso de cuotas de cálculo. Estas cuotas, normalmente horas de CPU, han de ser tenidas en cuenta en la planificación de cada tarea, pues pueden conllevar posibles sanciones. Si bien, *e-Infraestructuras* Federadas ofrecen IS para estas tareas, existen otras infraestructuras que también disponen de este tipo de limitaciones y que no disponen de tales mecanicismos, como en la RES. En estos casos, son los propios administradores de la infraestructura quienes a través del correo del contacto del usuario y/o del grupo al que pertenece este es notificado. Este *accounting* propio puede permitir al usuario manejar de manera más eficiente sus propias ejecuciones, conociendo el consumo y eficiencia de su aplicación en la infraestructura. Además, estos datos resultan también importantes de cara a solicitar futuras más horas de cálculo para nuevos proyectos.

Para adelantarse a estas situaciones, se han añadido dos variables a la descripción de cada recurso. Estas se denominan, *cpu_hours_soft* y *cpu_hours_hard*. La primera indica el límite a partir del cual ya no se planificarán más *jobs* para una determinada infraestructura. La segunda implica que una vez superado este valor todos los *jobs* existentes en la infraestructura, sea cual sea su estado, serán eliminados. Ambas variables son optativas y pueden combinarse de acuerdo con las necesidades del usuario. Por otro lado, si en la descripción del *job* se indica el *cputime* que se estima el *job* necesita, esto se tendrá en cuenta también en la planificación del mismo como se muestra en la figura 3.11.

3.4.1.1 HPC

```
1 if not job_cpu_time :
2     job_cpu_time = 0
3 if current_cpu_hours + job_cpu_time >= cpu_hours_hard :
4     delete_all_jobs()
5 elif current_cpu_hours + job_cpu_time < cpu_hours_foft :
6     scheduling_job()
```

Figura 3.11: Pseudocódigo para planificar jobs en recursos HPC.

3.4.1.2 Grid

La planificación en entornos *Grid* ha sido, y es, una de las que más problemas suscita, como se recoge en la sección 2.5.3.1, y como resultado numerosas propuestas en forma de *frameworks* han sido diseñadas para dar una solución. Entre ellas se han desarrollado varias basadas en lo que se conocen como *pilot jobs* [271] [272] [268] [273], con el objetivo de evitar los *overhead* de los tiempos de espera en las colas de los CEs y de proveer de información directa y veraz al usuario de los recursos *Grid*. Así, una vez que el *pilot job* es desplegado en un WN, este se conecta a un servidor que le provee de las tareas pendientes a ejecutar, evitando los tiempos de espera en las colas. Si bien son ampliamente utilizados en simulaciones de tipo Monte-Carlo [274] [230] y en la ejecución de *short jobs*, su uso se ve reducido en aplicaciones que implican ejecuciones de días o semanas o en aplicaciones que necesitan más de un WN para ejecutarse (MPI). Además el envío de *pilot jobs* supone el envío de un *job* "normal" que ha de superar todas las trabas existentes en G*rid*, previo paso a la ejecución de tareas.

Aunque GridWay ha sido ampliamente testeado en entornos G*rid* con excelentes resultados [275] [276] [227] [53], es necesario mejorar su planificación para poder competir con este tipo de *frameworks*. GridWay puede beneficiarse de paramentos de los IS, que aún no han sido explotados por el meta-planificador, para obtener una mejor selección de recursos. En concreto, nos estamos refiriendo al número de *jobs running* (GlueCEStateRunningJobs) de cada LRMS de cada CE, y en el número de *jobs* en estado *waiting* (GlueCEStateWaitingJobs). La figura 3.11 muestra el pseudocódigo de como estos valores se han integrado dentro del proceso de cálculo de *slots* libres en los recursos. Aunque aún existen ciertos problemas en la configuración los IS de EGI [277], igualmente existe un gran esfuerzo para mejorarlos como mediante técnicas de validación de los esquemas GLUE mostrados desde los IS [278].

```
1 for host in host_list :
2   for i in numer_of_queues :
3     if host.queue_maxjobsinqueue[i] > 0 :
4       slots=host.queue_maxjobsinqueue[i] – host.queue_running_jobs[i] -
5            host.queue_waiting_jobs[i]
6     else
7       slots=host.queue_max_running_jobs[j] – host.queue_active_jobs[j] -
8           host.queue_waiting_jobs[i]
```

Figura 3.12: Pseudocódigo para calcular el número de slots libres.

Por otro lado, la lista de recursos candidatos en GridWay es ordenada por idoneidad, que es calculada como la suma de las contribuciones de tres tipos de prioridad(ver figura 3.13): fija, dependiente de la historia de uso del recurso y la expresión de *ranking* que cada *job* permite configurar [279]. Una manera de mejorar la idoneidad de la lista es mediante los parámetros de disponibilidad y fiabilidad de cada *site*, que se en encuentran disponibles a través del servicio ARGO de EGI. En nuestro caso será el IM el encargado de testear de manera rutinaria los valores para que el planificador pueda incorporarlos al sumatorio de prioridades. Estos valores, que oscilan entre 0 y 100%, son normalizados al igual que el resto de valores, sus pesos son configurados mediante las variables AV_WEIGHT y RB_WEIGHT, y el número de valores de la serie histórica a considerar por AV_HISTORY_WINDOW y AV_RB_WEIGHT (ver tabla 3.2). De tal manera, que el planificador justo desde el momento de inicio dispone de un *background* de los recursos a seleccionar.

$$P_h = \Sigma_i \, w_i \, p_{ih}$$

Donde i es cada política, w_i es el peso de cada política y p_{ih} es la prioridad normalizada de cada política

Figura 3.13: Expresión para el cálculo de la prioridad de los recursos en GridWay.

3.4.1.2 Grid

Variable	Significado
AV_WEIGHT	Peso de disponibilidad de un recurso
AV_HISTORY_WINDOW	Número de días de la serie histórica a computar para el cálculo de la disponibilidad
RB_WEIGHT	Peso de la fiabilidad de un recurso
RB_HISTORY_WINDOW	Número de días de la serie histórica a computar para el cálculo de la fiabilidad

Tabla 3.2: Variables para la configuración de los parámetros de disponibilidad y fiabilidad en recursos Grid.

3.4.1.3 Cloud

Una de las diferencias fundamentales entre la planificación aplicada a recursos de tipo IaaS C*loud* y otro tipo de recursos es la implicación que supone planificar *jobs* para recursos que como tal aún pueden no existir durante proceso de planificación. Esto en la práctica supone *instanciar* una o varias VMs, con sus consiguientes tiempos de arranque, previo paso a la ejecución de los *jobs*. Situación que a su vez, genera cierta incertidumbre, debido a que el proceso de arranque de la VM no siempre es satisfactorio y depende de múltiples factores [280]. Además, la existencia de diferentes *flavours* de instancias, unido a los distintos modelos de cobro (*reserved, on-demand* y *on-spot*) por parte de los proveedores de las infraestructuras, hacen compleja su planificación. Por consiguiente, se añaden una serie de variables extra que no son contempladas en otro tipo de recursos, en los cuales siempre es asumido la existencia del recurso previa a la planificación y envío de *jobs.* En consecuencia, existen múltiples estudios en este sentido que tratan de mejorar la eficiencia en la planificación de recursos IaaS [281] [282] [283] [284].

En el modelo de planificación planteado en DRM4G, se ha optado por una planificación multinivel, en el que se diferencia entre la planificación de *jobs* y la de instancias. La planificación de *jobs* se realiza sobre recursos que ya se encuentran disponibles y, por lo tanto, la planificación será similar a la de otros tipos de recursos. Y es que los *jobs*, son planificados si existe previamente un recurso, evitando así los tiempos de espera en la creación de las instancias, que ya se encuentran disponibles en el momento de selección del recurso. La idea principal detrás de dividir el proceso de ejecución de *jobs* en dos niveles es desacoplar la creación y

destrucción de instancias, de la ejecución de *jobs*. Esta estructura desacoplada permite que los usuarios puedan mantener un control total sobre el número, tipo y coste de las instancias, a la vez que estas pueden ser reutilizadas por múltiples *jobs*.

En la planificación de *jobs* se tendrá en cuenta variables como los valores de *billing* asociados a la ejecución de tareas. Se han considerado dos variables o límites que se asocian a cada recurso si el usuario así lo considera: *billing_soft* y *billing_hard*. El superar la barrera impuesta por el *billing_soft* supone que el recurso no será susceptible para su planificación por ningún *job*. Además, los *jobs* que están siendo ejecutados serán oportunamente migrados a otros recursos y las instancias existentes tras este proceso serán destruidas. Mientras, la superación del nivel *billing_hard* supondrá la eliminación inmediata de las instancias existentes sin atender a los *jobs* en ejecución en ellas. Para realizar estas consideraciones, el usurario deberá indicar el gasto por hora de la instancia en el recurso. Si este no se indica, DRM4G estimá el gasto en función del tamaño de la VM seleccionada y su precio a través de una serie de estimaciones por defecto. Exceptuando los casos de los proveedores de infraestructuras AWS y Google Cloud [285], donde DRM4G obtendrá de manara dinámica el precio actual de la instancia seleccionada.

Por otro lado, la planificación de VMs conlleva el desarrollo de un *scheduler* especifico para esta tarea. La idea es que el usuario se beneficie de una de las características más notables de los recursos *Cloud*, la elasticidad. De este modo, el número de VMs también será elástico, dando una respuesta adaptada a las necesidades del usuario en todo momento de manera dinámica. Las políticas de planificación de VMs han sido implementadas con los objetivos de minimizar los costes asociados a las infraestructuras C*loud* y los tiempos de espera de los *jobs.* El objetivo ha sido buscar un compromiso entre el coste del uso de la infraestructura y el tiempo de espera de los *jobs* a ejecutar. En este entorno, las políticas de planificación vendrán determinadas por cuándo es necesario crear una instancia y cuándo es necesario destruirla. Esta implementación contrasta por ejemplo con otros desarrollos donde también se ofrecen servicios de computación *Cloud* a través de planificación, pero donde el usuario no dispone del control y la personalización que se ofrece en DRM4G para el control de VMs [286] [287] [288].

A continuación se indican las consideraciones necesitaras para que una VM sea creada en DRM4G:

- El usuario puede determinar un número mínimo continuo de instancias para un recurso. Esta opción permite disponer siempre de número de instancias disponibles para la

ejecución de *jobs*. De esta manera, se evita que los *jobs* enviados tengan que esperar mientras se crean VMs para su ejecución. Esta opción suele ser útil si el usuario considera va ha realizar un envío continuo de ráfagas de *jobs* y, por lo tanto, las instancias no se van a mantener ociosas durante periodos largos. Para la definición de este valor el usuario debiera tener en cuenta el tiempo necesario crea una instancia, el cual suele variar dependiendo del proveedor y del tamaño de la misma [289]. Si el número mínimo de VMs no se define este será 0 y la creación de VMs recaerá en las otras políticas de planificación.

- Si existe un determinado número de *jobs* en estado *pending* durante un tiempo determinado. En esta caso, el sistema aumentará el número de instancias hasta el máximo indicado por el usuario, si este valor no es indicado se asume un valor máximo de 100. Esta política puede ser emplea en picos de carga de trabajo en los que se quiere puntualmente ejecutar tareas de manera automática en infraestructuras *Cloud*, o simplemente para reducir el tiempo de finalización de una tarea.
- Si un *job* se mantiene en un estado *de espera* por un tiempo determinado. Esta política trata de evitar que haya *jobs* esperando si existen recursos disponibles sin *instanciar*. Debe tenerse en consideración que si su valor es reducido se puede ocasionar la creación de instancias de manera innecesaria con el consiguiente coste asociado.

Por otro lado está el proceso de destrucción de instancias, el cual puede resultar aún más complejo. La destrucción de una instancia puede implicar que para atender la ejecución de un próximo *job* quizás sea necesario crear una nueva instancia, y a su vez esta situación implicaría un consumo extra en tiempo y coste. En consecuencia, para destruir una VM en DRM4G se realizan las siguientes consideraciones:

- Si se ha superado el límite de gasto impuesto por el usuario. Si se supera el *billing_soft* se esperará hasta que los *jobs* sean migrados a otros recursos. En cambio, si el que se supera es el *billing_hard* las instancias serán destruidas independientemente del estado de los *jobs* existentes en las instancias.
- Si no existen *jobs* ejecutándose en una instancia y no hay *jobs* pendientes para ejecutarse. Con esta política se trata de evitar que existiendo *jobs* pendientes de ejecución se destruyan instancias, reducido el tiempo de espera. También se intenta evitar así destruir

una VM que pudiera ser necesaria a muy corto plazo. Por lo tanto, mientras existan *jobs* pendientes de ejecución no se destruirán VMs.

- El modelo más extendido de facturación entre los proveedores IaaS *Cloud* es el precio por hora y por tipo de instancia. Eso significa que el precio por instancia es cuantificado al principio de cada hora durante la ejecución de la instancia y, si la instancia se ejecuta durante una hora y un minuto, el número total de horas cobradas será dos. Por consiguiente para el uso de *Cloud* públicos, una vez que un nuevo intervalo de uso ha sido pagado, la instancia no será destruida antes que este tiempo expire. De tal forma, que se mantenga el máximo número de instancias sin incrementar el coste asociado.

Un detalle importante a destacar tanto en la creación y como en la destrucción de instancias es la gestión paralela de las mismas mediante *threads*. Por consiguiente, el tiempo necesario tanto para crear como para destruir VMs es independiente del número indicado, dependiendo únicamente de la capacidad de la infraestructura para gestionar varias peticiones de manera concurrente.

Respecto al tipo de instancias o *flavours* que DRM4G puede emplear, ha indicarse que DRM4G no permite el uso de instancias *on-spot*; como ocurre con gUSE/WS-PGRADE que sólo facilita el empleo de instancias *on-demand* [290]. Mientras que DIRAC y Condor son capaces de desplegar instancias *on-demand* y *on-spot* [291] [292]. Esta limitación viene dada por la propia librería Apache Libcloud, que actualmente no dispone de esta implementación. Nótese que las instancias de tipo *reserved* no han sido mencionadas, esto debe a que estas instancias se emplean de la misma manera que las de tipo *on-demand*. La diferencia es que en el tipo *reserved*, el usuario previamente a la ejecución de instancias, ya ha reservado una cierta capacidad (en este caso horas de cálculo) y, por lo tanto, el coste por hora de una instancia es menor a la vez que existe una mayor disponibilidad a la hora de *instanciar* las VM.

Para finalizar, la figura 3.10 recoge la secuencia del proceso de planificación multinivel aplicado para ejecutar *jobs* en entornos IaaS *Cloud,* en el que se aprecian las siguientes acciones (en esta secuencia se supone que no existen *jobs* previos en el sistema y que tampoco existe ninguna VM creada con anterioridad):

1) Configuración de la infraestructura o infraestructuras *Cloud* a utilizar.
2) El usuario indica a partir de que instante los recursos son susceptibles de ser usados.

3.4.1.3 Cloud

3) Se comprueba si existen recursos libres para la creación de VM a través del IS de la infraestructura y se mostrarán al usuario. Sino existe esta información, como en el caso de AWS, se supondrá una capacidad ilimitada de recursos y esta fase será obviada.
4) Se evalúan las condiciones de creación de VMs para determinar si existe un número mínimo de VMs es necesario crear. Si existe y hay recursos libres disponibles se iniciará el proceso de creación de VMs usando como plantilla la imagen indicada por el usuario. Este proceso incluye también la contextualización de la instancia: gestión de usuarios, configuración de servicios, instalación de aplicaciones, etc.
5) El usuario envía los *jobs* a ejecutar.
6) Los *jobs* serán planificados de acuerdo a las instancias disponibles. Si el valor del *billing_soft es* superado los *jobs* existentes en las instancias serán migrados. En el caso de no existir recursos disponibles, los *jobs* se quedarán en un estando *pending* a la espera de posibles recursos.
7) Se evalúan las condiciones para crear VMs. Si se cumple alguna de ellas y si existen recursos disponibles se desplegarán nuevas instancias.
8) Se evalúan las condiciones para destruir VMs. Si alguna de ellas se cumple se destruirán tantas como sea necesario.
9) Los *jobs* a ejecutar son enviados sobre las instancias disponibles.
10) Este proceso se repite para cada *job*.
11) Cuando el usuario indica que el recurso deja de ser valido se destruyen todas las VMs existentes.

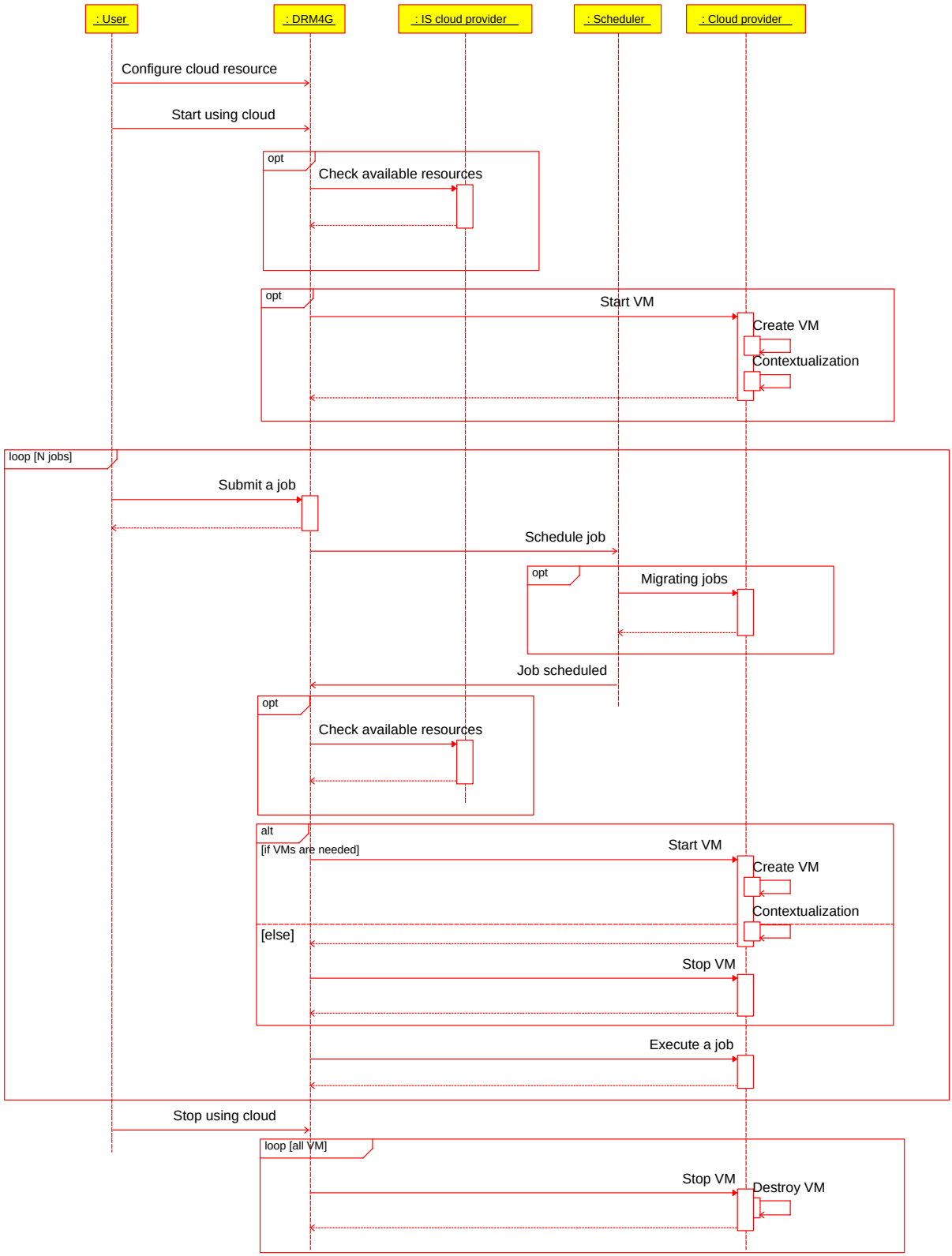

Figura 3.14: Diagrama de secuencia de ejecución de jobs en entornos Cloud en DRM4G.

3.4.2 Prioridad de los jobs

Una situación que se contempla habitualmente en el envió de *jobs* es que estos dispongan de una determinada prioridad fija inicial (que suele ser por defecto 0), la cual es tenida en cuenta para el cálculo de la prioridad de *jobs* por parte del planificador. Esta junto a otras, como por ejemplo el tiempo de espera para la asignación de un recurso, son empleadas por el *scheduler* para el cálculo de la prioridad de un *job*. En GridWay, una vez que los *jobs* están en el sistema no se contempla la posibilidad de que la prioridad de un *job* pueda ser modificada por el usuario. Aunque existen determinadas situaciones en las que esta característica podría ser de gran utilidad.

Así por ejemplo, en el caso de envíos de cientos o miles de *jobs*, cuando estos se encuentran esperando para la asignación de un recurso; el usuario puede querer modificar la prioridad de un *job* o un grupo de *jobs* para que estos se ejecuten antes sin alterar o eliminar el resto de *jobs.* Para satisfacer esta posible necesidad, DRM4G, al igual que Condor [293], permiten modificar de manera dinámica la prioridad de un *job* una vez que este se encuentra en el sistema.

3.4.3 Dependencias entre jobs

A menudo, la ejecución de un experimento está compuesto de múltiples ejecuciones cortas o largas que deben procesarse en secuencia. Un método para resolver esta problemática es crear una secuencia de *jobs* en la que se establezcan dependencias entre los *jobs*. Esta característica implementada por cada *framework* analizado presenta una carencia en GridWay a la hora de su planificación.

Aunque GridWay contempla internamente que los *jobs* pueden ser dependientes entre sí. Durante el cálculo de planificación de tareas, este no toma en consideración si existe una dependida de algún tipo entre los *jobs*. En otras palabras, si existe una dependencia entre *jobs*, el estado del *job* que ha finalizado no es tenido en cuenta para la planificación de los *jobs* que dependen de él. Por consiguiente, en DRM4G se han implementado tres tipos de dependencias, basándose en los modelos LRMS, para que el usuario defina la dependencia entre *jobs* a nivel del *scheduler*:

- ***afterok***: El *job* sólo se planificará si el *job* o los *jobs* de los que dependen ha completado satisfactoriamente.

- ***afternotok***: El *job* se ejecutá si el *job* o los *jobs* de los que dependen han terminado con errores.
- ***afterany***: El *job* será planificado si el *job* o los *jobs* ha finalizado independientemente si lo ha hecho con o sin errores.

La figura 3.15 muestra un ejemplo del uso de las dependencias entre *jobs* citadas anteriormente. Como se observa en la figura 3.15, el *job* con ID 1 sólo se ejecutará si el *job* con ID 0 termina de manera satisfactoria, y el *job* con ID 2 se ejecutará en cualquier caso una vez terminado el *job* 0.

```
[user@local~]$ drm4g job submit test.job
JOB ID: 0

[user@local~]$ drm4g job submit –-dep afterok:0 test.job
JOB ID: 1

[user@local~]$ drm4g job submit –-dep afterany:0 test.job
JOB ID: 2
```

Figura 3.15: Envío de tres jobs con DRM4G. El primer job sin ningún tipo dependencia, el segundo dependiente del primero con una dependencia de tipo afterok y el tercero dependiente primero con una dependencia de tipo afterany.

3.4.4 Meta-planificación multinivel

En [294] [295] se presenta como configurar el meta-planificación GridWay mediante diferentes topologías de planificación (sección 2.5.4) en el acceso a recursos *Grid*. En concreto, [295] resalta los beneficios de la meta-planificación multinivel como son: el control descentralizado, la *interoperabilidad*, la transparencia en el acceso, la *escalabilidad*, la reconfigurabilidad y la seguridad. Así, el trabajo en paralelo de múltiples planificadores distribuidos, en la misma o en diferentes infraestructuras, facilita una mayor autonomía mejorando la difusión de *jobs*. Para esta técnica, se emplea *middlewares* como Globus y aplicaciones como GridGateWay [295].

Al igual que GridWay, DRM4G puede ser utilizado en un modelo de meta-planificación multinivel, pero en este caso sin la necesidad de aplicaciones o *middleware* intermedios. DRM4G dispone de un RM concreto, denominado *drm4g*, que permite inter-conectar todos los meta-planificadores que un usuario tenga desplegados a lo largo de sus infraestructuras creando

una topología híbrida (ser sección 2.5.4). El resultado, es una infraestructura de meta-planificación en la que todos los meta-planificadores pueden ser accedidos por el usuario y por el meta-planificador o meta-planificadores que se encuentren en un nivel superior.

Esta meta-planificación a su vez puede ir un paso adelante y configurarse de manera dinámica si se despliega en entornos *Cloud*. DRM4G puede instalarse y configurase en el proceso de *contextualización* de las VMs, mediante la herramienta *pip,* por ejemplo. En consecuencia, cada instancia dispondrá de su propio planificador que gestionará de manera local los *jobs* a ejecutar en la instancia, llegando generando un árbol dinámico de meta-planificadores.

3.5 Conclusiones

En este capítulo se ha presentado el *framework* DRM4G como extensión del meta-planificador GridWay para el acceso distribuido a recursos HPC, *Grid* y *Cloud*. Su diseño, características, así como funcionamiento interno también han sido detallados. Además, DRM4G ha sido comparado con una selección de los *frameworks* actuales utilizados actualmente en producción como: Condor, DIRAC, gUSE/WS-PGRADE y el propio GridWay.

Por lo mostrado en el capítulo, DRM4G ha demostrado:

- Un diseño centrado en las necesidades de los usuarios.
- *Interoperabilidad* entre diversas infraestructuras y paradigmas computaciones.
- Diseño escalable y robusto.
- Disponer de un acceso concurrente y *multiplexado* a los recursos de manera independiente a su tipo.
- Adaptabilidad, facilitando la personalización de plantillas para el envío de *jobs*.
- Uso los dos tipos de instancias *Cloud*: *on-reserved* y *on-demand*.
- Gestión dinámica del número y tipo de VMs.
- Mejora del planificador actual de GridWay.
- Facilidad de instalación.
- CLI para la gestión de identidades en las infraestructuras.

- Acceso distribuido a *datasets*.
- Acceso a *e-Infraestructuras* federadas como: EGI, PRACE y RES; y no federadas: AWS, *clusters* HPC, PCs, etc.

Como contrapartida, DRM4G presenta limitaciones en la gestión de instancias en los recursos *Cloud*, no pudiendo utilizar instancias *on-spot* debido al uso de la librería Apache Libcloud. Debe recordarse que estas instancias tienen una gran demanda en la actualidad por la capacidad que disponen para reducir costes en la ejecución de *jobs* [296].

Una vez descrito y analizado el *framework* en forma teórica, el siguiente capítulo la Tesis se centrará en analizar de forma práctica, mediante la ejecución de dos experimentos, tres características fundamentales de un meta-planificador como son: la *escalabilidad*, la robustez y la *interoperabilidad*.

Capítulo 4. Experimentos de validación

4.1 Introducción

En este capítulo se mostrará dos experimentos diseñados para demostrar las capacidades de *escalabilidad*, robustez e *interoperabilidad* de DRM4G. En el primer experimento, someteremos a DRM4G a una situación de estrés en la que se enviarán de manera consecutiva dos tareas compuestas de 10K y 100K *jobs* para su ejecución. De esta forma, se comparará el diseño actual de DRM4G con el propuesto por GridWay. Por otro lado, el segundo experimento se centrará en el análisis de imágenes de galaxias aplicando *deep neural learning* [297] [298] para su catalogación. En esta simulación se emplearán 20K *jobs* que de manera concurrente serán distribuidos entre múltiples infraestructuras, las cuales pertenecerán a distintos paradigmas de computación. Se pretende mostrar cómo un usuario puede aprovechar diferentes tipos de entornos de computación para reducir el tiempo total de ejecución de una simulación.

4.2 Escalabilidad y robustez

Para medir la *escalabilidad* y robustez de DRM4G, realizaremos dos experimentos que intenten saturar todos los componentes de DRM4G mediante el envío de *jobs* de manera consecutiva. En el primer experimento en 10K *jobs*, y en el segundo 100K *jobs*. El objetivo de ambos experimentos es también comparar los resultados que se obtengan con los obtenidos por GridWay en [53], donde el meta-planificador era sometido al envío de 10K *jobs* de manera consecutiva. Por consiguiente, se ha tratado de reproducir el entorno de ejecución tanto en el número de recursos disponibles como en la maquina donde se ha desplegado el meta-planificador. El número de recursos tampoco se ha aumentado a pesar de aumentar el número de *jobs* en el segundo de los experimentos, pues se pretende observar también la ejecución y la planificación de *jobs* durante un periodo de tiempo más extenso.

4.2.1 Entorno de pruebas

Los recursos utilizados en este experimento se encuentran dentro de una red local y están compuestos por cuatro *workstations* y un PC con un procesador Intel(R) Pentium(R) D CPU 3.00 GHz con 2 GB de RAM; todos ellos con un sistema operativo CentOS 6. DRM4G será desplegado en el PC, mientras que las cuatro *workstations* se configurarán para la ejecución de los *jobs*. En concreto, estos recursos se accederán a través de SSH utilizando *fork* como LRMS de ejecución y, cada uno tendrá la capacidad de ejecutar 200 *jobs* de manera concurrente.

Por otro lado, el planificador se configurará con los siguientes parámetros:

- En cada iteración, el planificador planificará un máximo de 60 *jobs*.
- Cada iteración del planificador sucederá con un intervalo de 180 segundos.
- No habrá límite en la cantidad de *jobs* por usuario que se puedan ejecutar en al mismo tiempo.

Con respecto a los *jobs*, estos serán simples *sleep* con tiempos de ejecución entre 1800 y 5400 segundos. En concreto, estos tiempos variarán de acuerdo a la función representada en la figura 4.1. Estos *jobs* aunque no hacen uso de CPU sobre los recursos, permiten modelar miles de *jobs* sintéticos de diferentes tiempos de ejecución para evaluar la saturación del meta-planificador.

4.2.1 Entorno de pruebas

$$f(i) = 1800 + (20i) \% (5400 - 1800)$$

Donde i es el índice de cada *job*, que va de 0 a 9999 para el primer experimento y de 0 a 99999 para el segundo experimento.

Figura 4.1: Expresión para el cálculo de ejecución de cada job.

4.2.1.1 Resultados

El primer resultado a destacar es que DRM4G es capaz de gestionar el envío secuencial de 100K *jobs* sin ningún problema. Además, el tiempo medio necesario para el primer experimento fue de 75 segundos y de 962 segundos (~16 minutos) para el segundo. Esto contrasta con los 226 segundos que tarda de media GridWay en enviar 10000 *jobs* [53].

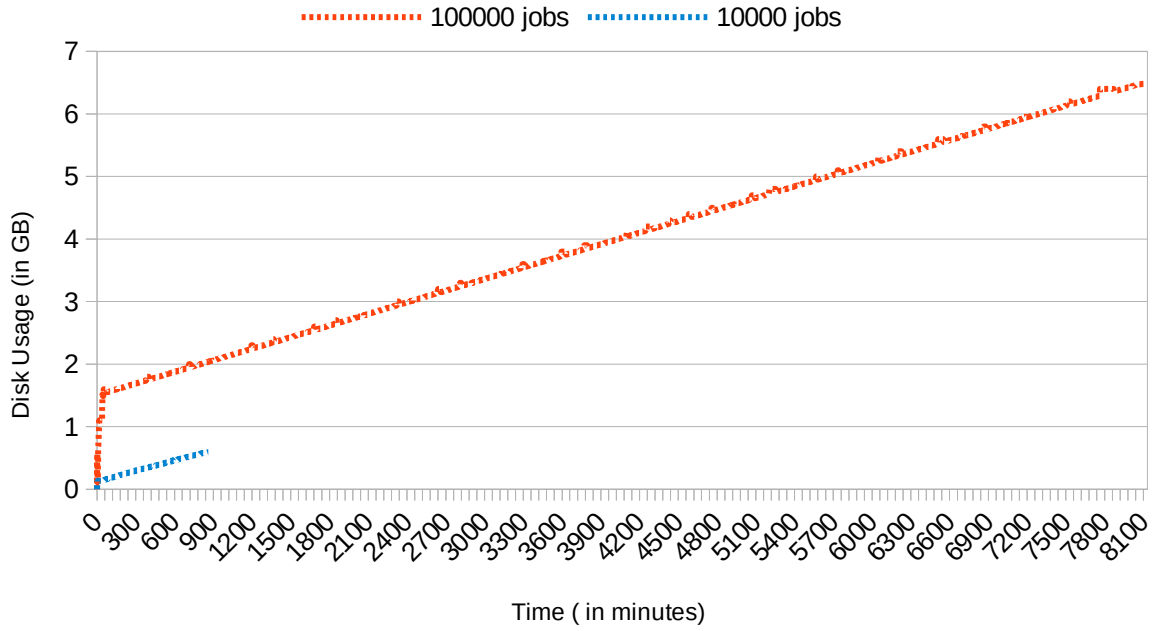

Figura 4.2: Espacio de almacenamiento empleado por los jobs.

La figura 4.2 muestra el consumo de disco asociado a los *jobs* para ambos experimentos. La linealidad que se observa en ambos casos es debida a que ningún *job* durante la ejecución falló y por lo tanto la secuencia de finalización de *jobs* fue prácticamente lineal. En la figura 4.2 se observa cómo una vez finalizados los todos ellos el total acumulado es de 6.5 GB para el primer experimento y de 616 MB para el segundo, lo que supone un promedio de 68 KB por *job*. Por consiguiente, un usuario puede determinar *a priori* el tamaño mínimo necesario para un conjunto

de *jobs*. Este tamaño medio incluye los ficheros de *log* asociados a cada *job*, así como todos los ficheros necesarios para su ejecución.

La figura 4.3 por su parte, muestra el porcentaje de consumo de memoria RAM para todos los componentes de DRM4G. Como se puede apreciar, valor máximo consumido por todos los componentes para el primer envío es como máximo entorno a un 15% (300 MB), un 5% menos que el consumo de memoria total mostrado por GridWay en [53], a pesar incluso que el número de *jobs* es 10 veces superior. Mientras que para el envío del mismo número de *jobs* este porcentaje puede aumentar hasta un 14%. Respecto a las oscilaciones o dientes de sierra mostrados, estas se deben a que el consumo de memoria de los MADs en DRM4G varía en función de las necesidades demandas y, por lo tanto, pueden aumentar o disminuir.

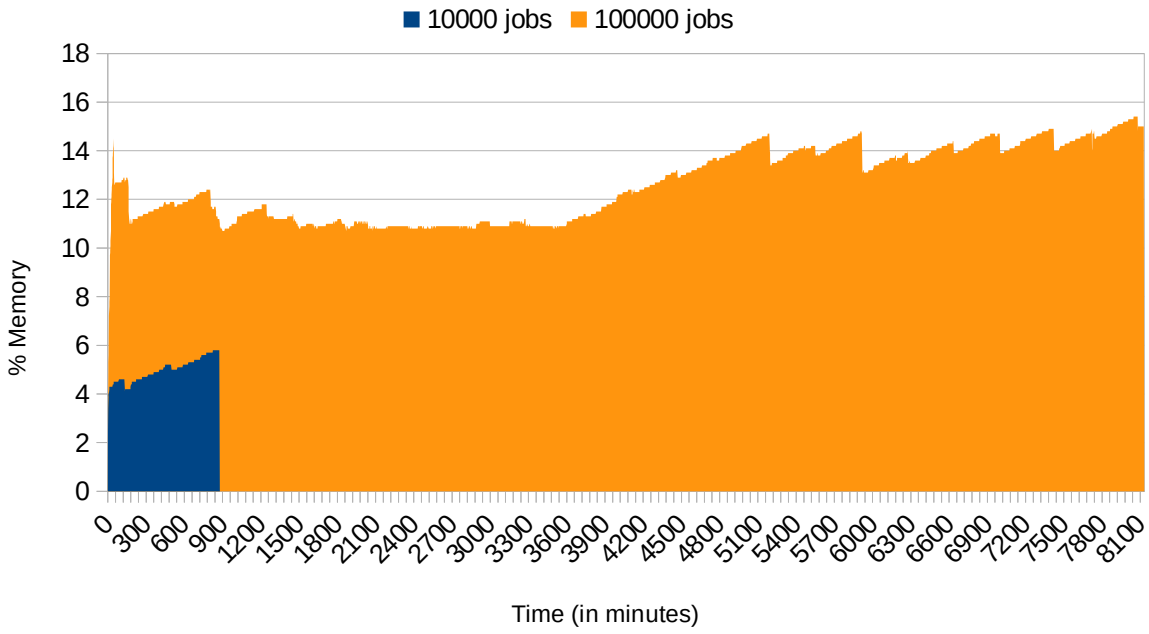

Figura 4.3: Consumo total de memoria de los componentes de DRM4G.

4.2.1.1 Resultados

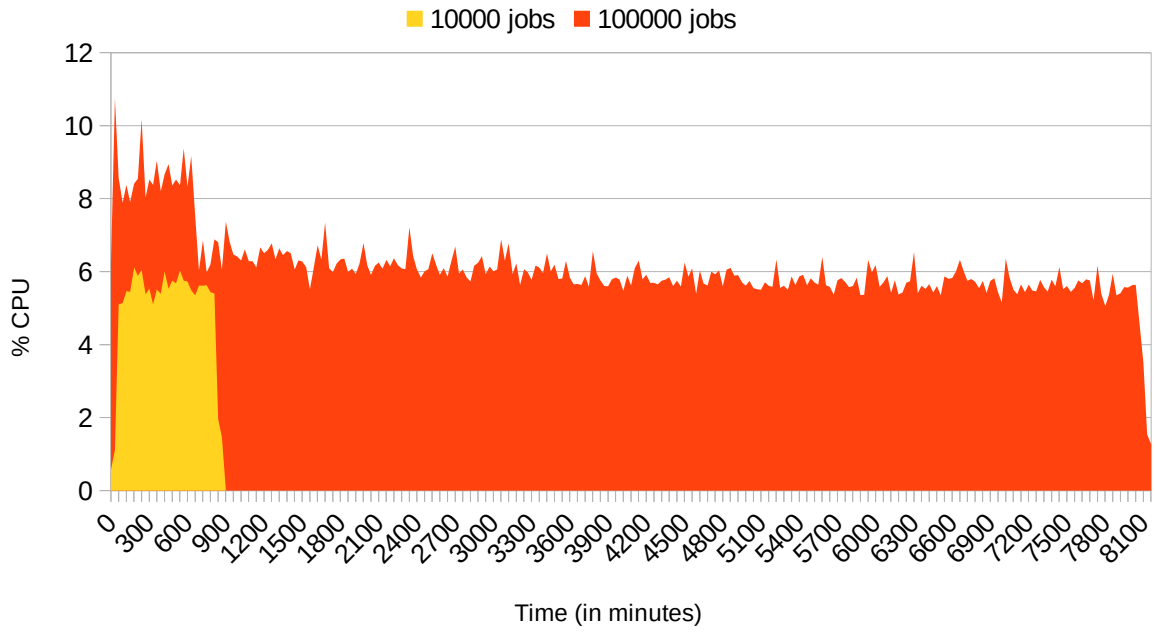

Figura 4.4: Consumo total de CPU de los componentes de DRM4G.

La figura 4.4 muestra el consumo de la CPU de DRM4G a lo largo de ambos experimentos. En el caso del primer experimento existe un consumo medio de CPU y continuo entorno al 5 %. En el segundo en cambio, se muestra un pico de consumo inicial durante la asimilación por parte de DRM4G de los *jobs*, para a continuación mantenerse estable entorno 8%, y posteriormente reducirse entorno al 6% de consumo hasta el final. Nuevamente, el consumo total de DRM4G en ambos casos es inferior al de GridWay en [53]. Estas diferencias radican en los MADs diseñados en DRM4G muestran consumos tanto en memoria como en CPU sensiblemente inferiores a los ofrecidos por los MADs de GridWay, aún teniendo en cuenta la diferencia de *jobs* utilizados. Un dato más a destacar, es el reducido consumo de CPU empleado por DRM4G a pesar de que el procesador del PC donde se desplegó es un modelo obsoleto en la actualidad.

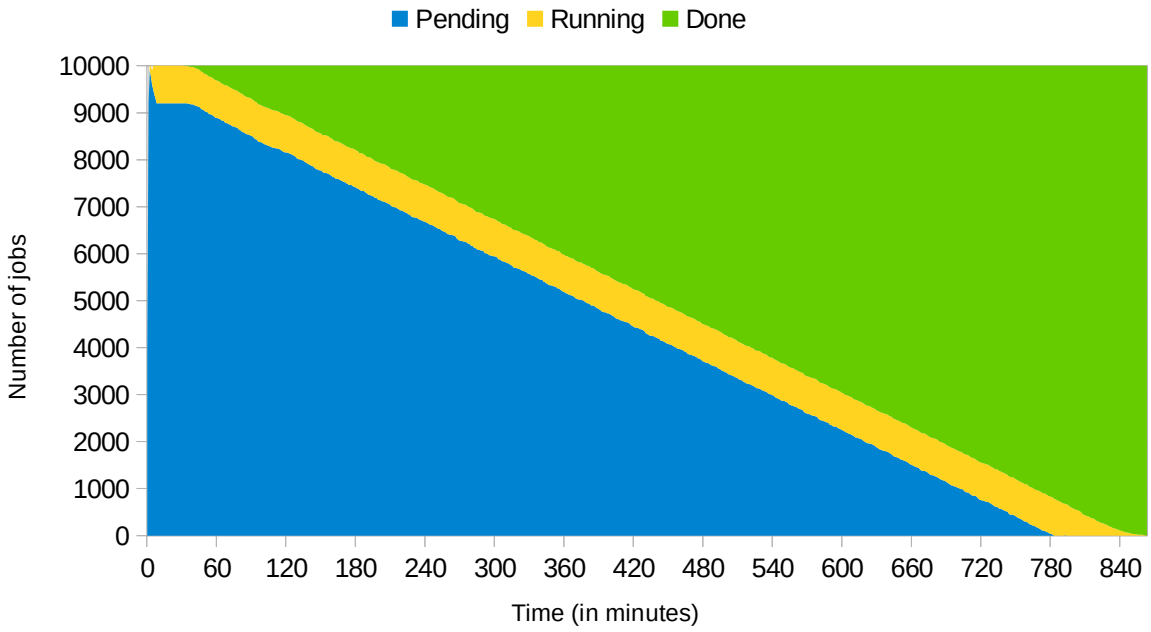

Figura 4.5: Estado de los jobs en el tiempo, cuando el número de jobs enviados es 10K.

Las figuras 4.5 y 4.6 muestran los estados de los *jobs* distribuidos a lo largo de los dos experimentos. En el primero el tiempo empleado por DRM4G para la finalización de todos los *jobs* fue de 782 minutos, manteniéndose casi constante la ejecución concurrente de 800 *jobs*. Un resultado similar al obtenido por GridWay [53] (800 minutos) aunque con un ligera mejora. En el segundo, los *jobs* fueron completados correctamente en 8126 minutos (más de 5 días y medio), manteniéndose constante la ejecución *jobs* como en el primer experimento. En ella además se puede observar cómo la ejecución de los primeros 10K *jobs* se ha completado entorno al minuto 856, cuando en el primer experimento se necesitaron 782 minutos. Por consiguiente, el aumento del envió de *jobs* ha producido que sea necesario más tiempo para ejecución del mismos número de *jobs*. Resultado lógico, si se considera que ya inicialmente el tiempo de envío de estos dos experimentos es diferente.

4.2.1.1 Resultados

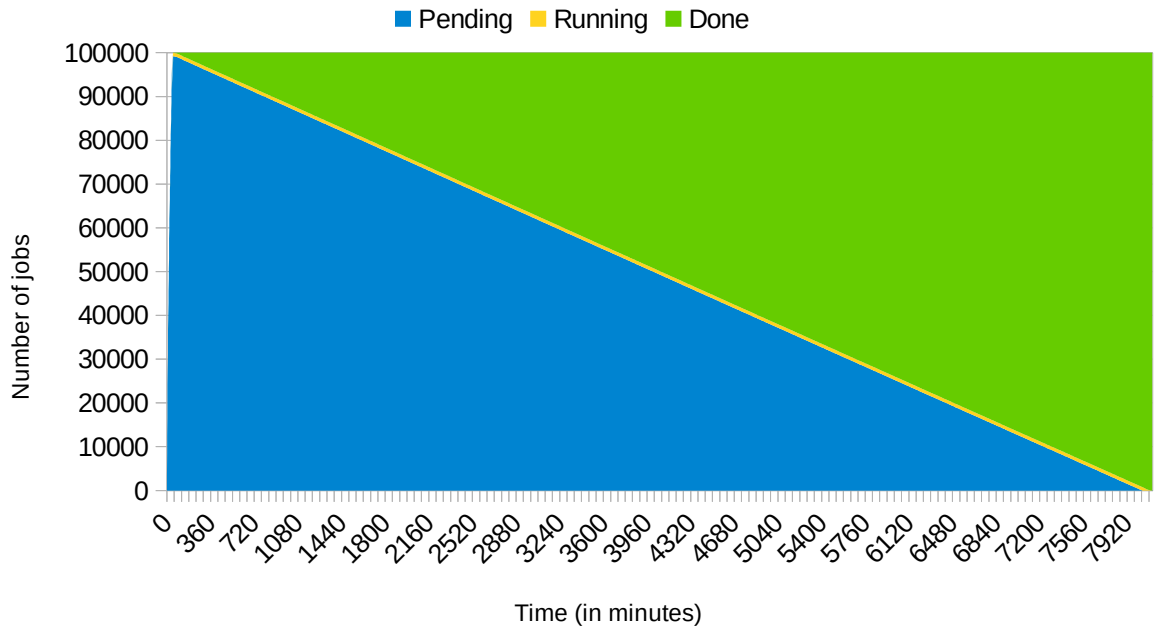

Figura 4.6: Estado de los jobs en el tiempo, cuando el número de jobs enviados en 100K.

Finalmente, cabe destacar que aunque en el presente experimento, sólo se ha analizado el envío máximo de 100K *jobs* de manera concurrente debido a la disponibilidad de recursos. DRM4G es capaz de gestionar el envío de tareas compuestas por un millón de *jobs*.

4.3 Interoperabilidad

En esta sección se prestará especial atención a la capacidad que interconexión entre infraestructuras computacionales para la ejecución de tareas de manera concurrente. El objetivo es mostrar cómo DRM4G es capaz de *interoperar* con diferentes tipos de infraestructuras que pertenecen a distintos tipos de paradigmas y, cómo a su vez es capaz también de aplicar una planificación avanzada sobre los *jobs* a ejecutar en un entrono distribuido.

Para el experimento a realizar en este *testbed* se emplearán imágenes individuales de 40K galaxias provenientes del repositorio *Cosmic Assembly Near-infrared Deep Extragalactic Legacy Survey* (CANDELS) [299]. Estas imágenes serán utilizadas para entrenar y validar un algoritmo de *deep neural learning* construido para analizar automáticamente las morfologías de galaxias de grandes conjuntos de galaxias y de estrellas a través de imágenes digitales. Las imágenes empleadas tienen un tamaño de 128 por 128 pixeles, una resolución de 0.06 pulgadas y

un tamaño de 133 KB cada una. Los detalles científicos del experimento ha realizar se encuentran descritos en [300].

Para tratar estos cientos de miles de galaxias de forma concurrente y reducir el tiempo total de ejecución del experimento global la carga de trabajo ha sido dividida en 20K *jobs.* Los cuales se emplearán paralelamente de forma independiente en diferentes e-*Infraestructuras* como EGI HTC o EGI FedCloud y en los recursos institucionales propios como se describe en la sección siguiente.

4.3.1 Entorno de pruebas

Para el experimento se configurarán diferentes tipos de recursos que traten de cubrir la casuística de recursos disponibles para un investigador actual. Por consiguiente, se han configurado: un *cluster* HPC que dispone de un sistema de colas PBS, las VOs *Grid iber.vo.ibergrid.eu* (IBERGRID [301]) y *prod.vo.eu-eela.eu* (GISELA) pertenecientes a la infraestructura EGI, y la VO de test *fedcloud.egi.eu* de la infraestructura EGI FedCloud. Por su parte, el meta-planificador DRM4G será desplegado en un PC con un i5-4300U (4 cores, 1.90GHz) y 4 GB de RAM con accesos a las citadas infraestructuras a través de SSH de forma similar a como refleja la figura 3.5. DRM4G accederá a los recursos *Grid* mediante el uso de un GUI debidamente configurado para ambas VOs. Debe resaltarse que todas las infraestructuras son compartirán con otros investigadores y no se realizá ningún tipo de reserva previa.

En el caso de la arquitectura FedCloud su configuración requiere además la elección de una imagen y un *flavour* para las VMs. Para asegurarnos utilizar el mayor número de *sites* con la misma imagen, se ha optado por el uso la imagen Centos 6 [302] que es la que mayor disponibilidad tiene, configurada en 17 de los 23 proveedores de recursos. Respecto al *flavour*, se ha optado por instancias con 2 *cores* y 4 GB de RAM, definidas normalmente bajo la etiqueta de *medium*, dadas las necesidades del experimento que demanda al menos de 2 GB de RAM por *job* y, por lo tanto, *flavours* de tipo *small* (1 *core* y 1 Gb de RAM) no eran susceptibles de uso.

En el experimento se empleará la versión 1.4 de GalSim [303], un software de código abierto diseñado para simular imágenes de galaxias de alta resolución, donde la mayor parte de los cálculos se realizan en C++, y que cuenta con un interfaz de usuario en Python. Además también serán necesarios paquetes de Python como *numpy* [304] o *astropy* [305] para la ejecución de las

simulaciones. Como este *software* no se encuentra disponible en ninguna de las infraestructuras indicadas, este deberá ser instalado previo paso a cada ejecución.

En la infraestructura local este puede ser instalado de manera temporal o permanente hasta la consecución del experimento, pero en las infraestructuras como las G*rid* será necesario la instalación del *software* en cada ejecución; situación que penalizará las ejecuciones en estas infraestructuras. Mientras en entornos *Cloud* la ventaja de la reutilización de VMs hará que sólo sea necesaria una instalación por cada VM *instanciada*. De forma que los tiempos de ejecución media esperados en la plataforma *Grid* serán mayores.

Por consiguiente, para los entornos *Grid* se ha optado por crear un empaquetado basado en Anaconda [306] que contenga todo el *software* necesario y que será almacenado en el repositorio LFC de cada VO junto a los datos de entrada (1.05 GB en total). En cambio en FedCloud dado que el usuario es a su vez administrador de las VMs, se ha optado por instalar el *software* en el tiempo de arranque de cada instancia a través de *cloud-init*. Mientras que los datos de entrada en este caso estarán disponibles en el contenedor OpenStack Swift del BIFI (333 MB).

En relación a los datos de salida, como su tamaño es inferior a los 100 KB por *job*, estos se adjuntarán las salidas estándar de cada *job* y serán almacenados en el propio PC donde se encuentra desplegado DRM4G.

4.3.2 Resultados

Como primera observación puede extraerse que DRM4G es capaz de *interoperar* entre diferentes infraestructuras de forma simultánea para el cálculo distribuido. Así, la figura 4.7 muestra la distribución de *jobs* entre las infraestructuras configuradas para el experimento. En ella se observa cómo más de un 60% de los *jobs* han sido ejecutados en la infraestructura FedCloud a pesar que inicialmente esta infraestructura sólo contaba con 20 *slots* disponibles, configurados mediante 10 VMs. Y sin embargo los recursos de las infraestructuras *Grid* que *a priori* eran mayores (~13000 *slots*) no llegan al 32% de los *jobs* ejecutados. Esta situación es en parte debida a que los recursos *Grid* y HPC no han sido capaces de asimilar la carga de trabajo del experimento (20K *jobs*) y, como consecuencia los *jobs* en estado *pending* han propiciado de forma automática la *instanciación* de nuevas VMs en la federación *cloud* para satisfacer las necesidades computacionales. Debe resaltarse que los CEs de las VOs configuradas son

compartidos con otras VOs e incluso entre ellas mismas y, por consiguiente, el número de *slots* ofertados pueden resultar a veces engañosos.

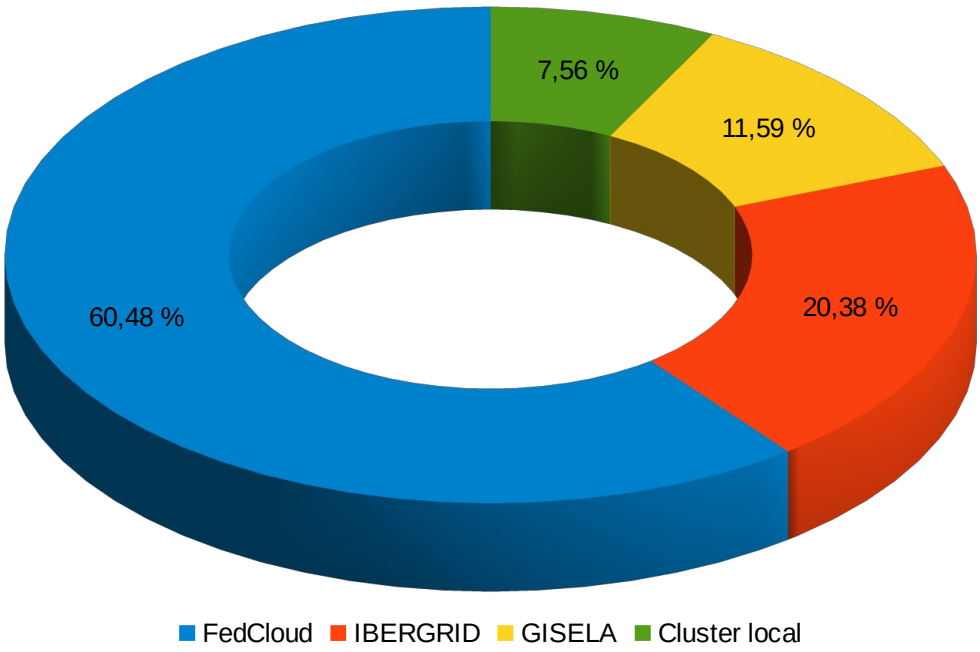

Figura 4.7: Distribución de jobs entre las infraestructuras.

La figura 4.7 además muestra un reflejo de la computación actual, en la que muchos *sites* que inicialmente publicaban recursos *Grid*, como por ejemplo los casos de BIFI o CETA-CIEMAT [307], han ido migrado total o parcialmente sus recursos computacionales a recursos de tipo *cloud*. Concretamente, el número de *sites Grid* disponibles en las VOs utilizadas ha descendido en los últimos años, así como el número de *slots* computaciones ofrecidos por ellas. Otro fenómeno que se observa, a la vista de los resultados, es la mejora de la fiabilidad de los proveedores de servicios de la infraestructura EGI FedCloud, que contrasta con estudios realizados en años anteriores [308]. Aunque aún persisten limitaciones detectadas durante la ejecución del experimento. Así, por ejemplo, ninguna imagen del repositorio EGI AppDB está disponible a la vez en todos los proveedores, la asignación de IPs públicas a cada VM no sigue ningún estándar y, por lo tanto, dependen de la configuración de cada proveedor, y además existe un número restringido de direcciones IP públicas en algunos *sites* muy inferior al número de recursos disponibles. Estas dificultades, unidas a los largos tiempos de espera necesarios para la activación de VMs de determinados *flavours* como *Large* o *XLarge*, siguen dificultando el uso de la infraestructura por parte de los usuarios.

4.3.2 Resultados

Finalmente, debe tenerse en cuenta que las infraestructuras utilizadas son compartidas con cientos de usurarios, y dada la heterogeneidad de las infraestructuras *Grid* y *Cloud* hace que la distribución mostrada en la figura 4.7 pueda variar dependiendo de las cargas de trabajo disponibles en cada momento en las infraestructuras.

A continuación, la tabla 4.1 muestra el número medio de *jobs* ejecutados por cada infraestructura por hora. Como era evidente, la mayor tasa es conseguida en EGI FedCloud, debido a la mayor disponibilidad de recursos. Si bien, esta infraestructura no dispone *a priori* de un nivel global de *slots* de cálculo mayor que el resto de infraestructuras configuradas, una vez que los recursos o VMs son *instanciados* estos dejan de ser compartidos con el resto de usuarios de la infraestructura. De esta forma, la ejecución de *jobs* es directa, sin la necesidad de espera en una cola como en los casos de HPC o *Grid*. Pero esta situación también puede repercutir negativamente en los propios usuarios, si por ejemplo los recursos adquiridos una vez que ya no son necesarios no son de nuevo liberados.

Infraestructura	Productividad (*jobs*/hora)
FedCloud	129,23
IBERGRID	43,53
GISELA	24,75
Cluster local	16,15

Tabla 4.1: Productividad por recurso.

La figura 4.8 muestra los tiempos medios de la ejecución de cada *job* en cada infraestructura, donde la infraestructura HPC ofrece un mejor rendimiento en la ejecución especialmente si se compara con FedCloud. Se trata un resultado esperado debido a los *overheads* introducidos por los los sistemas de *virtualización* que reducen el rendimiento de las aplicaciones [309]. En las infraestructuras IBERGRID y GISELA destacan los tiempos necesarios para la transferencia de ficheros debido a la necesidad de transferir el empaquetado con el *software* utilizado para la ejecución de los *jobs,* que supone un extra 744MB por cada *job*. En cambio, en la infraestructura FedCloud aunque también es necesario la configuración del *software* especifico (este se realiza en el tiempo de arranque de las VMs a través de *cloud-init*) y la transferencia los ficheros de entrada, estas operaciones sólo son necesarias una vez por cada VM *instanciada*. Así, sucesivas

ejecuciones de *jobs* no las necesitan realizar. En estos casos también hay que sumar el tiempo de *setup* de las VMs, el cual oscila dependiendo de factores como el número de VMs requeridas o el proveedor dentro de la infraestructura FedCloud. Mientras en el *cluster*, el *software* ya esta configurado y los ficheros de entrada no necesitan ser transferidos, de ahí que la ejecución no contenga ningún *overhead* añadido.

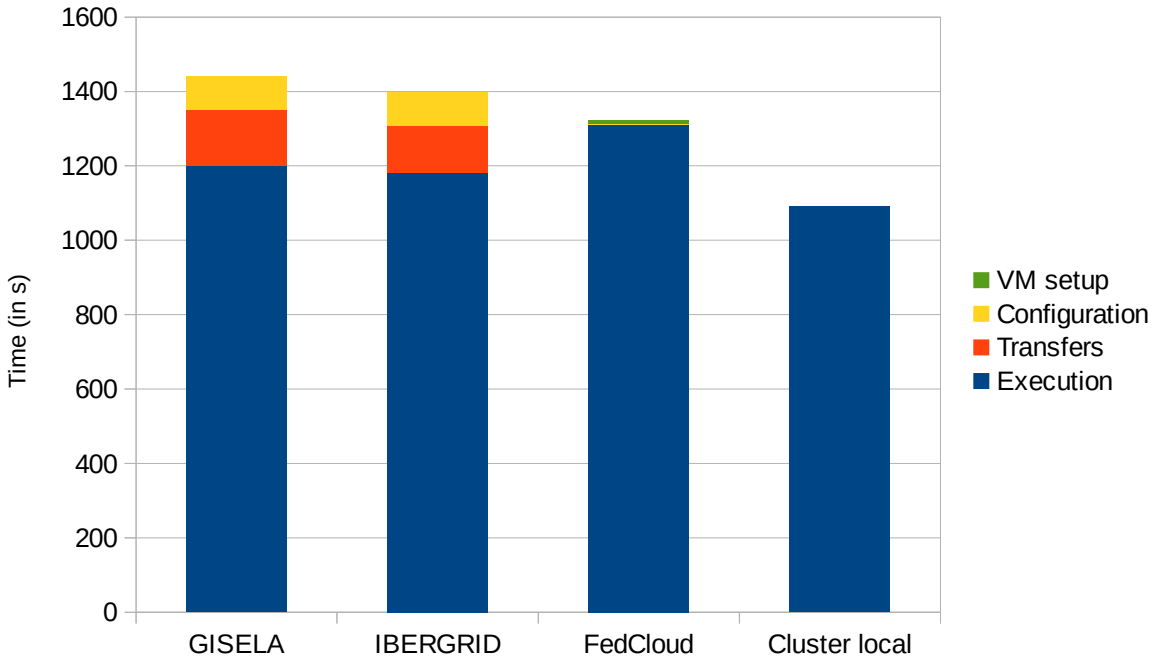

Figura 4.8: Desglose de los tiempos medios en la ejecución de jobs entre las infraestructuras computacionales.

Para concluir, el experimento ha tenido un *makespan* de 3 días 21 horas 35 minutos y 58 minutos lo que supone una mejora 64.75 veces más rápido la ejecución en un entorno distribuido usando DRM4G que si su ejecución se hubiese realizado de forma secuencial en un sólo nodo de cálculo en el *Cluster* local.

4.4 Conclusiones

En este capítulo se ha analizado la *escalabilidad* y robustez del *framework* DRM4G sometiéndole al envío y ejecución de tareas compuestas por 10K y 100K *jobs*. Además, se ha realizado una comparativa de los resultados obtenidos con los proporcionado por GridWay en [53], apreciándose cómo DRM4G tiene un menor consumo tanto en memoria como en CPU en condiciones similares.

Por otro lado, se ha analizado la *interoperabilidad* de DRM4G para le ejecución concurrente de *jobs* en infraestructuras pertenecientes a distintos tipos de paradigmas, proveedores e instituciones. Los resultados han mostrado la utilidad de la ejecución distribuida utilizando DRM4G para reducir el *makespan* de experimentos. Este experimento también ha servicio para analizar el estado actual de infraestructuras como: IBERGRID, GISELA o FedCloud.

Una vez analizado DRM4G de forma teórica y práctica, debe indicarse que las características de DRM4G para la ejecución de *jobs* pueden ser extendidas al uso de aplicaciones para la explotación de infraestructuras. En este sentido, el Capítulo 5. muestra cómo DRM4G es empleado por parte de aplicaciones como WRF4G [218], un *framework* de ejecución de simulaciones climáticas, para ejecutar sobre recursos de tipo HPC, *Grid* y *Cloud*. Además, en ese capítulo también se mostrará cómo DRM4G es capaz de desplegar infraestructuras complejas en entornos *Cloud* para el uso de aplicaciones como Hadoop [310] y Hive [32].

Capítulo 5. Aplicaciones

5.1 Introducción

En el Capítulo 4. se ha presentado DRM4G como una herramienta flexible para gestionar la meta-planificación en un entorno de inter-infraestructuras. Así mismo, se ha mostrado cómo el uso de la meta-planificación es importante para la ejecución de *jobs* a gran escala en sistemas distribuidos. En este capítulo avanzaremos en cómo complejas aplicaciones de diferentes áreas de investigación pueden beneficiarse del uso de DRM4G para eliminar las barreras que las infraestructuras imponen. Ya sea a través de su API estándar para la ejecución de *jobs*, o mediante el despliegue de aplicaciones de *Big Data* que DRM4G es capaz de desplegar de manera dinámica.

El objetivo de este capítulo es, por lo tanto, mostrar la idoneidad del uso de DRM4G para reducir la complejidad del uso concurrente y distribuidos de infraestructuras HPC, *Grid*, *Cloud* por parte de aplicaciones. Así, en este capítulo se muestran dos ejemplos de aplicaciones que utilizan DRM4G para el acceso a infraestructuras. La primera de ellas, hace referencia a la explotación de recursos distribuidos para realizar simulaciones climáticas; y la segunda, aplica la

escalabilidad y la elasticidad que las infraestructuras *Cloud* proveen para desplegar *clusters* Hadoop, para el análisis de datos.

5.2 Simulaciones climáticas: WRF4G

WRF4G es una herramienta que simplifica la ejecución de experimentos de simulación numérica atmosférica con el modelo WRF en infraestructuras distribuidas [155]. WRF es un modelo de área limitada, ampliamente utilizado debido a su flexibilidad y *modularidad*. De hecho, ha sido utilizado en una gran variedad de áreas de investigación y aplicación, desde la predicción del tiempo hasta la proyecciones de cambio climático.

WRF4G permite la ejecución de experimentos a gran escala combinando de manera eficientemente recursos heterogéneos y distribuidos, proporcionando un control total de las simulaciones y medios para reiniciar parte o todo el experimento en caso de fallo. También proporciona la capacidad de reproducir experimentos de manera total o parcial. Características muy útiles si el experimento está compuesto por miles de *jobs*.

5.2.1 Uso de infraestructuras en simulaciones climáticas

La complejidad de los modelos climáticos plantea desafíos no sólo para los meta-planificadores de los recursos, sino también para el *software* estándar utilizado en las infraestructuras computacionales [311]. La ejecución de los modelos climáticos suele implicar la gestión de flujos de trabajo complejos que producen grandes volúmenes de datos y, en ocasiones, largos períodos de duración de semanas e incluso meses. Además, las nuevas tendencias en el modelado climático que emplean predicciones de tipo *ensemble* [312] para muestrear las incertidumbres inherentes a las simulaciones, requieren ejecutar varias veces la misma simulación con parámetros variables y, por lo tanto, complica aún más la gestión del experimento.

El crecimiento en el número de simulaciones independientes necesarias para realizar la predicción *ensemble* ha obligado a la comunidad a encontrar nuevas fuentes de computo, no solamente las basadas en recursos HPC, ampliamente utilizadas por las VRCs del clima [313] [314]. La naturaleza independiente de estas simulaciones las hace adecuadas para ser ejecutadas

de manera concurrente en diferentes infraestructuras. La adaptación de estos modelos para aprovechar las infraestructuras distribuidas puede aumentar enormemente la potencia de cálculo accesible para los investigadores.

Así, con el fin de simplificar la ejecución de los experimentos climáticos y meteorológicos, algunas instituciones han creado sus propios *frameworks* que permiten a los usuarios realizar fácilmente experimentos en infraestructuras computacionales específicas. Estos *frameworks* proporcionan un conjunto de comandos y servicios que ocultan la complejidad de configurar, ejecutar y supervisar todas las simulaciones involucradas en un experimento. Aunque a diferencia de WRF4G no proveen de un acceso distribuido a infraestructuras heterogéneas para ejecutar concurrente las simulaciones, limitándose a recursos computacionales concretos.

5.2.2 Requerimientos y retos

Realizar un experimento climático requiere a menudo ejecutar varias veces la misma simulación con parámetros variables o permutaciones. Estos parámetros pueden ser diferentes fechas, datos de entrada, configuraciones de los modelos climáticos, etc. Por lo general, los requisitos de computación de una simulación dada no siempre se ajustan a las capacidades de los recursos. En algunos casos, los datos de entrada exceden la capacidad de disco de recursos de cálculo y, en otros, la simulación es más larga que el límite de *walltime* que los LRMS permiten. Para evitar estas restricciones, es necesario dividir las simulaciones en trozos más pequeños, denominados *chunks*, creando *checkpoints*.

Cada experimento comprende un conjunto de *realizaciones* independientes que se pueden dividir en varios *chunks* ejecutados como simulaciones dependientes (un *chunk* no comienza hasta que el *chunk* anterior haya terminado). Por lo tanto, si un experimento consiste en un conjunto de 100 *realizaciones* independientes y cada realización se divide en 50 *chunks*, un total de 5000 *chunks* tendrán que ser gestionados. Cada *chunk* presenta un *job* a enviar, y caso caso de fallo, varios *jobs* serán necesarios para finalizar un mismo *chunk.*

5.2.3 Acceso a los infraestructuras

5.2.3 Acceso a los infraestructuras

WRF4G se compone de tres servicios de gestión que permiten a usuario interactuar con las infraestructuras computacionales abstrayéndolo complejidad de su naturaleza:

- El servicio de gestión de experimentos que crea, supervisa y gestiona los experimentos de acuerdo con las solicitudes del usuario, con el fin de proporcionar un acceso transparente a los distintos tipos de infraestructuras. WRF4G incorpora un conjunto de herramientas para interactuar con los diferentes componentes del *framework*, permitiendo una monitorización en tiempo real del experimento durante su ejecución.
- El servicio de gestión de *jobs* se realiza a través de DRM4G mediante su API, resultando transparente para los usuarios. DRM4G es el encargado de resolver el problema de manejar de manera eficiente los *jobs* entre los diferentes tipos recursos, proporcionando un acceso homogéneo a los mismos. Debe considerarse que DRM4G ha sido diseñado siguiendo requisitos de *escalabilidad* y robustez, que son características esenciales para ejecutar experimentos climáticos a gran escala.
- El servicio de gestión de datos que facilita la combinación de diferentes infraestructuras y que implica el uso de diferentes protocolos de transferencia de datos. El objetivo es ocultar la complejidad de la gestión de diferentes protocolos de transferencia al usuario final. Por lo tanto, el repositorio de datos utilizado por un *job* depende de la infraestructura o el recurso donde se ejecute. WRF4G permite indicar la ubicación de los repositorios de datos sobre un *path* base por recurso. El usuario puede personalizar fácilmente los repositorios de datos de entrada y salida y, cuando un *job* empieza a ejecutarse en un recurso, el repositorio de datos se selecciona dependiendo de la infraestructura en la que se ejecute el *job*.

5.2.4 Testbed

El objetivo de este *testbed* es ilustrar la utilidad de WRF4G para ejecutar experimentos climáticos sobre infraestructuras distribuidas. En concreto, se ejecutará un experimento de simulación de viento que previamente había sido ejecutado en un recurso HPC. En este caso el uso de infraestructuras distribuidas permitirá aumentar significativamente la capacidad de cálculo disponible para la simulación. Así, se mostrará cómo, aprovechando diferentes

infraestructuras a través de DRM4G, el tiempo total de ejecución puede ser reducido hasta en un 75%.

El experimento realizado para este *testbed* simulará el comportamiento del viento en la cuenca mediterránea durante un año. Para ello, se ejecutará un conjunto de 365 realizaciones independientes (una por cada día). Los detalles científicos del experimento pueden encontrarse en [4]. Los datos de entrada para cada realización serán de 230 MB (84 GB para todo el experimento). Para cada realización, el modelo WRF producirá 2 GB de variables meteorológicas con una resolución de 15 km con un paso de tiempo de una hora. Como el objetivo principal del estudio se centra en el viento, se ha seleccionado sólo esta variable, reduciendo la producción efectiva producida a 40 MB por realización. Por lo tanto, la salida de datos para todo el experimento se redujo de 730 GB a 14 GB.

5.2.4.1 Setup

Se realizarán dos experimentos diferentes para probar los beneficios de usar recursos de manera distribuida. Además, los dos se realizarán utilizando el *framework* WRF4G y se ejecutarán *jobs* MPI, en concreto serán 8 procesos MPI por *job*.

El primer experimento se ejecutará en un solo *cluster*, mientras que el segundo se realizará combinado diferentes tipos infraestructuras todas ellas configuradas a través de DRM4G. La lista de recursos disponibles para ejecutar los experimentos serán: los *clusters* HPC1 y HPC2 que disponen de un LRMS de tipo PBS, el *cluster* HPC3 administrado con SGE, la VO ESR perteneciente a EGI y una infraestructura *Cloud* privada gestionada con el VIM OpenNebula y compuesta por 4 VMs. Los accesos a las infraestructuras se realizaran mediante SSH y el *framework* será desplegado en el UI del *cluster* HPC1 perteneciente al grupo de investigación *Santander Meteorology Group*(SMG) [315] (ver figura 5.1). Es importante señalar que todas las infraestructuras, excepto los recursos recursos *Cloud*, se compartirán con otros investigadores y realizar ningún tipo de reserva previa. Mostrando así un entorno real en la ejecución.

5.2.4.1 Setup

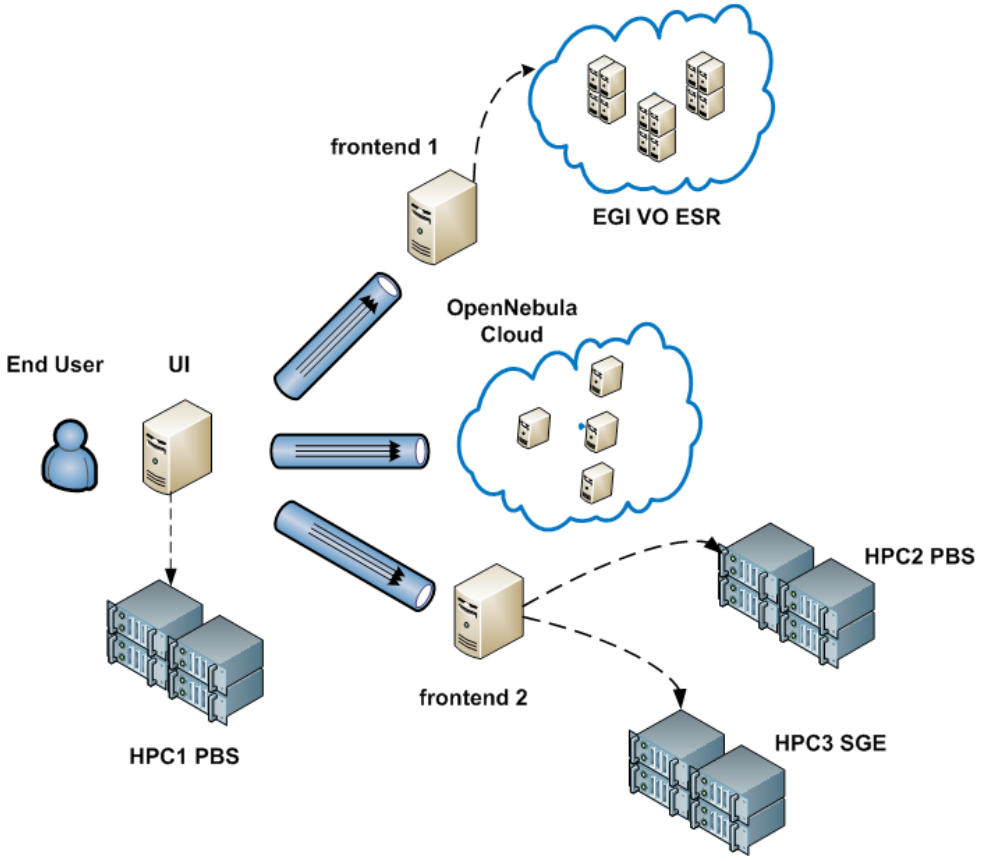

Figura 5.1: Setup para el experimento climático. En este caso WRF4G ha sido desplegado en UI del cluster HPC1. Los accesos a los clusters HPC2 y HPC3 se realiza mediante SSH a través del frontend 3 usando multiplexación. El acceso al Cloud privado OpenNebula se realiza utilizando SSH y multiplexación. Y el acceso a la e-Infraestructura EGI se realiza con el frontend 2 que es un GUI también a través de SSH y multiplexción.

Para demostrar que el uso de servicios de archivos de réplica es esencial cuando se ejecutan experimentos a gran escala en infraestructuras distribuidas, los datos de entrada se encontraban disponibles a través de SRM (replicado en 5 repositorios), NFS, y mediante servidores FTP. El proceso de replica de datos implicó la copia de 35 ficheros con una duración aproximada de 6 horas. En el caso de los datos de salida, estos se transferirán a medida que se producían, y como el tamaño de salida total es mucho menor que la entrada, los datos los experimentos se almacenarán en una ubicación común.

5.2.4.2 Resultados

El primer experimento fue ejecutado sólo una vez en el *cluster* HPC1 y concluyó en un tiempo de 60 horas y 15 minutos. El segundo experimento, en cambio, se ejecutó en todos los recursos anteriormente indicados y dos veces, para mostrar la importancia de la meta-

planificación cuando se ejecutan *jobs* en entornos distribuidos heterogéneos, mostrando dos tiempos de ejecución sensiblemente diferentes: 13 horas y 37 minutos y 33 horas y 10 minutos respectivamente. Como la ejecución del primer experimento no aporta complejidad a nivel de ejecución concurrente, nos centraremos en analizar el segundo de ellos.

Como muestra la figura 5.2 la distribución de *jobs* puede variar de una ejecución a otra. Y es que la política de meta-planificación empleada penaliza los recursos que no publican adecuadamente su información. Por lo tanto, cuando un recurso y/o infraestructura publica recursos disponibles pero los *jobs* en cambio son encolados, y no comienzan a ejecutarse hasta después de un tiempo determinado, el planificador los penaliza y pudiendo llegar a migrar los *jobs* a otros recursos. Lo mismo ocurre cuando los *jobs* fallan sin ninguna razón.

Esta situación se refleja perfectamente en la primera ejecución que muestra la figura 5.3. En ella se indica cómo durante la primera hora sólo un *job* se estaba ejecutando me manera concurrente. El proceso de planificación inicialmente comenzó a asignar *jobs* a recursos *Grid* que mostraban una gran capacidad disponible, todos ellos recursos de la VO ESR. Pero lamentablemente, sólo un *job* comenzó a ejecutarse. Después de una hora, el planificador comenzó a migrar todos los trabajos encolados a otros recursos.

Además, en la primera ejecución del segundo experimento, el planificador no tenía información previa sobre los recursos. Sin embargo, en la segunda ejecución una vez que el planificador ya estaba "entrenado" y disponía de un *background* de la ejecución anterior. En consecuencia, el comportamiento predeterminado del planificador es enviar los *jobs* a los recursos que mejor comportamiento han tenido según los datos almacenados.

Esto unido a que algunos *jobs* ejecutados en EGI fallaron porque alcanzaron las cuotas de almacenamiento de recursos, la pérdida de tiempo mientras los *jobs* estaban encolados y una mejor disponibilidad temporal de recursos disponibles en la segunda ejecución, son las principales razones por las que la primera ejecución fue dos veces más lenta (ver figura 5.3).

Dos elementos han de tenerse también en cuenta en ambas ejecuciones como son: el dinamismo de la propia infraestructura, EGI en este caso, puesto que existió un desfase temporal de una semana entre el inicio de cada ejecución; y los requerimientos de los *jobs* MPI a ejecutar, estos debían ejecutarse en un mismo nodo (PPN=8) para que la ejecución del WRF fuese eficiente, situación que limitaba a WNs de 8 *cores* libres.

5.2.4.2 Resultados

Finalmente, es importante resaltar que incluso la ejecución ineficiente en el entorno distribuido fue dos veces más rápida que la ejecución en un solo *cluster* (primer experimento), gracias en gran medida la *e-Infraestructura* EGI que favoreció de manera notable la ejecución de *jobs*.

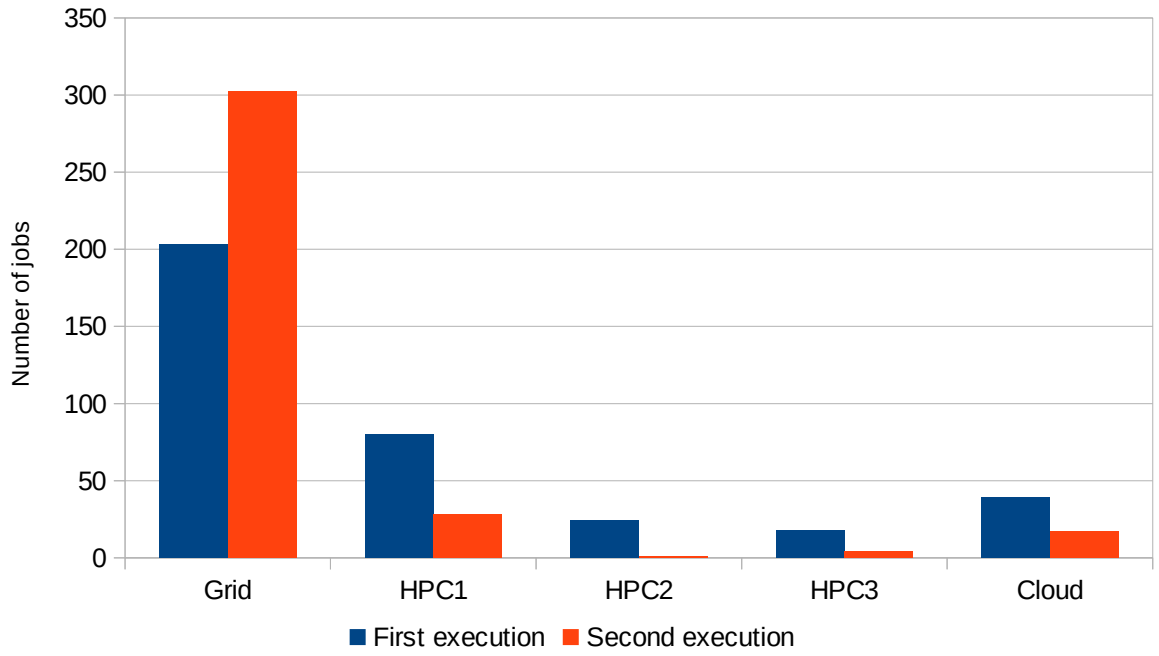

Figura 5.2: Distribución del número de jobs en función del tipo de recurso para cada ejecución en el segundo experimento.

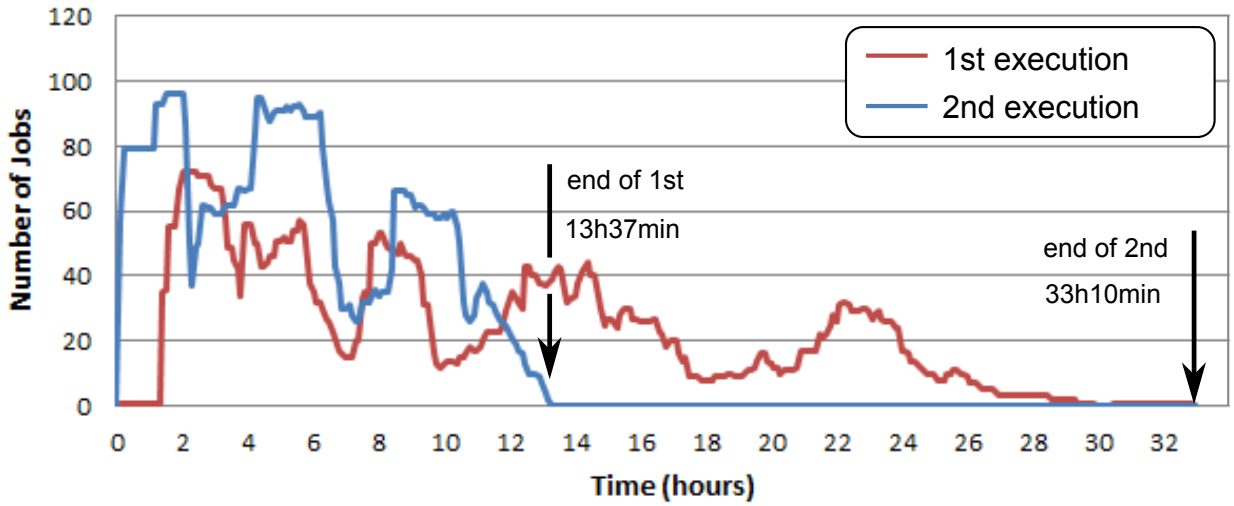

Figura 5.3: Número de jobs ejecutándose frente al tiempo en horas en el segundo experimento.

5.3 Big Data jobs: Hadoop y Hive

La computación *Cloud* es un paradigma de computación nuevo y emergente que tiene el potencial de cambiar completamente la forma en que las aplicaciones comerciales y científicas se implementan, alojan y ejecutan. Mediante el uso de la tecnología de virtualización, la computación C*loud* ofrece recursos y servicios escalables, fiables y flexibles. Sin embargo, aunque la capacidad computacional está aumentando con el paso del tiempo, el requisito de procesar la creciente cantidad de datos es cada vez más difícil. Los algoritmos tradicionales de procesamiento secuencial de datos no son lo suficientemente buenos para analizar este gran volumen de datos, y nuevos enfoques paralelos y algoritmos se proponen constantemente. Uno de estos ejemplos es Apache Hadoop, una implementación de código abierto del *framework* MapReduce [316] introducido por Google en 2004, que permite a los usuarios almacenar y procesar grandes cantidades de datos utilizando un entorno de *cluster* distribuido. En los últimos años, la combinación Hadoop y *Cloud* se ha convertido en un sistema muy popular para analizar *Big Data*. Muchas aplicaciones científicas como: predicción del tiempo [317], secuenciación del ADN [318] o dinámica molecular [319], han sido paralelizadas usando Hadoop y MapReduce.

Pero el uso de Hadoop no es fácil especialmente para aquellos usuarios que no están familiarizados con el entono MapReduce y quieren desarrollar programas de consulta y gestión, pues este carece de la capacidad de expresión de los lenguajes de consulta como SQL. Para facilitar el análisis de datos de una manera productiva, en 2007 Facebook diseñó Hive, cuyo objetivo era traer conceptos SQL como: tablas, columnas o particiones al entorno Hadoop.

Hive es una aplicación que se despliega sobre *clusters* Hadoop, diseñada para la consulta y análisis de datos masivos [320]. Además, es escalable y tolerante a fallos [321], y puede integrarse con algoritmos de compresión como: BZIP2 [322] , GZIP [323], LZO [324] o snappy [325]. Para la ejecución de operaciones utiliza un interfaz sencillo basado en sentencias SQL denominado lenguaje HQL(Hive SQL), cuyas operaciones se pueden transformar en tareas MapReduce [326]. Hive no es una base de datos relacional y no tiene formatos de datos especiales, pero si dispone de diferentes tipos de formatos de almacenamiento, incluyendo: TextFile [320], SequenceFile [320], RCFile [327] y ORC [328].

5.3.1 Enfoque genérico

5.3.1 Enfoque genérico

En las ejecuciones de *jobs* con capacidades de procesamiento de *Big Data* en *Cloud*, es necesario realizar tres tareas fundamentalmente. En primer lugar, el entorno de procesamiento específico, por ejemplo un *cluster* Hadoop, debe configurarse y desplegarse en la infraestructura *Cloud*. Esta configuración ha de ser automática, flexible y parametrizable fácilmente. En segundo lugar, las tareas de procesamiento de datos, *jobs* Hadoop y HQL *queries* por ejemplo, deben ejecutarse utilizando la infraestructura virtual establecida en el paso anterior. Por lo tanto, los ficheros de entrada de los *jobs* tienen que ser copiados al entorno de procesamiento HDFS [242] para poder ser utilizados. Finalmente, los resultados de los *jobs* deben ser copiados a infraestructuras de almacenamiento permanente, y los recursos de la infraestructura de procesamiento han de ser liberados. Este último paso, de nuevo debe ser automático.

5.3.2 Implementación

A pesar de los beneficios que aporta el uso de tecnologías como Hadoop o Hive, el despliegue y configuración de un *cluster* Hadoop suele estar muy por encima de las habilidades de los usuarios finales, levantándose así barreras significativas para el aprovechamiento de estas tecnologías en la investigación científica por ejemplo. Para paliar este problema y dotar al usuario final de la capacidades necesarias para la ejecución de *jobs*, DRM4G no sólo permite crear VMs en entornos *Cloud* (ver sección 3.4.1.3) sino también definir *clusters* Hadoop.

La implementación seguida para la creación, configuración y destrucción de instancias en el entrono *Cloud* se encuentra descrita en la figura 3.14. Aunque en este caso las instancias no son tratadas de manera individual, sino como un grupo o *cluster*. El proceso de creación es realizado de manera paralela haciendo viable el uso de *clusters* con un número elevado de instancias o nodos, evitando que este proceso sea directamente proporcional al número de nodos. La configuración del *cluster* se realiza en el tiempo de arranque de las VMs, durante el proceso de contextualización, de acuerdo a la configuración indicada por el usuario. Para reducir el tiempo de configuración se dispone de imágenes ya preconfiguradas en los repositorios EGI APPDB y AWS con Hadoop y Hive instalados, aunque también se facilita a los usuarios el *script* de instalación por si quieren realizar sus propias instalaciones. Cuando los nodos están creados y configurados, estos son inter-conectarlos para formar el *cluster* como tal, instanciando los servicios HDFS y YARN [329].

Pero un *cluster* Hadoop no es una infraestructura estática, sino todo lo contrario, es un *cluster* elástico que puede aumentar y reducir su tamaño aumentando o reduciendo su número de nodos por ejemplo. En consecuencia, DRM4G a través de su CLI ofrece también la posibilidad de, una vez creado el *cluster*, redimensionarlo para adaptarlo a las necesidades del usuario y la carga de trabajo disponible. Por lo tanto, no es necesario crear un nuevo *cluster* si la carga aumenta o destruirlo y volver a crear otro si disminuye, con las implicaciones que esto supone en tiempo y coste. Aunque este redimensionamiento elástico también implica un coste, añadir o eliminar nodos implica que el espacio de almacenamiento ha de ser redimensionado también y, por lo tanto, los bloques que componen los ficheros han de redistribuirse entre los nodos existentes, de forma que el reparto entre nodos sea equitativo.

Para finalizar, la liberación de recursos se realiza destruyendo los nodos de forma y paralela y ordenada cuando el usuario así lo requiere. Estos procesos automáticos para gestionar de Hadoop, abstraen al usuario de la complejidad de la infraestructura, que simplemente ha de indicar el tipo de *cluster* a utilizar (ver sección 5.3.3) y el *template* del *job* a ejecutar. A modo de ejemplo la figura 5.4 muestra la configuración necesaria para ejecutar el test WordCount [330] con DRM4G en un *cluster* Hadoop.

```
EXECUTABLE   = hadoop

ARGUMENTS    = -jar wordcount.jar org.myorg.WordCount /input /output -D
mapred.map.tasks=32 -D mapred.reduce.tasks=16

INPUT_FILES  = s3://user:key@bucket/input_file hdfs://input/input_file
OUTPUT_FILES = hdfs://output/output_file s3://user:key@bucket/output_file
```

Figura 5.4: Descripción de un job Mapreduce en DRM4G cuyos datos de entada y salida se encuentran en un bucket S3.

5.3.3 Configuración de recursos Hadoop

Para describir en DRM4G que un recurso a crear es un *cluster* Hadoop se emplea la variable *cluster_type* con el valor *hadoop* (ver figura 5.5). Una vez indicado el tipo de *cluster* a desplegar, una de las consideraciones más importantes al configurar es el número y el tipo de instancias que van a componer el *cluster*. Esto determinará la potencia de cálculo y la capacidad de almacenamiento de este. Es importante ajustar estos parámetros en función de la carga de trabajo

a procesar. Así, demasiada carga puede causar que el *cluster* se ejecute lentamente sobrecargandolo, o por el contrario si la carga es baja se puede infrautilizar generando costes innecesarios.

La cantidad de almacenamiento de un *cluster* Hadoop depende de tres factores: el número de nodos, la capacidad de almacenamiento del tipo de instancia, y la factor de réplica [242]. El factor de réplica (*hdfs_replicas*) es el número de veces que se almacena cada bloque de datos en HDFS, y que tiene una doble función, proveer de redundancia al sistema y facilitar que sea escalable (el factor de réplica por defecto es 1). El factor de réplica debe utilizarse adecuadamente ya que reduce la redundancia de los datos HDFS y la capacidad del *cluster* de recuperarse de bloques HDFS perdidos o dañados.

```
[HADOOP_CESNET]
cloud_provider      = fedcloud
endpoint            = https://carach5.ics.muni.cz:11443
image               = os_tpl#uuid_apache_hive_14_04_lts_warg_169
size                = resource_tpl#mammoth
cluster_type        = hadoop
instancies          = 10
extra_volume        = 100
public_key          = ~/.ssh/id_rsa.pub
lrms                = mp
max_jobs_running    = 1
```

Figura 5.5: Descripción de la configuración mínima necesaria para desplegar un cluster Hadoop en el site CESNET MetaCloud perteneciente a la VO fedcloud.egi.eu. Los parámetros necesarios para la su configuración serían los necesarios para acceder a la infraestructura: el tipo de proveedor Cloud (fedcloud), el nombre de la VO (vo) y el contact endpoint del site (endpoint); los necesarios para describir el cluster: identificador de la imagen el el repositorio EGI APPDB (image), las características físicas de los nodos (size), el número de instancias a desplegar (instancies), el almacenamiento extra que dispondrá cada nodo (extra_volume), la clave pública a utilizar (public_key); y los necesarios para el envío de jobs: el RM identificativo para jobs Hadoop (lrms) y el número de jobs máximo a ejecutar de manera concurrente (max_jobs_running).

Para calcular la capacidad de almacenamiento HDFS del *cluster*, ha de multiplicarse el número de nodos por la capacidad de almacenamiento del tipo de instancia que se haya seleccionado y dividir el total por el factor de replica. Si el valor de capacidad HDFS que se obtiene es menor que el tamaño de datos a procesar, se puede aumentar la cantidad de almacenamiento HDFS creando instancias con volúmenes adicionales mediante la variable *extra_storage* (GB).

A modo de resumen, la figura 5.5 describe la configuración a indicar para crear un *cluster* Hadoop en el *site* CESNET MetaCloud [331] perteneciente a la *e-Infraestructura* EGI FedCloud. Nótese, que al igual que un sistema LRMS típico, DRM4G permite también configurar un sistema de colas (*mp* para recursos MapReduce) indicando variables como *max_jobs_running* o *max_jobs_in_queue*.

5.3.4 Testbed

El objetivo de esta sección es mostrar un caso de uso real (ver figura 5.6) centrado en el despliegue de *clusters* Hadoop en *e-Infraestructuras* federadas para aplicaciones de *Big Data*. El experimento propuesto tratará de demostrar la vialidad de desplegar Hadoop *clusters on-demand* en entornos federados para el procesado de *datasets* en *batch processing*. La figura 5.6 recoge las infraestructuras que forman parte del caso de uso son: la infraestructura existente para la recogida y almacenamiento de datos a cargo del proyecto Euro-Argo [332] (*Physical Infrastructure EuroArgo*), la *e-Infraestructura* donde se procesarán los datos (EGI FedCloud), y el interfaz que dispone el usuario para visualizar los datos y gestionar los *jobs* de procesado (*User Intreface*). Concretamente, se utilizará la aplicación DRM4G (instalada en el PC del usuario) para desplegar *clusters* Hadoop de manera dinámica, sobre los cuales se importará el *dataset* (la importación y exportación de datos al sistema HDFS es gestionada de manera automática por DRM4G), para a continuación analizarlo mediante HQL.

En este caso de uso, el *dataset* acumulado es aproximadamente de 2 TB, y está compuesto por 1380 archivos CSV de un tamaño aproximando de 1,4 GB cada uno, con un número medio de registros de 18637819 y 12 parámetros por registro. Y aunque el tamaño de los datos es relativamente pequeño, debe considerarse que este aumenta a razón de 2,6 GB cada mes y, por lo tanto, la solución a emplear ha de ser escalable en el volumen de datos a procesar.

Teniendo en cuenta los requerimientos para el procesado de grandes volúmenes de datos en entornos Hadoop que en [327] se describen: rápida velocidad de lectura, rápido procesado de *queries*, alta eficiencia en el espacio de almacenamiento, y alta adaptabilidad a los diferentes tipos de análisis; y las conclusiones obtenidas en los estudios [333] [334] sobre la eficiencia del uso de datos comprimidos en Hive se ha optado por:

- Comprimir el *dataset* con el algoritmo LZO de forma que este sea de 256 GB.

5.3.4 Testbed

- Utilizar el formato de almacenamiento ORC, una versión mejora de RCFile, con compresión Snappy para mejorar la eficiencia en la ejecución de *queries*. Además, esta configuración permite mantener un tamaño total de almacenamiento HDFS, relativamente inferior al tamaño del *dataset* sin comprimir.

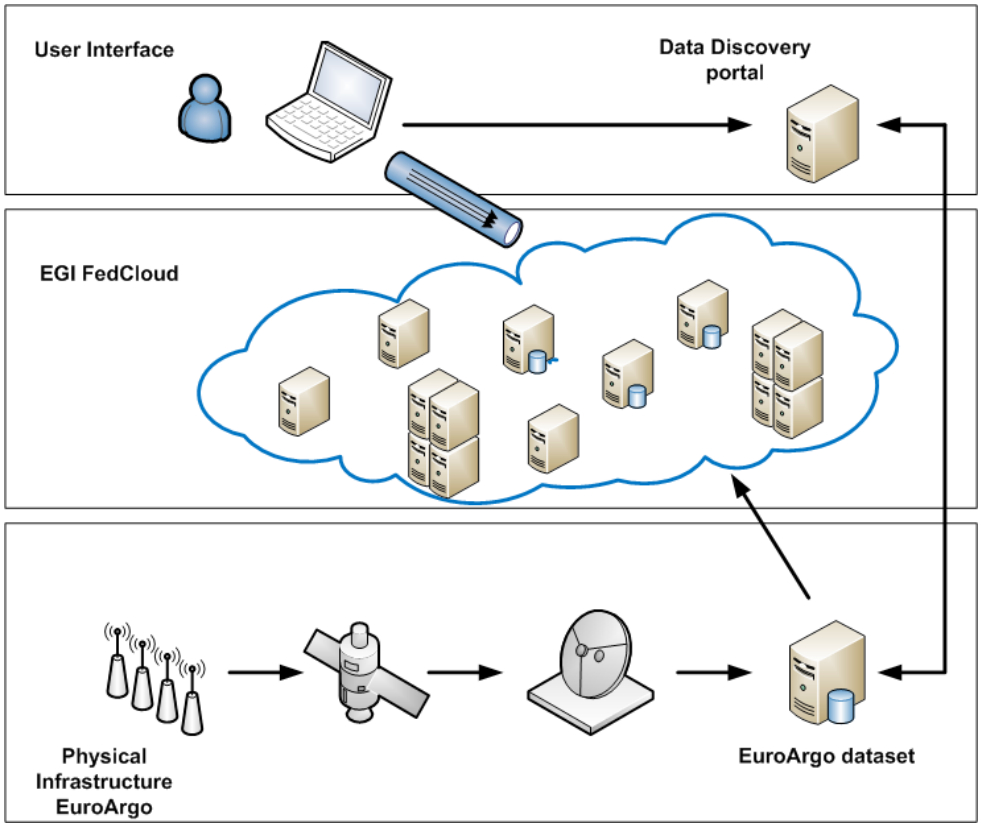

Figura 5.6: Caso de uso para el procesado de los datos provenientes del sistema mundial de boyas marinas recogidos a través de e-Infraestructuras europeas por proyecto Euro-Argo.

5.3.4.1 Setup

Previo paso al despliegue de *clusters* para procesar el *dataset*, se configuró y publicó una imagen de Ubuntu 14.04 *trusty server* en el repositorio AppDB Cloud Marketplace con Hive 1.2.1 y Hadoop 2.7.2, de acuerdo a las instrucciones facilitas por la propia *e-Infraestructura* [335]. Evitándose así que en cada despliegue ambas aplicaciones tuviesen que ser descargadas y configuradas desde cero. A continuación, esta imagen, disponible para los usuarios de la VO de test *fedcloud.egi.eu*, fue testada en todos proveedores *Cloud* de la federación que la desplegaron (nótese que aunque un usuario/desarrollador publique una imagen en *AppDB Cloud Marketplace*,

esto no implica que todos los proveedores *Cloud* de la federación la tengan disponible sus repositorios de imágenes).

El objetivo de este pretest era conocer la fiabilidad de los proveedores y su disponibilidad para desplegar infraestructuras complejas compuestas por varias instancias. De entre todos ellos, sólo CESNET-MetaCloud, BIFI, RECAS-BARI y IISAS-FEDCLOU demostraron una disponibilidad de instancias suficiente para desplegar *clusters* formados por al menos 16 instancias de tamaño *XLarge* (8 *virtual cores* y 16GB RAM) en la VO de test. De entre ellos, CESNET-MetaCloud, BIFI, y IISAS-FedCloud obtuvieron un 90% de fiabilidad sobre 20 despliegues de Hadoop *clusters* compuestos por 6 instancias. Finalmente, sólo el proveedor BIFI fue elegido para este test por la disponibilidad que ofrece de un contenedor OpenStack Swift. Si bien esto no implica que el contenedor no pueda ser usado desde otras instancias desplegadas en otros *sites* de la infraestructura, la cercanía física del mismo a las propias instancias desplegadas en el *site* permite tasas se transferencias mayores y una mayor fiabilidad. Una vez selecciono BIFI como único proveedor *Cloud* dentro de la *e-Infraestructura*, y haciendo uso de DRM4G para el despliegue de los *clusters* se realizarán dos test.

En el primer test, se desplegarán varios *clusters* de diferentes tamaños, donde se variará el número de nodos y en el tipo de *flavour* de la instancia, y sobre cada uno de ellos se ejecutará un *script* Hive compuesto por una *query* HQL. Esta consulta obtendrá todo el grupo de valores del *dataset* entre un rango de fechas, concretamente entre 2000-01-28 a las 20:00:00 horas y 2000-05-19 a las 20:00:00 horas, y cuyo resultado que deberá ser almacenado en el propio *cluster* con el mismo formato que los ficheros de entrada (CSV comprimido con LZO). El objetivo que se pretende es observar el comportamiento de los diferentes *clusters* para el mismo *dataset*.

En el segundo test en cambio, se mantendrá el tamaño del *cluster* y se variará el volumen del *dataset*, para observar cómo afecta su tamaño a un conjunto de *queries* seleccionadas que los futuros usuarios pueden demandar. El objetivo en este caso es comprobar la vialidad del uso combinado de Hive y Hadoop para el caso de uso. Debe recordarse que el *dataset* no es estático y que aumenta cada mes.

Como elemento común a ambos test, los *clusters* tendrán siempre un sistema HDFS con un factor de replica de 3, excepto si el *cluster* tiene menos de 3 nodos, en este caso será uno. Además el tamaño del almacenamiento hábil disponible será fijo (600GB), de tal forma que en cada configuración el tamaño adicional de cada instancia será diferente.

5.3.4.2 Resultados

La figura 5.7 recoge los resultados del primer test, donde se han desplegado cuatro *clusters* con diferente número de esclavos (nótese que un *cluster* Hadoop esta compuesto por un nodo maestro, encargado de la gestión, y un número N de esclavos encargados de almacenamiento HDFS y la ejecución de *jobs* MapReduce). En ella se observa cómo a media que el número de esclavos aumenta independientemente del tipo de instancia el tiempo de ejecución de la *query* también disminuye, escalando con el número de esclavos. Por otro lado, exceptuando el caso de un solo esclavo, en el que se observa una clara saturación del sistema, los tiempos de ejecución con las instancias de tipo *Large* son aproximadamente el doble que en los *clusters* compuestos por instancias *Xlarge*, donde se ejecutan el doble de procesos en cada esclavo. Debe indicarse que los tiempos representados incluyen también el tiempo el almacenamiento del resultado de la *query* en ficheros CSV, comprimiros en formato LZO, para su posterior exportación y análisis por parte del usuario.

Dados los resultados observados en la figura 5.7, se seleccionó el *cluster* compuesto por 5 esclavos de *flavour XLarge* para someterlo a un segundo test. Su elección se debe al compromiso de ofrece esta configuración entre consumo de recursos (número de esclavos y tamaño) y tiempo de procesado. Así, una vez fijada la configuración, en este *cluster* se ejecutaron seis *queries* que se simulaban posibles requerimientos de los usuarios (ver tabla 5.1). Estas *queries* realizadas para los datos entre las fechas 2000-01-28 a las 20:00:00 horas y 2000-05-19 a las 20:00:00 horas se mostraron independientes del resto del tamaño del *dataset*. El cual fue reducido (1000 ficheros) y aumentado con datos redundantes (2500 ficheros) sin que los tiempos variasen con un margen máximo de décimas de segundo. Esto se debe a la utilización del uso combinado de *partition* y *bucketing* [32] que permite agrupar grupos de datos de manera independiente. En concreto, en este test los ficheros fueron particionados por meses.

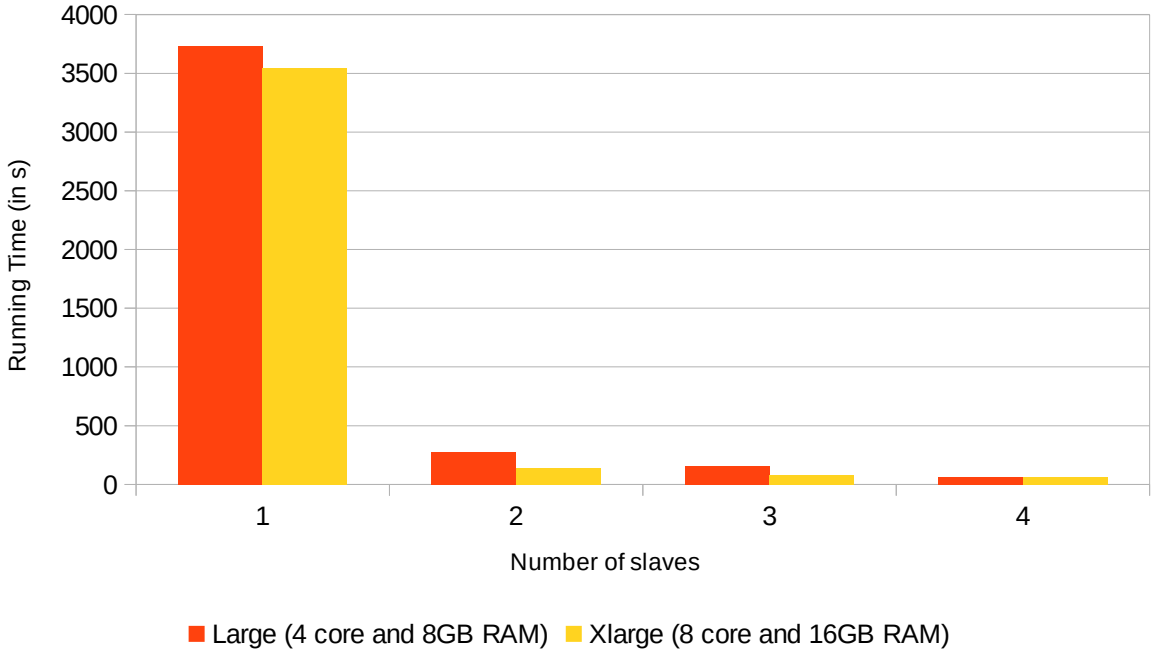

Figura 5.7: Tiempo de ejecución de la consulta HQL frente al número de esclavos del cluster.

Todas las variables el *dataset*	~140 segundos
Una variable del *dataset*	~99 segundos
Todas las variables del *dataset* que no incluya el mar Mediterráneo	~28 segundos
Todas las variables del *dataset* que comprenda el mar Mediterráneo	~28 segundos
Todas las variables del *dataset* de la boyas Marel-Iroise	~22 segundos
Todo las variables del *dataset* de la boya Dyfamed	~15 segundos

Tabla 5.1: Tiempos de ejecución de queries con distintos volúmenes de datos.

5.4 Conclusiones

En este capítulo se ha mostrado cómo DRM4G a través de su API DRMAA puede ser utilizado por parte de otras aplicaciones, como WRF4G, para la ejecución por ejemplo de simulaciones climáticas. Favoreciendo el uso distribuido de distintos tipos de infraestructuras y paradigmas comunicacionales de manera concurrente. Esta experiencia además no sólo puede ser extendida a otro tipo de modelos climáticos, sino a otras aplicaciones de otras VRCs que tengan una alta de manda de recursos para la ejecución de *jobs*. Así su uso combinado de diversas

infraestructuras podría suponer un salto cualitativo y cuantitativo en los tipos de experimentos a realizar.

Por otro lado, se ha visto la extensión de DRM4G para aplicaciones de *Big Data*, mostrada a través del caso de uso EuroAgo. Donde se ha utilizado la aplicación Hive para analizar *datasets* compuestos por 27 millones de *records* en entornos Hadoop. El uso combinado de DRM4G, Hadoop y Hive junto la *e-Infraestructura* EGI FedCloud ha sido demostrado viable para el análisis de volúmenes medios de datos, que pueden ser extendidos de forma escalable a mayores volúmenes de manera dinámica y flexible.

Finalmente, un elemento a tener en cuenta en el trabajo futuro sería la extensión de los tipos de *cluster* (TORQUE, SLURM, etc.) que DRM4G pudiera crear de manera dinámica en entornos *Cloud*, no sólo limitándose a *clusters* Hadoop para *Big Data*. Aunque esta situación se puede mitigar, mediante el uso combinado de DRM4G con otras aplicaciones (EC3 por ejemplo). Estas aplicaciones facilitarían el despliegue de infraestructuras de cálculo, para posteriormente el usuario habilitar el recursos a en DRM4G. La ventaja del uso de estas herramientas se centra en el despliegue optimizado de infraestructuras sobre *Cloud*, permitiendo por ejemplo que el tamaño o tipo de instancia del *frontend* del *cluster* desplegado sea distinto al de las instancias utilizadas por los WNs, crecimientos horizontales y verticales de los *clusters*, balanceos de carga en la red, grupos de seguridad, etc. Características todas ellas deseables para integrar en DRM4G.

Capítulo 6. Conclusions and Future Work

6.1 Conclusions

The main contribution of this research is the integration of computing services providing full interoperability among computing infrastructures. Although the implementation has been based on a Grid meta-scheduler, it is generic enough and applicable to any HPC environment and Cloud. This applicability has been demonstrated in several experiments by executing thousands of jobs on different type of resources.

As a result of this thesis, a framework called DRM4G was developed to define, submit, and manage computational jobs. DRM4G, which is based on GridWay, is an open platform, written in Python, that provides a single point of control for computing resources without installing any intermediate middlewares. Thus, any end user is able to use the same job template for running jobs on distributed infrastructures.

6.1 Conclusions

Regarding scheduling, it has been shown that the designed scheduler is capable of efficiently using hybrid infrastructures, and in particular, takes into account the differences between computing paradigms. For instance, economic savings are considered for commercial Cloud providers like AWS, consumed CPU hours are taken into account in HPC resources, and historical availability and reliability of each site are evaluated in Grid resources, among others. In the case of Cloud resources, it uses a dynamic deployment to instance elastic VMs and allow users to adapt this kind of resource to their own specific needs. It is important to remember that the scheduling process still has GridWay capabilities like dynamic scheduling, reporting, accounting, advanced policies, checkpointing support, performance slowdown detection, and opportunistic migration, among others. Furthermore, it can make an in-depth study of scheduling scalability, which is one of most important requirements of a successful meta-scheduler, proving analytically and experimentally that the solution enables the management of up to 100,000 jobs simultaneously.

The framework presented has been used for cutting edge areas of science such as climate simulations and Big Data, where the application has been applied to realistic use cases with success. In order to demonstrate these achievements, the framework has been tested with applications such as WRF4G performing climate calculations on HPC, Grid and Cloud infrastructures in production; and with the EuroArgo using a case scenario where several queries were run over a large dataset in a distributed storage by using Hadoop and Hive. It has been particularly challenging specially running climate simulations, as the execution of climate models usually involves managing complex workflows that produce large volumes of data as well [218].

In order to maximise the iterative design, which progressively refines the design through evaluation from the early stages of design, DRM4G code is hosted on GitHub, and all bundle packages are available on PyPi. As a result the evaluation steps enable developers and users to incorporate feedback, contributing to the application's development.

Finally, it is important to mention that the designed approach has been focused, instead of on fancy features, on the end user demands, from the perspective of researcher use and comprehension. As a consequence of employing a user-centered approaches, the system is an easy-to-learn, user-friendly interface that aims to be satisfying and engaging.

6.2 Future work

In future work, it is planned to extend DRM4G with new features in order to satisfy current user requirements regarding Cloud infrastructures. Specifically, this includes extension towards Amazon EC2's spot instances, which are particularly interesting for a completive performance per price rate. DRM4G does not support them for resource provisioning because the library that it uses (Apache Libcloud) does not support these types of instances.

Due to the implemented improvements in virtualization and the important benefits offered, deployment of virtual clusters in an IaaS *Cloud* provider is becoming more and more popular [210]. Thus, the possibility of managing dynamic clusters is also an extended feature. In this regard there are two options: first, to extend the experience deploying Hadoop clusters (see section 5.3) to other clusters; and second, to integrate consolidated applications such as EC3 with DMR4G.

Finally, it would be worthwhile to study the use of pilot jobs with DRM4G. In recent years, pilot jobs have had an important impact on scientific and distributed computing **[336]**. In fact, there is a pilot job application named GWPilot **[337]** that can be integrated with GridWay as a MAD. Thus, the combination of DRM4G with GWPilot can be useful in reducing the overhead due to waiting time in queues, especially when the workload is comprised of thousands of small tasks.

Capítulo 7. Bibliografía

[1] G. Andronico *et al.*, «e-Infrastructures for e-Science: A Global View», *J. Grid Comput.*, vol. 9, n.º 2, pp. 155-184, mar. 2011.

[2] Q. Snell, M. Clement, D. Jackson, y C. Gregory, «The performance impact of advance reservation meta-scheduling», en *Workshop on Job Scheduling Strategies for Parallel Processing*, 2000, pp. 137–153.

[3] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, y E. Varvarigos, «A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in Grid networks», *Comput. Commun.*, vol. 32, n.º 7–10, pp. 1172-1184, may 2009.

[4] M. Menendez, M. García-Díez, L. Fita, J. Fernández, F. J. Méndez, y J. M. Gutiérrez, «High-resolution sea wind hindcasts over the Mediterranean area», *Clim. Dyn.*, vol. 42, n.º 7-8, pp. 1857-1872, sep. 2013.

[5] J. Andreeva *et al.*, «Experiment Dashboard for monitoring computing activities of the LHC virtual organizations», *J. Grid Comput.*, vol. 8, n.º 2, pp. 323–339, 2010.

[6] C. M. S. Collaboration, R. Adolphi, y others, «The CMS experiment at the CERN LHC», *JInst*, vol. 3, n.º 08, p. S08004, 2008.

[7] G. Juve *et al.*, «Scientific workflow applications on Amazon EC2», en *2009 5th IEEE International Conference on E-Science Workshops*, 2009, pp. 59-66.

[8] A. Fox, «Cloud Computing—What's in It for Me as a Scientist?», *Science*, vol. 331, n.º 6016, pp. 406-407, ene. 2011.

[9] I. Foster y C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[10] K. Dowd, *High Performance Computing*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1993.

[11] M. Livny *et al.*, «Mechanisms for High Throughput Computing», *SPEEDUP*, vol. 11, 1997.

[12] G. George, M. R. Haas, y A. Pentland, «Big Data and Management», *Acad. Manage. J.*, vol. 57, n.º 2, pp. 321-326, abr. 2014.
[13] L. S. Group y E. O. for N. Research, *Design study of the large hadron collider (LHC): a multiparticle collider in the LEP tunnel*. Not Avail, 1991.
[14] C. A. Stewart, S. Simms, B. Plale, M. Link, D. Y. Hancock, y G. C. Fox, «What is Cyberinfrastructure», en *Proceedings of the 38th Annual ACM SIGUCCS Fall Conference: Navigation and Discovery*, New York, NY, USA, 2010, pp. 37–44.
[15] «European Research Area - ERA». [En línea]. Disponible en: http://ec.europa.eu/research/era/index_en.htm. [Accedido: 31-ago-2016].
[16] R. Buyya y others, «High performance cluster computing: Architectures and systems (volume 1)», *Prentice Hall Up. SaddleRiver NJ USA*, vol. 1, p. 999, 1999.
[17] V. Alexandrov *et al.*, «2013 International Conference on Computational ScienceLeveraging e-Infrastructures for Urgent Computing», *Procedia Comput. Sci.*, vol. 18, pp. 2177-2186, ene. 2013.
[18] «Red Española de Supercomputación». [En línea]. Disponible en: https://www.bsc.es/marenostrum-support-services/res. [Accedido: 31-ago-2016].
[19] D. Kranzlmüller, J. M. de Lucas, y P. Öster, «The European Grid Initiative (EGI)», en *Remote Instrumentation and Virtual Laboratories*, F. Davoli, N. Meyer, R. Pugliese, y S. Zappatore, Eds. Springer US, 2010, pp. 61-66.
[20] K. Jeffery, D. Kyriazis, E. Fernández-del-Castillo, D. Scardaci, y Á. L. García, «The EGI Federated Cloud e-Infrastructure», *Procedia Comput. Sci.*, vol. 68, pp. 196-205, ene. 2015.
[21] «EGI». [En línea]. Disponible en: https://www.egi.eu/. [Accedido: 31-ago-2016].
[22] «PRACE Research Infrastructure», *PRACE Research Infrastructure*. [En línea]. Disponible en: http://www.prace-ri.eu/. [Accedido: 30-ago-2016].
[23] R. Pordes *et al.*, «The open science grid», *J. Phys. Conf. Ser.*, vol. 78, n.º 1, p. 012057, 2007.
[24] J. Towns *et al.*, «XSEDE: Accelerating Scientific Discovery», *Comput. Sci. Eng.*, vol. 16, n.º 5, pp. 62-74, sep. 2014.
[25] B. J. K. Evans *et al.*, «Computational Environments and Analysis methods available on the NCI High Performance Computing (HPC) and High Performance Data (HPD) Platform», en *AGU Fall Meeting Abstracts*, 2014, vol. 1, p. 01.
[26] N. Mangala, B. B. P. Rao, S. Chattopadhyay, R. Sridharan, y N. S. C. Babu, «GARUDA: Pan-Indian distributed e-infrastructure for compute-data intensive collaborative science», *CSI Trans. ICT*, vol. 1, n.º 2, pp. 193-204, jun. 2013.
[27] D. Lecarpentier *et al.*, «EUDAT: A New Cross-Disciplinary Data Infrastructure for Science», *Int. J. Digit. Curation*, vol. 8, n.º 1, pp. 279-287, jun. 2013.
[28] K. Jeffery *et al.*, «OpenAIRE Guidelines for CRIS Managers: Supporting Interoperability of Open Research Information through Established Standards», *Procedia Comput. Sci.*, vol. 33, pp. 33-38, ene. 2014.
[29] D. N. Williams *et al.*, «The Earth System Grid Federation: software framework supporting CMIP5 data analysis and dissemination», *CLIVAR Exch.*, vol. 56: 16, n.º 2, pp. 40-42, 2011.
[30] «Welcome to Apache™ Hadoop®!» [En línea]. Disponible en: http://hadoop.apache.org/. [Accedido: 30-ago-2016].
[31] M. N. Vora, «Hadoop-HBase for large-scale data», en *2011 International Conference on Computer Science and Network Technology (ICCSNT)*, 2011, vol. 1, pp. 601-605.
[32] A. Thusoo *et al.*, «Hive: A Warehousing Solution over a Map-reduce Framework», *Proc VLDB Endow*, vol. 2, n.º 2, pp. 1626–1629, ago. 2009.
[33] «GÉANT». [En línea]. Disponible en: http://www.geant.org/. [Accedido: 06-sep-2016].

[34] I. Foster, C. Kesselman, y S. Tuecke, «The Anatomy of the Grid: Enabling Scalable Virtual Organizations», *Int J High Perform Comput Appl*, vol. 15, n.º 3, pp. 200–222, ago. 2001.
[35] I. Foster y C. Kesselman, «What is the Grid», *Three Point Checkl.*, vol. 20, 2003.
[36] «West-Life Structural Biology VRE». [En línea]. Disponible en: http://about.west-life.eu/. [Accedido: 05-sep-2016].
[37] «MuG Virtual Research Environment – The MuG project will provide tools to integrate the navigation in genomics data from sequence to 3D/4D chromatin dynamics data.» [En línea]. Disponible en: http://www.multiscalegenomics.eu/MuGVRE/. [Accedido: 05-sep-2016].
[38] «EVER-EST». [En línea]. Disponible en: http://www.ever-est.eu/. [Accedido: 05-sep-2016].
[39] M. Prica, R. Pugliese, A. D. Linz, y A. Curri, «Adapting the Instrument Element to Support a Remote Instrumentation Infrastructure», en *Remote Instrumentation and Virtual Laboratories*, F. Davoli, N. Meyer, R. Pugliese, y S. Zappatore, Eds. Springer US, 2010, pp. 11-22.
[40] D. Adami, A. Cheptsov, F. Davoli, I. Liabotis, R. Pugliese, y A. Zafeiropoulos, «The DORII project test bed: Distributed eScience applications at work», en *5th International Conference on Testbeds and Research Infrastructures for the Development of Networks Communities and Workshops, 2009. TridentCom 2009*, 2009, pp. 1-6.
[41] D. F. McMullen *et al.*, «The Common Instrument Middleware Architecture», en *Grid Enabled Remote Instrumentation*, F. Davoli, N. Meyer, R. Pugliese, y S. Zappatore, Eds. Springer US, 2009, pp. 393-407.
[42] V. Welch *et al.*, «Security for Grid services», en *12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings*, 2003, pp. 48-57.
[43] D. Recordon y D. Reed, «OpenID 2.0: A Platform for User-centric Identity Management», en *Proceedings of the Second ACM Workshop on Digital Identity Management*, New York, NY, USA, 2006, pp. 11–16.
[44] K. Zeilenga, «Lightweight directory access protocol (ldap): Technical specification road map», 2006.
[45] M. D. de Assunção y R. Buyya, «Performance analysis of allocation policies for interGrid resource provisioning», *Inf. Softw. Technol.*, vol. 51, n.º 1, pp. 42-55, ene. 2009.
[46] S. Sotiriadis, N. Bessis, y N. Antonopoulos, «Towards Inter-cloud Schedulers: A Survey of Meta-scheduling Approaches», en *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, 2011, pp. 59-66.
[47] O. Wäldrich, P. Wieder, y W. Ziegler, «A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources», en *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, N. Meyer, y J. Waśniewski, Eds. Springer Berlin Heidelberg, 2005, pp. 782-791.
[48] N. Andrade, W. Cirne, F. Brasileiro, y P. Roisenberg, «OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing», en *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, y U. Schwiegelshohn, Eds. Springer Berlin Heidelberg, 2003, pp. 61-86.
[49] R. Buyya, R. Ranjan, y R. N. Calheiros, «InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services», en *Algorithms and Architectures for Parallel Processing*, C.-H. Hsu, L. T. Yang, J. H. Park, y S.-S. Yeo, Eds. Springer Berlin Heidelberg, 2010, pp. 13-31.
[50] C. Vázquez Blanco, «Arquitectura para el aprovisionamiento dinámico de recursos computacionales», Universidad Complutense de Madrid, 2013.
[51] «The FitSM Standard | www.fitsm.eu». [En línea]. Disponible en: http://fitsm.itemo.org/.

[Accedido: 12-sep-2016].
[52] T. as Erl, «Service-Oriented Architecture (SOA) Concepts, Technology and Design», 2005.
[53] C. Vázquez, E. Huedo, R. S. Montero, y I. M. Llorente, «Federation of TeraGrid, EGEE and OSG Infrastructures Through a Metascheduler», *Future Gener Comput Syst*, vol. 26, n.º 7, pp. 979–985, jul. 2010.
[54] M. Heikkurinen y O. Appleton, «SLM and SDM Challenges in Federated Infrastructures», en *Euro-Par 2011: Parallel Processing Workshops*, M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. D. Martino, M. Gerndt, E. Jeannot, R. Namyst, J. Roman, S. L. Scott, J. L. Traff, G. Vallée, y J. Weidendorfer, Eds. Springer Berlin Heidelberg, 2011, pp. 64-72.
[55] D. Petcu y J. L. Vzquez-Poletti, *European Research Activities in Cloud Computing*. United Kingdom: Cambridge Scholars Publishing, 2012.
[56] M. Riedel, A. Streit, D. Mallmann, F. Wolf, y T. Lippert, «Experiences and Requirements for Interoperability Between HTC and HPC-driven e-Science Infrastructure», en *Future Application and Middleware Technology on e-Science*, O.-H. Byeon, J. H. Kwon, T. Dunning, K. W. Cho, y A. Savoy-Navarro, Eds. Springer US, 2010, pp. 113-123.
[57] M. Riedel *et al.*, «Interoperation of world-wide production e-Science infrastructures», *Concurr. Comput. Pract. Exp.*, vol. 21, n.º 8, pp. 961-990, jun. 2009.
[58] M. Dias de Assunção, R. Buyya, y S. Venugopal, «InterGrid: A case for internetworking islands of Grids», *Concurr. Comput. Pract. Exp.*, vol. 20, n.º 8, pp. 997–1024, 2008.
[59] P. Mell y T. Grance, «The NIST definition of cloud computing», 2011.
[60] «Amazon Web Services (AWS) - Cloud Computing Services», *Amazon Web Services, Inc.* [En línea]. Disponible en: //aws.amazon.com/. [Accedido: 12-sep-2016].
[61] P. Kacsuk y G. Sipos, «Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal», *J. Grid Comput.*, vol. 3, n.º 3-4, pp. 221-238, ene. 2006.
[62] A. J. Ferrer *et al.*, «OPTIMIS: A holistic approach to cloud service provisioning», *Future Gener. Comput. Syst.*, vol. 28, n.º 1, pp. 66–77, 2012.
[63] N. Grozev y R. Buyya, «Inter-Cloud architectures and application brokering: taxonomy and survey», *Softw. Pract. Exp.*, vol. 44, n.º 3, pp. 369-390, mar. 2014.
[64] G. Mathieu, D. A. Richards, D. J. Gordon, C. D. C. Novales, P. Colclough, y Matthew Viljoen, «GOCDB, a topology repository for a worldwide grid infrastructure», *J. Phys. Conf. Ser.*, vol. 219, n.º 6, p. 062021, 2010.
[65] J. Shiers, «The Worldwide LHC Computing Grid (worldwide LCG)», *Comput. Phys. Commun.*, vol. 177, n.º 1–2, pp. 219-223, jul. 2007.
[66] «AAI - EGIWiki». [En línea]. Disponible en: https://wiki.egi.eu/wiki/AAI. [Accedido: 12-sep-2016].
[67] R. Alfieri *et al.*, «VOMS, an Authorization System for Virtual Organizations», en *Grid Computing*, F. F. Rivera, M. Bubak, A. G. Tato, y R. Doallo, Eds. Springer Berlin Heidelberg, 2004, pp. 33-40.
[68] R. Byrom *et al.*, «APEL: An implementation of Grid accounting using R-GMA», en *UK e-Science All Hands Conference*, 2005.
[69] M. Ellert, A. Konstantinov, B. Kónya, O. Smirnova, y A. Wäänänen, «The NorduGrid project: using Globus toolkit for building GRID infrastructure», *Nucl. Instrum. Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip.*, vol. 502, n.º 2–3, pp. 407-410, abr. 2003.
[70] «ARGO». [En línea]. Disponible en: http://argo.egi.eu/#. [Accedido: 12-sep-2016].
[71] P. Bhoj, S. Singhal, y S. Chutani, «SLA management in federated environments», *Comput. Netw.*, vol. 35, n.º 1, pp. 5-24, ene. 2001.

[72] «GSI-OpenSSH». [En línea]. Disponible en: http://toolkit.globus.org/toolkit/docs/5.0/5.0.4/security/openssh/. [Accedido: 13-sep-2016].
[73] «OpenSSH». [En línea]. Disponible en: http://www.openssh.com/. [Accedido: 13-sep-2016].
[74] J. Novotny, S. Tuecke, y V. Welch, «An online credential repository for the Grid: MyProxy», en *10th IEEE International Symposium on High Performance Distributed Computing, 2001. Proceedings*, 2001, pp. 104-111.
[75] G. Bell, J. Gray, y A. Szalay, *Petascale computations systems: Balanced cyberinfrastructure in a data-centric world*. 2005.
[76] «B2ACCESS - EUDAT». [En línea]. Disponible en: https://www.eudat.eu/services/b2access. [Accedido: 12-sep-2016].
[77] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, y T. Scavo, «Security assertion markup language (saml) v2. 0 technical overview», *OASIS Com. Draft*, vol. 2, p. 2008, 2008.
[78] D. E. Hammer-Lahav y D. Hardt, «The oauth2. 0 authorization protocol. 2011», IETF Internet Draft.
[79] G. K. Thiruvathukal y K. Laufer, «Plone and content management», *Comput. Sci. Eng.*, vol. 6, n.º 4, pp. 88–95, 2004.
[80] W. C. Skamarock *et al.*, «A Description of the Advanced Research WRF Version 2», jun. 2005.
[81] A. Shawish y M. Salama, «Cloud Computing: Paradigms and Technologies», en *Inter-cooperative Collective Intelligence: Techniques and Applications*, F. Xhafa y N. Bessis, Eds. Springer Berlin Heidelberg, 2014, pp. 39-67.
[82] D. P. Anderson y G. Fedak, «The Computational and Storage Potential of Volunteer Computing», en *Sixth IEEE International Symposium on Cluster Computing and the Grid, 2006. CCGRID 06*, 2006, vol. 1, pp. 73-80.
[83] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, y I. Brandic, «Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility», *Future Gener. Comput. Syst.*, vol. 25, n.º 6, pp. 599-616, jun. 2009.
[84] I. Foster, Y. Zhao, I. Raicu, y S. Lu, «Cloud Computing and Grid Computing 360-Degree Compared», en *2008 Grid Computing Environments Workshop*, 2008, pp. 1-10.
[85] Y. Huang, N. Bessis, P. Norrington, P. Kuonen, y B. Hirsbrunner, «Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm», *Future Gener. Comput. Syst.*, vol. 29, n.º 1, pp. 402-415, ene. 2013.
[86] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, y M. Livny, «Inter-operating Grids Through Delegated Matchmaking», en *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2007, p. 13:1–13:12.
[87] K. Leal, E. Huedo, y I. M. Llorente, «A decentralized model for scheduling independent tasks in Federated Grids», *Future Gener. Comput. Syst.*, vol. 25, n.º 8, pp. 840-852, sep. 2009.
[88] C. Evangelinos y C. N. Hill, «Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2», en *In The 1st Workshop on Cloud Computing and its Applications (CCA*, 2008.
[89] E. Roloff, M. Diener, A. Carissimi, y P. Navaux, «High Performance Computing in the Cloud: Deployment, Performance and Cost Efficiency», 2012. .
[90] M. Rodríguez, D. Tapiador, J. Fontán, E. Huedo, R. S. Montero, y I. M. Llorente, «Dynamic Provisioning of Virtual Clusters for Grid Computing», en *Euro-Par 2008 Workshops - Parallel Processing*, E. César, M. Alexander, A. Streit, J. L. Träff, C. Cérin, A.

Knüpfer, D. Kranzlmüller, y S. Jha, Eds. Springer Berlin Heidelberg, 2008, pp. 23-32.
[91] C. Vazquez, E. Huedo, R. S. Montero, y I. Llorente, «Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers», en *Grid and Pervasive Computing Conference, 2009. GPC '09. Workshops at the*, 2009, pp. 113-120.
[92] R. Bolze y E. Deelman, «Exploiting the Cloud of Computing Environments: An Application's Perspective», *Cloud Comput. Softw. Serv.*, p. 173, 2010.
[93] G. Mateescu, W. Gentzsch, y C. J. Ribbens, «Hybrid Computing—Where HPC meets grid and Cloud Computing», *Future Gener. Comput. Syst.*, vol. 27, n.º 5, pp. 440-453, may 2011.
[94] W. Gropp, E. Lusk, N. Doss, y A. Skjellum, «A high-performance, portable implementation of the MPI message passing interface standard», *Parallel Comput.*, vol. 22, n.º 6, pp. 789-828, sep. 1996.
[95] R. Chandra, *Parallel programming in OpenMP*. Morgan Kaufmann, 2001.
[96] A. Geist, *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.
[97] V. Fernández-Quiruelas, J. Fernández, C. Baeza, A. S. Cofiño, y J. M. Gutiérrez, «Execution management in the GRID, for sensitivity studies of global climate simulations», *Earth Sci. Inform.*, vol. 2, n.º 1-2, pp. 75-82, ene. 2009.
[98] V. Fernández-Quiruelas, J. Fernández, A. S. Cofiño, L. Fita, y J. M. Gutiérrez, «Benefits and requirements of grid computing for climate applications. An example with the community atmospheric model», *Environ. Model. Softw.*, vol. 26, n.º 9, pp. 1057-1069, sep. 2011.
[99] H. Casanova, D. Zagorodnov, F. Berman, y A. Legrand, «Heuristics for Scheduling Parameter Sweep Applications in Grid Environments», en *Proceedings of the 9th Heterogeneous Computing Workshop*, Washington, DC, USA, 2000, p. 349–.
[100] K. Binder, «Introduction: Theory and "Technical" Aspects of Monte Carlo Simulations», en *Monte Carlo Methods in Statistical Physics*, P. D. K. Binder, Ed. Springer Berlin Heidelberg, 1986, pp. 1-45.
[101] R. Buyya, «Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing as the 5th Utility», en *9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009. CCGRID '09*, 2009, pp. 1-1.
[102] M. D. de Assunção, A. di Costanzo, y R. Buyya, «A cost-benefit analysis of using cloud computing to extend the capacity of clusters», *Clust. Comput.*, vol. 13, n.º 3, pp. 335-347, abr. 2010.
[103] D. Montes, J. A. Añel, T. F. Pena, P. Uhe, y D. C. H. Wallom, «Enabling BOINC in Infrastructure as a Service Cloud Systems», *Geosci. Model Dev. Discuss.*, pp. 1-24, sep. 2016.
[104] J. L. Vázquez-Poletti, G. Barderas, I. M. Llorente, y P. Romero, «A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure», en *Applied Parallel and Scientific Computing*, K. Jónasson, Ed. Springer Berlin Heidelberg, 2010, pp. 33-42.
[105] J. L. Vázquez-Poletti, D. Santos-Muñoz, I. M. Llorente, y F. Valero, «A Cloud for Clouds: Weather Research and Forecasting on a Public Cloud Infrastructure», en *Cloud Computing and Services Sciences*, M. Helfert, F. Desprez, D. Ferguson, F. Leymann, y V. M. Muñoz, Eds. Springer International Publishing, 2015, pp. 3-11.
[106] N. Bessis, S. Sotiriadis, F. Pop, y V. Cristea, «An architectural strategy for meta-scheduling in Inter-clouds», en *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, 2012, pp. 1184–1189.
[107] T. Goodale *et al.*, «SAGA: A Simple API for Grid Applications. High-level application

programming on the Grid», *Comput. Methods Sci. Technol.*, vol. 12, n.º 1, pp. 7–20, 2006.
[108] P. Tröger, R. Brobst, D. Gruber, M. Mamonski, y D. Templeton, «Distributed resource management application API Version 2 (DRMAA)», 2012.
[109] I. Foster *et al.*, *OGSA basic execution service version 1.0*. 2007.
[110] J. Almond y D. Snelling, «UNICORE: uniform access to supercomputing as an element of electronic commerce», *Future Gener. Comput. Syst.*, vol. 15, n.º 5–6, pp. 539-548, oct. 1999.
[111]Y. Yan y B. Chapman, «Comparative Study of Distributed Resource Management Systems – SGE, LSF, PBS Pro, and LoadLeveler».
[112] A. B. Yoo, M. A. Jette, y M. Grondona, *SLURM: Simple Linux Utility for Resource Management*, vol. 2862. 2003.
[113] R. L. Henderson, «Job scheduling under the Portable Batch System», en *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson y L. Rudolph, Eds. Springer Berlin Heidelberg, 1995, pp. 279-294.
[114] W. Gentzsch, «Sun Grid Engine: towards creating a compute power grid», en *First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings*, 2001, pp. 35-36.
[115] S. Zhou, «LSF: Load sharing in large-scale heterogeneous distributed systems», presentado en Workshop on Cluster Computing, 1992.
[116] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, y J. F. Skovira, «Workload management with loadleveler», *IBM Redb.*, vol. 2, p. 2, 2001.
[117] S. Shepler *et al.*, «Network file system (NFS) version 4 protocol», *Network*, 2003.
[118] F. B. Schmuck y R. L. Haskin, «GPFS: A Shared-Disk File System for Large Computing Clusters.», en *FAST*, 2002, vol. 2, pp. 231–244.
[119] P. J. Braam y others, «The Lustre storage architecture», 2004.
[120] I. Foster y C. Kesselman, «Globus: A Metacomputing Infrastructure Toolkit», *Int. J. Supercomput. Appl.*, vol. 11, pp. 115–128, 1996.
[121] S. Burke, L. Field, y D. Horat, «Migration to the GLUE 2.0 information schema in the LCG/EGEE/EGI production Grid», *J. Phys. Conf. Ser.*, vol. 331, n.º 6, p. 062004, 2011.
[122] K. Czajkowski *et al.*, «A resource management architecture for metacomputing systems», en *Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62–82.
[123] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, y S. Tuecke, «GridFTP: Protocol extensions to FTP for the Grid», *Glob. Grid ForumGFD-RP*, vol. 20, pp. 1–21, 2003.
[124] E. Laure y others, «Programming the Grid with gLite. Computational Methods in Science and Technology», *Comput Methods Sci Technol*, vol. 12, 2006.
[125] M. Ellert *et al.*, «Advanced Resource Connector middleware for lightweight computational Grids», *Future Gener. Comput. Syst.*, vol. 23, n.º 2, pp. 219-240, feb. 2007.
[126] I. Foster, «Globus toolkit version 4: Software for service-oriented systems», en *IFIP international conference on network and parallel computing*, 2005, pp. 2–13.
[127] P. Andreetto *et al.*, «Cream: A simple, grid-accessible, job management system for local computational resources», *Proc. CHEP'06 MUMBAY*, 2006.
[128] B. Bockelman *et al.*, «Commissioning the HTCondor-CE for the Open Science Grid», *J. Phys. Conf. Ser.*, vol. 664, n.º 6, p. 062003, 2015.
[129] F. Paganelli, Z. Nagy, y O. Smirnova, «Arc computing element system administrator guide», 2011.
[130] A. Shoshani, A. Sim, y J. Gu, «Storage Resource Managers: Middleware Components for Grid Storage», en *Tenth Goddard Conference on Mass Storage Systems and Technologies*,

2002, p. 209.
[131] B. Sotomayor, R. S. Montero, I. Llorente, y I. Foster, «Virtual Infrastructure Management in Private and Hybrid Clouds», *IEEE Internet Comput.*, vol. 13, n.º 5, pp. 14-22, sep. 2009.
[132] K. Jackson, C. Bunch, y E. Sigler, *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2015.
[133] K. Keahey, I. Foster, T. Freeman, y X. Zhang, «Virtual workspaces: Achieving quality of service and quality of life in the grid», *Sci. Program.*, vol. 13, n.º 4, pp. 265–275, 2005.
[134] D. Nurmi *et al.*, «Eucalyptus: an open-source cloud computing infrastructure», *J. Phys. Conf. Ser.*, vol. 180, n.º 1, p. 012051, jul. 2009.
[135] «Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS», *Amazon Web Services, Inc.* [En línea]. Disponible en: //aws.amazon.com/ec2/. [Accedido: 18-sep-2016].
[136] «Amazon Simple Storage Service (S3) - Cloud Storage», *Amazon Web Services, Inc.* [En línea]. Disponible en: //aws.amazon.com/s3/. [Accedido: 18-sep-2016].
[137] «ISO/IEC 17826:2016 - Information technology -- Cloud Data Management Interface (CDMI)», *ISO*. [En línea]. Disponible en: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=70226. [Accedido: 19-sep-2016].
[138] S. N. I. Association y others, *Cloud data management interface (CDMI)*. 2010.
[139] M. Sain, «The future of cloud integration: Cloud Middleware?», en *2013 15th International Conference on Advanced Communication Technology (ICACT)*, 2013, pp. 1-1.
[140] «Open Grid Forum». [En línea]. Disponible en: https://www.ogf.org/ogf/doku.php. [Accedido: 19-sep-2016].
[141] D. Davis y G. Pilz, «Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP», *Vol DSP-0263 May*, 2012.
[142] «Home | DMTF». [En línea]. Disponible en: https://www.dmtf.org/. [Accedido: 02-ene-2017].
[143] «Universal Cloud Management Platform by RightScale». [En línea]. Disponible en: http://www.rightscale.com/. [Accedido: 02-ene-2017].
[144] «Scalr: Enterprise-Grade Cloud Management Platform». [En línea]. Disponible en: http://www.scalr.com/. [Accedido: 02-ene-2017].
[145] «Kaavo - Cloud Management Software». [En línea]. Disponible en: http://www.kaavo.com/. [Accedido: 02-ene-2017].
[146] T. A. S. Foundation, «Apache Libcloud is a standard Python library that abstracts away differences among multiple cloud provider APIs», *Apache Libcloud*. [En línea]. Disponible en: https://libcloud.apache.org/index.html. [Accedido: 02-ene-2017].
[147] «Apache jclouds® :: Home». [En línea]. Disponible en: http://jclouds.apache.org/. [Accedido: 02-ene-2017].
[148] «Supported Providers — Apache Libcloud 0.14.0-dev documentation». [En línea]. Disponible en: https://libcloud.readthedocs.io/en/latest/supported_providers.html. [Accedido: 02-ene-2017].
[149] E. eu IASA, «EGI Applications Database». [En línea]. Disponible en: https://appdb.egi.eu. [Accedido: 20-sep-2016].
[150] R. Bradshaw, N. Desai, T. Freeman, y K. Keahey, «A scalable approach to deploying and managing appliances», en *TeraGrid Conference*, 2007.
[151] K. Keahey y T. Freeman, «Contextualization: Providing One-Click Virtual Clusters», en *IEEE Fourth International Conference on eScience, 2008. eScience '08*, 2008, pp. 301-308.
[152] D. Armstrong, K. Djemame, S. Nair, J. Tordsson, y W. Ziegler, «Towards a Contextualization Solution for Cloud Platform Services», en *2011 IEEE Third International*

Conference on Cloud Computing Technology and Science (CloudCom), 2011, pp. 328-331.
[153] «cloud-init». [En línea]. Disponible en: https://launchpad.net/cloud-init. [Accedido: 19-sep-2016].
[154] U. G. Matlab, «The mathworks», *Inc Natick MA*, vol. 1992, 1760.
[155] V. Fernández-Quiruelas, «Simulación con modelos climáticos en infraestructuras de computación distribuida», Universidad de Cantabria, 2017.
[156] E. N. Lorenz, «The predictability of a flow which possesses many scales of motion», *Tellus*, vol. 21, n.º 3, pp. 289-307, jun. 1969.
[157] A. Arteaga, O. Fuhrer, y T. Hoefler, «Designing Bit-Reproducible Portable High-Performance Applications», en *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2014, pp. 1235–1244.
[158] J. Demmel y H. D. Nguyen, «Numerical Reproducibility and Accuracy at ExaScale», en *2013 IEEE 21st Symposium on Computer Arithmetic*, 2013, pp. 235-237.
[159] P. Langlois, R. Nheili, y C. Denis, «Numerical reproducibility: Feasibility issues», en *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1-5.
[160] J. D. Ullman, «NP-complete scheduling problems», *J. Comput. Syst. Sci.*, vol. 10, n.º 3, pp. 384-393, jun. 1975.
[161] L. Anand, D. Ghose, y V. Mani, «ELISA: An estimated load information scheduling algorithm for distributed computing systems», *Comput. Math. Appl.*, vol. 37, n.º 8, pp. 57-85, abr. 1999.
[162] N. Bessis, S. Sotiriadis, V. Cristea, y F. Pop, «Modelling Requirements for Enabling Meta-scheduling in Inter-Clouds and Inter-Enterprises», en *2011 Third International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, 2011, pp. 149-156.
[163] C. Daval-Frerot, M. Lacroix, y H. Guyennet, «Federation of resource traders in object-oriented distributed systems», en *International Conference on Parallel Computing in Electrical Engineering, 2000. PARELEC 2000. Proceedings*, 2000, pp. 84-88.
[164] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, y M. Livny, «Inter-operating Grids Through Delegated Matchmaking», en *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2007, p. 13:1–13:12.
[165] A. AuYoung, B. Chun, A. Snoeren, y A. Vahdat, «Resource allocation in federated distributed computing infrastructures», en *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure*, 2004, vol. 9.
[166] V. V. Korkhov, J. T. Moscicki, y V. V. Krzhizhanovskaya, «Dynamic workload balancing of parallel applications with user-level scheduling on the Grid», *Future Gener. Comput. Syst.*, vol. 25, n.º 1, pp. 28–34, 2009.
[167] G. Aceto, A. Botta, W. de Donato, y A. Pescapè, «Cloud monitoring: A survey», *Comput. Netw.*, vol. 57, n.º 9, pp. 2093-2115, jun. 2013.
[168] K. Wood y M. Anderson, «Understanding the Complexity Surrounding Multitenancy in Cloud Computing», en *2011 IEEE 8th International Conference on e-Business Engineering (ICEBE)*, 2011, pp. 119-124.
[169] S. Sotiriadis, N. Bessis, F. Xhafa, y N. Antonopoulos, «From Meta-computing to Interoperable Infrastructures: A Review of Meta-schedulers for HPC, Grid and Cloud», en *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, 2012, pp. 874-883.
[170] J. H. Abawajy, «Fault-Tolerant Dynamic Job Scheduling Policy», en *Distributed and Parallel Computing*, M. Hobbs, A. M. Goscinski, y W. Zhou, Eds. Springer Berlin

Heidelberg, 2005, pp. 165-173.
[171] J. Frey, T. Tannenbaum, M. Livny, I. Foster, y S. Tuecke, «Condor-G: A computation management agent for multi-institutional grids», *Clust. Comput.*, vol. 5, n.º 3, pp. 237–246, 2002.
[172] P. Andreetto *et al.*, «The gLite workload management system», en *Journal of Physics: Conference Series*, 2008, vol. 119, p. 062007.
[173] B. Bode, D. M. Halstead, R. Kendall, Z. Lei, y D. Jackson, «The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters.», en *Annual Linux Showcase & Conference*, 2000.
[174] D. Krishnamurthy, M. Alemzadeh, y M. Moussavi, «Towards automated HPC scheduler configuration tuning», *Concurr. Comput. Pract. Exp.*, vol. 23, n.º 15, pp. 1723-1748, oct. 2011.
[175] S. Venugopal, K. Nadiminti, H. Gibbins, y R. Buyya, «Designing a resource broker for heterogeneous grids», *Softw. Pract. Exp.*, vol. 38, n.º 8, pp. 793–825, 2008.
[176] R. Bazoubandi y F. Abdoli, «A Dynamic Pricing Algorithm for Super Scheduling of Computational Grid», en *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, T. Sobh y K. Elleithy, Eds. Springer International Publishing, 2015, pp. 453-460.
[177] H. Shan, L. Oliker, y R. Biswas, «Job superscheduler architecture and performance in computational grid environments», en *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, 2003, p. 44.
[178] R. Buyya, D. Abramson, y J. Giddy, «Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid», en *High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/Exhibition on*, 2000, vol. 1, pp. 283–289.
[179] L. Tomás, A. C. Caminero, O. Rana, C. Carrión, y B. Caminero, «A GridWay-based autonomic network-aware metascheduler», *Future Gener. Comput. Syst.*, vol. 28, n.º 7, pp. 1058-1069, jul. 2012.
[180] S. Yangui, I.-J. Marshall, J.-P. Laisne, y S. Tata, «CompatibleOne: The open source cloud broker», *J. Grid Comput.*, vol. 12, n.º 1, pp. 93–109, 2014.
[181] «SlipStream: Smart Cloud Application Management | SixSq». [En línea]. Disponible en: http://sixsq.com/products/slipstream/index.html. [Accedido: 24-sep-2016].
[182] A. Tsaregorodtsev, V. Garonne, y I. Stokes-Rees, «DIRAC: A scalable lightweight architecture for high throughput computing», en *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 19–25.
[183] V. M. Muñoz, A. C. Ramo, R. G. Diaz, y V. F. Albor, «How to run scientific applications with dirac in federated hybrid clouds», en *ADVCOMP 2013, The Seventh International Conference on Advanced Engineering Computing and Applications in Sciences*, 2013, pp. 73–78.
[184] A. Kertész, Z. Farkas, P. Kacsuk, y T. Kiss, «Grid Interoperability by Multiple Broker Utilization and Meta-Broking», en *Grid Enabled Remote Instrumentation*, Springer, 2009, pp. 303–312.
[185] A. Kertesz, P. K. Kacsuk, I. Rodero, F. Guim, y J. Corbalan, «Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management», 2007.
[186] I. J. Taylor, E. Deelman, D. B. Gannon, y M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
[187] E. Deelman *et al.*, «Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems», *Sci Program*, vol. 13, n.º 3, pp. 219–237, jul. 2005.

[188] Y. Zhao *et al.*, «Swift: Fast, reliable, loosely coupled parallel computation», en *Services, 2007 IEEE Congress on*, 2007, pp. 199–206.

[189] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, y S. Mock, «Kepler: an extensible system for design and execution of scientific workflows», en *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, 2004, pp. 423–424.

[190] T. Oinn *et al.*, «Taverna: a tool for the composition and enactment of bioinformatics workflows», *Bioinformatics*, vol. 20, n.º 17, pp. 3045-3054, nov. 2004.

[191] P. Kacsuk, «P-GRADE portal family for grid infrastructures», *Concurr. Comput. Pract. Exp.*, vol. 23, n.º 3, pp. 235-245, mar. 2011.

[192] J. Nagle, «On Packet Switches With Infinite Storage», en *RFC 970, FACC Palo Alto*, 1985.

[193] V. Hamscher, U. Schwiegelshohn, A. Streit, y R. Yahyapour, «Evaluation of Job-Scheduling Strategies for Grid Computing», en *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, London, UK, UK, 2000, pp. 191–202.

[194] E. Huedo, R. S. Montero, y I. M. Llorente, «Experiences on adaptive grid scheduling of parameter sweep applications», en *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, 2004, pp. 28–33.

[195] J. Cao, A. T. Chan, Y. Sun, S. K. Das, y M. Guo, «A taxonomy of application scheduling tools for high performance cluster computing», *Clust. Comput.*, vol. 9, n.º 3, pp. 355–371, 2006.

[196] L. Ilijašić y L. Saitta, «Characterization of a Computational Grid As a Complex System», en *Proceedings of the 6th International Conference Industry Session on Grids Meets Autonomic Computing*, New York, NY, USA, 2009, pp. 9–18.

[197] A. Kertész y P. Kacsuk, «A taxonomy of grid resource brokers», en *Distributed and Parallel Systems*, Springer, 2007, pp. 201–210.

[198] K. Krauter, R. Buyya, y M. Maheswaran, «A taxonomy and survey of grid resource management systems for distributed computing», *Softw. Pract. Exp.*, vol. 32, n.º 2, pp. 135–164, 2002.

[199] S. Fitzgerald, «Grid Information Services for Distributed Resource Sharing», en *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, Washington, DC, USA, 2001, p. 181–.

[200] N. Palatin, A. Leizarowitz, A. Schuster, y R. Wolff, «Mining for Misconfigured Machines in Grid Systems», en *Data Mining Techniques in Grid Computing Environments*, W. Dubitzky, Ed. John Wiley & Sons, Ltd, 2008, pp. 71-89.

[201] R. M. Piro, A. Guarise, G. Patania, y A. Werbrouck, «Using historical accounting information to predict the resource usage of grid jobs», *Future Gener. Comput. Syst.*, vol. 25, n.º 5, pp. 499-510, may 2009.

[202] S. Bhardwaj, L. Jain, y S. Jain, «Cloud computing: A study of infrastructure as a service (IAAS)», *Int. J. Eng. Inf. Technol.*, vol. 2, n.º 1, pp. 60–63, 2010.

[203] M. Mahjoub, A. Mdhaffar, R. B. Halima, y M. Jmaiel, «A Comparative Study of the Current Cloud Computing Technologies and Offers», en *2011 First International Symposium on Network Cloud Computing and Applications (NCCA)*, 2011, pp. 131-134.

[204] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, y J. Wettinger, «Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies», en *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, R. Meersman, H. Panetto, T. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. D. Leenheer, y D. Dou, Eds. Springer Berlin Heidelberg, 2013, pp. 130-148.

[205] V. K. Singh y K. Dutta, «Dynamic Price Prediction for Amazon Spot Instances», en *2015 48th Hawaii International Conference on System Sciences*, 2015, pp. 1513-1520.
[206] «Spot Instances - Amazon Elastic Compute Cloud». [En línea]. Disponible en: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html. [Accedido: 12-nov-2016].
[207] A. Haeberlen, «A case for the accountable cloud», *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, n.º 2, pp. 52–57, 2010.
[208] P. Kunszt *et al.*, «Accelerating 3D Protein Modeling Using Cloud Computing: Using Rosetta as a Service on the IBM SmartCloud», en *e-Science Workshops (eScienceW), 2011 IEEE Seventh International Conference on*, 2011, pp. 166–169.
[209] M. Caballer, I. Blanquer, G. Moltó, y C. de Alfonso, «Dynamic Management of Virtual Infrastructures», *J. Grid Comput.*, vol. 13, n.º 1, pp. 53-70, abr. 2014.
[210] M. Caballer, C. De Alfonso, F. Alvarruiz, y G. Moltó, «EC3: Elastic cloud computing cluster», *J. Comput. Syst. Sci.*, vol. 79, n.º 8, pp. 1341–1351, 2013.
[211] G. Kecskemeti, M. Gergely, A. Visegradi, Z. Nemeth, J. Kovacs, y P. Kacsuk, «One Click Cloud Orchestrator: Bringing Complex Applications Effortlessly to the Clouds», en *European Conference on Parallel Processing*, 2014, pp. 38–49.
[212] C. Vázquez, E. Huedo, R. S. Montero, y I. M. Llorente, «On the use of clouds for grid resource provisioning», *Future Gener. Comput. Syst.*, vol. 27, n.º 5, pp. 600-605, may 2011.
[213] B. Liu, B. Sotomayor, R. Madduri, K. Chard, y I. Foster, «Deploying bioinformatics workflows on clouds with galaxy and globus provision», en *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 1087–1095.
[214] A. Vulpe y M. Frincu, «Exploring Scalability in Pattern Finding in Galactic Structure using MapReduce», en *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*, 2016, pp. 582–587.
[215] N. Bessis, S. Sotiriadis, F. Xhafa, F. Pop, y V. Cristea, «Meta-scheduling Issues in Interoperable HPCs, Grids and Clouds», *Int J Web Grid Serv*, vol. 8, n.º 2, pp. 153–172, ago. 2012.
[216] I. M. Carrión, E. Huedo, y I. M. Llorente, «Interoperating grid infrastructures with the GridWay metascheduler», *Concurr. Comput. Pract. Exp.*, vol. 27, n.º 9, pp. 2278-2290, jun. 2015.
[217] R. Nandakumar *et al.*, «The LHCb computing data challenge DC06», *J. Phys. Conf. Ser.*, vol. 119, n.º 7, p. 072023, 2008.
[218] V. Fernández-Quiruelas, C. Blanco, A. S. Cofiño, y J. Fernández, «Large-scale climate simulations harnessing clusters, grid and cloud infrastructures», *Future Gener. Comput. Syst.*, vol. 51, pp. 36-44, oct. 2015.
[219] Z. Farkas y P. Kacsuk, «P-GRADE Portal: A generic workflow system to support user communities», *Future Gener. Comput. Syst.*, vol. 27, n.º 5, pp. 454-465, may 2011.
[220] «DRM4G – Wiki». [En línea]. Disponible en: https://meteo.unican.es/trac/wiki/DRM4G. [Accedido: 03-abr-2017].
[221] «GitHub DRM4G». [En línea]. Disponible en: https://github.com/SantanderMetGroup/DRM4G. [Accedido: 15-feb-2017].
[222] E. Huedo, R. S. Montero, y I. M. Llorente, «A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services», *Future Gener. Comput. Syst.*, vol. 23, n.º 2, pp. 252-261, feb. 2007.
[223] D. Thain, T. Tannenbaum, y M. Livny, «Distributed computing in practice: the Condor experience», *Concurr. Comput. Pract. Exp.*, vol. 17, n.º 2-4, pp. 323–356, 2005.
[224] T. Tannenbaum, D. Wright, K. Miller, y M. Livny, «Condor: a distributed job scheduler»,

en *Beowulf cluster computing with Linux*, 2001, pp. 307–350.
[225] «European Union Public License, version 1.1 (EUPL-1.1) | Open Source Initiative». [En línea]. Disponible en: https://opensource.org/licenses/EUPL-1.1. [Accedido: 15-feb-2017].
[226] E. Huedo, R. S. Montero, y I. M. Llorente, «A framework for adaptive execution in grids», *Softw. Pract. Exp.*, vol. 34, n.º 7, pp. 631-651, jun. 2004.
[227] I. M. Carrión, E. Huedo, y I. M. Llorente, «Interoperating grid infrastructures with the GridWay metascheduler», *Concurr. Comput. Pract. Exp.*, p. n/a-n/a, nov. 2012.
[228] J. Herrera, E. Huedo, R. S. Montero, y I. M. Llorente, «Developing Grid-Aware Applications with DRMAA on Globus-Based Grids», en *Euro-Par 2004 Parallel Processing*, M. Danelutto, M. Vanneschi, y D. Laforenza, Eds. Springer Berlin Heidelberg, 2004, pp. 429-435.
[229] K. Leal, E. Huedo, y I. M. Llorente, «Performance-based scheduling strategies for HTC applications in complex federated grids», *Concurr. Comput. Pract. Exp.*, p. n/a-n/a, 2009.
[230] A. J. Rubio-Montero, M. A. Rodríguez-Pascual, y R. Mayo-García, «Evaluation of an Adaptive Framework for Resilient Monte Carlo Executions», en *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, New York, NY, USA, 2015, pp. 448–455.
[231] «DRM4G CLI». [En línea]. Disponible en: https://meteo.unican.es/trac/wiki/DRM4G/available_commands. [Accedido: 08-nov-2016].
[232] «DRM4G/API». [En línea]. Disponible en: https://meteo.unican.es/trac/wiki/DRM4G/API. [Accedido: 05-dic-2016].
[233] E. Huedo, R. S. Montero, y I. M. Llorente, «A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services», *Future Gener. Comput. Syst.*, vol. 23, n.º 2, pp. 252-261, feb. 2007.
[234] «Welcome to Python.org», *Python.org*. [En línea]. Disponible en: https://www.python.org/. [Accedido: 08-nov-2016].
[235] «PyPI - the Python Package Index : Python Package Index». [En línea]. Disponible en: https://pypi.python.org/pypi. [Accedido: 25-nov-2016].
[236] «Easy Install — Setuptools documentation». [En línea]. Disponible en: http://setuptools.readthedocs.io/en/latest/easy_install.html. [Accedido: 25-nov-2016].
[237] «pip — pip 9.0.1 documentation». [En línea]. Disponible en: https://pip.pypa.io/en/stable/. [Accedido: 15-nov-2016].
[238] «DRM4G/Installation». [En línea]. Disponible en: https://meteo.unican.es/trac/wiki/DRM4G/Installation. [Accedido: 25-nov-2016].
[239] R. Sezov, *Liferay in action: the official guide to Liferay portal development*. Manning Shelter Island, NY, 2012.
[240] «HTCondor Debian Repository». [En línea]. Disponible en: https://research.cs.wisc.edu/htcondor/debian/. [Accedido: 25-nov-2016].
[241] R. A. Kulkarni, M. K. Damade, y S. Bano, «HADOOP ARCHITECTURE: A distributed file system».
[242] K. Shvachko, H. Kuang, S. Radia, y R. Chansler, «The hadoop distributed file system», en *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, 2010, pp. 1–10.
[243] «Welcome to Swift's documentation! — swift 2.10.1.dev108 documentation». [En línea]. Disponible en: http://docs.openstack.org/developer/swift/. [Accedido: 14-nov-2016].
[244] C. Munro y B. Koblitz, «Performance comparison of the LCG2 and gLite file catalogues», *Nucl. Instrum. Methods Phys. Res. Sect. Accel. Spectrometers Detect. Assoc. Equip.*, vol. 559, n.º 1, pp. 48–52, 2006.
[245] A. Celesti, F. Tusa, M. Villari, y A. Puliafito, «Security and cloud computing: intercloud

identity management infrastructure», en *Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on*, 2010, pp. 263–265.

[246] S. Langella, S. Oster, S. Hastings, F. Siebenlist, T. Kurc, y J. Saltz, «Dorian: Grid Service Infrastructure for Identity Management and Federation», en *19th IEEE Symposium on Computer-Based Medical Systems (CBMS'06)*, 2006, pp. 756-761.

[247] F. Berman, G. Fox, y A. J. Hey, *Grid computing: making the global infrastructure a reality*, vol. 2. John Wiley and sons, 2003.

[248] A. Cavoukian y others, «Privacy and Digital Identity: Implications For The Internet», en *Proceedings from Identity in the Information Society Workshop*, 2008.

[249] Z. Farkas *et al.*, «AutoDock Gateway for Molecular Docking Simulations in Cloud Systems», *Cloud Comput. E-Sci. Appl.*, p. 217, 2015.

[250] «GISELA Grid». [En línea]. Disponible en: http://www.gisela-grid.eu/. [Accedido: 03-mar-2017].

[251] A. J. Stell, R. O. Sinnott, y J. P. Watt, «Comparison of advanced authorisation infrastructures for grid computing», en *19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*, 2005, pp. 195–201.

[252] «OpenBSD manual pages». [En línea]. Disponible en: http://man.openbsd.org/OpenBSD-current/man1/ssh-agent.1. [Accedido: 10-ene-2017].

[253] P. Kacsuk, «P-GRADE portal family for grid infrastructures», *Concurr. Comput. Pract. Exp.*, vol. 23, n.º 3, pp. 235-245, mar. 2011.

[254] P. Russom y others, «Big data analytics», *TDWI Best Pract. Rep. Fourth Quart.*, pp. 1–35, 2011.

[255] S. Walton, *Linux socket programming*. Sams, 2001.

[256] «Support Knowledge Center @ BSC-CNS». [En línea]. Disponible en: https://www.bsc.es/user-support/mn3.php. [Accedido: 17-nov-2016].

[257] «MareNostrum User's Guide». [En línea]. Disponible en: https://www.bsc.es/user-support/mn3.php. [Accedido: 04-abr-2017].

[258] «ESR VO Introduction | European Earth Science Grid». [En línea]. Disponible en: http://www.euearthsciencegrid.org/content/esr-vo-introduction. [Accedido: 16-nov-2016].

[259] A. J. Rubio-Montero, E. Huedo, R. S. Montero, y I. M. Llorente, «Management of Virtual Machines on Globus Grids Using GridWay», en *2007 IEEE International Parallel and Distributed Processing Symposium*, 2007, pp. 1-7.

[260] B. Parák, Z. Šustr, F. Feldhaus, P. Kasprzakc, y M. Srbac, «The rOCCI Project–Providing Cloud Interoperability with OCCI 1.1», en *International Symposium on Grids and Clouds (ISGC)*, 2014, vol. 23.

[261] «HOWTO11 How to use the rOCCI Client - EGIWiki». [En línea]. Disponible en: https://wiki.egi.eu/wiki/HOWTO11_How_to_use_the_rOCCI_Client. [Accedido: 23-nov-2016].

[262] «HOWTO09 How to use Federated Cloud Storage - EGIWiki». [En línea]. Disponible en: https://wiki.egi.eu/wiki/HOWTO09_How_to_use_Federated_Cloud_Storage. [Accedido: 21-nov-2016].

[263] «Amazon Elastic Block Store (EBS) – Block Storage for EC2», *Amazon Web Services, Inc.* [En línea]. Disponible en: //aws.amazon.com/ebs/. [Accedido: 10-dic-2016].

[264] «Institute for Biocomputation and Physics of Complex Systems - Instituto de Biocomputación y Física de Sistemas Complejos (BIFI)». [En línea]. Disponible en: http://bifi.es/en/. [Accedido: 22-nov-2016].

[265] A. Balasko, Z. Farkas, y P. Kacsuk, «Building science gateways by utilizing the generic

WS-PGRADE/gUSE workflow system», *Comput. Sci.*, vol. 14, n.º 2), pp. 307–325, 2013.
[266] A. C. Ramo, R. G. Diaz, y A. Tsaregorodtsev, «DIRAC RESTful API», *J. Phys. Conf. Ser.*, vol. 396, n.º 5, p. 052019, 2012.
[267] P. Troger, H. Rajic, A. Haas, y P. Domagalski, «Standardization of an API for distributed resource management systems», en *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, 2007, pp. 619–626.
[268] A. Tsaregorodtsev *et al.*, «DIRAC: a community grid solution», *J. Phys. Conf. Ser.*, vol. 119, n.º 6, p. 062048, 2008.
[269] C. V. Blanco, E. Huedo, R. S. Montero, y I. M. Llorente, «Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers», en *Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference*, Washington, DC, USA, 2009, pp. 113–120.
[270] A. Lorca, J. Martín-Caro, R. Núñez-Ramírez, y J. Martínez-Salazar, «Merging on-demand HPC resources from amazon EC2 with the grid: a case study of a Xmipp application», 2012.
[271] S. Bagnasco *et al.*, «AliEn: ALICE environment on the GRID», en *Journal of Physics: Conference Series*, 2008, vol. 119, p. 062012.
[272] T. Maeno, «PanDA: distributed production and distributed analysis system for ATLAS», en *Journal of Physics: Conference Series*, 2008, vol. 119, p. 062036.
[273] A. Luckow, L. Lacinski, y S. Jha, «Saga bigjob: An extensible and interoperable pilot-job abstraction for distributed applications and systems», en *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 135–144.
[274] M. Rodriguez-Pascual, R. M. Mayo-García, y I. M. Llorente, «Montera: a framework for efficient execution of Monte Carlo codes on grid infrastructures», *Comput. Inform.*, vol. 32, n.º 1, pp. 113–144, 2013.
[275] J. L. Vázquez-Poletti, E. Huedo, R. S. Montero, y I. M. Llorente, «A Comparison Between Two Grid Scheduling Philosophies: EGEE WMS and Grid Way», *Multiagent Grid Syst*, vol. 3, n.º 4, pp. 429–439, dic. 2007.
[276] E. Huedo, R. Montero, I. M. Llorente, y others, «The GridWay framework for adaptive scheduling and execution on grids», *Scalable Comput. Pract. Exp.*, vol. 6, n.º 3, 2001.
[277] F. Michel, J. Montagnat, y T. Glatard, «Technical support for Life Sciences communities on a production grid infrastructure», *Heal. Appl. Technol. Meet Sci. Gatew. Life Sci.*, vol. 175, p. 81, 2012.
[278] S. Burke, M. A. Pradillo, L. Field, y O. Keeble, «GLUE 2 deployment: Ensuring quality in the EGI/WLCG information system», en *Journal of Physics: Conference Series*, 2014, vol. 513, p. 032012.
[279] A. D. Peris, J. Hernandez, E. Huedo, y I. M. Llorente, «Data location-aware job scheduling in the grid. Application to the GridWay metascheduler», *J. Phys. Conf. Ser.*, vol. 219, n.º 6, p. 062043, 2010.
[280] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, y D. Epema, «Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing», *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, n.º 6, pp. 931-945, jun. 2011.
[281] S. Shen, K. Deng, A. Iosup, y D. Epema, «Scheduling jobs in the cloud using on-demand and reserved instances», en *European Conference on Parallel Processing*, 2013, pp. 242–254.
[282] U. Sharma, P. Shenoy, S. Sahu, y A. Shaikh, «A cost-aware elasticity provisioning system for the cloud», en *Distributed Computing Systems (ICDCS), 2011 31st International*

Conference on, 2011, pp. 559–570.
[283] B. Nicolae, F. Cappello, y G. Antoniu, «Optimizing multi-deployment on clouds by means of self-adaptive prefetching», en *European Conference on Parallel Processing*, 2011, pp. 503–513.
[284] D. Villegas, A. Antoniou, S. M. Sadjadi, y A. Iosup, «An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds», en *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, 2012, pp. 612–619.
[285] «Google Cloud Computing, Hosting Services & APIs», *Google Cloud Platform*. [En línea]. Disponible en: https://cloud.google.com/. [Accedido: 03-ene-2017].
[286] R. G. Diaz, A. C. Ramo, A. C. Agüero, T. Fifield, y M. Sevior, «Belle-DIRAC Setup for Using Amazon Elastic Compute Cloud», *J. Grid Comput.*, vol. 9, n.º 1, pp. 65-79, ene. 2011.
[287] P. Mhashilkar, A. Tiradani, B. Holzman, K. Larson, I. Sfiligoi, y M. Rynge, «Cloud Bursting with GlideinWMS: Means to satisfy ever increasing computing needs for Scientific Workflows», *J. Phys. Conf. Ser.*, vol. 513, n.º 3, p. 032069, 2014.
[288] S. J. E. Taylor, A. Anagnostou, T. Kiss, G. Terstyanszky, P. Kacsuk, y N. Fantini, «A tutorial on Cloud computing for Agent-based Modeling amp; Simulation with Repast», en *Proceedings of the Winter Simulation Conference 2014*, 2014, pp. 192-206.
[289] S. Gugnani, C. Blanco, T. Kiss, y G. Terstyanszky, «Extending Science Gateway Frameworks to Support Big Data Applications in the Cloud», *J. Grid Comput.*, vol. 14, n.º 4, pp. 589-601, dic. 2016.
[290] G. Terstyanszky *et al.*, «Validating Scanned Foot Images and Designing Customized Insoles on the Cloud», en *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3288–3296.
[291] R. Grzymkowski, T. Hara, y B. I. computing group, «Belle II public and private cloud management in VMDIRAC system.», *J. Phys. Conf. Ser.*, vol. 664, n.º 2, p. 022021, 2015.
[292] «The Grid Universe». [En línea]. Disponible en: http://research.cs.wisc.edu/htcondor/manual/v7.9/5_3Grid_Universe.html. [Accedido: 23-nov-2016].
[293] «2.6 Managing a Job». [En línea]. Disponible en: http://research.cs.wisc.edu/htcondor/manual/v7.7/2_6Managing_Job.html#sec:job-prio. [Accedido: 05-ene-2017].
[294] E. Huedo, R. Montero, y I. Llorente, «Grid Architecture from a Metascheduling Perspective», *Computer*, vol. 43, n.º 7, pp. 51-56, jul. 2010.
[295] E. Huedo, R. S. Montero, y I. M. Llorente, «A recursive architecture for hierarchical grid resource management», *Future Gener. Comput. Syst.*, vol. 25, n.º 4, pp. 401-405, abr. 2009.
[296] S. Yi, D. Kondo, y A. Andrzejak, «Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud», en *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 236-243.
[297] K. Fukushima, «Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position», *Biol. Cybern.*, vol. 36, n.º 4, pp. 193-202, abr. 1980.
[298] Y. LeCun, Y. Bengio, y G. Hinton, «Deep learning», *Nature*, vol. 521, n.º 7553, pp. 436-444, may 2015.
[299] N. A. Grogin *et al.*, «CANDELS: the cosmic assembly near-infrared deep extragalactic legacy survey», *Astrophys. J. Suppl. Ser.*, vol. 197, n.º 2, p. 35, 2011.
[300] D. Tuccillo, M. Huertas-Company, E. Decenciere, y S. Velasco-Forero, «Deep learning

for studies of galaxy morphology», *ArXiv170105917 Astro-Ph*, ene. 2017.
[301] «IBERGRID - Iberian Grid Infrastructure Conference». [En línea]. Disponible en: http://ibergrid.lip.pt/. [Accedido: 03-mar-2017].
[302] E. eu IASA, «EGI Centos 6». [En línea]. Disponible en: https://appdb.egi.eu/store/vappliance/egi.centos.6. [Accedido: 17-mar-2017].
[303] B. T. P. Rowe *et al.*, «GALSIM: The modular galaxy image simulation toolkit», *Astron. Comput.*, vol. 10, pp. 121–150, 2015.
[304] S. van der Walt, S. C. Colbert, y G. Varoquaux, «The NumPy array: a structure for efficient numerical computation», *Comput. Sci. Eng.*, vol. 13, n.º 2, pp. 22–30, 2011.
[305] T. P. Robitaille *et al.*, «Astropy: A community Python package for astronomy», *Astron. Astrophys.*, vol. 558, p. A33, oct. 2013.
[306] «Continuum», *Continuum*. [En línea]. Disponible en: https://www.continuum.io/. [Accedido: 16-mar-2017].
[307] «CETA-Ciemat», *Centro Extremeño de Tecnologías Avanzadas - CETA-Ciemat*. [En línea]. Disponible en: http://www.ceta-ciemat.es/. [Accedido: 14-mar-2017].
[308] A. J. Rubio-Montero, E. Huedo, y R. Mayo-García, «Scheduling multiple virtual environments in cloud federations for distributed calculations», *Future Gener. Comput. Syst.*
[309] N. Huber, M. von Quast, M. Hauck, y S. Kounev, «Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments.», en *CLOSER*, 2011, pp. 563–573.
[310] J. Nandimath, E. Banerjee, A. Patil, P. Kakade, S. Vaidya, y D. Chaturvedi, «Big data analysis using Apache Hadoop», en *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, 2013, pp. 700–703.
[311] R. Ford, G. Riley, R. Budich, y R. Redler, *Earth System Modelling-Volume 5: Tools for Configuring, Building and Running Models*. Springer Science & Business Media, 2012.
[312] T. N. Palmer, «The economic value of ensemble forecasts as a tool for risk assessment: From days to decades», *Q. J. R. Meteorol. Soc.*, vol. 128, n.º 581, pp. 747–774, 2002.
[313] G. Shainer *et al.*, «Weather Research and Forecast (WRF) model performance and profiling analysis on advanced multi-core HPC clusters», *10th LCI Int. High-Perform. Clust. Comput. Boulder CO*, 2009.
[314] C. Arshad Shameem y T. N. Venkatesh, «High resolution simulation and visualizations of atmospheric flow using coupled models on HPC system», en *Proceedings of the 16th AeSI Annual CFD Symposium*, 2014, pp. 11–12.
[315] «Santander Meteorology Group | A multidisciplinary approach for weather & climate». [En línea]. Disponible en: http://meteo.unican.es/en/main. [Accedido: 06-dic-2016].
[316] J. Dean y S. Ghemawat, «MapReduce: Simplified Data Processing on Large Clusters», *Commun ACM*, vol. 51, n.º 1, pp. 107–113, ene. 2008.
[317] L. Li, Z. Ma, L. Liu, y Y. Fan, «Hadoop-based ARIMA Algorithm and its Application in Weather Forecast», *Int. J. Database Theory Appl.*, vol. 6, n.º 5, pp. 119-132.
[318] M. C. Schatz, «CloudBurst: highly sensitive read mapping with MapReduce», *Bioinformatics*, vol. 25, n.º 11, pp. 1363–1369, 2009.
[319] S. Jiao, C. He, Y. Dou, y H. Tang, «Molecular dynamics simulation: Implementation and optimization based on Hadoop», en *2012 Eighth International Conference on Natural Computation (ICNC)*, 2012, pp. 1203-1207.
[320] A. Thusoo *et al.*, «Hive - a petabyte scale data warehouse using Hadoop», en *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, 2010, pp. 996-1005.
[321] R. Kaur y M. Goyal, «A survey on the different text data compression techniques», *Int J*

Adv Res Comput Eng Technol, vol. 2, n.º 2, pp. 711–714, 2013.
[322] J. Seward, «bzip2 and libbzip2», *Avaliable Httpwww Bzip Org*, 1996.
[323] L. P. Deutsch, «GZIP file format specification version 4.3», 1996.
[324] M. Oberhumer, «LZO real-time data compression library», *User Man. LZO Version 028 URL Httpwww Infosys Tuwien Ac AtStaffluxmarcolzo Html Febr. 1997*, 2005.
[325] «Snappy by google». [En línea]. Disponible en: http://google.github.io/snappy/. [Accedido: 15-dic-2016].
[326] R. P. Padhy, «Big data processing with Hadoop-MapReduce in cloud systems», *Int. J. Cloud Comput. Serv. Sci.*, vol. 2, n.º 1, p. 16, 2013.
[327] Y. He *et al.*, «RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems», en *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 1199-1208.
[328] R. Kumar y N. Kumar, «Improved join operations using ORC in HIVE», *CSI Trans. ICT*, pp. 1-7, dic. 2016.
[329] V. K. Vavilapalli *et al.*, «Apache Hadoop YARN: Yet Another Resource Negotiator», en *Proceedings of the 4th Annual Symposium on Cloud Computing*, New York, NY, USA, 2013, p. 5:1–5:16.
[330] F. Tian y K. Chen, «Towards optimal resource provisioning for running mapreduce programs in public clouds», en *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 155–162.
[331] «CESNET | CESNET, zájmové sdružení právnických osob». [En línea]. Disponible en: https://www.cesnet.cz/?lang=en. [Accedido: 10-dic-2016].
[332] «Euro-Argo». [En línea]. Disponible en: http://www.euro-argo.eu/. [Accedido: 10-dic-2016].
[333] Y. Huai *et al.*, «Major Technical Advancements in Apache Hive», en *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2014, pp. 1235–1246.
[334] Z. Qu y G. Chen, «Big data compression processing and verification based on Hive for smart substation», *J. Mod. Power Syst. Clean Energy*, vol. 3, n.º 3, pp. 440-446, sep. 2015.
[335] «EGI-FCTF/fedcloud-userinterface», *GitHub*. [En línea]. Disponible en: https://github.com/EGI-FCTF/fedcloud-userinterface. [Accedido: 19-dic-2016].
[336] M. Turilli, M. Santcroos, y S. Jha, «A Comprehensive Perspective on Pilot-Job Systems», *ArXiv150804180 Cs*, ago. 2015.
[337] A. J. Rubio-Montero, E. Huedo, F. Castejón, y R. Mayo-García, «GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs», *Future Gener. Comput. Syst.*, vol. 45, pp. 25-52, abr. 2015.