USO DE INFRAESTRUCTURAS HÍBRIDAS GRID Y CLOUD PARA LA COMPUTACIÓN CIENTÍFICA

por

Amanda Calatrava Arroyo

Máster en Computación Paralela y Distribuida

2012

Director: Dr. D. Germán Moltó Martínez

Fecha: Febrero 2012

RESUMEN

El auge de las técnicas de virtualización en los últimos años ha propiciado la aparición del Cloud Computing. Esta nueva tecnología ha abierto un camino hacia el empleo de infraestructuras computacionales híbridas en el ámbito científico, basadas en potentes recursos Grid combinados con las infraestructuras virtuales dinámicas y elásticas que proporciona el Cloud. Pero esta combinación de recursos para dar soporte a ejecuciones de aplicaciones científicas intensivas no es trivial, propiciando la aparición de nuevos retos y oportunidades en áreas como la provisión de recursos o la metaplanificación.

En esta tesis de máster, en primer lugar, se han desarrollado modelos teóricos de metaplanificación híbrida Grid/Cloud que permiten la integración y aprovechamiento de ambas infraestructuras por aplicaciones científicas HTC (*High Throughput Computing*) de acuerdo al estado del arte actual. Estos modelos teóricos se han puesto en práctica a través del desarrollo de herramientas que permiten el despliegue y ejecución concurrente de aplicaciones científicas sobre plataformas Grid y Cloud (incluyendo Clouds privados y públicos). En segundo lugar, se ha realizado un estudio de la sobrecarga que supone el proceso de virtualización con respecto a una máquina física. Finalmente, para poder valorar y poner en práctica la efectividad de los modelos, se ha incluido un caso de estudio para una aplicación científica computacionalmente compleja capaz de realizar el proceso de diseño de proteínas de propósito específico.

ABSTRACT

The advent of virtualization techniques in recent years has led to the emergence of Cloud Computing. This new technology has paved the way towards the use of hybrid computing infrastructures in science, based on powerful Grid resources combined with dynamic and elastic virtual infrastructures that provides the Cloud. But this combination of resources to support the execution of computationally intensive scientific applications is not trivial, giving rise to new challenges and opportunities in areas such as the provision of resources or meta-scheduling.

This master's thesis, has first developed theoretical models of hybrid Grid/Cloud meta-scheduling that enable the integration and use of both infrastructures by scientific HTC (*High Throughput Computing*) applications according to the current state of art. These theoretical models have been implemented through the development of prototype implementations that allow the deployment and concurrent execution of scientific applications on Grid and Cloud platforms (including private and public Clouds). Secondly, we have made a study of the overheads of the virtualization process with respect to a physical machine. Finally, it assesses the effectiveness of the models. For that, we have included a case study that involves a computationally intensive scientific application that is able to perform the optimization of proteins with target properties.

AGRADECIMIENTOS

Me gustaría expresar mi más sincero agradecimiento a Germán Moltó, como ya hice en mi proyecto final de carrera, pero esta vez con una mención especial ya que no sé cómo agradecerle que renovase su confianza en mí. Él me ha dado la oportunidad de trabajar a su lado, lo cual ha supuesto para mí una experiencia emocionante y enriquecedora, y de esta colaboración ha surgido el trabajo que se presenta en la presente memoria. Su apoyo durante todo este tiempo ha sido clave para mí. También quiero agradecer a los profesores que he tenido oportunidad de conocer durante el máster, los conocimientos y experiencias que me han transmitido, ya que gracias a ellos he logrado alcanzar mis objetivos satisfactoriamente.

Gracias también a Vicente Hernández, el responsable del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP), y a todos sus componentes por acogerme entre ellos como uno más y proporcionarme todo el soporte necesario para desarrollar mi trabajo. Gracias a todos por hacer tan agradable mi estancia en el laboratorio.

También he de agradecer a la Universidad Politécnica de Valencia la financiación para llevar a cabo el proyecto PAID-06-09-2810 "Ejecución Escalable de Aplicaciones Científicas en Infraestructuras Virtualizadas Multicapa Mediante Tecnologías Grid y Cloud", sobre el cual se enmarca esta tesis de máster.

Y no podía acabar este apartado sin agradecer a mis familiares y amigos, y más concretamente a mis padres, el apoyo recibido durante toda mi carrera, y en especial en este último año, en el que he aprendido mucho. Y también he de citar a mi abuela, por prepararme esas comidas que me daban la energía suficiente para afrontar las clases hasta tan tarde.

Índice general

LISTA DE TABLAS

Número *Página*

LISTA DE FIGURAS

Número *Página*

1. INTRODUCCIÓN

En esta primera sección de la memoria se pretende situar al lector en el contexto del trabajo realizado, describiendo los motivos que han llevado a realizar este proyecto y los objetivos a alcanzar. Para cerrar este capítulo, se presenta la estructura que seguirá el presente documento.

1.1. MOTIVACIÓN

La computación en Grid ha propiciado en la última década el despliegue de infraestructuras distribuidas de gran escala entre diferentes instituciones de investigación, situadas en cualquier parte del mundo, permitiendo la compartición de recursos con el objetivo de abordar problemas de gran dimensión. Un ejemplo de ello es WLCG [1] (*Worldwide LHC Computing Grid*), un gran Grid de recursos distribuido entre 35 países y 140 centros de computación que permite manejar la enorme cantidad de datos que genera el LHC (*Large Hadron Collider*, Gran Colisionador de Hadrones).

Este tipo de computación permite utilizar todo tipo de recursos, principalmente de cómputo y de almacenamiento, que no están sujetos a un control centralizado. Así, permite desplegar infraestructuras distribuidas que persiguen el uso colectivo de los recursos heterogéneos que la componen, administrados por diferentes instituciones.

Pero también es cierto que, si bien son muchos los beneficios de este tipo de infraestructuras, también cuenta con inconvenientes significativos que muchas veces puede llevar a los científicos a cuestionar el uso de las infraestructuras Grid. Actualmente, el acceso a este tipo de recursos requiere la aprobación previa de comités científicos que evalúan el impacto de las aplicaciones científicas que quieren

llevar a cabo ejecuciones en el Grid. Esto supone un retraso y puede ser un problema para investigadores que necesiten acceso inmediato a los recursos. Otros inconvenientes que presentan las infraestructuras Grid son la falta de versatilidad en los recursos o la administración remota de los mismos ya que, en el Grid, son los proveedores de recursos los que determinan el entorno de ejecución de las aplicaciones. Ello conlleva a los científicos la necesidad de contar con un conocimiento importante sobre la arquitectura del recurso y el middleware Grid para poder adaptar con éxito el proceso de compilación y/o ejecución de sus aplicaciones en los recursos remotos.

Figura 1.1 – El Grid del WLCG distribuido en Europa.

El auge en las técnicas de virtualización durante estos últimos años ha propulsado la aparición del Cloud Computing, una tecnología que ofrece el acceso bajo demanda y a través de Internet a un conjunto de recursos virtualizados configurables, y que cuyo aprovisionamiento y repliegue se realiza sin mayor esfuerzo del usuario. Todo ello ha propiciado el surgimiento de nuevas ideas que pretenden construir infraestructuras híbridas computacionales basadas en potentes recursos Grid combinados con infraestructuras virtuales bajo demanda, dinámicas y elásticas, que proporcionan los despliegues Cloud. La combinación de ambos tipos de recursos puede dar lugar a la aparición de un nuevo un entorno completo y robusto, donde se

aprovechen los beneficios de ambas infraestructuras y se atenúen los inconvenientes concretos de cada una.

Sin embargo, la combinación de ambos tipos de recursos para la ejecución de aplicaciones científicas computacionalmente intensas no es una tarea trivial, abriendo el camino a nuevos retos y oportunidades en áreas como la provisión y gestión de recursos o la metaplanificación, donde es necesario determinar qué trabajos obtendrán mejores prestaciones en qué infraestructuras para poder reducir el tiempo de obtención de la solución, minimizar costes (en lo que a asignación de recursos se refiere) o manejar situaciones inesperadas.

En este contexto surge la presente tesis de máster, que se enmarca dentro del proyecto PAID-06-09-2810 "Ejecución Escalable de Aplicaciones Científicas en Infraestructuras Virtualizadas Multicapa Mediante Tecnologías Grid y Cloud", y que persigue la abstracción de la ejecución de aplicaciones científicas de forma simultánea tanto sobre despliegues computacionales Grid como Cloud.

1.2. OBJETIVOS

El principal objetivo que se plantea en esta tesis de máster consiste en diseñar modelos teóricos de integración de infraestructuras Grid y Cloud, en los que se pongan de manifiesto las ventajas que proporciona el uso combinado de ambos tipos de entornos para ejecuciones en el ámbito científico. Ello conllevará, en primer lugar, la realización previa de un estudio del estado del arte en este contexto que ponga de manifiesto el trabajo realizado en este campo hasta la fecha y las necesidades que quedan por cubrir.

Tras analizar los avances en el campo en el que se enmarca el presente trabajo, será necesario seleccionar las estrategias de metaplanificación híbrida a emplear. La metaplanificación sobre infraestructuras Grid típicamente involucra modelos de prestaciones que persiguen la reducción del tiempo de ejecución de las aplicaciones

mediante el aprovechamiento de los recursos disponibles, que se suelen asumir como ya disponibles al comienzo del proceso de asignación de tareas. Por el contrario, el uso de plataformas Cloud precisa el despliegue de la infraestructura virtual previa a la ejecución de las tareas. Esto requiere decidir el número apropiado de máquinas virtuales a desplegar considerando criterios adicionales como el coste económico de los recursos o el presupuesto disponible por el usuario. Además, la elasticidad es un factor clave a tener en cuenta en este tipo de infraestructuras, puesto que dependiendo de la carga de trabajo del metaplanificador, debería ser posible adaptar la infraestructura (desplegar y replegar nuevos recursos) para satisfacer la demanda de cómputo.

Posteriormente se abordará la implementación de un conjunto de herramientas que, haciendo uso de los modelos teóricos previamente desarrollados, permitan la ejecución de aplicaciones científicas sobre ambas infraestructuras. Estas herramientas servirán como base para la ejecución de un caso de estudio que demuestre los beneficios de los modelos propuestos y la efectividad de los mismos y se desarrollarán sobre el código del ya existente metaplanificador Grid GMarte [49]. Para ampliar el campo de acción de la herramienta implementada, se propone el uso de una infraestructura Cloud pública, como es Amazon EC2 [18] (*Elastic Compute Cloud*, Computación Cloud Elástica), utilizando así los dos modelos de servicio Cloud más importantes, el privado y el público.

También se plantea la realización de un estudio de las sobrecargas que supone el proceso de virtualización sobre una máquina física. Este estudio ayudará a analizar las sobrecargas de la virtualización en tiempo de ejecución de las aplicaciones para entender qué tipo de trabajos será más conveniente para cada tipo de infraestructura, ya que el uso de capas de virtualización introduce sobrecargas en el proceso de cómputo, tanto para aplicaciones limitadas por CPU (*Central Processing Unit*, Unidad Central de Procesamiento) como aquellas limitadas por E/S (*Input/Output*, Entrada/Salida).

Finalmente, el desarrollo de un caso de estudio empleando una aplicación científica computacionalmente completa, servirá para poner de manifiesto la efectividad de los modelos desarrollados y su implementación.

1.3. ESTRUCTURA DE LA MEMORIA

El presente documento se encuentra estructurado mediante capítulos que analizan todos los aspectos del trabajo realizado. En el capítulo 1 se realiza la introducción a la tesis de máster, que abarca aspectos como la motivación y objetivos que han llevado a la realización de este trabajo. A continuación se encuentra el capítulo 2, que contiene además de los conceptos más relevantes para situar al lector en el contexto adecuado, las tecnologías actuales empleadas en la realización de este trabajo.

Seguido al anterior se encuentra el capítulo 3, que recoge el principal trabajo en el que se centra esta tesis de máster. El primer subcapítulo analiza el estado del arte en el contexto de la planificación y ejecución híbrida de trabajos científicos sobre infraestructuras Grid/Cloud. Tras presentar la visión general donde se va a enmarcar el estudio de los modelos teóricos en el siguiente apartado, el subcapítulo 4.3 repasa los parámetros relevantes a tener en cuenta a la hora de realizar un proceso de planificación de trabajos sobre una infraestructura híbrida, ordenándolos por el entorno al que afectan. Para finalizar el capítulo se presentan los modelos diseñados, analizando uno a uno su comportamiento y los parámetros que afectarán a su comportamiento, de los presentados anteriormente.

El capítulo 4 repasa los principales aspectos de implementación de las herramientas desarrolladas. En primer lugar se expone la arquitectura implementada, sobre la que van a funcionar las herramientas desarrolladas, entrando en detalle en sus principales componentes. Además, se presenta un esquema general de las clases que componen el desarrollo y el flujo de ejecución en un entorno Cloud. También recoge este capítulo el comportamiento de los orquestadores desarrollados, regido por los

modelos previamente expuestos. Finalmente, el último apartado se centra en la conexión con el proveedor Cloud público Amazon EC2, estudiando para ello el estado del arte sobre los principales APIs (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) de agregación que existen actualmente. A continuación se encuentra el capítulo 5, que contiene el estudio realizado sobre la sobrecarga que supone la virtualización de una máquina física para dar soporte a la computación Cloud. Este capítulo está dividido principalmente en dos subcapítulos que tratan por separado los *benchmarks* de CPU y los de E/S. Esto es así, porque los dos parámetros más relevantes que limitan a las aplicaciones son las operaciones de cómputo y las de entrada/salida.

Posteriormente, en el capítulo 6 se recoge un caso de estudio que pone de manifiesto las ventajas del uso de una infraestructura híbrida para las aplicaciones científicas computacionalmente intensas. Para ello, en primer lugar se analiza la infraestructura empleada para la realización del caso de estudio, al igual que se hace con la aplicación seleccionada. El comportamiento de la aplicación bajo los distintos modelos previamente desarrollados se expone a continuación.

Finalmente, en el capítulo 7 se exponen las conclusiones extraídas tras la realización de la tesis de máster, además de las publicaciones en el ámbito científico a las que ha dado lugar. El documento se cierra con un par de apéndices que recogen aspectos adicionales a los tratados en el grueso del mismo.

2. CONCEPTOS INICIALES Y TECNOLOGÍAS

En este apartado de la memoria se analizan, sin profundizar en exceso puesto que no es el objetivo, los conceptos que se han considerado más relevantes para poder situar al lector de forma correcta en el ámbito de la tesis de máster. Además se repasan las principales tecnologías relacionadas con los conceptos tratados que se han empleado en el desarrollo de la tesis.

2.1. GRID COMPUTING

El término Grid Computing empezó a utilizarse a mediados de los 90 para referirse a una infraestructura de computación distribuida concebida con el ánimo de poder dar soporte a la investigación colaborativa [2]. Su principal objetivo se centra en la compartición de recursos para que las colaboraciones científicas puedan llevar a cabo la resolución de grandes retos científicos en campos como la simulación, la biomedicina o la física de altas energías. Estos recursos son principalmente de cómputo (clústers de PCs, supercomputadores, etc.) y de almacenamiento aunque también se incluyen fuentes de información, como bases de datos.

Inicialmente, el concepto de Grid Computing se refería la computación bajo demanda, haciendo el símil con la red eléctrica de consumo, que proporciona servicio a sus clientes sin que éstos sepan de donde proviene ni cómo se genera la electricidad. Pero poco a poco se ha ido alejando de esta primera definición para que, actualmente, el concepto de Grid se refiera a "coordinar recursos que no son sujetos a un control centralizado utilizando protocolos e interfaces estándar, abiertos y de propósito general, para proporcionar una calidad de servicio no trivial" [3].

Además del objetivo señalado en el primer párrafo, una infraestructura Grid persigue la fiabilidad y alta disponibilidad de sus recursos, proporcionada gracias a la redundancia de los mismos, así como gestionar y planificar los recursos eficientemente, para reducir los periodos ociosos de los mismos y proporcionar el mejor soporte a las aplicaciones, y organizar a los usuarios en VOs (*Virtual Organizations*, Organizaciones Virtuales), para facilitar el proceso de autorización cuando los usuarios acceden a la infraestructura Grid.

Existen numerosos proyectos que han desarrollado y empleado infraestructuras Grid, como el recién finalizado EGEE (*Enablig Grids for E-SciencE*, Habilitando Grids para la E-Ciencia) [4], que dio lugar a la construcción de un Grid por toda Europa utilizado hasta el momento para computación científica, EGI (*European Grid Infrastructure*, Infraestructura Grid Europea), o el Grid que se encuentra desplegado en España, conocido como ES-NGI (*Spanish National Grid Initiative*, Iniciativa Grid Nacional) [5] y que forma parte del anterior.

2.1.1. MIDDLEWARES GRID

Un middleware es una capa de abstracción situada entre las aplicaciones distribuidas y el sistema operativo en red, capaz de ocultar la heterogeneidad de éstos últimos. En el ámbito Grid existen distintos middlewares, como UNICORE [6], pero en este documento repasaremos los dos más utilizados: Globus Toolkit (GT) [7] y gLite [8].

2.1.1.1. GLOBUS TOOLKIT

Globus Toolkit es el middleware Grid de código abierto más empleado. Fue desarrollado en Argonne por un equipo liderado por Ian Foster. Actualmente se encuentra por la versión 5, pero ha sido la versión 2 la más utilizada, seguida de la 4

que está orientada a los servicios web empleando la especificación WSRF (*Web Services Resource Framework*).

Globus Toolkit 2 proporciona cuatro servicios esenciales en toda infraestructura Grid mediante sus cuatro componentes principales:

- **Seguridad**: un Grid debe proporcionar mecanismos de autenticación y autorización para asegurar la integridad de datos y recursos. El componente encargado de ello en Globus Toolkit es el GSI (*Grid Security Interface*, Interfaz Grid de Seguridad), que emplea certificados X.509 para realizar la autenticación y VOs para el proceso de autorización.

- **Sistema de información**: El servicio MDS (*Monitoring and Discovery Service*, Servicio de Monitorización y Descubrimiento) de Globus es capaz de recoger dinámicamente el estado de los recursos, su ubicación y sus características, para proporcionar un servicio completo de información de todos los recursos que componen el Grid, servicio indispensable para que la asignación de trabajos a recursos sea lo más eficiente posible. Este servicio está compuesto a su vez de dos componentes: GIIS (*Grid Index Information Service*, Servicio Grid de Índice de Información), basado en LDAP (*Lightweight Directory Access Protocol*, Protocolo Ligero de Acceso a Directorios) y encargado de publicar la información del estado de los recursos; y GRIS (*Grid Resource Information Service*, Servicio Grid de Información de Recursos) que es capaz de obtener la información de estado de cada recurso y delegarla en el GIIS.

- **Gestión de recursos**: La gestión de los recursos en el Grid debe ser lo más transparente posible al usuario. De ello se encarga el componente GRAM (*Grid Resource Allocation Manager*, Gestor Grid de Asignación de Recursos). Proporciona servicios de ejecución remota y monitorización del estado de los trabajos empleando los gestores de recursos locales a cada

recurso Grid (PBS (*Portable Batch System*, Sistema Batch Portable), SGE (*Sun Grid Engine*), etc.).

- **Gestión de datos**: GASS (*Global Access to Secondary Storage*, Acceso Global al Almacenamiento Secundario) es el componente de Globus Toolkit encargado de gestionar los datos almacenados en el Grid definiendo rutas virtuales a los mismos. Los datos son accedidos empleando el protocolo Grid FTP, una versión modificada del original FTP (*File Transfer Protocol*, Protocolo de Transferencia de Ficheros). Este componente está diseñado para que los procesos puedan acceder de forma ubicua a los ficheros independientemente del recurso de cómputo donde se ejecuten.

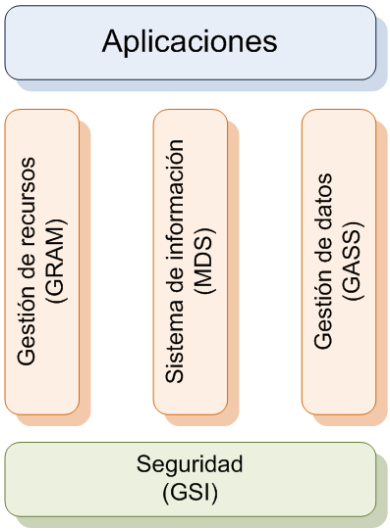

Figura 2.1 – Principales componentes de Globus Toolkit 2.

2.1.1.2. GLITE

gLite es un middleware que soporta la creación de ciberinfraestructuras a gran escala. Este middleware ha sido desarrollado en diferentes lenguajes, como C, C++, Python y Java, entre otros, y por multitud de colaboradores de diferentes países bajo

el proyecto EGEE. Actualmente se encuentra disponible su versión 3.2 y dispone de APIs para usuario en C, C++ y Java.

Sus principales objetivos coinciden con los de cualquier middleware Grid, como son la alta disponibilidad, tolerancia a fallos, robustez, seguridad, altas prestaciones y eficiencia. Al tratarse de un middleware desarrollado bajo un proyecto de investigación, otros objetivos son la gestión de múltiples comunidades de usuarios, la modularidad y el soporte continuo.

Un Grid creado a partir de este middleware está formado por distintos componentes, como se puede observar en la Figura 2.2.

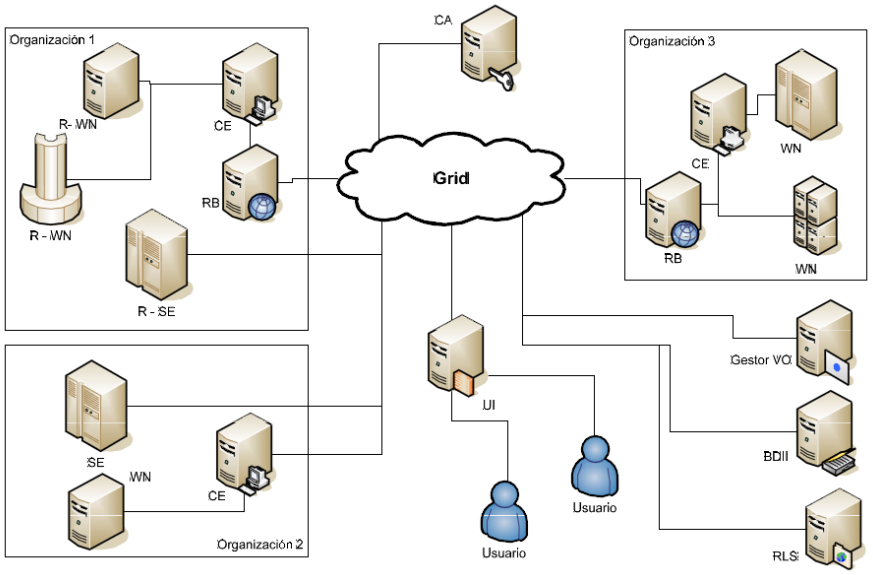

Figura 2.2 – Principales componentes de gLite.

- **Interfaz de usuario (*User Interface,* UI)**: es el elemento que proporciona el punto de entrada al Grid para los usuarios. Ofrece los comandos necesarios para interactuar con gLite y poder lanzar trabajos, monitorizarlos, almacenar datos, etc.

- **Entidad certificadora (*Certificate Authority*, CA)**: autoridad capaz de firmar los certificados digitales, tanto de usuarios como de máquinas, necesarios para poder autentificarse en el Grid. No tiene por qué estar vinculada al propio Grid.

- **Servicio de información (*Information Service*, IS o *Berkeley Database Information Index*, BDII)**: la información sobre los recursos disponibles en gLite es proporcionada mediante un servidor LDAP que es actualizado mediante procesos externos. Este elemento también contiene las listas de usuarios de las VOs.

- **Catálogo de réplicas (*Replica Catalog*, RC, *LCG File Catalog*, LFC o *Replica Location Service*, RLS)**: este elemento gestiona los datos almacenados en el Grid. Es capaz de ubicar un fichero y sus réplicas, facilitando esta información cuando sea necesaria.

- **Planificador de recursos (*Resource Broker*, RB o *Workload Management System*, WMS)**: este componente es capaz de gestionar los trabajos que se ejecutan en el Grid, seleccionando el mejor CE para ejecutar cada uno de ellos.

- **Elementos de cómputo (*Computing Element*, CE)**: estos elementos son los recursos de computación del Grid. Suelen ser los nodos de los clústers que dan acceso a los nodos de trabajo (*front-ends*).

- **Nodos de trabajo (*Worker Node*, WN)**: los componentes del Grid que se encargan de ejecutar los trabajos de los usuarios son los *worker nodes*. Suelen ser los nodos internos de los clústers.

- **Elementos de almacenamiento (*Storage Element*, SE)**: los recursos de almacenamiento son los encargados de almacenar los datos de los usuarios del Grid (ficheros de entrada para los trabajos, etc.).

2.2. CLOUD COMPUTING

El término Cloud Computing (que en castellano significa "Computación en la nube", donde "nube" es una metáfora del concepto de Internet) hace referencia a un nuevo paradigma informático que surge para ofrecer servicios de cómputo a los usuarios. Al ser un servicio, como lo es la electricidad, los usuarios pueden acceder a él sin la necesidad de tener conocimientos sobre la gestión de los recursos que usan y sin tener que preocuparse por el proveedor del servicio para disfrutar de él, es decir, acceden al servicio de forma transparente. Todo ello se consigue a partir de la virtualización de los recursos físicos, tecnología que constituye la base del Cloud Computing.

Las infraestructuras de tipo Cloud se apoyan en Internet para ofrecer su servicio, aprovechándose de las ventajas que ofrece esta red de redes. Los proveedores pueden ofrecer un gran número de servicios de forma rápida y eficiente, y los usuarios acceder a ellos de forma cómoda y segura.

Esta es una tecnología centrada en ofrecer cómputo bajo demanda como cualquier otro servicio. En cambio, a diferencia del Grid, Cloud Computing no persigue ni la investigación colaborativa ni la integración de recursos en dominios administrativos ajenos ni la resolución de grandes retos científicos, sino que se centra en lograr un uso más eficiente, económico y escalable de los recursos disponibles por un proveedor.

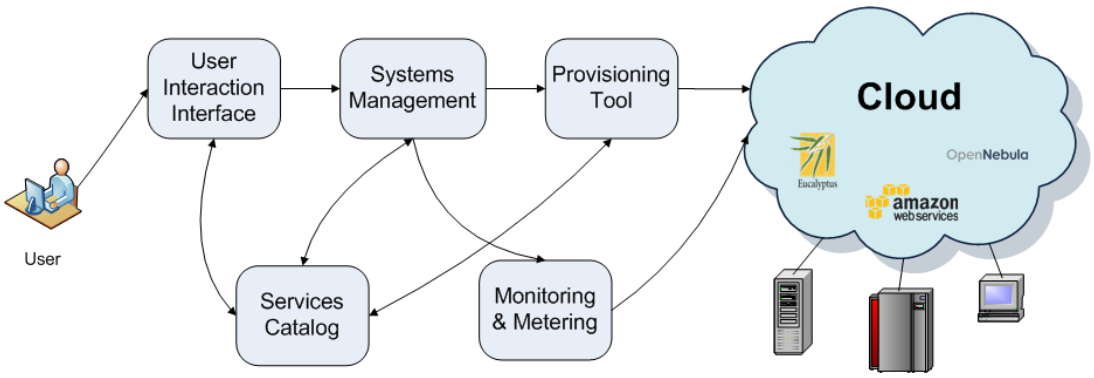

Figura 2.3 – Esquema de funcionamiento de una nube.

Según la definición del NIST [10] (*National Institute of Standards and Thecnology*, Instituto Nacional de Estándares y Tecnología), el término Cloud Computing se puede interpretar como "un modelo para permitir un acceso ubicuo, conveniente, bajo demanda y a través de la red, a un conjunto compartido de recursos informáticos configurables (como redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con el mínimo esfuerzo de gestión o de interacción con el proveedor del servicio".

Una de las características más relevantes de este tipo de computación es su elasticidad. Los entornos Cloud son escalables, es decir, capaces de ajustarse a la demanda de los usuarios. Ante un pico elevado de requisitos de cómputo, un usuario puede solicitar mayor capacidad de cálculo, que provocará el despliegue automático de nuevas VMs (*Virtual Machine*, Máquina Virtual) sobre la infraestructura física existente. Por el contrario, si lo que se detecta es una bajada de la demanda de cómputo, el usuario puede decidir prescindir de parte de su infraestructura, pudiéndola ajustar a su uso de forma dinámica. Esta es una gran ventaja, puesto que uno de los problemas que existían hasta la aparición del Cloud Computing era la creación de una infraestructura ad hoc preparada para soportar demandas no previsibles de cómputo, ya que esto suponía una elevada inversión y además no se podía asegurar que fuese capaz de responder correctamente a estos incrementos repentinos en la demanda de los recursos de cómputo.

Además de esta característica, en el mismo documento donde se define el término tratado en este apartado, el NIST cita otros cuatro rasgos que caracterizan a este tipo de tecnología:

- **Servicio automático bajo demanda**: un usuario puede proveerse los recursos necesarios de forma unilateral y automática sin la necesidad de interaccionar con personal del proveedor del servicio.

- **Amplio acceso a través de la red**: las capacidades están disponibles a través de la red y pueden ser accedidas desde cualquier tipo de dispositivo

conectado a Internet. Esto supone mínimos requerimientos por parte del cliente.

- **Agrupación de recursos**: el proveedor de servicio asigna dinámicamente el acceso a los servicios a diferentes usuarios, proporcionando transparencia de ubicación, aunque el usuario tiene la posibilidad de determinar la localización en un nivel mayor de abstracción.

- **Servicio medido**: los recursos utilizados se contabilizan de forma independiente y precisa para poder implementar un método de pago por uso. El uso de los recursos puede ser monitorizado y controlado, proporcionando transparencia entre el proveedor del servicio y el usuario sobre el gasto de los mismos.

La arquitectura (o modelos de servicio según el NIST) que presenta este paradigma de computación está dividida en tres capas fundamentales, como podemos apreciar en la Figura 1.3.

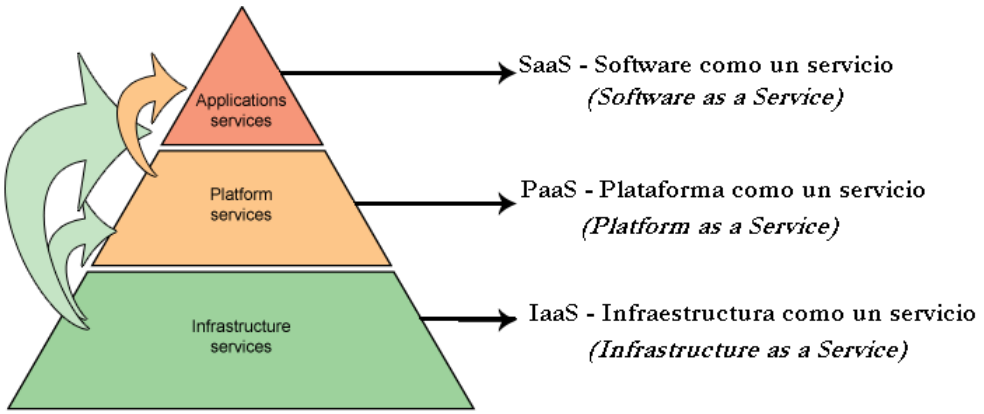

Figura 2.4 – Arquitectura del Cloud Computing.

La capa superior, que se corresponde con el nombre de SaaS (*Software as a Sservice*, Software como un Servicio) es la que aloja a las aplicaciones que se ejecutan en la nube y se ofrecen bajo demanda a modo de servicio a los usuarios. Estas aplicaciones

son escalables y evitan a los usuarios tener que instalar y mantener el software, ya que pueden acceder a él a través de Internet. La empresa distribuidora aporta el servicio de mantenimiento y soporte del software solicitado por el cliente y sus beneficios se basan en el concepto de pago por el uso. Un ejemplo de ello pueden ser las variadas aplicaciones que ofrece la empresa Google, como GMail [12] o Google Sites [13], todas ellas reunidas en un conjunto de aplicaciones que recibe el nombre de Google Apps [14].

La capa intermedia, que recibe el nombre de PaaS (*Platform as a Service*, Plataforma como un Servicio) se encarga de proporcionar la infraestructura y el entorno de ejecución necesario a las aplicaciones de la capa superior como un servicio más. En otras palabras, esta capa permite el desarrollo de aplicaciones Cloud evitándole al usuario tener que preocuparse de la gestión y administración de la infraestructura. Por ello, esta capa se orienta a los desarrolladores de aplicaciones y suele ofrecer APIs de programación para que los desarrolladores gestionen la programación de sus aplicaciones en la nube. Suele estar virtualizada, para poder garantizar la escalabilidad. Un ejemplo práctico puede ser Google App Engine [15], o Windows Azure [16].

Por último se encuentra la capa IaaS (*Infrastructure as a Service*, Infraestructura como un Servicio) que se corresponde con la capa inferior de la arquitectura. En ella se ubican los recursos físicos, tales como servidores, bases de datos, o discos de almacenamiento que se ofrecen como un servicio al usuario. También suelen utilizar técnicas de virtualización consolidando los servidores para poder albergar un número elevado de máquinas virtuales y conseguir reducir costes gracias a la utilización más eficiente de los recursos. Una muestra de esta capa serían GoGrid [60] o Amazon EC2 [18] aunque, en los últimos meses, este proveedor está dando un salto hacia la capa superior ofreciendo servicios de plataforma como el auto-escalado, la monitorización de recursos, etc.

Para finalizar esta introducción al concepto de Cloud Computing, cabe señalar que existen cuatro modelos de despliegue o tipos de nube:

- **Privada**: cuyo uso queda restringido a una sola institución, dueña de los recursos físicos.

- **Pública**: accesible para todo el mundo y que suele emplear políticas de pago por uso (como Amazon EC2).

- **Colectiva**: compartida entre varias organizaciones.

- **Híbrida**: una combinación de las anteriores.

Esta tesis de máster se centra en el uso de los dos primeros, ya que son los más comunes.

2.2.1. HIPERVISORES

Los hipervisores o VMM (*Virtual Machine Monitor*, Monitor de Máquina Virtual) son herramientas software capaces de comunicarse con el sistema anfitrión, ya sea hardware o un SO (*Operating System*, sistema operativo), para interoperar entre un sistema de virtualización y el sistema físico. Conforman una pieza fundamental en la tarea de ejecución de VMs. Su objetivo se basa en lograr la abstracción del hardware que requieren los SOs para permitir ejecutar una o más VMs en un mismo equipo.

Esta plataforma de virtualización se encarga de gestionar los cuatro recursos principales de un computador (CPU, memoria, red y discos de almacenamiento) repartiendo dinámicamente dichos recursos entre todas las VMs definidas en el computador anfitrión.

Existen dos tipos fundamentales de hipervisores, correspondientes con los dos tipos de virtualización existentes (paravirtualización y virtualización completa):

- **Hipervisores de tipo 1 o *Bare-Metal*:** son capaces de ejecutarse directamente sobre el hardware real de la máquina sin necesidad de tener

instalado ningún SO. Este tipo de hipervisores son más eficientes que los de tipo 2, ya que consiguen dar un mayor rendimiento y escalabilidad, además de suponer menos sobrecarga para el computador. La desventaja es que no todo el hardware soporta este tipo de software, pero son los que se suelen emplear normalmente en grandes infraestructuras virtualizadas como es el Cloud Computing. Ejemplos de este tipo de hipervisores son Xen [23], KVM [24] o VMware [25] ESXi, aunque cabe destacar que la clasificación de Xen y KVM es un tema discutido ya que ambos emplean algún tipo de software intermedio entre en hardware y el propio hipervisor, pero realmente no necesitan un SO anfitrión como en el siguiente tipo de hipervisores.

- **Hipervisores de tipo 2 o *Hosted:*** necesitan un SO anfitrión para poder ejecutarse. Permiten la creación de VMs dentro del mismo SO. Al ejecutarse sobre un SO y no directamente sobre el hardware, el rendimiento de este tipo de hipervisores es menor que los de tipo 1. Suelen utilizarse en entornos de escritorio y no en grandes infraestructuras virtualizadas. Algunos de ellos son VirtualBox [27], VMware Player [26] o Virtual PC [28].

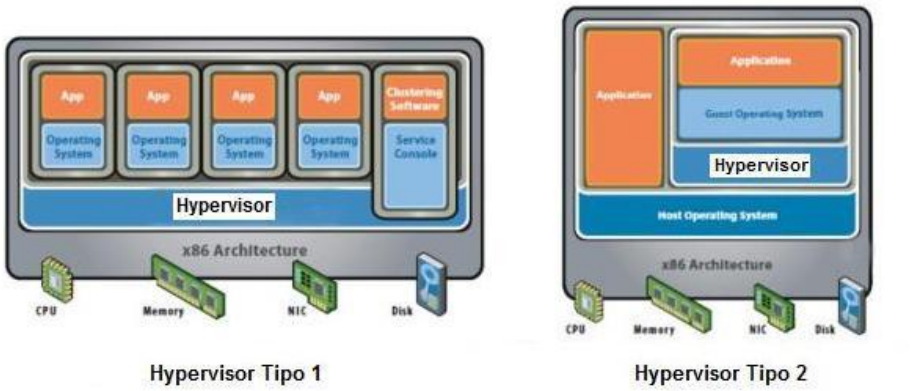

Figura 2.5 – Diferentes tipos de hipervisores.

Como se puede observar, existe un gran número de hipervisores disponibles actualmente. Por ello, con el objetivo de facilitar la tarea de elección entre diferentes hipervisores y portabilidad de las imágenes de las máquinas virtuales, surge el estándar OVF [21] (*Open Virtualization Format*, Formato Abierto de Virtualización), una iniciativa en la que colaboran los grandes desarrolladores de hipervisores, como VMware o Xen.

El estándar OVF persigue la construcción de un formato abierto, extensible, independiente de la plataforma y eficiente para el empaquetamiento y distribución de máquinas virtuales. Así, una máquina virtual que cumpla con este estándar podrá ser desplegada en cualquier hipervisor que de soporte a OVF.

Un paquete OVF contiene los siguientes ficheros:

- Un descriptor OVF, con extensión *.ovf*, que contiene todos los metadatos necesarios para describir el paquete, incluyendo el hardware virtual y el sistema operativo. También contiene términos de licencia de la VMI (*Virtual Machine Image*, Imagen de Máquina Virtual).

- Un fichero *manifest* OVF con el mismo nombre y extensión *.mf*, que contiene los SHA-1 digest de los ficheros individuales del paquete.

- Un conjunto de imágenes de disco.

- Un conjunto de recursos adicionales, como imágenes ISO.

- Un certificado del paquete, elemento opcional.

2.2.1.1. VMWARE

El hipervisor VMware es un software capaz de virtualizar los recursos de una máquina física permitiendo lanzar sobre ellos VMs. Esta plataforma comercial de virtualización dispone de numerosas versiones, de entre las que destacan VMware ESX y VMware ESXi. Como se ha comentado anteriormente, este tipo de hipervisor es capaz de ejecutarse directamente sobre el hardware real de la máquina sin necesidad de tener instalado ningún SO, permitiendo obtener un mejor rendimiento para las VMs desplegadas. Para su ejecución, estos hipervisores se apoyan en un sistema Linux basado en Red Hat Entreprise Linux [22] modificado. Anteriormente estas herramientas basaban su código ejecutable en 32 bits, pero actualmente, su versión más reciente conocida como vSphere, basa su código ejecutable en 64 bits por lo que sus requerimientos pasan a ser mayores ofreciendo a su vez un rendimiento superior.

Figura 2.6 – Arquitectura de trabajo de los hipervisores VMware ESX y ESXi.

Otra versión cuyo uso se encuentra bastante extendido es VMware Server. Esta versión, a diferencia de la anterior, se ejecuta sobre un sistema operativo (hipervisor de tipo 2 o *hosted*).

2.2.1.2. KVM

KVM (*Kernel-based Virtual Machine*, Máquina virtual basada en el núcleo) es un hipervisor de código abierto, desarrollado por Qumranet (empresa propiedad de

Redhat desde 2008), empleado para soportar la virtualización en entornos Linux sobre hardware cuya arquitectura sea x86. Está formado por un módulo de núcleo (que recibe el nombre de *kvm.ko*) que proporciona la infraestructura de virtualización de base y un módulo de procesador específico, *kvm intel.ko* o *kvm amd.ko*, dependiente del tipo de procesador físico del que se disponga.

Utilizando KVM, se pueden lanzar múltiples máquinas virtuales que se ejecutan sin modificar las imágenes de Linux o Windows (virtualización completa). Pero también soporta la paravirtualización, es decir, permite a un software *guest* o invitado especialmente modificado, ejecutarse de forma independiente sobre la máquina virtual, permitiendo la ejecución directa de determinadas operaciones sobre el hardware específico. En cualquier caso, cada máquina virtual tiene el hardware virtualizado privado (tarjeta de red, disco, tarjeta gráfica, etc.). También proporciona migración de máquinas virtuales en caliente, es decir, permite que las VMs puedan ser migradas entre equipos físicos sin pararlas.

El componente KVM está incluido en el kernel de Linux desde la versión 2.6.20, aunque es un software que está en continuo desarrollo. Actualmente se trabaja, por ejemplo, en intentar aprovechar al máximo las características de los nuevos procesadores de Intel y AMD.

2.2.1.3. XEN

Xen es una plataforma de virtualización de código abierto, desarrollado por la Universidad de Cambridge, para arquitecturas x86 que soportan la ejecución de múltiples máquinas virtuales con aislamiento de los recursos. Al igual que KVM, este hipervisor también proporciona migración de máquinas virtuales en caliente. Durante este proceso, la memoria de la máquina virtual se copia literalmente al destino sin detener su ejecución. Es necesario realizar una parada muy breve, de alrededor de 60

a 300 ms, para poder realizar la sincronización final antes de que la máquina virtual vuelva a ejecutarse en su destino final.

Xen utiliza la técnica de paravirtualización para alcanzar alto rendimiento obteniendo una sobrecarga en proceso razonablemente reducida. Intel también ha desarrollado la virtualización completa con Xen, para dar soporte a sus extensiones de arquitectura *VT-X Vanderpool*.

Esta herramienta ejecuta un hipervisor sobre la plataforma host (dom0). Sobre esta plataforma, se ejecuta un software especialmente modificado (domU), como se puede observar en la Figura 2.7.

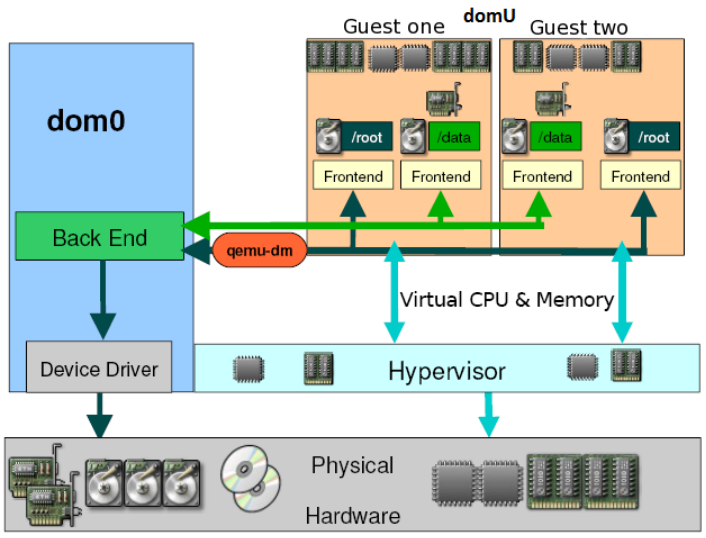

Figura 2.7 – Arquitectura de trabajo del hipervisor Xen.

2.2.2. GESTORES DE MÁQUINAS VIRTUALES

Los GMVs (*Virtual Infrastructure Managers*, Gestores de Máquinas Virtuales) son herramientas software que permiten crear y gestionar las nubes localizadas en la capa de IaaS, ya sean públicas, privadas, colectivas o híbridas. Desacoplan la VM de la

localización física disponible creando una capa de virtualización distribuida que se sitúa entre la infraestructura física existente y el servicio que se presta a los usuarios, como se puede apreciar en la Figura 2.8. Esta capa permite extender los beneficios de las plataformas de virtualización a múltiples recursos y permite gestionar de forma independiente los recursos físicos y los servicios.

La utilización de un GMVs aporta ventajas tales como la gestión centralizada, el escalado y particionamiento de recursos dinámico, monitorización y la provisión bajo demanda de VMs.

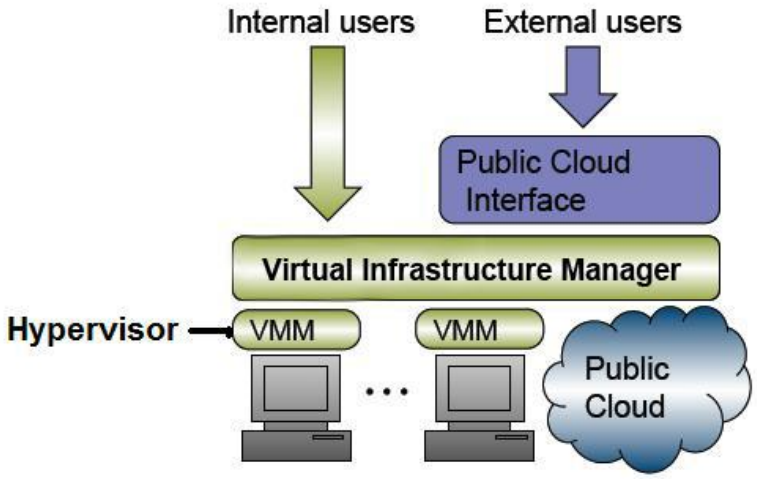

Figura 2.8 – Esquema de localización del GMVs.

Actualmente existe gran variedad de GMVs disponibles en el mercado. Algunos de los más conocidos actualmente son OpenNebula [19], Openstack [29], Eucalyptus [30] o Globus Nimbus [31]. A continuación se analiza el gestor seleccionado para la realización del presente trabajo.

2.2.2.1. OPENNEBULA

OpenNebula es un GMVs de código abierto, desarrollado por el grupo de Arquitectura de Sistemas Distribuidos de la Universidad Complutense de Madrid. Permite crear Clouds privados, públicos e híbridos, y es capaz de gestionar almacenamiento, red y tecnologías de virtualización permitiendo la creación dinámica de VMs sobre infraestructuras físicas.

Este GMVs está diseñado para soportar varios usuarios, pero sólo existe uno que puede llevar a cabo todas las operaciones que OpenNebula ofrece, el propietario de la cuenta *oneadmin*. Éste será el administrador del sistema, el único capaz de llevar a cabo acciones como la gestión del resto de usuarios.

OpenNebula define una VM mediante una plantilla, que incluye datos como la capacidad de memoria y CPU, la red virtual a la que pertenece, el sistema operativo o la imagen en disco. Un ejemplo de plantilla válida se expone en la Figura 2.9.

```
NAME = holaMundo
CPU = 1.0
MEMORY = 512
OS = [ ARCH = "i686" ]
DISK = [ source = "/usr/local/soft/images/myImage", target = "sda", bus = "scsi", driver = "lsilogic" ]
NIC  = [ NETWORK = "publica"]
GRAPHICS = [ type = "vnc", listen = "127.0.0.1" ]
```

Figura 2.9 - Ejemplo de plantilla de un VM para OpenNebula.

Cada VM en OpenNebula se identifica por un número único, el identificador (VID). Además, el usuario puede asignar un nombre a ésta en la plantilla; el nombre predeterminado para cada VM es su identificador. El ciclo de vida de toda VM incluye las siguientes etapas:

- ***Pending*:** por defecto una VM se inicia en este estado, en espera de un recurso que ejecutar.

- ***Hold:*** el usuario ha declarado la VM y no será puesta en marcha hasta que se libere.

- ***Prolog:*** el sistema transfiere los archivos de la VM (imágenes de disco y el archivo de recuperación).

- ***Running:*** en este estado, la VM se está ejecutando.

- ***Migrate:*** la VM está migrando de un recurso a otro. Puede realizarse de dos formas: *livemigration* (la VM se transfiere a otro recurso sin que se aprecie tiempo de inactividad) o la migración en frío, donde se guarda la actual VM y los archivos transfieren al nuevo recurso.

- ***Epilog:*** en esta fase se eliminan los archivos de la máquina utilizada para alojar la VM y se realiza una copia de las imágenes de disco que hay que guardar.

- ***Stopped:*** la VM se detiene. Su estado se ha guardado y ha sido transferido de vuelta junto con las imágenes de disco.

- ***Suspended:*** estado similar al anterior, pero los archivos permanecen en el recurso remoto para reiniciar más tarde la VM.

- ***Failed:*** la máquina virtual falló como consecuencia de algún error.

- ***Unknown:*** la VM no pudo ser localizada, puesto que se encuentra en un estado desconocido.

- ***Done:*** la máquina virtual finaliza. Las VMs en este estado no serán mostradas en la lista de VMs creadas, pero se mantienen en la base de datos a efectos contables.

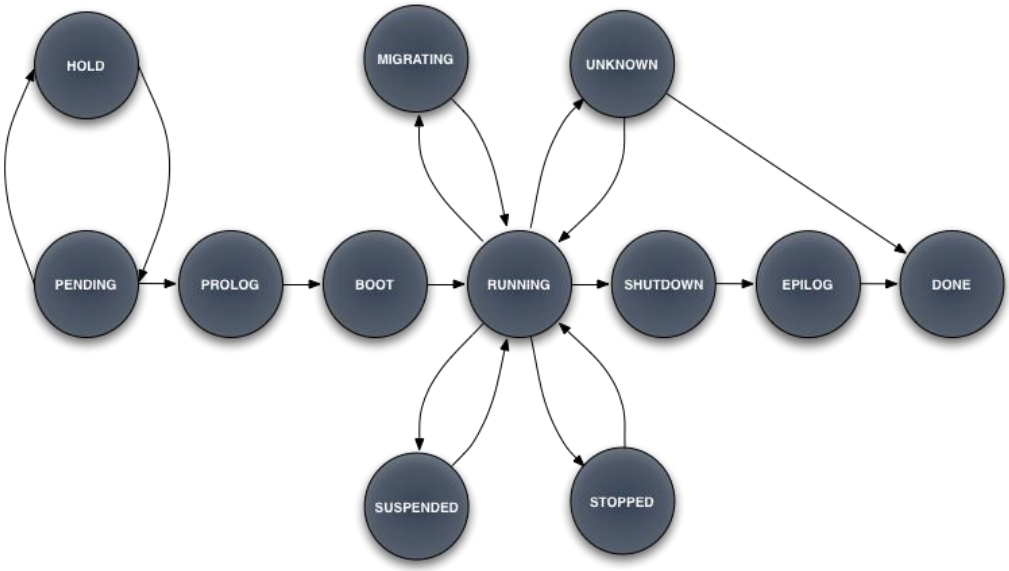

Figura 2.10 – Ciclo de vida de una VM.

La instalación típica de este GMVs se realiza sobre un clúster de PCs de manera que el demonio de OpenNebula se encarga de la distribución y lanzamiento de las VMs sobre los nodos internos del clúster. Estos nodos contienen el hipervisor previamente instalado, que deberá ser KVM, Xen o VMware, los tres hipervisores soportados por OpenNebula. De esta manera, es posible transformar un clúster en una infraestructura virtualizada elástica, que dé soporte a la visión de Cloud computing.

2.2.2.2. AMAZON WEB SERVICES

Amazon Web Services (AWS) es uno de los proveedores de Cloud Computing más representativos en la actualidad, que basa su funcionamiento en una serie de servicios ofrecidos a través de la web que son accesibles con facilidad por los usuarios. Dentro de este conjunto de servicios web de distinta naturaleza destacan Amazon EC2 [18] y Amazon S3 (*Simple Storage Service*, Servicio Simple de

Almacenamiento) [91] de entre otros servicios, como Amazon ElasticBlock Storage [92] o Amazon SimpleDB [93].

Amazon EC2 es un servicio computacional que implementa la modalidad de pago por uso para proporcionar capacidad de cómputo redimensionable. El servicio se encuentra dividido en regiones, correspondientes con la localización geográfica de los distintos centros de proceso de datos que Amazon pone a disposición de sus usuarios. Se ofrecen distintos tipos de instancias o máquinas virtuales, donde cada una de ellas tiene sus características preconfiguradas, tal y como se puede observar en la Tabla 2.1.

Name	Memory	Compute Units *	Storage	Platform	I/O Perf	Linux cost
Standard Small	1.7 GB	1 (1 core x 1 unit)	160 GB	32-bit	Moderate	$0.085 per hour
Standard Large	7.5 GB	4 (2 cores x 2 units)	850 GB	64-bit	High	$0.34 per hour
Standard Extra Large	15 GB	8 (4 cores x 2 units)	1690 GB	64-bit	High	$0.68 per hour
Micro	0.6 GB	2 (only for short bursts)	EBS only	32/64-bit	Low	$0.02 per hour
High-Memory Extra Large	17.1 GB	6.5 (2 cores x 3.25 units)	420 GB	64-bit	Moderate	$0.50 per hour
High-Memory Double Extra Large	34.2 GB	13 (4 cores x 3.25 units)	850 GB	64-bit	High	$1.00 per hour
High-Memory Quadruple Extra Large	68.4 GB	26 (8 cores x 3.25 units)	1690 GB	64-bit	High	$2.00 per hour
High-CPU Medium	1.7 GB	5 (2 cores x 2.5 units)	350 GB	32-bit	Moderate	$0.17 per hour
High-CPU Extra Large	7 GB	20 (8 cores x 2.5 units)	1690 GB	64-bit	High	$0.68 per hour
Cluster Compute Quadruple Extra Large	23 GB	33.5 (2 x Intel Xeon X5570)	1690 GB	64-bit	Very High	$1.60 per hour
Cluster GPU Quadruple Extra Large	22 GB	33.5 (2 x Intel Xeon X5570)	1690 GB	64-bit	Very High	$2.10 per hour

Tabla 2.1 – Tipos de instancias en Amazon EC2. (*) Amazon mide en n procesadores de m EC2 Cores. 1 EC2 core≈ Xeon1-1.2 GHz. (Precios consultados a fecha de 12/12/2011)

Para poder acceder al servicio de cómputo, es imprescindible utilizarlo en conjunción con S3, puesto que para leer las imágenes de las máquinas virtuales se utiliza este servicio de almacenamiento. Amazon fuerza a que las imágenes de máquina virtual de EC2 se almacenen en S3 para garantizar la fiabilidad. Amazon S3 básicamente consiste en un sistema de almacenamiento para el entorno Cloud, accesible mediante protocolos estándar, como HTTP (*Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto).

Con respecto al almacenamiento de datos de las instancias, es decir, su disco duro, cuando se ejecutan las instancias de Amazon EC2, el usuario tiene la opción de almacenar los datos del dispositivo raíz en Amazon EBS o en el almacén de instancias local, asociado a la instancia y que desaparece cuando ésta deja de estar activa. Si utiliza Amazon EBS, los datos en el dispositivo raíz persistirán independientemente del ciclo de vida de la instancia. Esto permite detener y reiniciar la instancia posteriormente, de forma parecida a concluir un portátil y reiniciarlo cuando sea necesario, asegurando que los datos permanecen en la propia instancia al igual que lo hacen en el disco duro del portátil. De manera alternativa, el almacén de instancias local sólo persiste durante la vida de la instancia. Esta es una forma poco costosa de ejecutar instancias, en la que los datos no se almacenan en el dispositivo raíz, pero que dejan de ser persistentes al detener la instancia.

En el Apéndice B de este mismo documento se puede encontrar una pequeña guía en la que se exponen los primeros pasos para el empleo de Amazon EC2 y cómo configurar AMIs para poder posteriormente emplearlas.

3. LA METAPLANIFICACIÓN HÍBRIDA

En este capítulo se realiza un estudio teórico sobre los parámetros y modelos de uso que surgen al emplear las infraestructuras Grid y Cloud conjuntamente. Para ello, en primer lugar, se presenta un estudio del estado del arte en metaplanificación híbrida, poniendo de manifiesto el trabajo realizado hasta la fecha en este campo y los principales campos donde actuar. En segundo lugar, tras repasar la visión general de la arquitectura donde se enmarca el desarrollo del metaplanificador, se detallan todos los parámetros que se consideran necesarios a tener en cuenta para poder construir un metaplanificador que sea capaz de gestionar recursos pertenecientes a las dos infraestructuras, y decidir el recurso más conveniente a la hora de ejecutar una aplicación científica en este entorno híbrido. Finalmente, se exponen distintos modelos teóricos de planificación en los que se tienen en cuenta los parámetros expuestos anteriormente y donde se repasa la importancia de cada uno de ellos para un modelo concreto.

3.1. ESTADO DEL ARTE

Si se analiza la literatura reciente, se pueden encontrar diferentes trabajos previos que animan a combinar el uso de las infraestructuras Grid y Cloud para realizar ejecuciones científicas. En este apartado se examinan las principales aportaciones, tanto artículos científicos como proyectos y herramientas funcionales, al campo de la ejecución híbrida Grid/Cloud.

Tomando como referencia los artículos de investigación, encontramos distintas aproximaciones al uso de infraestructuras híbridas, como por ejemplo en [32], donde los autores enfocan su trabajo en la integración de Grids de alto rendimiento con

entornos Cloud, determinando diferentes modelos de uso desde el punto de vista de las aplicaciones (aceleración, conservación y resistencia). Para ello, utilizan el software CometCloud [45], un *framework* desarrollado en Java que surge con el propósito de soportar la interacción con Clouds públicos (Amazon EC2), privados (Eucalyptus o Nimbus), Grids (TeraGrid) y centros de datos. Esta herramienta es la empleada en este trabajo para ejecutar aplicaciones en infraestructuras híbridas dinámicas basadas en Grid y Cloud. De los mismos autores que el trabajo anterior, también podemos encontrar en [33] una discusión sobre la conveniencia del uso de una infraestructura según el tipo de aplicación que se desee ejecutar y un estudio algo más exhaustivo de las ventajas de emplear el Cloud como acelerador de las ejecuciones Grid. Algunas de las ideas recogidas en estos trabajos ya empezaban a esbozarse en trabajos anteriores como [34], aunque de forma muy introductoria debido a la inmadurez del Cloud en la fecha de publicación.

En [35] estudian el uso de infraestructuras híbridas para la ejecución de *workflows*, donde los criterios de planificación son seleccionados a través del SLA (*Service Level Agreement*, Acuerdo de Nivel de Servicio) que proporciona el usuario de entre los siguientes: *runtime, execution cost, reliability and network bandwidth*. También debe seleccionar el propio usuario la calidad de servicio que desea, facilitándole la tarea al planificador. Siguiendo con la misma estrategia, en [36] los autores explican cómo negociar con los servicios Grid/Cloud a partir de un SLA. Como prerrequisito para realizar la negociación, es necesario que el usuario construya un documento MND (*Meta-Negociation Document)* donde indique los prerrequisitos y el acuerdo al que se desea llegar.

En otros trabajos de la literatura también se concibe el empleo de conceptos propios del Cloud en infraestructuras Grid, como la virtualización de los recursos Grid, como se recoge en [37] o la ilusión de proporcionar infraestructuras Grid elásticas apoyándose en el uso de proveedores Cloud cuando se detecten picos elevados de demanda de cómputo, propuesta recogida en [38] y más desarrollada por los mismos autores en [39]. También se propone un metaplanificador Grid en [40]

que emplea técnicas de virtualización para facilitar la tarea de planificación de trabajos en esta infraestructura, implementando protocolos propios para soportar la virtualización en los recursos físicos del Grid.

También existen diferentes aproximaciones funcionales como en [41] donde ProActive [42], un middleware desarrollado en Java que proporciona acceso a clusters, Grids y Clouds, es empleado para realizar ejecuciones HPC (*High Performance Computing*, Computación de Alto Rendimiento) de aplicaciones sobre infraestructuras híbridas basadas en recursos Grid y Cloud. SAGA [43] es un API desarrollado en C++ y Python que da sustento al lanzamiento de aplicaciones sobre Grid y Cloud sin tener que variar la aplicación, pero aunque soporta distintos proveedores Cloud como Amazon EC2, Eucalyptus o Nimbus, la decisión de la infraestructura a emplear la debe llevar a cabo el usuario. También podemos encontrar a Swarm [44], un gestor de trabajos que permite la ejecución de las aplicaciones sobre tres infraestructuras distintas (Grid, Windows Server Cluster y Cloud), pero que no tiene en cuenta las sobrecargas típicas del Cloud, como son el despliegue y la contextualización de las VMs, ya que se supone que las máquinas ya están listas para su uso, simplificando así el proceso de planificación.

Para finalizar este apartado, cabe destacar algunos proyectos de investigación que se están llevando a cabo y que pretenden el uso combinado de infraestructuras Grid y Cloud de forma eficiente. Por ejemplo, podemos encontrar el proyecto FutureGrid [46], cuyo objetivo es desarrollar una infraestructura de alto rendimiento que proporcione un banco de pruebas a los científicos para desarrollar y probar sobre infraestructuras paralelas, Grid y Cloud sus aplicaciones científicas. Emplea tecnologías de virtualización para ofrecer una amplia gama de SOs facilitando a los investigadores a identificar la ciberinfraestructura que mejor se adapte a sus necesidades científicas. StratusLab [47] es un proyecto europeo en el que participa Telefónica I+D y la Universidad Complutense de Madrid, entre otros, en el que están desarrollando una distribución Cloud de código abierto que permita tanto a los centros de recursos Grid como no-Grid ofrecer una nube IaaS, mejorando

infraestructuras distribuidas ya existentes como EGI. Por último, también es posible encontrar un proyecto [48] que persigue la integración de las infraestructuras Grid y Cloud. Su estrategia se basa en delegar la ejecución de los trabajos al Grid y si éste se satura, obtienen nuevos recursos empleando las técnicas que proporciona el Cloud (empleando Eucalyptus), virtualizando otros recursos del Grid.

Algunas de las ideas citadas en los trabajos anteriores se han tenido en cuenta a la hora de realizar el trabajo que se presenta a continuación, pero adaptadas a la metaplanificación híbrida para aplicaciones tipo *High Throughput Computing* (HTC) en lugar de *High Performance Computing* (HPC) y puestas en marcha mediante la construcción de un prototipo de implementación. Además, el presente trabajo persigue el uso de herramientas de catalogación de imágenes y la integración con herramientas de contextualización de entornos de ejecución. Todo ello con el objetivo de simplificar la labor del usuario, que únicamente debe centrarse en la definición de trabajos, sin preocuparse por contar con los conocimientos necesarios para poder manejar la infraestructura.

3.2. VISIÓN GENERAL DEL ESCENARIO

En los siguientes subcapítulos se llevan a cabo estudios sobre los parámetros y modelos que pueden ser aplicados a una infraestructura híbrida Grid/Cloud. Por ello, para poder situar correctamente al lector en el entorno sobre el que se van a realizar los mismos, es necesario presentar la arquitectura donde se enmarca el trabajo. Para ello, la Figura 3.1 recoge el escenario simplificado compuesto por tres tipos distintos de infraestructuras (Grid, Cloud privado y Cloud público) y los componentes necesarios para realizar la metaplanificación. Ambas infraestructuras Cloud se corresponden con un modelo de servicio IaaS, que ofrecen al usuario la capacidad de lanzar VMs sobre las máquinas físicas.

Un usuario que desee ejecutar su aplicación HTC en el sistema, como por ejemplo una aplicación multiparamétrica compuesta por un grupo de tareas independientes, deberá definir los requisitos de la misma, empleando el API del metaplanificador. Estos requerimientos, como dependencias de librerías, sistema operativo o características hardware concretas, serán determinantes a la hora de escoger el entorno más apropiado para la ejecución de la aplicación. Todo ello constituirá la entrada al proceso de metaplanificación, capaz de determinar a qué infraestructura de las que se dispone es más conveniente el lanzamiento del trabajo. Para tomar esta decisión, se basará en los parámetros definidos en la siguiente sección, como el tiempo de ejecución de las tareas o el tipo de trabajo.

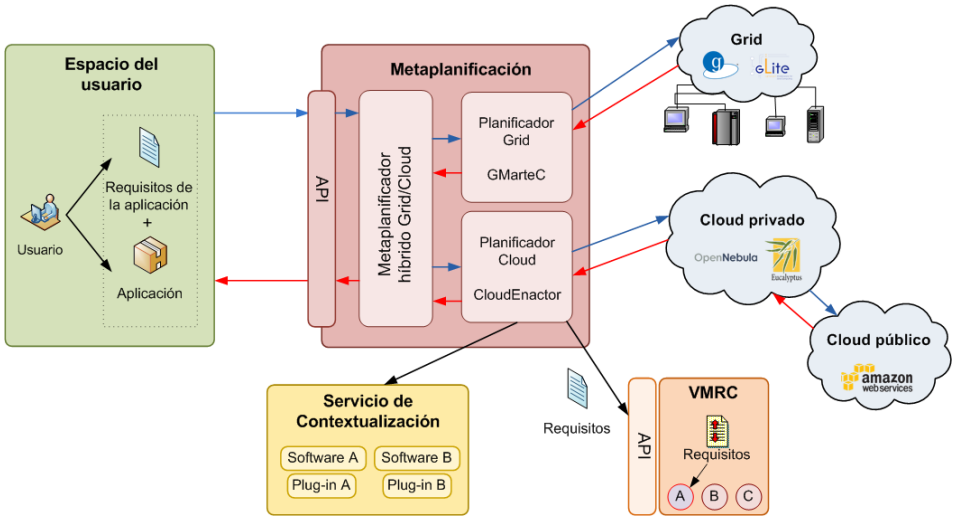

Figura 3.1 – Visión general del escenario híbrido.

Una vez seleccionado el mejor entorno de ejecución para la aplicación, ésta se delegará al planificador concreto de ese entorno. En el caso de una ejecución sobre la infraestructura Grid, el metaplanificador GMarte (*Grid Middleware to Abstract Remote Task Execution,* Middleware Grid para abstraer la ejecución remota de tareas) [49] será el encargado de encontrar el mejor recurso Grid y de ejecutar el trabajo, transfiriendo sus posibles ficheros de entrada, monitorizando el estado del mismo y recuperando los resultados de salida. Este sistema de computación en Grid fue desarrollado para

soportar la ejecución de aplicaciones científicas sobre infraestructuras de computación distribuidas basadas en Globus Toolkit.

Para poder realizar la gestión eficiente de la ejecución de un conjunto de tareas sobre los recursos computacionales disponibles en las organizaciones involucradas en una infraestructura Grid, GMarte proporciona distintas capas de abstracción para ocultar a los usuarios la complejidad de la infraestructura subyacente que, como se comentó en la introducción de esta memoria, resulta uno de los principales problemas para los usuarios del Grid.

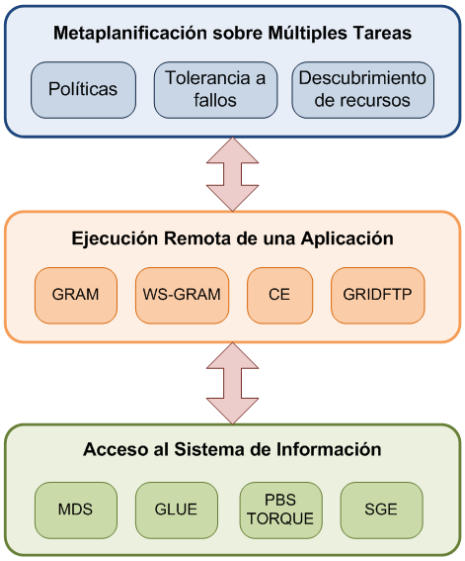

Figura 3.2 – Capas de abstracción proporcionadas por GMarte.

La Figura 3.2 muestra las tres principales capas de abstracción ofrecidas por GMarte. Cada una de ellas proporciona una interfaz de alto nivel para comunicarse con el resto de capas o permitir la realización de componentes que interactúen sólo con una de ellas. El nivel más bajo proporciona un acceso uniforme a los diferentes sistemas de información de los recursos del Grid, que ofrece información sobre el número de procesadores, memoria RAM disponible, etc. La capa intermedia permite la ejecución remota de tareas sobre recursos computacionales, ocultando las diferencias existentes en este contexto debido al empleo de distintos middlewares.

Finalmente, el nivel más alto lo ocupa la capa que ofrece abstracción sobre el proceso de metaplanificación. El principal objetivo de este nivel consiste en permitir al usuario la definición de las tareas a ejecutar, y dejar que el sistema realice de forma transparente todas las acciones necesarias para la ejecución eficiente de las mismas.

En esta tesis de máster no se va a entrar en más detalles sobre este metaplanificador Grid, ya que no forma parte del desarrollo de la misma, además de que existe suficiente material sobre el mismo disponible en [50] y [51].

Si, por el contrario, se considera conveniente realizar la ejecución del trabajo en una infraestructura Cloud, la aplicación la recibirá Cloud Enactor, un componente de orquestación de los servicios involucrados con la infraestructura Cloud. A diferencia del Grid, en un entorno Cloud la planificación se debe llevar a cabo en tres fases principales. En la primera de ellas, la infraestructura creada por las VMs debe ser desplegada. Seguidamente, la segunda fase se encargará de asignar los trabajos a las VMs donde serán ejecutados. Finalmente, la infraestructura de VMs desplegada en la primera fase deberá ser replegada si no existen más trabajos pendientes.

Para ello, el flujo de funcionamiento de Cloud Enactor, que se puede observar en la Figura 3.3, comienza con el cálculo del número apropiado de máquinas virtuales necesarias para realizar la ejecución de las tareas del usuario, una vez este último ha definido y sometido los trabajos a este componente. Seguidamente, Cloud Enactor requiere obtener una VMI que se adapte a los requerimientos de los trabajos. Para ello, adicionalmente en la arquitectura se dispone de un servicio de repositorio y catálogo de imágenes de máquinas virtuales, como VMRC [52] (*Virtual Machine Image Repository & Catalog*), que facilitará la búsqueda y despliegue de la VMI más apropiada para el trabajo del usuario. Este servicio tiene la capacidad de, a partir de unos requerimientos hardware y software dados, devolver la VMI más apropiada a los mismos, haciendo uso de su capacidad de *matchmaking*.

Una vez Cloud Enactor dispone de la VMI más apropiada para los trabajos, es momento de desplegar tantas VMs como sean necesarias. Las políticas de alojamiento

de las VMs sobre la infraestructura física vienen determinadas por el GMVs empleado. Tras desplegar la infraestructura Cloud a emplear, es necesario configurar adecuadamente el entorno donde se ejecutarán las tareas del usuario. Para ello, la arquitectura propuesta cuenta con un servicio de contextualización capaz de adaptar las VMs a los trabajos del usuario, instalando todas las dependencias software necesarias y configurando el entorno de ejecución. Nótese que el servicio del catálogo y repositorio puede incluso reducir los tiempos de contextualización para VMs involucradas en la resolución de problemas multiparamétricos, donde la contextualización de la VM puede realizarse la primera vez y, aprovechando este servicio, puede salvarse la VMI contextualizada en el catálogo para ser empleada de nuevo ahorrando el tiempo de contextualización de la máquina.

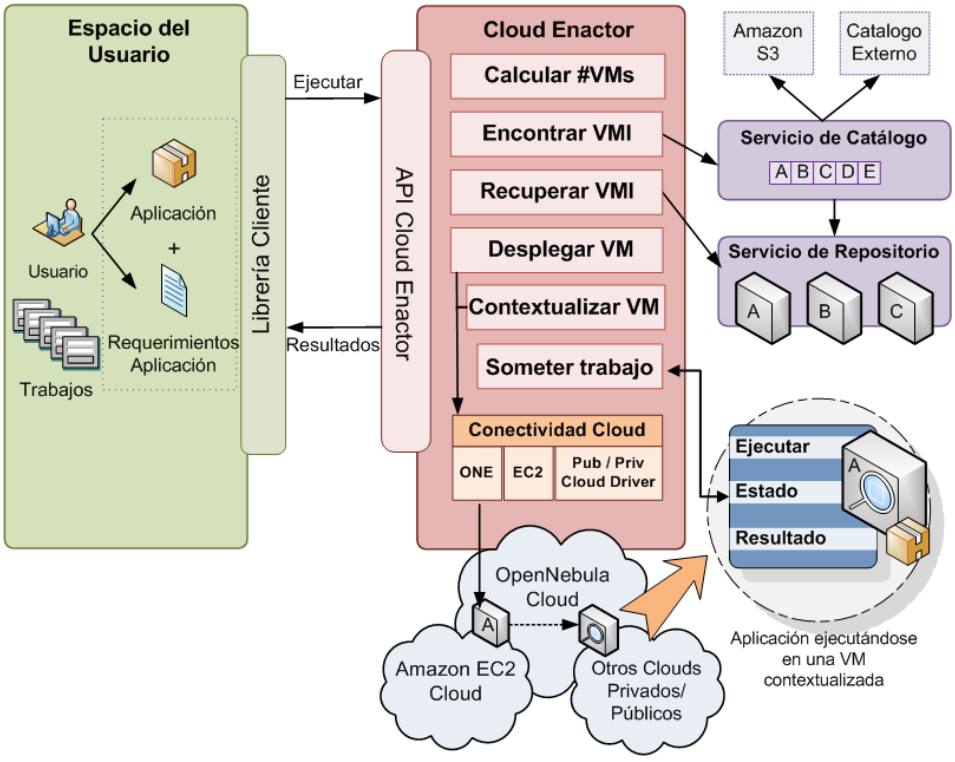

Figura 3.3 – Esquema de funcionamiento de Cloud Enactor.

Una vez la infraestructura está desplegada y configurada correctamente, el proceso de alojamiento de los trabajos da comienzo. Durante la ejecución de los

trabajos se realiza la monitorización del estado de los mismos, hasta que finalizan y los resultados se recuperan para ser devueltos al usuario, dejando las VMs preparadas para albergar nuevas ejecuciones.

Las políticas de despliegue y repliegue de la infraestructura Cloud vendrán definidas a partir de ciertos parámetros, como el número de trabajos pendientes, su duración estimada, o el presupuesto disponible para las ejecuciones, cuando las VMs dependan de un proveedor Cloud como Amazon EC2. Estos parámetros se analizan en la siguiente sección.

En el siguiente capítulo se encuentra detallada la arquitectura implementada, analizando uno a uno sus principales componentes, como el contextualizador o el VMRC.

3.3. PARÁMETROS

En este apartado se analiza cada parámetro considerado de importancia en el proceso de metaplanificación sobre infraestructuras Grid/Cloud. Para definir los parámetros a tener en cuenta a la hora de planificar recursos en una infraestructura híbrida, hay que diferenciar entre los factores que afectarán a ambas infraestructuras y los que son específicos de cada una. Por ello, este apartado se divide en tres categorías, permitiendo clasificar los parámetros dependiendo de la infraestructura a la que afectan.

Un aspecto destacable reside en que no todos los parámetros detallados a continuación podrán ser calculados por el sistema, algunos de ellos deberán ser indicados por parte del usuario como entrada al proceso de metaplanificación.

3.3.1. PARÁMETROS COMUNES

Los parámetros que afectan tanto a una infraestructura Grid como a una infraestructura Cloud y que, por lo tanto, se deben tener siempre en cuenta en tiempo de planificación, son los siguientes:

- **Tiempo estimado de ejecución**: para poder llevar a cabo una buena planificación, es imprescindible conocer a priori el tiempo estimado que consume la aplicación en ser ejecutada. Este parámetro debe ser indicado en primera instancia por el usuario, que podrá ser determinarlo bajo una máquina con unas características concretas, pero que servirá como referencia. Teniendo en cuenta que los trabajos más típicos que se ejecutarán serán trabajos tipo *batch* multiparamétricos, este parámetro podrá ser recalculado dinámicamente permitiendo realizar una planificación más precisa para cada ejecución. El cálculo dinámico se puede llevar a cabo en base a un histórico de ejecuciones actualizado cada vez que se produce una nueva ejecución. Un valor preciso de este parámetro permitirá, por ejemplo, planificar los trabajos más cortos a las máquinas más lentas cuando se disponga de un grupo de recursos heterogéneo.

- **Tiempo de transferencia de ficheros**: es muy común que las aplicaciones requieran ficheros de entrada necesarios para su ejecución, y que generen resultados que habrá que devolver al usuario. Por ello, es necesario considerar el tiempo que se consume en realizar las transferencias de datos entre los recursos que ejecutan la aplicación y la máquina del usuario. Para determinar el valor de este parámetro habrá que tener en cuenta tanto el número y tamaño de ficheros a transferir como el ancho de banda disponible para realizar esa transmisión. El ancho de banda se puede recalcular dinámicamente tras cada transferencia, al igual que pasaba con el tiempo de ejecución, permitiendo dotar al planificador de la capacidad para detectar cambios en el estado de la red. En resumen, este parámetro involucra tanto el

tiempo de transferencia de ficheros de entrada y salida, como posibles resultados parciales y el ancho de banda.

- **Dependencias de la aplicación**: las aplicaciones científicas suelen tener muchas dependencias software, como pueden ser librerías matemáticas, compiladores de lenguajes específicos, paquetes de software determinados, e incluso sistemas operativos poco comunes o versiones específicas de núcleo o *kernel*. Es por ello, que otro parámetro a tener en cuenta en el proceso de planificación son las dependencias de la aplicación, que deberán ser cubiertas, al menos parcialmente, por los recursos donde se vaya a ejecutar la aplicación. Dentro de este parámetro también se tienen en cuenta las dependencias hardware, que suelen ser más fáciles de suplir, pero que son clave para conseguir una ejecución eficiente y exitosa.

- **Tipo de trabajo**: el tipo de aplicación determinará si resulta apropiado un recurso computacional, ya que dependiendo del valor de este parámetro, el trabajo podrá aprovechar mejor las características de una determinada infraestructura. Por ejemplo, una aplicación paralela de grano grueso obtendrá mejores resultados en una máquina física que pueda estar disponible en el Grid que en una máquina virtual, aprovechando la baja latencia de las comunicaciones entre los nodos. En cambio, una aplicación más ligera de cómputo con dependencias concretas, como por ejemplo una aplicación secuencial, se adaptará mejor a una infraestructura Cloud.

3.3.2. PARÁMETROS QUE AFECTAN A UNA INFRAESTRUCTURA GRID

Analizando detalladamente cuáles son los parámetros que determinan exclusivamente la ejecución de un trabajo en una infraestructura Grid, encontramos los siguientes:

- **Tiempo de espera de respuesta del sistema de información:** el planificador necesitará conocer las características de las máquinas que residen en el Grid, para poder determinar cuál es la que mejor se adapta al trabajo. Para ello, es necesario contactar con el sistema de información del middleware Grid para obtener el estado y características de los recursos disponibles. Esto supone una latencia en tiempo de aprovisionamiento de recursos, pero que será paliada con la ejecución de los trabajos científicos que típicamente consumen mucho tiempo de ejecución.

- **Tiempo medio de colas:** este parámetro hace referencia al tiempo que pasan las tareas del usuario en las colas de trabajos locales al recurso seleccionado como apropiado para la ejecución, pendientes de ser ejecutadas. En la práctica, un valor estimado de dicho tiempo lo puede proporcionar el sistema de información del middleware Grid empleado, pero nunca se puede obtener un valor exacto para este parámetro que, en ocasiones, puede afectar de forma notoria a la ejecución de una aplicación en un entorno Grid.

3.3.3. PARÁMETROS QUE AFECTAN A UNA INFRAESTRUCTURA CLOUD

Si se centra la atención en el análisis de los parámetros que condicionan el comportamiento de una aplicación sobre una infraestructura Cloud exclusivamente, encontramos aspectos, como el tiempo de despliegue de la máquina virtual, que carecen de sentido en otro tipo de entorno. Dependiendo del proveedor Cloud empleado, ya sea un GMVs privado como OpenNebula o uno público como

Amazon EC2, algunos de los parámetros siguientes carecerán de sentido. Por ejemplo, el presupuesto del usuario tendrá un rol importante en un entorno Cloud público, pero no en el caso de una infraestructura Cloud privada. Por todo ello, los parámetros concretos de este tipo de infraestructura son los siguientes:

- **Tiempo de acceso al catálogo**: como se ha mencionado anteriormente, se dispone de un catálogo de VMIs que permite seleccionar la imagen más apropiada en base a de los requisitos de la aplicación. Pero el acceso a este servicio y su posterior obtención de los resultados, que puede involucrar la descarga de la imagen, suponen un tiempo que es necesario tener en cuenta a la hora de decidir qué infraestructura es la recomendada. La transferencia de la imagen dependerá de la calidad y el estado de la red en el momento del envío.

- **Tiempo de despliegue de la VM**: otro parámetro concreto de este entorno hace referencia al tiempo consumido por el GMVs para poner en marcha una VM. Este periodo dependerá del gestor de VMs que empleemos (OpenNebula, Eucalyptus, Nimbus,...), además del estado en el que se encuentre la infraestructura donde se van a desplegar las VMs. Es más, si se utilizan técnicas de Green Computing, el despliegue de la VM puede provocar el encendido de los nodos, tiempo que habrá que contabilizar en este parámetro.

- **Tiempo de contextualización**: para conseguir ejecutar con éxito el trabajo del usuario en la máquina virtual, será necesario aplicar un proceso de contextualización basado en la ejecución de scripts para dotarla de los requerimientos software necesarios y preparar el entorno de ejecución. Además, este proceso se encargará de dejar instalada la aplicación para poder ser posteriormente ejecutada. La instalación típicamente consistirá en la descompresión y compilación del trabajo. El valor de este parámetro puede ser elevado si las dependencias de la

aplicación son extensas y costosas de instalar, es por ello que para reducir este parámetro se puede emplear el VMRC para guardar VMIs previamente contextualizadas.

- **Existencia de VMs previamente desplegadas**: un usuario puede disponer de VMs ya desplegadas y contextualizadas, por ejemplo, debido a una ejecución previa. En este caso, la sobrecarga que supone el empleo de la infraestructura Cloud se vería considerablemente reducida.

- **Presupuesto del usuario**: en un escenario en el que se disponga de acceso a Clouds públicos, es necesario que el usuario indique el presupuesto disponible para emplear este tipo de entornos. Este parámetro será determinante a la hora de calcular el número de VMs que se utilizarán y el tiempo que estarán en marcha, ya que no se podrá exceder. El presupuesto no tiene porqué ser solo dinero, sino que un usuario podría disponer de créditos que permitan el acceso a la infraestructura y que van disminuyendo de acuerdo a su uso.

- **Políticas de cobro**: al igual que el parámetro anterior, si surge la necesidad de emplear un Cloud público, el metaplanificador debería tener en cuenta las políticas de cobro de los proveedores. Por ejemplo, si el proveedor Cloud cobra por horas completas, como es el caso de Amazon EC2, el proceso de alojamiento de las tareas en los recursos de cómputo debería refinarse para evitar ejecuciones de 61 minutos, ya que se facturarán dos horas de cómputo completas**.** Alternativamente, este parámetro también puede ser considerado a la hora de apagar las VMs, ya que si queda gran parte de la hora facturada disponible, la VM debería seguir en marcha para aceptar posibles ejecuciones a lo largo de esa hora.

- **Franjas horarias**: el proveedor Cloud puede tener definidas ciertas franjas horarias en las que se apliquen diferentes precios dependiendo de la hora en la que se ejecuten los trabajos en el Cloud. Será interesante que el planificador tenga en cuenta este parámetro para conseguir el máximo rendimiento posible a partir del presupuesto del usuario respetando la calidad del servicio (QoS, *Quality of Service*).

- **Confianza y reputación**: a la hora de decidir qué proveedor Cloud público será más conveniente emplear, se debe tener en cuenta la confianza que ofrecen los mismos, para poder elegir con garantías el mejor proveedor para el trabajo. La confianza puede estimarse por medio de la reputación del proveedor (experiencias históricas basadas en registros de fallos con ese proveedor), el cumplimiento con el SLA (*Service Level Agreement*, Acuerdo de Nivel de Servicio) equivalente a un "contrato", y la calidad de servicio prestada. Este parámetro cobra mayor sentido en entornos de Sky Computing [94], donde se plantea el aprovisionamiento de recursos a diferentes proveedores Cloud.

3.4. MODELOS DE METAPLANIFICACIÓN HÍBRIDA

En este apartado se presentan tres escenarios en los que la planificación híbrida cobra sentido, aportando beneficios potenciales para la ejecución de aplicaciones científicas, y en los que se definen las estrategias de metaplanificación para cada uno de los casos. Para explicar el comportamiento del metaplanificador en los distintos modelos, nos apoyaremos en el esquema que se plantea en la Figura 3.1.

Aunque, en ocasiones, durante la explicación de los modelos se hable del Cloud sin concretar si se trata de un Cloud público o privado, es necesario destacar que los modelos que se describen a continuación abarcan ambos tipos de infraestructuras Cloud, tanto una de ellas como las dos, convirtiendo el proceso de planificación en

una metaplanificación híbrida a tres bandas en el caso de contar con acceso a Grid, Cloud privado y Cloud público.

3.4.1. MODELO 1: COMPORTAMIENTO ANTE ELEVADOS PICOS DE DEMANDA

Como ya se ha mencionado a lo largo de este documento, es bien sabido que un entorno Grid es estático, es decir, ante elevados picos de demanda no puede escalar sus recursos, dejando a los usuarios sin servicio hasta que se liberen de carga de trabajo. También es conocido que un entorno Cloud sí que es capaz de atender a elevados picos de demanda de cómputo, aprovechando una de sus características más importantes, la elasticidad.

Si bien es cierto que una infraestructura Grid suele ser de gran escala, es decir, que cuenta con un número elevado de recursos, sus usuarios típicamente no tienen acceso a todos ellos, quedando restringidos a los recursos a los que tiene acceso la VO a la que pertenecen que, dependiendo de la demanda de los mismo, pueden llegar a saturarse temporalmente. Es por ello que, empleando conjuntamente ambos tipos de infraestructuras, se pueden ampliar las capacidades del Grid, dotándolo de elasticidad para tener un buen comportamiento frente a elevados picos de demanda a través de entornos Cloud. Este es el principal objetivo del modelo 1, que sitúa su funcionamiento en la detección de saturación de las distintas infraestructuras involucradas en la arquitectura empleada, con el objetivo de mantener la calidad de servicio apropiada.

El funcionamiento de este modelo es el siguiente: el proceso de metaplanificación ha determinado que el trabajo debería ejecutarse en la infraestructura Grid, en base a los parámetros analizados en el apartado anterior. Para ello, el trabajo se delega al planificador del entorno seleccionado disponible en el sistema (GMarte en nuestro caso). Cuando el trabajo se intenta lanzar a la infraestructura Grid, el planificador de este entorno detecta que ésta se encuentra

saturada, a partir de la respuesta del sistema de información (dependiente del middleware Grid empleado). Al no poder ejecutar inmediatamente el trabajo, el planificador Grid avisa al metaplanificador de que no puede ejecutar el trabajo debido a la saturación de la infraestructura Grid.

Si no dispusiésemos de acceso a la infraestructura Cloud, el lanzamiento del trabajo se prorrogaría un tiempo determinado y se volvería a intentar lanzar trascurrido ese tiempo, con la esperanza del que la infraestructura Grid se encuentre más liberada de carga de trabajos. En cambio, si existe la posibilidad de delegar la ejecución de trabajos a una infraestructura Cloud, tras recibir el metaplanificador el aviso de que el Grid se encuentra saturado, inmediatamente delega el trabajo en el planificador Cloud, en nuestro caso Cloud Enactor, el cual podrá ejecutar el trabajo desplegando la VMI adecuada que habrá sido seleccionada través del catálogo.

Nótese que, aunque en primera instancia el metaplanificador había decidido que era más conveniente lanzar el trabajo al Grid, tras detectar la saturación de éste, se gana tiempo ejecutándolo en el Cloud, ya que el tiempo que deberíamos estar esperando para poder ejecutar el trabajo en el entorno Grid podía ser indefinido. Una vez se detecte la liberación de recursos en el Grid, se volverán a lanzar los trabajos sobre esta infraestructura.

En las siguientes figuras, se puede observar el comportamiento de este modelo de forma visual gracias a las gráficas que muestran el reparto de tareas entre ambas infraestructuras a lo largo del tiempo. Se ha asumido un esquema de HTC, donde un cierto grupo de trabajos está disponible para su ejecución o existe un flujo de trabajos lo suficientemente extenso para que el metaplanificador pueda actuar sobre ellos. También se asume que los trabajos tendrán una duración muy similar, comportamiento encontrado precisamente en las aplicaciones multiparamétricas (o *parameter sweep*) o en las aplicaciones tipo bolsa de tareas (*bag of tasks*), a su vez, aplicaciones muy comunes en los casos de estudio y ejecuciones científicas.

Por un lado, en la Figura 3.4 se asumen condiciones ideales donde no existen sobrecargas cuando se interactúa con una infraestructura y el proceso de metaplanificación es perfecto, distribuyendo la carga de trabajo sobre ambas infraestructuras disponibles, y no consumiendo tiempo. El alojamiento de los trabajos empieza en el instante 0 (separado del eje de coordenadas para que la gráfica sea más clara). Bajo estas condiciones ideales, la única acción que consume tiempo es la ejecución de los trabajos, de los cuales, el primer grupo de los mismos se aloja en los recursos computacionales de la infraestructura Grid, hasta que todos los recursos se encuentran ocupados por las tareas. Inmediatamente, la saturación de la infraestructura Grid es detectada, el proceso de despliegue de máquinas virtuales se completa en un tiempo despreciable (condiciones ideales) y los siguientes trabajos son alojados en las nuevas VMs. Conforme se van liberando los recursos de ambas infraestructuras, los trabajos pendientes van ocupando su lugar, hasta que finaliza la ejecución de todos ellos.

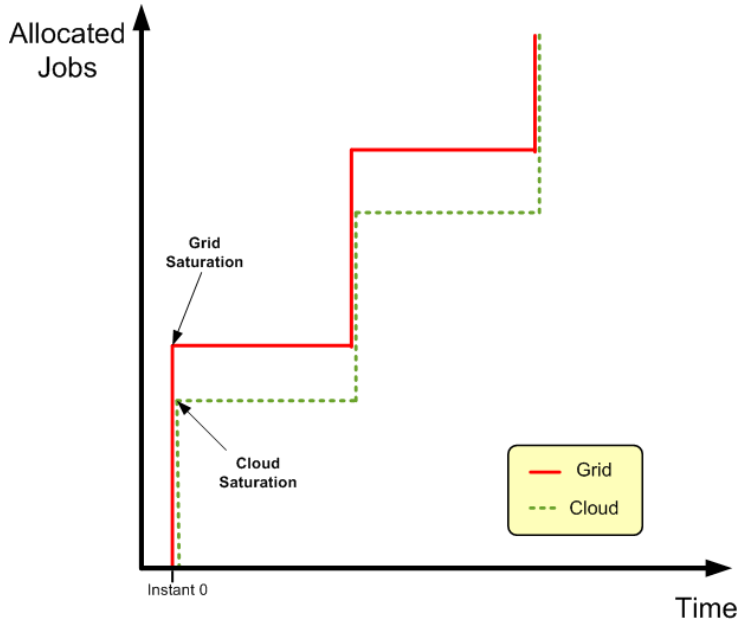

Figura 3.4 – Comportamiento ideal del modelo 1.

Por otro lado, la Figura 3.5 muestra el comportamiento del modelo 1 considerando todas las sobrecargas que se producen durante el proceso de

metaplanificación y ejecución de aplicaciones científicas. En ella, se puede observar que el proceso de metaplanificación comienza en el instante 0 de nuevo, pero consume un determinado período de tiempo *A* para realizar la configuración inicial de las tareas. Este tiempo depende en mayor parte de la infraestructura Grid, donde se requiere obtener información actualizada sobre el estado de sus recursos, a través de los distintos sistemas de información disponibles (dependientes del middleware Grid empleado) para poder llevar a cabo el proceso de metaplanificación correctamente. Seguidamente, los trabajos deben ser sometidos para la ejecución, acción que requiere un tiempo *G3*. El número de recursos disponibles para el usuario puede imponer un límite sobre cuántos trabajos se ejecutan de forma concurrente, definiendo el número máximo de trabajos que se pueden ejecutar de forma concurrente en el Grid (*G1*). El período de tiempo dedicado a la ejecución de los trabajos en el Grid, descrito en la Figura 3.5 como *G2*, depende de la carga computacional de los trabajos y las capacidades de los recursos Grid. Cabe destacar que las diferencias entre los distintos recursos computacionales de esta infraestructura pueden introducir variabilidad en el tiempo de ejecución de los trabajos. Ello produciría un efecto escalera en la pendiente de asignación de trabajos al Grid, ya que los trabajos terminados crean espacios de ejecución libres para la asignación de nuevos trabajos.

Cuando el metaplanificador detecta la situación de sobrecarga del Grid, decide aprovisionar y usar recursos Cloud. Esto lleva un tiempo *C1*, en el que están implicadas las acciones de contactar con el GMVs o el proveedor Cloud y desplegar las máquinas virtuales necesarias, dependientes del presupuesto del usuario (entre otros parámetros) y que limitarán de nuevo el número de trabajos concurrentes que se van a poder ejecutar sobre los recursos Cloud (*C2*). Una vez la infraestructura Cloud se ha desplegado, es tiempo de contextualizar las nuevas VMs para soportar la ejecución de la aplicación científica y alojar los *C2* trabajos sobre ellas. Este paso lo lleva a cabo el planificador Cloud Enactor, y consume un tiempo *C4*.

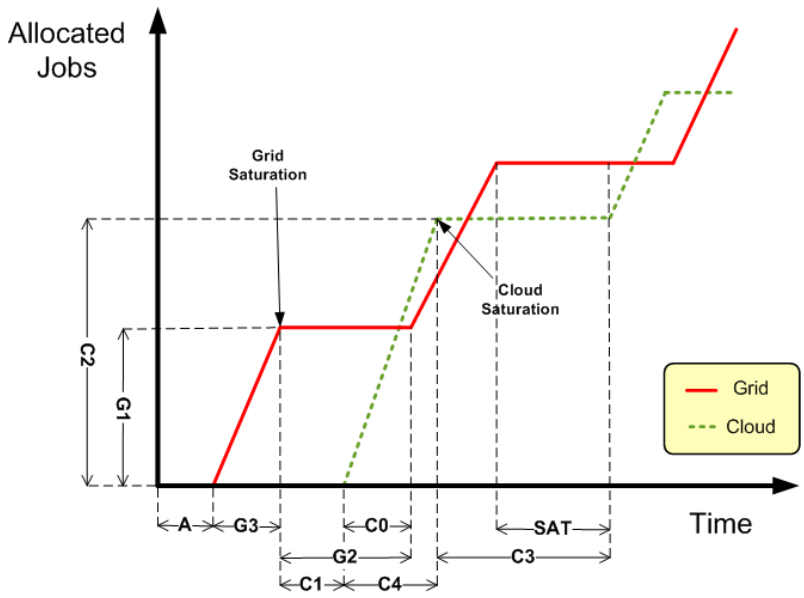

Figura 3.5 – Comportamiento real del modelo 1.

El intervalo *C3* se corresponde con el tiempo invertido en la ejecución de los trabajos en la infraestructura Cloud. Este tiempo típicamente puede ser mayor al tiempo invertido en la ejecución del mismo número de trabajos en la infraestructura Grid, debido al sobrecoste de la virtualización y al hecho de que los recursos Grid tienden a ser más potentes computacionalmente que las VMs dentro de un Cloud. *C0* indica el intervalo de tiempo desde que los trabajos han sido delegados a la infraestructura Cloud hasta que existen nuevos espacios de ejecución disponibles en el entorno Grid. Un período corto de *C0* indicará que no es necesario desplegar la infraestructura Cloud puesto que el tiempo consumido durante su despliegue y contextualización será mayor que el tiempo que tarden en liberarse los recursos Grid.

El intervalo de tiempo etiquetado como *SAT* indica que ambas infraestructuras se encuentran saturadas y, por lo tanto, no pueden aceptar nuevos trabajos hasta que (al menos alguno de ellos) los trabajos activos finalicen. Esta es una situación ideal desde el punto de vista de la utilización de los recursos.

Es necesario destacar que la Figura 3.4 asume que la infraestructura Grid tiene un mayor número de recursos que la infraestructura Cloud, sin embargo, en la Figura 3.5 se puede observar el comportamiento inverso, donde se dispone de un número más elevado de recursos Cloud que de recursos Grid. Esta distinción permite analizar el comportamiento desde dos puntos de vista diferentes, dependiendo del presupuesto del usuario para aprovisionar recursos Cloud y de los recursos Grid disponibles para la ejecución de trabajos, que típicamente dependerá de la VO a la que pertenezca el usuario.

3.4.2. MODELO 2: USANDO EL CLOUD CUANDO LOS REQUISITOS DEL TRABAJO NO SE SATISFACEN

Como se ha mencionado anteriormente, los trabajos más comunes que se van a ejecutar en la arquitectura propuesta serán aplicaciones científicas que típicamente tienen gran cantidad de dependencias, tanto hardware como software, y que se debe encontrar un recurso disponible para su ejecución que cumpla la mayor parte de ellos para asegurar que la ejecución se realizará con éxito.

En una infraestructura Grid, el sistema de información intentará encontrar el recurso que mejor se adapte a las dependencias de la aplicación, pero a veces puede no ser suficiente. El problema aparece cuando este software complejo que se quiere ejecutar tiene que hacerlo en una máquina que escapa de los dominios de los creadores de la aplicación, ya que no se podrá adaptar el recurso al trabajo, sino que es éste último el que deberá adaptarse al recurso si no existe un recurso computacional que se adapte a los requisitos de la aplicación. Suplir los requisitos hardware es realmente imposible. En el caso de los requisitos software, hasta ahora, los usuarios Grid tienden a utilizar archivos ejecutables enlazados estáticamente que encapsulan las dependencias software de la aplicación. Estos archivos ejecutables dependen, en gran medida, de la arquitectura hardware subyacente y la versión del sistema operativo, además de que su tamaño incrementa considerablemente. Esta es

la razón por la que distintos archivos ejecutables para diferentes arquitecturas se pueden almacenar en un elemento de almacenamiento (SE) con el fin de decidir, en tiempo de ejecución, cuál de ellos utilizar dependiendo de la plataforma subyacente donde se va a ejecutar el trabajo. Este es un gran inconveniente para el usuario.

Por el contrario, en un entorno Cloud, no existe esta limitación ya que el usuario tiene la oportunidad de construir una VMI a la carta y decidir las características de su máquina, tanto hardware como software, en caso de que no se encuentre una VMI que se adapte a las dependencias del trabajo disponible en el catálogo. Esta habilidad libera al desarrollador de la aplicación científica de tener que construir un gran ejecutable tal y como se ha comentado anteriormente y que no siempre tendrá las garantías de que el proceso de ejecución finalice con éxito. Por esta razón surge el segundo escenario, donde la aplicación que se desea ejecutar no encuentra ningún recurso en el Grid que se adapte a sus dependencias, pero tiene la posibilidad de construir una VM a su medida.

El comportamiento de este segundo modelo transcurre como sigue: tras analizar las dependencias hardware y software de la aplicación, el metaplanificador Grid consulta al servicio de información del middleware Grid para conocer si existe algún recurso que se adapte a estos requisitos. Si la respuesta obtenida no es satisfactoria, la ejecución de la aplicación será delegada al planificador Cloud, quien se encargará de encontrar una VMI apropiada que satisfaga las necesidades de la aplicación y ejecutarla con éxito.

Cabe destacar que la delegación a la infraestructura Cloud puede implicar tanto el uso de una nube privada como pública, aspecto que se encargará de decidir el metaplanificador Cloud (Cloud Enactor en este caso) en base a los parámetros analizados anteriormente.

3.4.3. MODELO 3: USANDO GRID Y CLOUD PARA DISTRIBUIR LA CARGA DE TRABAJOS

El tercer modelo que se presenta en este documento tiene como objetivo principal el empleo de la infraestructura Cloud como complemento de cómputo al entorno Grid en situaciones en las que se pretenda balancear la carga de trabajos entre ambas infraestructuras para evitar exceder cuotas o simplemente acceder a un mayor número de recursos para acelerar la ejecución de las tareas. Un usuario puede tener restringido el número de recursos a emplear tanto en infraestructuras Cloud como en infraestructuras Grid, donde dependiendo de las VOs a las que pertenezca puede sufrir una limitación de los recursos disponibles. En el caso del Cloud, estas restricciones pueden afectar al número de VMs que puede desplegar diarias o de forma simultánea. Claramente en una infraestructura Cloud pública el mayor limitante es el presupuesto disponible por parte del usuario, pero incluso empleando una infraestructura Cloud privada, el número de máquinas virtuales desplegables por un usuario puede estar limitado para favorecer el compartimiento de la infraestructura.

Por ello, este escenario implica el uso tanto de la infraestructura Cloud como de la infraestructura Grid, actuando ambas como las proveedoras de potencia computacional para la ejecución simultánea de trabajos.

Una distribución de la carga de trabajo entre infraestructuras separadas permite la ejecución de los trabajos con un mayor rendimiento, ya que se utiliza un mayor número de recursos al mismo tiempo (dependiendo de la disponibilidad y el presupuesto del usuario). Esto es de especial importancia para grupos de tareas multiparamétricas, donde se pueden ejecutar múltiples trabajos independientes de forma concurrente.

En este escenario, el metaplanificador está encargado de repartir los trabajos entre las infraestructuras Grid y Cloud, tratando de lograr el equilibrio de la de carga entre ambas infraestructuras. El resultado deseable en este modelo consiste en maximizar el rendimiento de las infraestructuras en términos de uso de los recursos,

tratando de aumentar la tasa trabajos terminados por unidad de tiempo. Esto puede implicar enfoques voraces o *greedy* que intentan aprovechar al máximo las capacidades de cómputo para obtener tantos recursos como sea posible de ambas infraestructuras.

En el caso de contar con acceso a un Cloud público, el reparto de tareas que deberá llevar a cabo el metaplanificador involucrará a las tres infraestructuras, haciendo más complejo el proceso de metaplanificación pero reduciéndo aún más el tiempo de ejecución total de las tareas, al poder acceder a un mayor número de recursos.

4. IMPLEMENTACIÓN DE LAS HERRAMIENTAS DE METAPLANIFICACIÓN

En este capítulo, se presentan los principales aspectos de implementación de las herramientas de metaplanificación híbridas desarrolladas en esta tesis de máster. En primer lugar, se presenta la arquitectura del sistema, desde el punto de vista Cloud, analizando sus principales componentes. En segundo lugar, se exponen las clases implementadas que intervienen en un ciclo de ejecución Cloud. Seguidamente, se realiza un análisis exhaustivo de los orquestadores híbridos desarrollados, capaces de realizar ejecuciones de aplicaciones científicas en tres tipos de infraestructuras distintas: Grid, Cloud privado y Cloud público. Finalmente, el capítulo se cierra con un estudio sobre el estado del arte en APIs de agregación, necesario para poder determinar con criterio el que mejor se adapta a la infraestructura híbrida.

4.1. ARQUITECTURA DE LA INFRAESTRUCTURA CLOUD

En este apartado se analiza la infraestructura sobre la que se va a realizar la implementación de las herramientas de metaplanificación. Los detalles de la parte Grid no se analizan en esta memoria puesto que son parte de la tesis del director de este trabajo [51], cuyo metaplanificador Grid GMarte ha servido como base para el desarrollo de esta tesis de máster.

Tras la visión general, se han incluido apartados específicos para analizar con más detalle cada uno de los componentes que actúan en una ejecución.

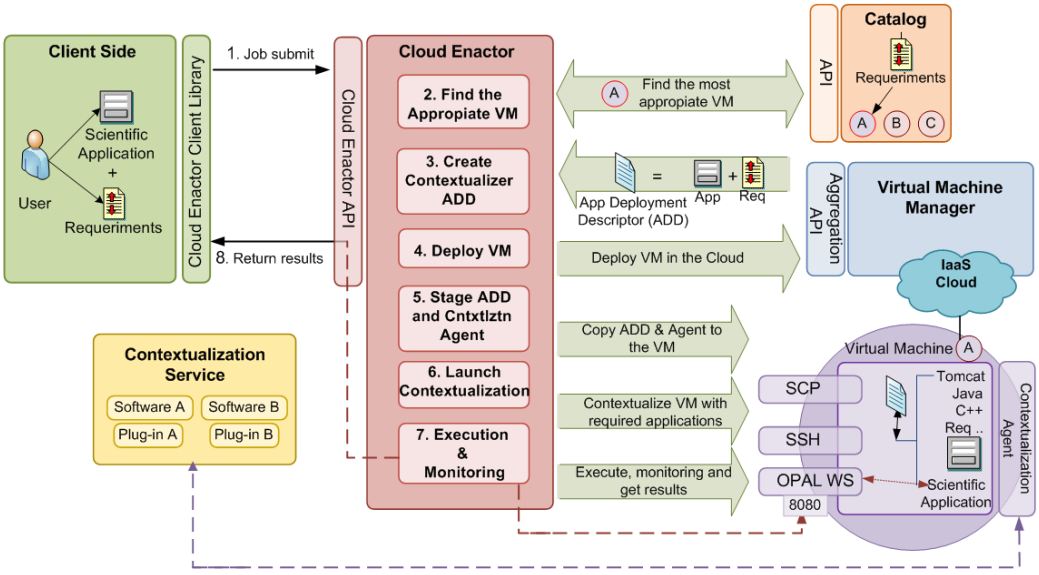

Figura 4.1 – Esquema ampliado del funcionamiento de Cloud Enactor.

En la Figura 4.1 se puede observar el esquema de la arquitectura diseñada e implementada para interactuar con una infraestructura Cloud. En primer lugar, el usuario envía la aplicación junto con los requisitos de la misma (descritos mediante el API de GMarte) al Cloud Enactor (CE). Éste contacta con el servicio VMRC con el fin de encontrar la VMI que más se ajuste a los requisitos de los trabajos. La capacidad de *matchmaking* del catálogo permitirá realizar esta tarea sin mayor esfuerzo por parte del orquestador, que obtendrá una lista priorizada de las VMIs que más se ajustan a estos requisitos.

Una vez seleccionada la VMI más adecuada, el CE crea un descriptor de despliegue de la aplicación, un fichero generado dinámicamente que servirá al servicio de contextualización para llevar a cabo la configuración del entorno permitiendo la instalación de manera desatendida de las dependencias software en la VM. A continuación, se delega la puesta en marcha de la VM a un GMVs o un proveedor Cloud (como OpenNebula o Amazon EC2) que se encarga de elegir el recurso físico más apropiado para el despliegue y ejecución de la VM. La interacción con los gestores de máquinas virtuales se realiza mediante el API adecuado (en el caso de OpenNebula se emplea el API desarrollado en el proyecto final de carrera de la

autora de este documento [53] y en el caso de Amazon EC2 se emplea el API de agregación jclouds, del cual se habla más tarde). Cuando la VM ha arrancado y tiene en marcha el servicio SSH (*Secure SHell*, Intérptete de Órdenes Seguro), el CE transfiere la aplicación junto con su descriptor de despliegue a la VM vía SSH y pone en marcha el contextualizador, por lo que el proceso de instalación de software comienza de manera desatendida. Cuando culmina, la VM se ha convertido en una VA (*Virtual Appliance*, Aplicación Virtualizada) que incluye la instalación de la aplicación junto con todos los requisitos software necesarios para su ejecución.

Llegados a este punto, es momento de lanzar la ejecución de la aplicación, tarea que se lleva a cabo empleando la herramienta Opal [54], que realiza una generación automática de un envoltorio basado en WS (*Web Services*, Servicios Web) para la aplicación. Mediante esta herramienta se puede poner en marcha de forma remota la aplicación, consultar el estado del trabajo y, finalmente recoger la salida para devolverle los resultados al usuario.

Con el ánimo de reducir el tiempo de contextualización, es posible llevar a cabo una estrategia consistente en guardar la VMI contextualizada en el VMRC, para poder utilizarla posteriormente evitando realizar de nuevo este proceso. Esta estrategia, como se ha comentado en el capítulo anterior, cobra vital importancia en el caso de aplicaciones multiparamétricas, donde se puede reutilizar la VMI para las distintas tareas.

4.1.1. VMRC

El catálogo y repositorio VMRC [52] (*Virtual Machine image Repository and Catalog*, Repositorio y Catálogo de imágenes de Máquinas Virtuales) es un servicio que permite a los usuarios indexar y almacenar VMIs junto con los metadatos apropiados que describen las características hardware y software de las máquinas, de manera que éstas puedan ser posteriormente buscadas y descargadas por parte de otros usuarios.

Además, cuenta con capacidades de *matchmaking*, basadas en los metadatos que acompañan a las VMIs, permitiendo a los usuarios especificar los requerimientos que necesitan al catálogo y el sistema se encarga de devolver la VMI más apropiada a ellos.

La catalogación de las VMIs se hace en base a los metadatos que aporta el usuario y que describen las características de las mismas: sistema operativo (tipo, distribución y versión), arquitectura de la CPU, imprevisor para el que ha sido construida y las aplicaciones que en ella están instaladas (nombre y versión). Con esta información es posible, por ejemplo, almacenar una VMI basada en GNU/Linux Ubuntu 9.10 con Java JDK 6.0 y Apache Tomcat 5.3.

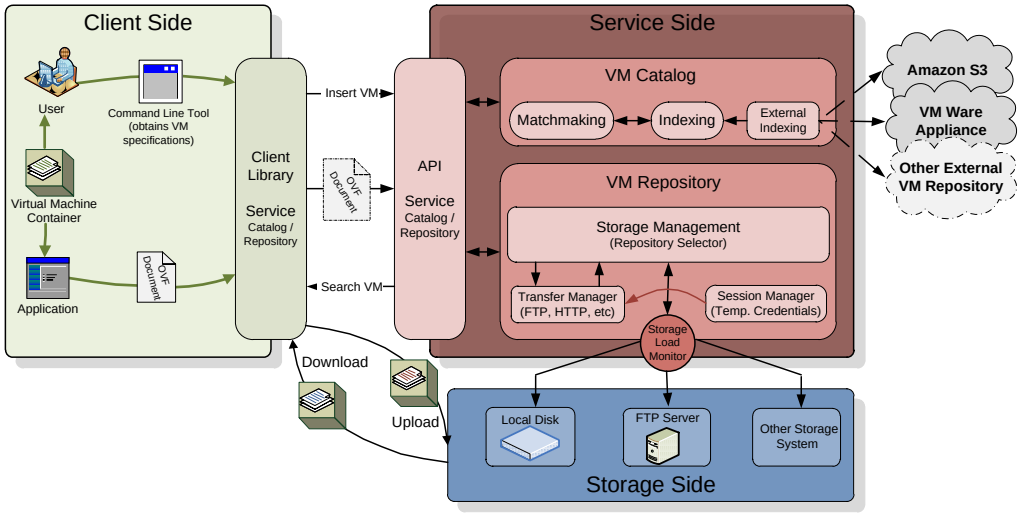

Figura 4.2 – Arquitectura del repositorio de imágenes de máquinas virtuales.

La Figura 4.2 resume la arquitectura del sistema VMCR. Su funcionalidad está dividida en los módulos cliente, catálogo, repositorio y almacenamiento:

- El módulo cliente se encarga proporcionar una interfaz de alto nivel al usuario para que éste pueda interactuar con los principales servicios del VMRC.

- El módulo de catálogo permite indexar las VMIs de acuerdo a sus principales características (hipervisor, SO y aplicaciones instaladas). Además, este módulo no está restringido a indexar las VMIs almacenadas en su repositorio, sino que es posible indexar las imágenes almacenadas en otros repositorios, facilitando de esta manera la integración con otros repositorios externos.

- El módulo de repositorio proporciona soporte para diferentes protocolos de transferencia de ficheros, como FTP o HTTP. Además, se encarga de gestionar las credenciales de sesión emitidas de forma temporal para permitir las subidas de VMIs.

- El módulo de almacenamiento se encarga de abstraer los detalles de diferentes medios de almacenamiento de imágenes de máquina virtual, como pueden ser discos locales o servidores FTP.

Para llevar a cabo la especificación de requisitos, el VMRC proporciona un lenguaje sencillo de basado en un esquema de *clave-valor*. Este lenguaje diferencia entre requisitos *duros* (aquellos que deben ser cumplidos obligatoriamente por la VMI para ser considerada una de las candidatas) y requisitos *blandos* (se valora positivamente su cumplimiento por parte de una VMI). Estos últimos llevan asociada una ponderación por parte del usuario de lo necesario que resulta dicha característica. A continuación, en la Figura 4.3, se muestra un ejemplo de este lenguaje en el que se precisa de una VMI basada en Linux, preferiblemente una versión Ubuntu 9.10 o superior, creada mediante el hipervisor KVM. Se requiere que esté instalado MySQL 5.0 y Tomcat 5.0. Opcionalmente, interesaría que la imagen tuviera instalado Java JDK 5.0 o superior.

```
vm.type='kvm'
os.name='linux'
os.name='linux'&& os.flavour='ubuntu' && os.version>='9.10',soft,20
app.name='mysql' && app.version='5.0'
app.name='tomcat' && app.version='5.0'
app.name='java-jdk' && app.version>='5.0'soft,40
```

Figura 4.3 - Ejemplo de lenguaje para la interacción con el VMRC.

El sistema VMRC ha sido desarrollado como un Servicio Web, desplegable por encima de un servidor de aplicaciones, como puede ser Apache Tomcat. Consta de un servidor FTP embebido que se activa automáticamente ante las peticiones de carga o descarga de VMIs por parte de los usuarios. Utiliza el estándar de la industria OVF para la descripción del contenido de la VMI. Además, consta de una librería cliente en Java para facilitar la interacción con el catálogo, y también proporciona una interfaz web para permitir a los usuarios la autenticación de los mismos vía usuario y contraseña para poder listar y descargar las VMIs junto con los metadatos asociados a la misma.

Cuenta con soporte a múltiples usuarios empleando listas de control de acceso que además introducen ciertos niveles de seguridad y previenen la distribución de software malintencionado en las propias VMs. El VMRC tiene una cuenta de administrador, el cual es capaz de crear nuevos usuarios. Cuando un usuario registra una VMI, puede opcionalmente especificar una lista de usuarios autorizados para interaccionar con la nueva VMI, o dejarla pública para ser accedida por cualquier usuario. Este aspecto cobra mayor importancia cuando se sitúa en el contexto de la investigación colaborativa.

4.1.2. EL CONTEXTUALIZADOR *CNTXTLZR*

Una VM base a menudo no contiene todo el soporte software necesario para la ejecución de una aplicación científica. Por ejemplo, una aplicación basada Java requiere la instalación de una JVM (*Java Virtual Machine*, Máquina Virtual Java), una dependencia que no tiene porqué encontrarse en las VMIs. Tradicionalmente, la instalación de este tipo de dependencias se ha realizado de forma manual por parte del usuario. También hay que tener en consideración el hecho de que una VM ya configurada con todo el software necesario para la ejecución de una aplicación puede ser reaprovechada. Sin embargo, el uso coordinado de diferentes proveedores Cloud con diferentes tecnologías de hipervisores (como Xen, KVM, etc.) provoca que las

VMIs no sean fácilmente trasladables de una infraestructura a otra, ya que, aunque se está trabajando en el uso de estándares como OVF, todavía queda camino por recorrer.

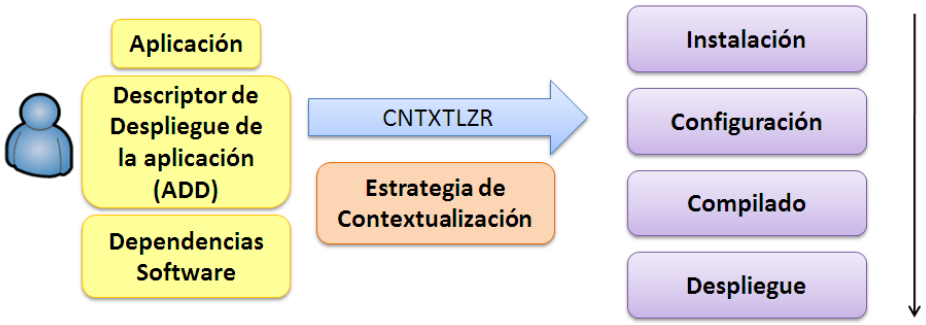

Figura 4.4 - Esquema para la contextualización de aplicaciones científicas.

Con el fin de evitar procedimientos manuales de instalación y configuración del entorno de ejecución (conocido como proceso de contextualización), tras desplegar las VMs se emplea una herramienta de contextualización, llamada *cntxtlzr*, que permite automatizar el flujo de despliegue de aplicaciones científicas. Esta herramienta de contextualización, desarrollada en Python bajo el proyecto en el que se enmarca el trabajo presentado en esta memoria, permite la instalación desatendida del soporte de ejecución necesario para las aplicaciones así como el despliegue de la propia aplicación. Su funcionamiento se basa en procesar un conjunto de ficheros XML (*eXtensible Markup Language*, Lenguaje de Marcas eXtensible) o *plugins* de instalación empleados para poder automatizar el proceso de contextualización. La Figura 4.4 resume la funcionalidad de esta herramienta, donde el proceso de contextualización se lleva a cabo en cuatro fases principales:

- **Instalación**: esta fase instala los paquetes de software de los que depende la aplicación. Además resuelve las dependencias de estos paquetes para instalar el resto de componentes software necesarios. Los paquetes de software pueden ser accesibles por el proceso de contextualización a través de una URL, instalables vía *yum* o *apt-get* o

simplemente encontrarse dentro de la propia máquina virtual, transferidos junto con la herramienta de contextualización.

- **Configuración**: Permite al usuario detallar el proceso de configuración del paquete software. Esto se logra mediante la especificación de acciones comunes, tales como la copia de ficheros, el cambio propiedades en ficheros de configuración, declarar variables de entorno, etc.

- **Compilado**: esta fase es la encargada de compilar el paquete de software utilizando el compilador más apropiado (Configurar + Make, Apache Ant, SCons, etc.).

- **Despliegue/Ejecución**: Para facilitar la ejecución remota de la aplicación, así como su monitorización, en esta fase se instala Opal, una herramienta que permite el despliegue de aplicaciones científicas como servicios web abstrayendo al usuario de tener conocimientos sobre estos servicios. Para que Opal se ejecute correctamente, es necesario instalar un contenedor de servicios web, como es Tomcat, y Java (requisito indispensable para el funcionamiento de los anteriores). Finalmente, se inicia Tomcat, provocando que la aplicación científica sea desplegada y los trabajos queden preparados para ser iniciados por el planificador Cloud Enactor.

En cuanto a la labor del usuario, tal y como se observa en la Figura 4.4, es el encargado de proporcionar una descripción de alto nivel del proceso de instalación de la aplicación científica, junto con las dependencias software de la misma. El contextualizador soporta un pequeño lenguaje declarativo basado en XML utilizado para crear un descriptor de despliegue de aplicaciones (ADD, *Application Deployment Description*), que detalla las acciones comunes empleadas para desplegar una aplicación científica, junto con sus requisitos software. Un ejemplo de este lenguaje de descripción se puede observar en la Figura 4.5.

```
<DeployableApp name="MyApp" requires="mpich opal">
  <Package name="MyApp" file="MyApp.tgz" overwrite="yes" />
  <Opalize exec_file="MyApp" default_args="--gra1
@MyApp#INSTALL_PATH@/casestudies/input_file.bin  --nliters 100 -t 10 --
lambda_min 0 --lambda_max
1 --lambda_delta 0.5" parallel="False"/>
  <Build type="make"/>
  <Build type="ant" args="-f @opal#INSTALL_PATH@/build.xml deploy -
DserviceName=MyAppServicePort -Da
ppConfig=@MyApp#INSTALL_PATH@/opal2/MyApp.xml"/>
</DeployableApp>
```

Figura 4.5 - Ejemplo de lenguaje de descripción de despliegue de una aplicación.

En ella se indica que el código fuente de la aplicación *MyApp* está en el fichero *MyApp.tgz*, sus argumentos de línea de comandos por defecto, *default_args*, y que debe compilarse (con *make*) previamente a la ejecución. Como dependencias, la aplicación requiere de la instalación de MPICH, una implementación del estándar MPI, y Opal, común a todas las tareas que se ejecuten bajo la arquitectura planteada.

4.2. ESQUEMA GENERAL DE CLASES

En este apartado se pretende hacer un resumen de las principales clases que intervienen en el ciclo de ejecución Cloud. Las clases encargadas de orquestar todas las fases expuestas en la Figura 3.3 del anterior capítulo se explican con detalle en la sección 4.3. En esta sección se centra la atención en las clases que realizan cada paso necesario para lograr la ejecución satisfactoria de la aplicación en el entorno Cloud, sin entrar en aspectos de metaplanificación.

Antes de comenzar con la explicación de las clases desarrolladas es importante destacar que se ha tomado como base el código desarrollado para GMarte, añadiéndole acceso a las infraestructuras Cloud (tanto privadas como públicas) y desarrollando nuevos planificadores híbridos, pero en todo momento manteniendo la estructura de clases con la que ya se contaba. Todo el desarrollo se ha llevado a cabo

empleando el lenguaje orientado a objetos Java [55], lenguaje en el que ya se encontraba desarrollado GMarte.

Por ello, la definición de tareas la lleva a cabo el usuario empleando el API de GMarte. Se ha trabajado en mantener la definición de tareas común a Grid y Cloud, por ello en el diagrama de clases que muestra la Figura 4.6 se pueden ver clases cuyo nombre incluye "Grid" pero es simplemente nomenclatura, ya que todas las tareas se definen de igual forma, puesto que el usuario no debe saber qué tareas se van a ejecutar en qué infraestructura. Este aspecto debe ser completamente transparente para él, ya que es tarea del sistema decidir en qué entorno será más conveniente la ejecución.

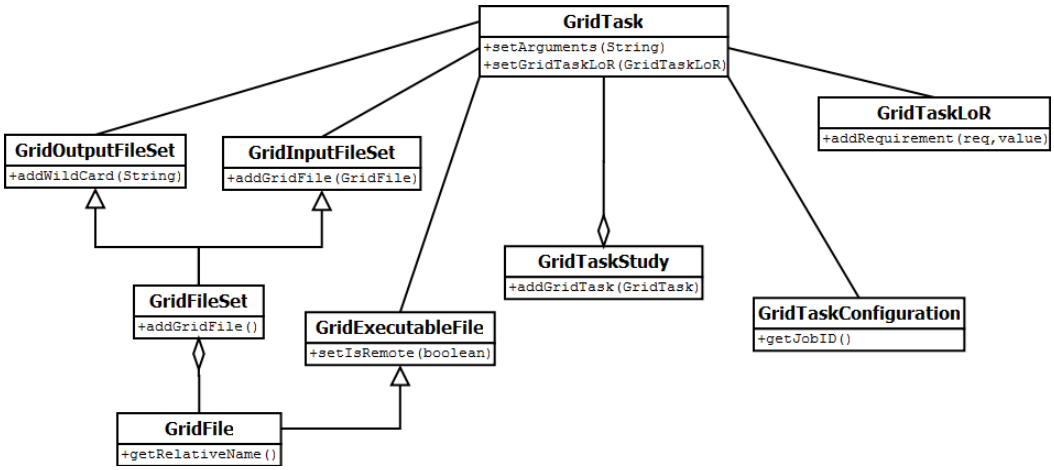

Figura 4.6 - Diagrama de clases simplificado para la definición de tareas.

En el diagrama, un objeto *GridTask* representa una tarea a ejecutar. Un conjunto de tareas ya definidas se agrupan en un objeto *GridTaskStudy*. Un *GridTask* representa la unidad de ejecución en la infraestructura híbrida y lleva asociados varios componentes, representados todos ellos como objetos. *GridExecutableFile* representa el fichero ejecutable de la aplicación. *GridInputFileSet* permite especificar los ficheros de entrada que requiere la aplicación para su ejecución. Se puede especificar tanto una ruta completa como un conjunto de ficheros de un directorio. *GridOutputFileSet*

permite especificar los ficheros de resultados que genera la aplicación que se desean recuperar en la máquina local tras finalizar la ejecución de la tarea. Típicamente engloba los ficheros generados por la aplicación, así como los ficheros de salida estándar y salida estándar de error.

Además, un objeto *GridTask* permite especificar una lista de requisitos que debe cumplir un recurso computacional para poder realizar la ejecución de la tarea. Para ello, se emplea un objeto de la clase *GridTaskLoR*, que permite especificar requisitos como el sistema operativo o la memoria RAM disponible. Por último, un objeto de tipo *GridTaskConfiguration* se puede emplear para especificar comandos previos a la ejecución de la tarea, como descomprimir algún fichero o configurar una variable de entorno.

Para la activación del *TestBed* o grupo de recursos computacionales sobre los que realizar las ejecuciones de los trabajos se dispone de las clases recogidas en la Figura 4.7. La clase principal que coordina la creación y contextualización de los recursos Cloud es *CloudManager*. Ésta es la encargada de contactar con el catálogo y repositorio de máquinas virtuales (VMRC), con el fin de obtener la VMI que mejor se adapte a los requerimientos de las tareas. Es necesario recordar que este elemento permite almacenar VMIs que pueden ser empleadas en infraestructuras Cloud privadas, pero también es capaz de albergar la descripción de las AMIs de Amazon EC2, lo cual resulta imprescindible para integrar el uso indistinto de ambos tipos de nube (privada y pública). Para ello, emplea la clase *VMRCApi*. Para poder acceder a una infraestructura Cloud se precisan de unas credenciales de acceso que el usuario debe proporcionar como entrada al proceso de metaplanificación, mediante la creación de un objeto de tipo *AccessCredentials*. Dependiendo de estas credenciales el propio metaplanificador, que se analiza más adelante, será capaz de determinar con qué tipo infraestructura va a poder interactuar.

Dependiendo del tipo de infraestructura donde se vaya a llevar a cabo la ejecución del trabajo, *CloudManager* delegará la tarea en un administrador Cloud concreto. En el caso de realizar la ejecución sobre un Cloud privado, la tarea la

recibirá la clase *ONECloudManager*, que creará la plantilla o *template* apropiado de la VM en base a los requerimientos de la tarea y contactará con el GMVs OpenNebula para ponerla en marcha. Si, por el contrario, la ejecución se va a llevar a cabo sobre una nube pública, la tarea será delegada a la clase *EC2CloudManager*, la cual se encargará de contactar con el proveedor Cloud Amazon EC2.

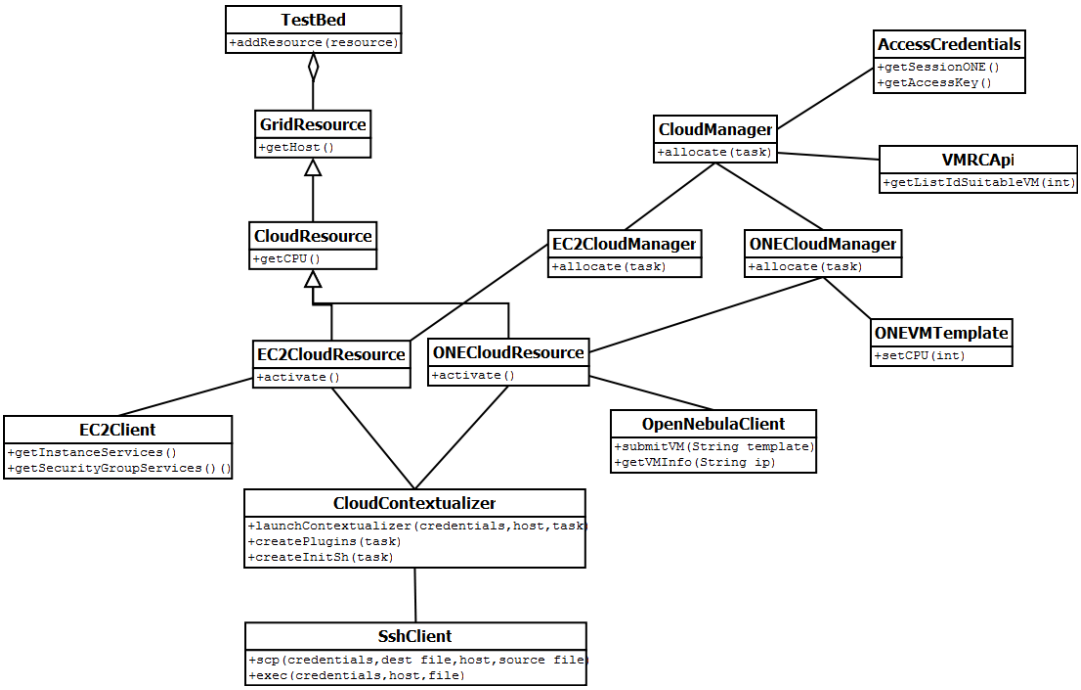

Figura 4.7 - Diagrama de clases simplificado para creación y contextualización de las VMs.

Un objeto *CloudResource* permite abstraer el concepto de máquina virtual. Para integrar con el código ya existente en GMarte, este tipo de objeto hereda sus principales características del objeto *GridResource*, que representa un recurso computacional cualquiera, añadiéndole las características propias de una VM, como el hipervisor. Para diferenciar entre un recurso de un Cloud público y otro de un Cloud privado, se emplean las clases *EC2CloudResource* y *ONECloudResource*, cada una de ellas asociada con el API concreto que permite la interacción con el proveedor adecuado.

Como se ha comentado anteriormente, para la interacción con el GMVs OpenNebula se ha empleado el API desarrollado en el proyecto final de carrera (PFC) de la autora de esta tesis de máster. Si bien es cierto que actualmente OpenNebula ofrece un API Java para interactuar con este gestor, la interfaz de alto nivel desarrollada en el PFC permite la construcción de clústers elásticos de máquinas virtuales, que son de especial beneficio en aplicaciones científicas en las que típicamente se requiere desplegar más de una VM de iguales características. En el caso de utilizar las infraestructuras Cloud de producción, que ofrecen computación bajo demanda previo pago por parte del usuario, como Amazon EC2, el API elegido ha sido jclouds [62], por adaptarse mejor a los requisitos que presentaba nuestra arquitectura y disponer de buena documentación, ejemplos y guías para su uso bajo los principales proveedores. Si bien es cierto que Amazon EC2 ya ofrece un API Java para utilizar sus servicios de forma programática, se ha desechado esta idea puesto que, al adaptar la arquitectura a un API de agregación, se facilita la tarea de emplear cualquier otro proveedor Cloud soportado por el API de agregación sin realizar mayor esfuerzo. Al final del presente capítulo se incluye un estudio del estado del arte en APIs de agregación que justifica la elección de *jclouds*.

La clase *CloudContextualizer* se encarga de llevar a cabo la contextualización de los recursos Cloud, ya sean públicos o privados. Para ello, construye de forma automática el *plugin* de la aplicación a ejecutar en base a la definición de la tarea que ha llevado a cabo el usuario. También se encarga de crear un pequeño script que pondrá en marcha la ejecución del contextualizador y llevará a cabo las tareas previas que el usuario haya definido en el objeto *GridTaskConfiguration*. La conexión con la máquina virtual se lleva a cabo mediante la clase *SshClient*, que proporciona las operaciones necesarias para transferir y ejecutar órdenes de forma remota. También cuenta con soporte a certificados, necesario para llevar las conexiones vía SSH y ejecutar comandos de forma remota en los recursos que provisiona Amazon EC2.

Finalmente, para la puesta en marcha de la ejecución de la aplicación una vez ésta se encuentra alojada en un recurso computacional y configurado su entorno de

ejecución, se emplea el ejecutor apropiado dependiendo del tipo de recurso sobre el que se vaya a realizar la ejecución. Para ello, la clase *RunnableGridTask* se encarga de determinar qué clase es la apropiada basándose en el tipo de recurso por el que está formada. Si la ejecución se realiza sobre una VM perteneciente a una nube privada, la ejecución se delega a la clase *ONECloudExecutor*. Por el contrario, si se trata de una ejecución sobre un recurso proporcionado por un proveedor Cloud público, la ejecución de la tarea se delega sobre la clase *EC2CloudExecutor*. La Figura 4.8 muestra el esquema simplificado de las clases encargadas de realizar la ejecución de las tareas.

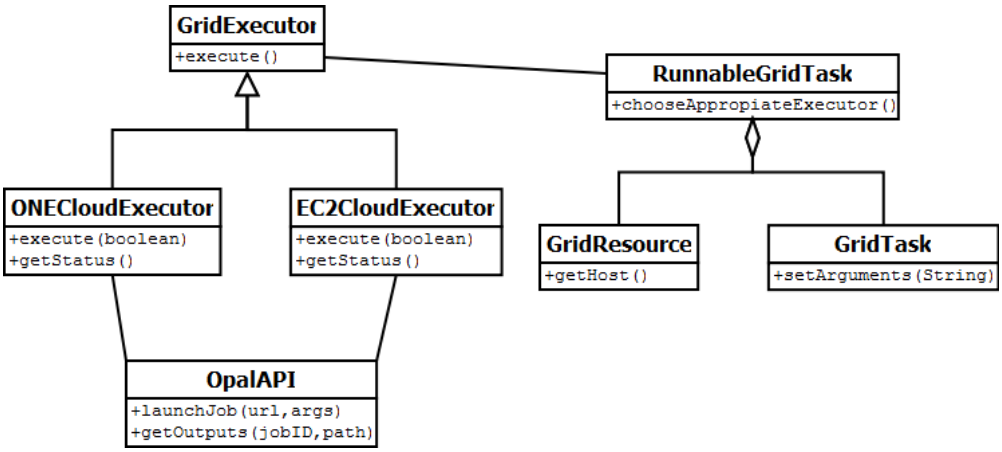

Figura 4.8 - Diagrama de clases simplificado para la ejecución de la aplicación en el Cloud.

Como se ha comentado en el apartado 4.1.2 de este mismo capítulo, para llevar a cabo la ejecución remota de las tareas en un recurso Cloud se emplea la herramienta Opal, que envuelve a la aplicación permitiendo dotarla de capacidades propias de WS, facilitando así su interacción desde el exterior. Para ello, ambos ejecutores interaccionan con la clase *OpalAPI*, una pequeña librería de instrucciones para interaccionar con Opal utilizada, por ejemplo, para poner en marcha la ejecución del trabajo, monitorizarlo y obtener los resultados de salida.

4.3. METAPLANIFICADORES HÍBRIDOS

Tras analizar las principales clases que componen el desarrollo para la ejecución de tareas en un entorno Cloud, ya sea privado o público, en este apartado se analiza la funcionalidad que ofrecen los orquestadores híbridos desarrollados, que son los componentes principales dentro de la ejecución.

En primer lugar, es necesario destacar en este apartado los principales estados por los que atraviesa una tarea al ser ejecutada por el metaplanificador GMarte, ya que en las ejecuciones Cloud se seguirán respetando estos estados. La Figura 4.9 muestra el diagrama de estados por los que puede pasar una tarea durante su ejecución.

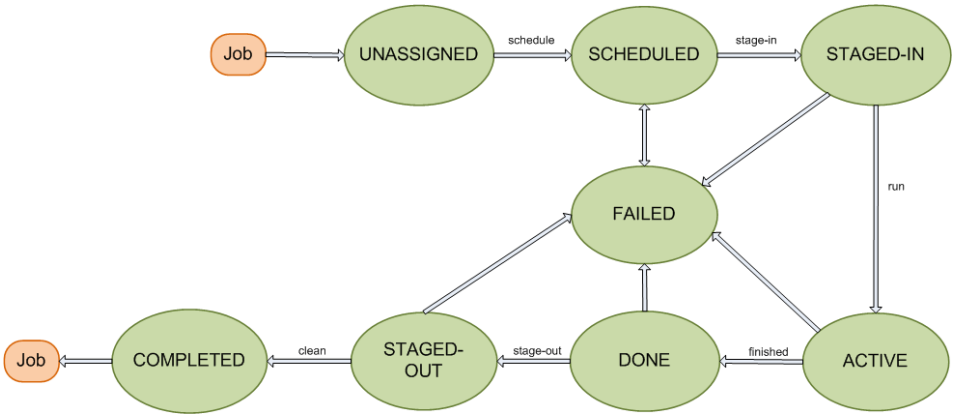

Figura 4.9 - Diagrama de estados de un trabajo.

Cada estado proporciona información sobre la fase que está atravesando la tarea, cuyo significado es el siguiente:

- ***Unassigned***: es el estado inicial de una tarea, que todavía no ha sido seleccionada por el metaplanificador.

- ***Scheduled***: la tarea ha sido seleccionada por el metaplanificador, que habrá seleccionado el mejor recurso para la misma, a la espera de comenzar la transferencia de datos al recurso destino.

- ***Staged-in***: la fase de transferencia inicial de archivos ha sido completada satisfactoriamente y en la máquina destino se encuentra tanto la aplicación como los archivos dependientes.

- ***Active***: la tarea se encuentra en ejecución en el recurso remoto.

- ***Done***: la tarea ha finalizado su ejecución en el recurso remoto. Alcanzar este estado no siempre significa que la tarea haya concluido satisfactoriamente, simplemente que ha terminado su ejecución.

- ***Staged-out***: los resultados de la ejecución se han transferido a la máquina local del usuario.

- ***Completed***: la tarea ha acabado satisfactoriamente y el proceso de metaplanificación no debe realizar ninguna acción adicional con la misma. Por lo tanto, este es el último estado que alcanza una tarea.

- ***Failed***: la tarea ha fallado. Se puede alcanzar este estado por varios motivos, como porque falló la fase de transferencia de archivos, no pudo ser ejecutada por la máquina remota o porque se produjo un fallo durante la ejecución.

Los metaplanificadores desarrollados se encargan de realizar el despliegue y la activación de los recursos en el momento adecuado, empleando para ello las clases comentadas anteriormente, y de realizar el reparto de tareas apropiado. Siguiendo con los modelos teóricos planteados en el capítulo anterior, donde se ponen de manifiesto tres casos de uso en los que la ejecución híbrida proporciona potenciales beneficios a la ejecución de aplicaciones científicas, se han desarrollado las clases necesarias para reproducir el comportamiento teórico planteado. La Figura 4.10 recoge el diagrama de clases simplificado donde se muestra la integración de los nuevos metaplanificadores con los que ya existían desarrollados en GMarte.

La clase *OrchestratorFirstModel* reproduce las políticas de metaplanificación puestas de manifiesto en el modelo 1. Por ello, para seleccionar el mejor recurso (*chooseResource*) una vez seleccionada la tarea, esta clase delega las tareas a la infraestructura Grid hasta que ésta se encuentra saturada, detección que lleva a cabo en tiempo real y que provoca el despliegue, contextualización y selección de los recursos disponibles en el entorno Cloud. Nótese que el despliegue de las VMs se realiza en el preciso instante de detección de la saturación Grid, puesto que sería innecesario desplegar la infraestructura Cloud antes de saber si el Grid va a poder albergar la ejecución completa de las tareas.

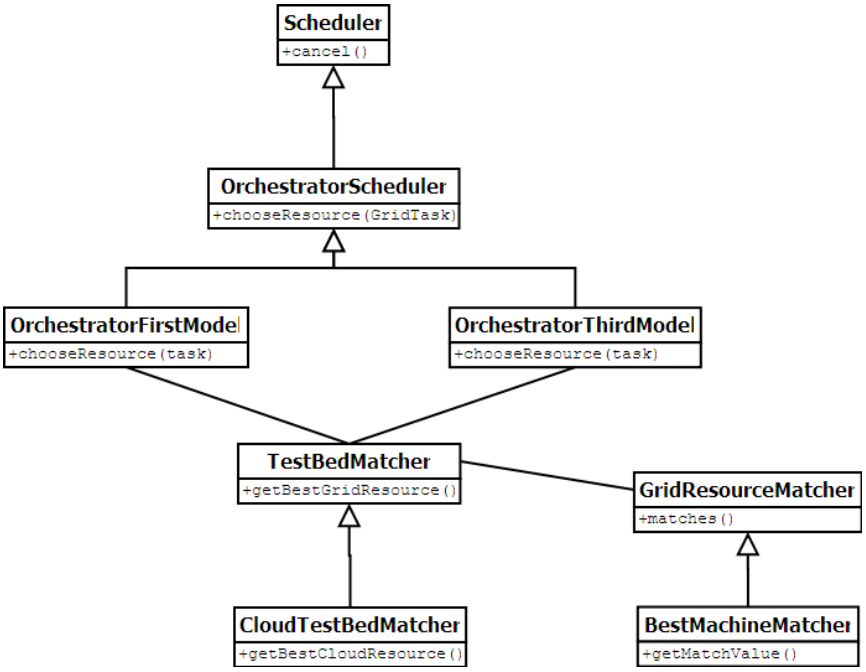

Figura 4.10 - Diagrama de clases simplificado de los metaplanificadores desarrollados.

Por un lado, para llevar a cabo la selección del mejor recurso Grid cuando esta infraestructura cuenta con recursos libres se emplea la clase *TestBedMatcher*, que aplica políticas de selección basadas en un modelo de prestaciones que abarca tanto a la propia aplicación (tiempo de ejecución y requisitos), como a los recursos computacionales disponibles (a través de las clases *GridResourceMatcher* y

BestMachineMatcher). Este modelo de prestaciones se encuentra descrito en [51] y no forma parte de este trabajo, luego no se profundizará más en el mismo. Por otro lado, en el caso de una ejecución sobre la infraestructura Cloud, la clase C*loudTestBedMatcher* es la encargada de determinar qué VM es la más apropiada para ejecutar la tarea. En este caso, es muy probable que si existe más de una VM disponible, todas tengan las mismas características, luego la aplicación del modelo de prestaciones no sería suficiente para balancear la carga de trabajo entre los recursos. Es por ello que esta clase se encarga de aplicar una política circular, semejante a la estrategia de *Round-robin* (RR) sobre las máquinas, con el fin de repartir la carga computacional entre todas.

Si el usuario ha proporcionado las credenciales y presupuesto necesarios para acceder a una infraestructura Cloud pública, cuando se detecte la saturación del Cloud privado, la ejecución de las tareas se delegará a los recursos proporcionados por Cloud público, empleando de nuevo una política circular para repartir la carga computacional entre las VMs y permitiendo ejecutar un mayor número de trabajos de forma concurrente.

Cabe mencionar que mediante la clase *OrchestratorFirstModel* también se implementa el comportamiento del segundo modelo, donde si no se encuentra un recurso apropiado en el Grid, o éste está saturado, la ejecución se delega a la infraestructura Cloud, ya sea privada o pública.

En el Anexo A se puede analizar una posible clase de un usuario que desea emplear el planificador *OrchestratorFirstModel* para la ejecución de una aplicación muy sencilla.

La clase *OrchestratorThirdModel* se corresponde con una de las posibles implementaciones que se pueden realizar del modelo 3. En ella, se ha seguido una estrategia de balanceo de carga que persigue el reparto equitativo de la carga computacional entre infraestructuras Grid y Cloud. Se ha considerado oportuno que la implementación del modelo 3 atienda principalmente a tres criterios: equidad, o garantizar que las tareas sean tratadas de manera igualitaria, rendimiento, o maximizar

el número de tareas procesadas por unidad de tiempo, y utilización, donde los recursos se han de mantener tan ocupados como sea posible respetando los límites de uso impuestos por el usuario.

Es por ello que la selección del mejor recurso responde a la utilización de un algoritmo que permite realizar un reparto de las tareas de forma circular entre, como mínimo, dos infraestructuras (que puede ampliarse a tres en el caso de contar con acceso a un Cloud público), asegurando el criterio de equidad. Este algoritmo tiene en cuenta la posible saturación de las infraestructuras, detectando esta situación y adaptando su comportamiento a este hecho. Si quedan tareas pendientes, la infraestructura que no se encuentra saturada albergará un número mayor de trabajos hasta que se vayan liberando los recursos y pueda volverse a aplicar un reparto equitativo. De esta forma, se cumple con los criterios de rendimiento y utilización de recursos.

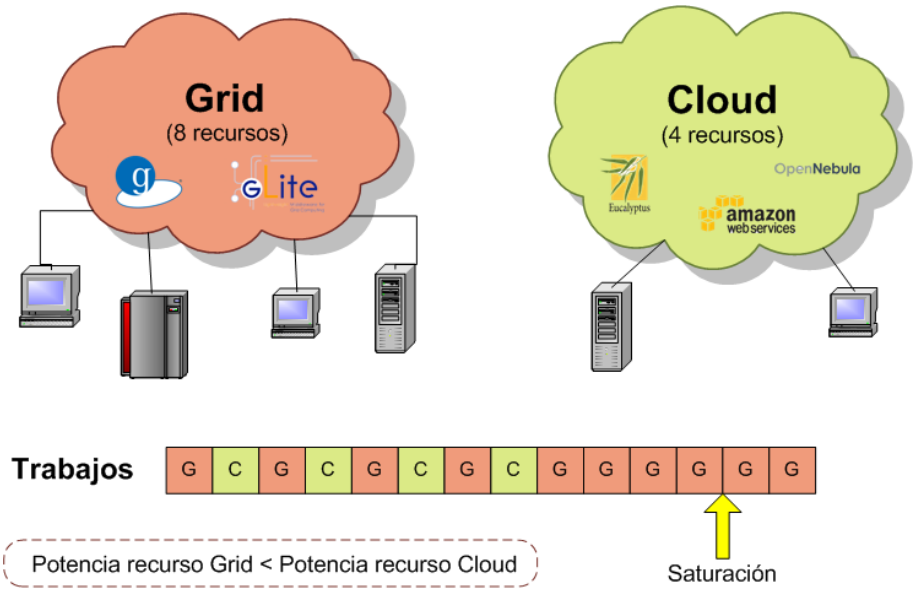

Figura 4.11– Comportamiento de la implementación del modelo 3.

La Figura 4.11 muestra un ejemplo del comportamiento que sigue el algoritmo empleado, que realiza un reparto equitativo de las tareas mientras ambas infraestructuras tienen nodos libres. En ella se ha asumido que las tareas, aunque sean

de la misma duración, en los recursos Grid se van a ejecutar más rápido que en los recursos Cloud, comportamiento habitual que se analiza en el capítulo 6, debido a la diferente potencia computacional de los recursos. Es por ello que tras ejecutar la primera ronda de tareas, las restantes se van a ejecutar en el Grid puesto que sus tareas acabarán antes.

Existen otras posibles políticas, como la planificación por grupos de tareas, donde agrupar las tareas menos costosas computacionalmente con el fin de enviarlas a los recursos computacionales más lentos (típicamente las VMs) y hacer lo mismo con las tareas más costosas, para ejecutarlas en los recursos Grid que típicamente son más potentes computacionalmente. Un algoritmo por prioridades también podría aplicarse, atendiendo a tiempo de ejecución de las tareas (concediendo prioridad a tareas más cortas o por el contrario, a las de mayor duración), o a dependencias entre ellas (donde las tareas encargadas de generar los datos de los que dependen otras tendrían mayor prioridad). La categorización de los recursos, empleando un *benchmark* previo a la ejecución de las tareas capaz de evaluar los recursos heterogéneos (típicamente las infraestructuras Grid tienden a ser heterogéneas), podría ser una estrategia a seguir para obtener una aproximación del rendimiento que ofrecerán para la ejecución de las tareas, y así planificar de acuerdo a la capacidad de cómputo de los recursos.

Sin embargo, la decisión de la política implementada ha estado condicionada principalmente por el tipo de tareas que se van a ejecutar (multiparamétricas, de muy similar duración e independientes entre sí). Cabe señalar que, como se ha mencionado anteriormente, esta es una posible implementación de las muchas que se podían llevar a cabo empleando el modelo 3, pero debido a las consideraciones realizadas, se ha llevado a cabo una implementación sencilla que se adapta perfectamente a los requerimientos dados y que es capaz de justificar el comportamiento del modelo 3 para entornos híbridos bajo los recursos a los que se tiene acceso.

Resulta necesario destacar que, a diferencia de la clase *OrchestratorFirstModel*, el despliegue y contextualización de las VMs que conforman la infraestructura Cloud se

lleva a cabo previo a la inicialización del reparto de las tareas, puesto que es necesario contar con los recursos disponibles de ambas infraestructuras para dar comienzo al algoritmo de planificación.

Nótese que ambos orquestadores heredan de la clase *OrchestratorScheduler*, que proporciona la funcionalidad de un metaplanificador de tareas multi-hilo, implementando la interfaz *Scheduler*.

4.4. ESTADO DEL ARTE EN APIs DE AGREGACIÓN

Como se ha explicado en el segundo capítulo de esta tesis de máster, existen numerosos GMVs disponibles actualmente en el mercado. Es por ello que surge la necesidad de construir interfaces de programación capaces de interaccionar a la vez con distintos gestores, facilitando la labor de programación e interacción con los mismos al usuario. En este contexto, resulta necesario realizar un estudio del estado del arte en APIs de agregación, para poder seleccionar el que más se adecúe a las necesidades que plantea nuestra infraestructura híbrida.

La organización Apache cuenta con su propio API de agregación Cloud, LibCloud [56], que es posible utilizar empleando el lenguaje de programación Python [57]. Esta librería soporta la interacción con distintos GMVs, como Eucalyptus, OpenNebula, OpenStack o Amazon EC2, entre otros. También da soporte a distintos servicios de almacenamiento Cloud, como Amazon S3. Actualmente, existen diversos proyectos que están empleando este API, pero el lenguaje de programación empleado no encaja con nuestro proyecto.

DeltaCloud [58] es un API, también desarrollado bajo la organización Apache, que aplica la filosofía REST (*Representational State Transfer*, Transferencia de Estado Representacional) para dar soporte a la interacción con numerosos GMVs. Aunque está desarrollado en Ruby [59], al ser un REST API, pueden construirse clientes en otros lenguajes que se comuniquen con el servidor del mismo. Funciona a través de la

instalación de distintos drivers (o *gemas* hablando en la jerga de Ruby), uno para cada GMVs que se desee ejecutar. Actualmente, da soporte a varios proveedores Cloud, como Amazon EC2, GoGrid [60], Eucalyptus u OpenNebula, pero no soporta todas las operaciones para todos los proveedores (por ejemplo, en EC2 no se pueden poner en marcha instancias EBS que fueron paradas empleando este API). Este hecho es el que mayores inconvenientes supone a la hora de emplearlo en el proyecto. Fog [61], es otro API de agregación desarrollado en Ruby, pero que soporta un número de proveedores menor. Además, la documentación sobre el mismo resulta escasa, debido quizás a que se encuentra poco desarrollado.

En cuanto a APIs de agregación desarrollados en Java, se puede encontrar a jclouds [62] que cuenta con soporte a múltiples proveedores Cloud. Este es el API seleccionado para el desarrollo del proyecto, por ser el que mejor se adapta a nuestros requisitos. Además, está muy bien documentado y existen múltiples ejemplos y guías para los principales proveedores, como Eucalyptus, GoGrid, Openstack o Amazon EC2, y también cuenta con soporte para los principales servicios de almacenamiento, como Amazon S3 o Azure Storage Service [63]. Otros APIs, como Dasein Cloud API [64], también están desarrollados en Java, pero su lista de proveedores soportados es menor. Además, en este caso, algunas de las operaciones con los GMVs se delegan al anterior, jclouds.

Existen otros APIs, como Simple Cloud [65], desarrollados para dar soporte sólo a los servicios de almacenamiento y gestión de ficheros, como Amazon S3 o Niervanix [66]. Esta librería está desarrollada en PHP.

Por último, existe un API que se encuentra en desarrollo por la empresa RightScale, y que de momento se encuentra en su versión beta [67]. De nuevo, esta librería sigue una estrategia REST y está pensado para ser utilizado desde línea de comandos o empleando el lenguaje Ruby. Actualmente da soporte al proveedor de AWS (*Amazon Web Services*, Servicios Web de Amazon), Amazon EC2.

5. ANÁLISIS DE LAS SOBRECARGAS DEL PROCESO DE VIRTUALIZACIÓN

Antes de lanzar un trabajo a una infraestructura tipo Cloud, es conveniente realizar un estudio a priori para conocer si se obtendrán beneficios al ejecutar el trabajo en este tipo de entorno. Para ello es necesario conocer, además de la duración aproximada del tiempo de ejecución del trabajo, el impacto que supone la creación y despliegue de una VM (máquina virtual, *Virtual Machine*).

Es por ello que en este capítulo se plantea el estudio del impacto en las prestaciones que supone lanzar y desplegar una VM sobre una máquina física en contraste con las prestaciones obtenidas directamente sobre la infraestructura física. En concreto, se prestará mayor atención a la sobrecarga que supone el uso de virtualización en lo que a memoria (operaciones de E/S) y tiempo de ejecución (CPU) se refiere.

5.1. INFRAESTRUCTURA EMPLEADA

La evaluación de las sobrecargas de la virtualización se va a realizar sobre una de las máquinas de que se dispone en el laboratorio del GRyCAP, un clúster formado por 4 nodos de tipo Blade donde cada nodo consta de dos procesadores Intel(R) Xeon(R) CPU L5430 a 2.66GHz, con tecnología *Hyper-Threading* y arquitectura x86-64, de 4 cores cada uno. Cuenta con un total de 16 Gigabytes de memoria RAM (*Random Access Memory*, Memoria de Acceso Aleatorio). El sistema operativo que corre en esta máquina es Linux Ubuntu Server 9.10. Con respecto a las tecnologías empleadas para crear un Cloud privado, está máquina cuenta con la instalación del

hipervisor KVM y OpenNebula 2.0 como gestor de máquinas virtuales. Las pruebas se han realizado sobre uno de los nodos del clúster.

5.2. BENCHMARKS SINTÉTICOS DE CPU

El consumo de CPU supone una de las principales limitaciones que existen a la hora de ejecutar una aplicación científica. Es por ello que este apartado se dedica íntegramente a analizar las sobrecargas que supone el empleo de una máquina virtual en contraste con una máquina física para este aspecto. En primer lugar se realiza una pequeña revisión del estado del arte en *benchmarks* sintéticos de CPU, para seleccionar con criterio el que se empleará en el estudio que se presenta posteriormente. Seguidamente se exponen las pruebas que se van a llevar a cabo para finalmente, presentar los resultados obtenidos.

5.2.1. ESTADO DEL ARTE

Los test sintéticos no tienen la intención de representar a un "mundo real" en lo que a carga de trabajo se refiere, pero son útiles en la medición de la capacidad máxima de los aspectos específicos de un sistema. Analizando el estado del arte en *benchmarks* sintéticos de CPU, se pueden encontrar distintas soluciones para medir las prestaciones de cómputo de una máquina, desde versiones de pago, hasta versiones libres o específicas para un tipo de computador o computación (HPC, etc.).

El primer *benchmark* que se puede encontrar en el mercado es el desarrollado por SPEC (*Standard Performance Evaluation Corporation*, Corporación de Evaluación de Estándares de Rendimiento), una entidad sin fines de lucro formada para establecer, mantener y apoyar un conjunto estandarizado de *benchmarks* relevantes que se pueden aplicar a la nueva generación de computadores de alto rendimiento. La versión actual del *benchmark* de CPU que ofrecen es SPEC CPU2006 [68], un test diseñado para

proporcionar medidas de rendimiento que se pueden utilizar para comparar las cargas de trabajo de computación intensiva en sistemas informáticos diferentes. Este *benchmark* está formado por dos grupos de herramientas diferentes: CINT2006 para medir y comparar el rendimiento de operaciones enteras de computación intensiva, y CFP2006 para medir y comparar el rendimiento de cómputo intensivo de operaciones en coma flotante. Además, también se ofrecen test para medir el rendimiento de la virtualización, como SPECvirt sc2010 [69], pero el principal inconveniente a la hora de emplear estos *benchmarks* es que son de pago.

Otro de los *benckmarks* más conocidos en el entorno de la supercomputación es el test de Linpack [70], basado en la resolución de distintos sistemas de ecuaciones para obtener el rendimiento de las operaciones en coma flotante. Este test es ampliamente utilizado en arquitecturas paralelas. Por ejemplo, es el *benchmark* que se emplea cada año para determinar los supercomputadores más potentes del mundo (la lista del TOP500 [71]). Para su funcionamiento requiere tener instalada una implementación de MPI, la librería matemática BLAS o VSIPL.

BYTEmark o Nbench [72], es un *benchmark* desarrollado en el lenguaje C, inicialmente pensado para entornos Linux, que está compuesto por diferentes test diseñados para exponer las capacidades de la CPU, FPU (*Floating-Point Unit*, Unidad de Punto Flotante) y memoria del sistema. Algunos de los algoritmos que aplica son la descomposición LU o la ordenación de vectores de números o caracteres. En [73] se puede encontrar una descripción más exhaustiva de los algoritmos que emplea el test.

Finalmente, se puede encontrar Unixbench [74], un *benchmark* que, a pesar de no ser estrictamente de CPU (ya que está especializado en medir las prestaciones del sistema), incluye test encargados de medir las prestaciones de la CPU mediante la medición de la velocidad y eficiencia de las operaciones en coma flotante (*Whetstone* [75]) y operaciones específicas para tipos de variables concretos (desde cadenas de caracteres, hasta tipos aritméticos, empleando por ejemplo el test de *Dhrystone* [76]). Está implementado principalmente en C, aunque incluye rutinas desarrolladas en Perl.

5.2.2. PRUEBAS Y PARÁMETROS A MEDIR

Tras analizar el estado del arte sobre los *benchmarks* sintéticos de CPU, se ha optado por la elección de UnixBench, ya que facilita la elección de los tipos de operaciones que se pretender medir. Se empleará la última versión disponible, 5.1.3.

UnixBench está formado por distintos test individuales encargados de medir información diversa sobre el sistema. No todas las pruebas que ofrece son de interés para el proyecto, luego se ha procedido a realizar una selección de los más apropiados. En concreto, las pruebas utilizadas son las que se recogen en la tabla 5.1.

Nombre del test	**Breve descripción**
Whetstone	Mide el rendimiento de las operaciones enteras y en coma flotante.
Dhrystone	Mide el rendimiento de las operaciones de cadenas de caracteres. Muy dependiente del compilador.
Short	Operaciones aritméticas sobre variables tipo short.
Int	Operaciones aritméticas sobre variables tipo int.
Long	Operaciones aritméticas sobre variables tipo long.
Float	Operaciones aritméticas sobre variables tipo float.
Double	Operaciones aritméticas sobre variables tipo double.

Tabla 5.1 – Selección de test empleados del *benchmark* UnixBench.

El test de *Whetstone* expresa sus resultados en MWIPS (Millones de Instrucciones *Whetstone* Por Segundo). Se puede definir a una instrucción *Whetstone* como una instrucción de punto flotante promedio. Su cálculo se llevaría a cabo siguiendo la fórmula de la Figura 5.1. El resto de test expresan sus resultados en LPS (*Loops Per Second*).

$$\text{Instrucción Whetstone} = \frac{100 * \text{Cantidad de iteraciones} * \text{Cantidad de WIPS por iteración}}{\text{Tiempo de ejecución}}$$

Figura 5.1 – Fórmula para obtener el valor de una instrucción *Whetstone*.

Los pasos seguidos para compilar y ejecutar este *benchmark* son tan sencillos como ejecutar el comando "*make*", dotar de los permisos necesarios al fichero "*Run"* para poder ejecutarlo, y hacerlo con las opciones deseadas. Si no se le indica ninguna opción, se ejecutarán todos los test que forman el *benchmark*. Para poder seleccionar uno en concreto es necesario ejecutar "*Run*" con el nombre del test deseado.

5.2.3. RESULTADOS

Tras la ejecución parcial del *benchmark* UnixBench, ejecutado con los mismos parámetros tanto en la máquina física como en la virtual, se han obtenido los resultados que se recogen en la tabla 5.2.

Cabe señalar que estos resultados los obtiene el *benchmark* repitiendo las pruebas varias veces (en concreto 10 veces para los test *Whetstone* y *Dhrystone* y 3 para el resto de test analizados) y obteniendo la media de los mismos.

	Whestone	**Drhystone**	**short**	**int**	**long**	**float**	**double**
Máquina física	3179,8	20813809,8	2538918,7	2565064,7	715808,3	2137011,2	1315369,9
Máquina virtual	3003,7	18439819,2	2183684,9	2159692,7	807196,9	1722840,4	1087000,2
Unidades	MWIPS	lps	lps	lps	lps	lps	lps

Tabla 5.2 – Resultados obtenidos con el *benchmark* UnixBench.

En ellos se puede observar cómo, en general, la máquina física ofrece mejor rendimiento de CPU que la máquina virtual. Este resultado era el que, previamente a ejecutar las pruebas, se esperaba.

Analizando cada una de las pruebas por separado, se puede obtener el porcentaje de ganancia/pérdida que supone el empleo de la máquina virtual con respecto a la física. En el caso del test *Whetstone*, el empleo de la VM supone una pérdida de rendimiento de las operaciones en coma flotante y enteras del 5.5%.

En el caso de las operaciones sobre cadenas de caracteres, el porcentaje ronda el 10%, algo superior al de las operaciones numéricas, pero teniendo en cuenta que el test de *Dhrystone* está fuertemente condicionado por el compilador empleado y que la versión del compilador empleado en la máquina física (gcc versión 4.4.1) es superior al de la virtual (gcc versión 4.3.3), la diferencia puede deberse a este hecho.

El resto de pruebas realizadas empleando el *benchmark* de CPU, reflejan la diferencia de rendimiento de operaciones aritméticas sobre tipos de datos concretos (cada tipo de dato se corresponde directamente con el nombre que recibe el test) entre la máquina física y virtual.

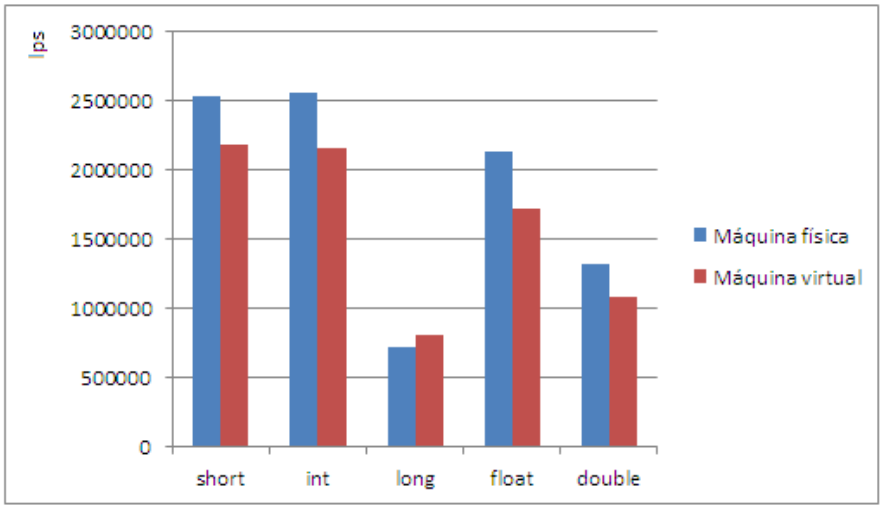

Figura 5.2 – Gráfica de los resultados del *benchmark* UnixBench.

En general, la máquina física obtiene un 15% más que la VM en las operaciones con datos de tipo entero, exceptuando el caso de las operaciones con datos de tipo *long* (enteros de 4 bytes), en el que la VM supera en un 11.5% a la máquina física. El motivo de este resultado puede ser la versión del sistema operativo, y en consecuencia, la posible distinta versión de la implementación de las operaciones con este tipo de datos.

En el caso de las operaciones con tipos de datos en coma flotante, la pérdida de rendimiento al emplear una máquina virtual sobre una máquina física supone un 19% en el caso de variables tipo *float* y un 15% en el caso de *doubles*.

5.3. BENCHMARKS SINTÉTICOS DE E/S

La otra limitación principal que sufren las aplicaciones es la Entrada/Salida (E/S). Los procesos que cuentan con requerimientos intensivos de E/S ocupan mucho de su tiempo en espera de operaciones de este tipo. Es por ello que este apartado se dedica íntegramente a analizar las sobrecargas que supone el empleo de una máquina virtual en contraste con una máquina física en lo que a operaciones de E/S supone, típicamente de lectura y escritura en disco. Siguiendo el patrón del subcapítulo anterior, en primer lugar se realiza una pequeña revisión del estado del arte en *benchmarks* sintéticos de E/S, para seleccionar con criterio el que se empleará en el estudio que se presenta posteriormente. Seguidamente se exponen las pruebas que se van a llevar a cabo para finalmente, presentar los resultados obtenidos.

5.3.1. ESTADO DEL ARTE

Al igual que con los *benchmarks* de CPU, se ha realizado un estudio del estado del arte sobre los *benchmarks* disponibles a día de hoy capaces de medir las prestaciones de las operaciones de E/S.

IOzone [77] es un *benchmark* implementado en el lenguaje C, capaz de proporcionar información sobre el rendimiento de las operaciones *read, write, re-read, re-write, read backwards, read strided, fread, fwrite, random read, pread ,mmap, aio_read, aio_write* sobre la memoria del sistema. Además, presenta la información en un formato que luego puede ser procesado por programas capaces de generar gráficas con los resultados obtenidos, como Microsoft Office Excel o *gnuplot*, facilitando la tarea de procesado de los resultados al usuario.

Otro *benchmark* empleado para obtener las prestaciones de una máquina en cuanto a operaciones de E/S se refiere, es Bonnie [78], centrado en obtener el rendimiento de las operaciones sobre el sistema de ficheros de UNIX. Este *benchmark* lleva a cabo una serie de test con ficheros de tamaño conocido, proporcionando para cada una de las pruebas informes de los bytes procesados por segundo, por segundo de CPU y el uso en tanto por ciento de la CPU. Existe una versión desarrollada en C++, Bonnie++ [79], capaz de evaluar las prestaciones con ficheros de mayor tamaño que los que acepta originalmente Bonnie. IObench [80] es una herramienta que sirve para estresar las operaciones de E/S. Durante la ejecución, IObench proporciona informes de sus estadísticas internas, ayudando a entender los límites de subsistema de E/S.

También existen *benchmarks* específicos para la computación paralela, como FLASH I/O [82], que utiliza grandes datos numéricos almacenados en el formato HDF-5 (*Hierarchical Data Format*, Formato de Datos Jerárquico) o IOR [83], un test capaz de evaluar las prestaciones de las operaciones de lectura y escritura sobre uno o varios ficheros de forma paralela empleando MPI-IO. Este tipo de test no han sido abordados en este estudio, pero en [81] se incluye una lista de los más relevantes en este campo.

5.3.2. PRUEBAS Y PARÁMETROS A MEDIR

Para el estudio del rendimiento de las operaciones de entrada/salida y la diferencia existente entre la ejecución de este tipo de instrucciones en una máquina física con respecto a una virtualizada, se ha empleado el *benchmark* IOzone, por ser el test que mejor se adaptaba a los objetivos iniciales. Este *benchmark* proporciona los resultados de las operaciones más habituales de E/S (como leer o escribir un fichero) en KB/s (Kilobytes por segundo), permitiendo variar el tamaño del fichero (de 64 KB a 512 MB) y el tamaño de bloque (de 4KB a 16 MB) de forma que cada incremento supone el doble del valor anterior. El tamaño de bloque indica el número de bytes del fichero a los que se accede de forma simultánea en cada operación.

Los resultados se pueden recoger en un fichero en formato *.wks*, donde se representa la información obtenida de forma que se pueda realizar una gráfica de los mismos. Un uso más extendido de este *benchmark*, además de su compilación, se encuentra recogido en su guía de uso [84]. En nuestro caso, se va a realizar una ejecución completa del *benchmark*, incluyendo todos los test que ofrece la versión más reciente en el momento de la realización de las pruebas, IOzone 3_373.

Para su empleo basta con compilar los ficheros fuente con el compilador *make*, indicándole el tipo de computador sobre el que vamos a hacer la ejecución, en este caso *linux-ia64*. En la ejecución se indica que genere un fichero *output.wks* con los datos calculados y que realice la ejecución automática de todos los test, tal y como se muestra a continuación:

```
user@ubuntu:~/iozone3_373/src/current$ ./iozone -Rab output.wks
```

5.3.3. RESULTADOS

Tras la ejecución completa de la versión 3_373 del *benchmark* IOzone, los resultados obtenidos para cada operación medida son los siguientes:

- **Lectura *(read)***: el rendimiento de la operación de lectura sobre un fichero ya existente supone, en general, un aumento del mismo que llega hasta el 50% empleando una VM, en contraste con la máquina física. Esto ocurre en casos concretos como con un tamaño de fichero de 16 MBytes y un bloque de 4 MBytes (este hecho puede observarse en la Figura 5.3). Pero a partir de un tamaño de fichero superior a 256 MBytes, las prestaciones se degradan de forma visible.

- **Escritura (*write*)**: la escritura de un nuevo fichero supone una pérdida de prestaciones del orden de un 35% para tamaños de fichero que oscilan entre 64 KBytes y 32 MBytes. A partir de 64 MBytes se observa un decaimiento de las prestaciones que llega hasta un 99% en el caso de un fichero de tamaño 512 MBytes y un tamaño de bloque de 2 MBytes, como se puede observar en la gráfica que recoge la Figura 5.4.

- **Re-lectura (*re-read*)**: en esta operación, se puede observar que para tamaños de bloque de 4 y 8 KBytes, la máquina física es, en general, un 15% más eficiente que la virtual. Pero a partir de 16 KBytes, la VM comienza a obtener mejores prestaciones, alcanzando el 50% más que la máquina física, exceptuando un tamaño de fichero de 512 MBytes, donde las prestaciones decaen un 85%.

- **Re-escritura (*re-write*)**: a la hora de escribir en un fichero ya existente, los resultados que se obtienen son similares para ambas máquinas, llegando incluso a obtener un rendimiento superior por la VM, en contraste con la operación de escritura, donde claramente la máquina física obtenía mejores prestaciones. Cuando se alcanzan tamaños de fichero de 64 MBytes, las prestaciones disminuyen hasta un 99% en la VM.

- **Lectura aleatoria (*random read*)**: si la operación que se lleva a cabo es un acceso a una posición aleatoria en un fichero para realizar una lectura,

los resultados obtenidos señalan que la máquina física es superior con bloques de 4 KBytes y tamaños de fichero pequeños (hasta 4MBytes). A partir de estos valores, los resultados indican que las prestaciones de la VM aumentan en comparación con la máquina física, llegando a alcanzar un 50% para ficheros de tamaño 128 MBytes y bloques de 16 MBytes. Con ficheros de 512 MBytes sigue habiendo una diferencia importante entre VM y máquina física, en detrimento de la primera.

- **Escritura aleatoria (*random write*)**: en esta operación se obtienen los resultados más parejos de todo el test, aunque se presenta un mayor número de casos en los que la máquina física supera a la virtual con, alrededor de un 20% más de rendimiento. De nuevo, se observa una caída significativa de prestaciones a partir de ficheros de 64 MBytes en la VM.

- **Lectura al revés (*read backwards*)**: en esta operación, donde se lleva a cabo la lectura de un fichero de final a principio (empleado en aplicaciones como MSC Nastran [98], un programa de cálculo estructural que aplica el método de los elementos finitos sobre ficheros de gran tamaño) la máquina virtual supera por hasta un 50% a la física (salvo en casos puntuales que pueden ser debidos a que haya habido otra operación concurrente en la máquina), hasta alcanzar un tamaño de fichero de 512 Mbytes, donde de nuevo las prestaciones se degradan notablemente.

- **Lectura empleando la función *fread***: nuevamente, la máquina virtual es superior en lo que a operaciones de lectura se refiere. En este caso el porcentaje alcanza el 50% en casos concretos como cuando se dispone de un fichero de 16 MBytes y un tamaño de bloque del mismo valor, donde se alcanzan 3043539 KB/s en la VM. En esta operación también

se puede apreciar la decaída de prestaciones que sufre la VM con ficheros de tamaño 512 MBytes.

- **Escritura empleando la función *fwrite***: mediante el empleo de la función de librería *fwrite* para escribir en un fichero, las diferencias entre ambas máquinas se reducen en comparación con la escritura normal. Con tamaños de bloque que oscilan desde los 4 KBytes hasta los 2 MBytes, la máquina física es superior a la virtual, en un porcentaje que ronda el 15%. A partir de tamaños de bloque superiores, la VM supera a la física. Nuevamente, la VM obtiene unas prestaciones muy bajas a partir de ficheros con peso de 64 MBytes en adelante.

- **Lectura *strided* (*read strided*)**: en este tipo de lectura, la VM es sobre un 35% más eficiente con respecto a la máquina física. Como repetidamente ocurre, al alcanzar valores de 512 MBytes en lo que a tamaño se refiere, las prestaciones sufren una decaída importante.

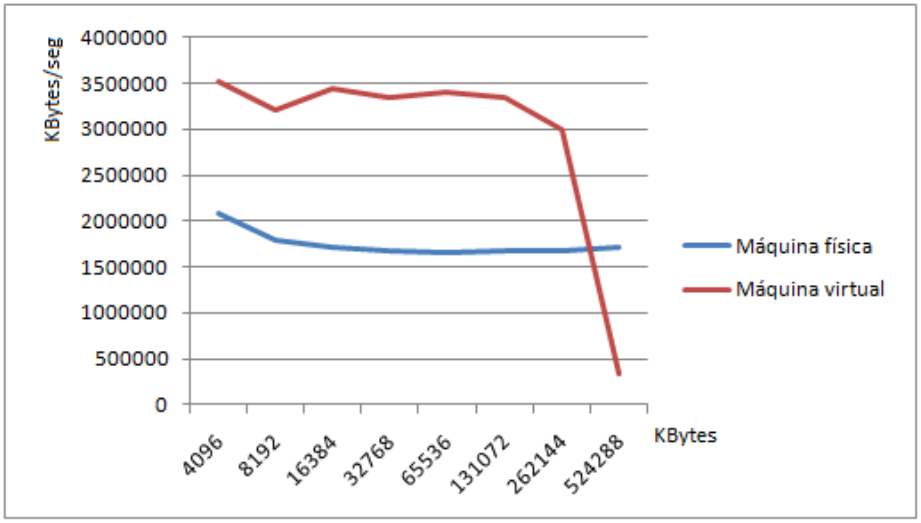

Figura 5.3 – Gráfica de los resultados del *benchmark* IOzone (*read*, tam. bloque = 4 MB).

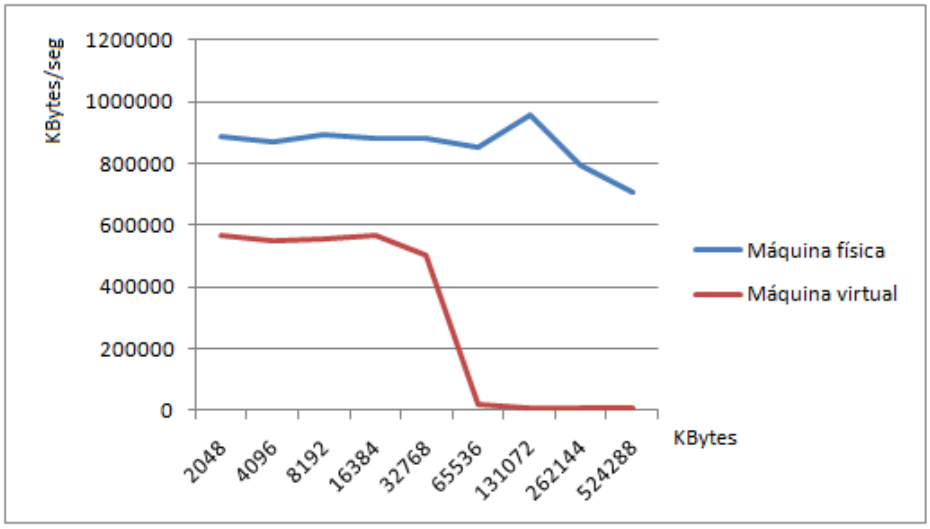

Figura 5.4 – Gráfica de los resultados del *benchmark* IOzone (*write*, tam. bloque = 2 MB).

Resumiendo los resultados obtenidos, en general las operaciones de escritura obtienen mejor rendimiento en la máquina física. Todo lo contrario ocurre con las operaciones de lectura, donde la máquina virtual presenta mejores resultados. Las causas de este comportamiento pueden radicar en el hecho de que al cargar la imagen de la máquina virtual sobre la máquina física, también se cargue información a la que se accede posteriormente, evitando tener que acceder a disco en las operaciones de lectura. En cambio, en las operaciones de escritura es estrictamente necesario acceder a disco, de ahí las diferencias obtenidas. Además, esta conclusión cobra fuerza al analizar que la operación de re-escritura llega a obtener unos resultados muy similares en ambos entornos (tanto el físico como el virtual) debido a que el fichero ya se había cargado en memoria previamente.

Asimismo, se observa un decrecimiento importante de prestaciones en la VM cuando se alcanzan tamaños de fichero de 64 MBytes en el caso de las operaciones de escritura, y de 512 MBytes en lo que a operaciones de lectura se refiere. Este comportamiento se repite en la mayoría de las operaciones analizadas, y puede ser debido a las configuraciones del hipervisor KVM, que permite limitar el tamaño de bloque y de disco que emplearán las VM desplegadas bajo este hipervisor. Además,

las configuraciones del hipervisor permiten emplear o no la memoria caché y decidir la política de escritura (*write-back* o *write-through*), configuraciones que pueden repercutir sobre los resultados obtenidos.

Los resultados mostrados en esta memoria se han repetido un número adecuado de veces como para catalogarlos de consistentes, luego el comportamiento aquí mostrado se corresponde con la realidad y no con una conducta anómala sucedida bajo una situación inesperada. Aun así, con el objetivo de contrastar los resultados obtenidos, se ha revisado el estado del arte en benchmarking sobre virtualización, centrando la atención en los trabajos realizados sobre el hipervisor KVM. Muchos de los trabajos existentes analizan el rendimiento de varios hipervisores mediante pruebas de CPU, de E/S y de red. En [99] y en [100] se muestran gráficas obtenidas con el *benchmark* IOzone para las operaciones de lectura y escritura. Ambos trabajos limitan los resultados mostrados en las gráficas para unos valores concretos de tamaño de fichero y de bloque, que no alcanzan los valores mostrados en esta memoria para los cuales se obtienen las peores prestaciones. Aun así, los autores destacan de ambos trabajos destacan la caída de prestaciones con respecto a la máquina física en las operaciones de escritura. Más concretamente, los autores de [100] destacan que "KVM expone una gran reducción en el ancho de banda de memoria en escrituras" y que "KVM sufre una tremenda caída de 6 a 7 veces en comparación con la máquina física", afirmaciones que justifican argumentando que la MMU (*Memory Management Unit*, Unidad de Gestión de Memoria) de este hipervisor "emplea un método de virtualización de la memoria que se asemeja virtualización completa, que debe pre-escanear el código que será ejecutado para reemplazar a las sensibles instrucciones privilegio".

Otros trabajos, como [101], muestran que KVM es el hipervisor que peores resultados ofrece de los tres analizados (KVM, Xen y OpenVZ). En [102] los autores afirman que "KVM presenta un rendimiento inesperadamente pobre en las pruebas de disco y red.", y en [103] se resaltan los buenos resultados para las operaciones de lectura y lectura aleatoria que obtiene el *benchmark* IOzone, en contraste con los

obtenidos para las operaciones de escritura y escritura aleatoria. Ninguno de estos tres trabajos destaca los tamaños de fichero y bloque empleados para obtener estas conclusiones, ni siquiera en las gráficas que muestran.

Por último, en el trabajo recogido en [104], también se destaca la caída de prestaciones de KVM que en las operaciones de escritura alcanza hasta a un 60% - 70% de decrecimiento en el rendimiento. En este caso sí que se señalan los tamaños de bloque y fichero empleados, que no alcanzan los 512 MB mostrados en este trabajo.

En conclusión, la revisión del estado del arte demuestra que los autores han experimentado comportamientos similares a los obtenidos en esta tesis de máster, y que las gráficas expuestas en los trabajos de la literatura no alcanzan a mostrar los resultados para los valores concretos de tamaño de fichero y bloque para los que se observa la degradación de las prestaciones, como si se ha hecho en esta memoria. Aun así, los resultados de los test que miden el rendimiento de las máquinas, ya sea sobre operaciones de CPU, de E/S, o de cualquier otra característica, se encuentran condicionados por muchos factores y son fácilmente susceptibles a la variabilidad.

6. CASO DE ESTUDIO

Con el objetivo de demostrar el funcionamiento de los modelos teóricos propuestos en el capítulo 3, a continuación se expone un caso de estudio que analiza la ejecución sobre un entorno híbrido Grid/Cloud de una aplicación de optimización de proteínas de propósito específico que emplea técnicas de computación de altas prestaciones. Para ello, en primer lugar se analizan, una a una, las diferentes infraestructuras empleadas para el desarrollo y realización de las pruebas que se presentan en el presente capítulo. Tras ello se exponen las principales características de la aplicación seleccionada, para proseguir con el comportamiento de la aplicación bajo los distintos modelos, definiendo previamente las pruebas a realizar y presentando posteriormente los resultados obtenidos.

6.1. INFRAESTRUCTURA EMPLEADA

En este subcapítulo se analizan las diferentes máquinas que se han empleado para llevar a cabo la ejecución del caso de estudio, proporcionadas por el Grupo de Grid y Computación de Altas Prestaciones (GRyCAP) de la Universidad Politécnica de Valencia (UPV).

Por un lado, la infraestructura que dará soporte al despliegue de máquinas virtuales es un clúster, denominado Kefren, que consta de 20 nodos biprocesadores Pentium Xeon a 2 Ghz, interconectados mediante una red SCI con topología de Toro 2D en malla de 4x4. Cada nodo consta de 1 Gigabyte de memoria RAM. Uno de estos nodos actúa como *front-end*, que es el punto de entrada al clúster, y consta de 4 discos duros Ultra160 SCSI de 36 Gigabytes a 10000 rpm *hot-plug*. El resto de nodos constan de un disco duro IDE de 40 Gigabytes a 7200 rpm. El sistema operativo del clúster es

CentOS 5.2. Este clúster de PCs tiene instalada la versión 2.2 completa de OpenNebula, gestor de máquinas virtuales que se empleará en este caso de estudio y que determinará en qué nodo se lanza cada VM. Cuenta con la instalación de VMware Server 2.0.2 como hipervisor. Cabe señalar que este clúster emplea técnicas de *Green computing* [85], con lo que no todos los nodos se encuentran activos en todo momento.

Por otro, la infraestructura Grid está formada por un clúster, denominado Odín, de 50 nodos biprocesadores Intel Xeon CPU 2.80GHz, interconectados mediante una red SCI con topología de Toro 2D en malla de 10x5. Cada nodo consta de 2 Gigabytes de memoria RAM. Uno de estos nodos es el encargado de actuar como *front-end* dando acceso al clúster desde el exterior. El sistema operativo que corre en esta máquina es Linux CentOS 5.3 con una versión de *kernel* 2.6.18. Cuenta con la instalación completa de Globus Toolkit 4, versión 4.2.1, y la implementación NMPI 1.3.3.1 del estándar MPI, optimizada para la red de comunicaciones SCI. Al igual que Kefren, esta máquina también emplea técnicas de *Green Computing*, luego los nodos se encenderán bajo demanda del usuario.

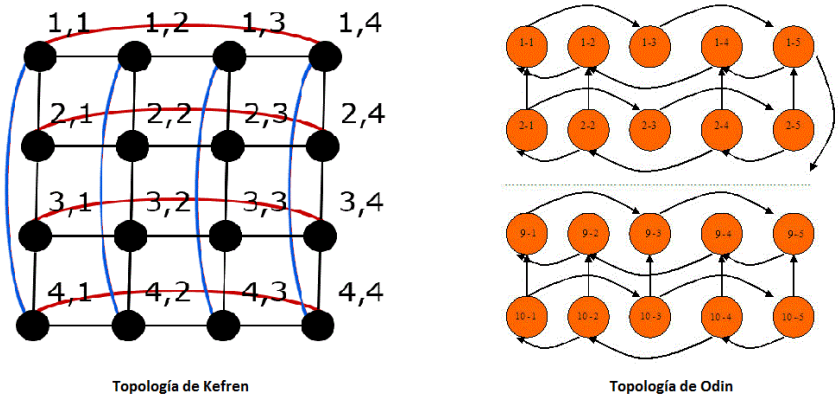

Figura 6.1 – Topología de las máquinas empleadas en el caso de estudio.

6.2. LA APLICACIÓN GBIOBJ

La aplicación elegida para llevar a cabo el caso de estudio, que recibe el nombre de *gBiObj* [86], realiza la exploración del espacio de búsqueda de conformaciones estructurales con diferentes semillas aleatorias para obtener diferentes configuraciones de la proteína en base a sus aminoácidos constituyentes. El código está basado en métodos de Monte Carlo y utiliza dos objetivos (plegado y enlace). La aplicación *gBiObj* es computacionalmente costosa ya que realiza decenas de miles de iteraciones para explorar el espacio de posibles conformaciones estructurales (rotámeros), lo que le hace requerir de gran capacidad de cómputo, y el gran tamaño de las matrices de energía que emplea para llevar a cabo los cálculos, condicionan el tamaño de la memoria necesario para conseguir una ejecución eficiente. Por ello, su coste computacional puede superar los recursos de una única organización, dando pie al uso de una infraestructura híbrida para su ejecución.

Más concretamente, la aplicación funciona de la siguiente forma: *gBiObj* realiza la optimización de una proteína sintética de 1000 posiciones, para este caso de estudio. En una iteración dada, cada posición recibe un rotámero de una lista de posibles rotámeros candidatos que podrían ocupar esa posición. Un rotámero es la estructura espacial en tres dimensiones de un aminoácido y su proceso involucra el empleo de unas matrices pre-computadas de interacción energética entre los rotámeros que guía el procedimiento de optimización en la proteína más estable. Por lo tanto, este es un proceso iterativo que evalúa las diferentes estructuras de la proteína tratando de llegar a una configuración de rotámeros con la mínima energía global. Cuanto menor es la energía, más estable es la proteína. Este enfoque estocástico para el diseño de proteínas por lo general converge hacia la estructura de la proteína más estable después de cientos de miles de iteraciones.

Con respecto a las dependencias de la aplicación, cabe decir que *gBiObj* está escrita en C, luego será necesario contar con un compilador C en la máquina destino. Este requisito es muy fácil de suplir bajo entornos Linux, el entorno que deberá ser empleado ya que también depende de librerías estándar del sistema. Como

dependencias externas, *gBiObj* precisa de las librerías MPI para su correcto funcionamiento. Este es un requisito más difícil de encontrar, aunque bien es cierto que los recursos disponibles en el Grid, al estar destinados a la investigación científica, suelen contar con este tipo de librerías. En cambio, sí es conveniente realizar la compilación de la aplicación en remoto, sobretodo en el caso de la ejecución en Grid, garantizando una mayor probabilidad de éxito de la misma, ya que existen muchas implementaciones del estándar MPI. Otra opción sería emplear ejecutables estáticos auto contenidos, pero el tamaño del ejecutable y la ausencia de garantías acerca de que la ejecución se va a realizar con éxito, decantan el uso de la compilación en remoto.

6.3. ANÁLISIS DE LAS SOBRECARGAS SOBRE EL TIEMPO DE EJECUCION

En este apartado se pretende analizar la diferencia en cuanto a tiempo de ejecución se refiere entre la ejecución de la aplicación *gBiObj* en un recurso Grid frente a un recurso Cloud. Se va a prestar especial atención a la desigualdad que se presenta entre el tiempo que consume la aplicación en ser ejecutada dentro del propio recurso y el tiempo total de ejecución que se observa desde el metaplanificador, donde afectan las sobrecargas que suponen ambas infraestructuras y la latencia del propio metaplanificador en detectar que la ejecución ha finalizado.

La Tabla 6.1 recoge los resultados del estudio de las sobrecargas que supone la ejecución de la aplicación en los dos tipos de infraestructuras estudiadas (Grid y Cloud). La ejecución de *gBiObj* se ha realizado con un valor fijo de iteraciones globales, *ngiters = 1*, y un valor variable de las iteraciones locales (*nliters*), lo que supone que se convierta en una aplicación de tipo *CPU-bound*. Como se puede observar, en la tabla se muestran dos valores para cada número de iteraciones locales: el primero se corresponde al valor total del tiempo de ejecución que observa el metaplanificador, con las sobrecargas propias de cada infraestructura (por ejemplo en el caso del Cloud la interacción con Opal y en el caso del Grid, el tiempo de colas). El segundo es el

tiempo que se consume en realizar la ejecución de la aplicación en el propio recurso, sin sobrecarga alguna, recibido en el propio fichero de salida de *gBiObj*. Ambos tiempos se encuentran expresados en segundos.

Infraestructura		**20000**	**40000**	**80000**	**160000**	**320000**	**640000**	**1280000**	**2560000**	**5120000**
Grid	**sin sobrecarga**	2.945	5.627	10.942	21.876	43.408	86.725	173.059	343.270	684.196
	con sobrecarga	13.013	13.201	23.486	33.679	54.629	96.166	179.099	355.666	697.779
Cloud	**sin sobrecarga**	4.082	7.955	15.841	31.079	62.142	122.705	246.414	486.437	979.368
	con sobrecarga	6.816	17.510	37.699	37.637	77.701	137.955	258.325	498.882	1040.15

Tabla 6.1 - Análisis de las sobrecargas sobre el tiempo de ejecución para diferentes valores de iteraciones locales.

Estos mismos resultados se pueden expresar en formato de gráfica (Figura 6.2), donde se puede observar fácilmente y de forma visual, que la ejecución en un recurso Grid es más rápida que en un recurso Cloud. Ello es debido a que, como se ha presentado en párrafos anteriores, los nodos Grid cuentan con características superiores a los nodos Cloud y además, a estos últimos hay que restarles las sobrecargas que suponen la virtualización, estudiadas en el capítulo 5.

Analizando detalladamente los resultados se puede extraer una conclusión importante: la ejecución en un recurso Cloud es más conveniente para trabajos pequeños, mientras que para trabajos computacionalmente más complejos es mejor emplear un recurso Grid. Ello es debido principalmente a la capacidad superior de los recursos Grid, pero también a que las sobrecargas de esta infraestructura se ven amortizadas cuanto más elevado sea el tiempo de ejecución de los trabajos (pasando de una sobrecarga del 77% para un trabajo cuyo tiempo de ejecución es muy pequeño (*nliters* = 20000), causada principalmente por el tiempo de colas, a un 2% para tareas que rondan los 12 minutos de duración (*nliters* = 5120000)). Esta conclusión cobra especial importancia bajo el modelo 3, donde se pretende balancear la carga de los

trabajos que, como se acaba de poner de manifiesto, interesa que los trabajos más pequeños caigan en un recurso Cloud y los más intensos computacionalmente, se ejecuten en un recurso Grid.

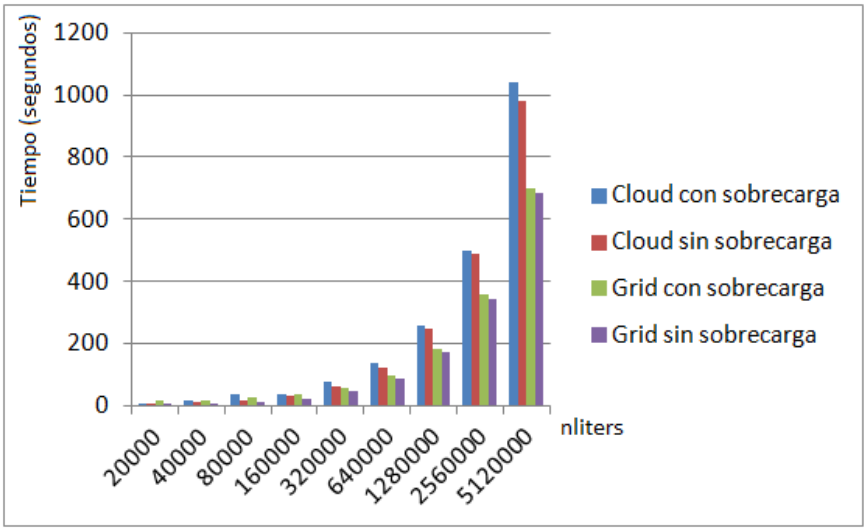

Figura 6.2. - Análisis de las sobrecargas sobre el tiempo de ejecución.

En el caso de las ejecuciones sobre los recursos Cloud, las sobrecargas principalmente se deben a la latencia del metaplanificador en detectar la finalización de la ejecución del trabajo, ya que es éste el encargado de consultar mediante Opal el estado del mismo, acción que se lleva a cabo cada cierto tiempo. Es por ello que dependiendo de la ejecución analizada, la diferencia entre ambos tiempos de ejecución discrepa de forma variable. La consulta del estado del trabajo forma parte de las sobrecargas del Cloud, pero su tiempo es despreciable.

6.4. COMPORTAMIENTO BAJO EL MODELO 1

Con el objetivo de demostrar el comportamiento del modelo 1, que pretende el uso de infraestructuras Cloud cuando la infraestructura Grid se ve superada en cuanto a carga de trabajos se refiere, en este apartado se propone la ejecución de la aplicación

gBiObj que facilitará el análisis del comportamiento de este modelo bajo un entorno real y las principales ventajas que aporta. Para ello en primer lugar se definen las pruebas que se van a llevar a cabo y finalmente se presentan los resultados obtenidos.

6.4.1. DEFINICIÓN DE LAS PRUEBAS

Bajo el modelo 1 se plantean fundamentalmente dos tipos de pruebas. La primera de ellas involucra solamente el uso de la infraestructura Grid y de la infraestructura Cloud privada. En la segunda se pretende ampliar el número de recursos accediendo para ello a una infraestructura Cloud pública, proporcionada por Amazon EC2, que amplíe el campo de acción del modelo.

Como aspectos comunes a ambos tipos de pruebas cabe destacar, en primer lugar, que para simplificar la exposición de los resultados, se asume que cada tarea se va a ejecutar en un nodo de cómputo. Así, podremos albergar tantas tareas concurrentes como el número de nodos que tenga la infraestructura híbrida, llegando a saturarla cuando no existan nodos disponibles. El resto de tareas se irán procesando conforme se vayan liberando los recursos de cómputo. Además, cada VM desplegada contará con una sola CPU y todas las VMs serán iguales para cada infraestructura Cloud, es decir, las VMs disponibles en el Cloud privado serán iguales entre sí, al igual que pasará con los recursos proporcionados por proveedor de recursos Cloud públicos. Ello, sumado a que no contamos con mecanismos de monitorización para conocer el estado de las VMs (a diferencia del Grid que, mediante los servicios MDS o BDII, puede proporcionar esta información), implica el uso de políticas de *Round-robin* para determinar qué recurso Cloud dentro de una infraestructura determinada es el más apropiado para la ejecución.

Para el despliegue de las VMs que albergarán la ejecución de *gBiObj* en la infraestructura Cloud privada, se va a emplear una VMI que ya contiene instalada una versión del estándar MPI, en concreto MPICH2, y la propia aplicación a falta de

instalar y configurar todas las dependencias necesarias para poder envolverla en un servicio web mediante Opal. Esta imagen se encuentra en el catálogo VMRC al que ya se hizo referencia anteriormente. En cuanto al Cloud público, la estrategia seguida es muy similar, ya que se ha construido previamente una AMI (*Amazon Machine Image*, Imagen de Máquina de Amazon) con las mismas características software que la VMI del Cloud privado y se ha registrado en el catálogo de imágenes de Amazon EC2 (tal y como recoge el Anexo B), para poder emplearla en las ejecuciones posteriores y reducir el tiempo de contextualización. Nótese que si este paso no se hubiese llevado a cabo, se podría haber empleado una AMI ya existente y realizar sobre ella el proceso de contextualización completo, pero para evitar la instalación completa de MPICH2 cada vez que se lanza una nueva VM se ha considerado oportuno seguir esta estrategia.

El número de VMs a desplegar vendrá determinado por el mínimo entre la capacidad de la infraestructura física y el número de tareas pendientes. Para un usuario, la capacidad de la infraestructura puede verse limitada a un número concreto de nodos. Por ello, el usuario debe indicar el número de nodos que puede emplear. Las máquinas permanecerán desplegadas hasta la finalización de la ejecución, evitando volver a repetir el proceso de despliegue y contextualización en un momento dado de la misma ejecución.

Aunque los recursos empleados para la ejecución del caso de estudio dispongan del número de nodos señalado en el apartado 6.1, en las distintas ejecuciones no se va a emplear la totalidad de ellos, ya que es lógico comprender que este tipo de recursos se encuentran compartidos entre distintos grupos y proyectos de investigación y es necesario respetar las necesidades de todos. Es por ello que, en la primera prueba se va a emplear un subconjunto de nodos de ambas máquinas que, en el caso de la infraestructura Grid serán 10 nodos y en el caso del Cloud privado, 4. Para el segundo caso, se emplearán los mismos recursos que en la primera prueba pero además se podrá acceder a la infraestructura proporcionada por Amazon EC2 en el momento necesario, pudiendo desplegar hasta 4 VMs.

El número de tareas en ambos casos de estudio será de 30. Cada tarea se ejecutará con una semilla aleatoria diferente, pero la duración de cada una de ellas será muy similar, empleando un valor de *nliters* igual a 2560000.

6.4.2. RESULTADOS: ACCESO A CLOUD PRIVADO

En este apartado se analizan los resultados obtenidos para la ejecución de *gBiObj* en el entorno híbrido descrito anteriormente y bajo las limitaciones y simplificaciones expuestas. Por un lado, se presentan los resultados obtenidos para la primera prueba, que involucra a las infraestructuras Grid y Cloud privado en la que, como se ha comentado anteriormente, se han empleado 10 nodos de la infraestructura Grid, permitiendo albergar como máximo 10 trabajos simultáneos, y 4 nodos de la infraestructura Cloud, que permite desplegar 4 VMs y en consecuencia, atender a 4 trabajos simultáneos. La ejecución completa consta de 30 tareas que requieren el mismo tiempo de cómputo para ser ejecutadas.

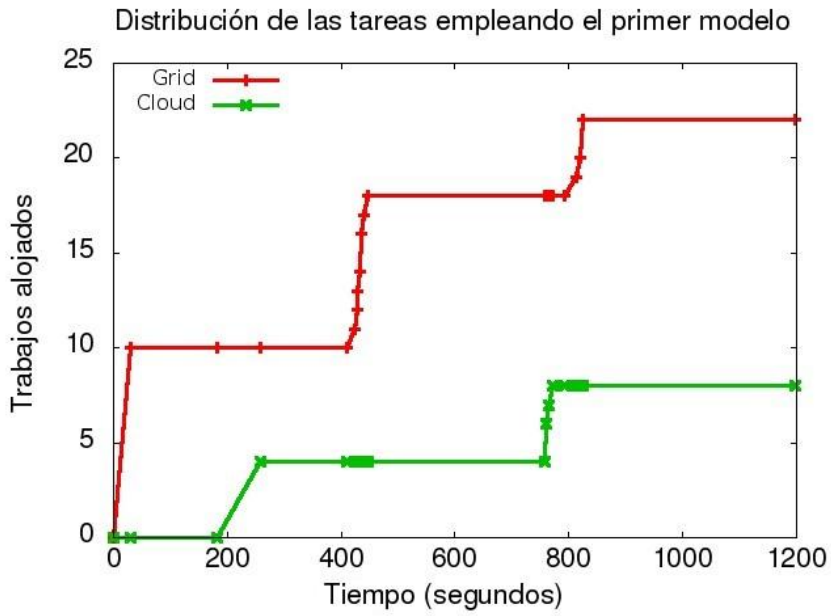

Figura 6.3 - Distribución de las tareas empleando el primer modelo sobre dos infraestructuras.

La Figura 6.3 muestra los resultados obtenidos, que se analizarán a continuación de acuerdo a los intervalos señalados en la descripción teórica del modelo para facilitar la compresión de su comportamiento. Estos resultados ponen de manifiesto que la infraestructura Grid es más potente en cuanto a capacidad de cómputo ya que la aplicación *gBiObj* consume unos 6 minutos de ejecución en un recurso Grid (G_2) y unos 8 en un recurso Cloud (C_3). Cabe recordar que los intervalos G_2 y C_3 se corresponden con el tiempo de ejecución del conjunto de las tareas enviadas a una infraestructura determinada. Sin embargo, dado que los trabajos son ejecutados al mismo tiempo, los valores representan también el tiempo de ejecución de una sola tarea en cada infraestructura. Esta diferencia de tiempo implica que, junto con el hecho de que el entorno Grid cuenta con mayor capacidad para albergar trabajos, gran parte de las tareas se ejecuten en esta infraestructura (en concreto, 22 tareas se han completado en esta infraestructura). Pero el uso del Cloud en los momentos de saturación de la infraestructura anterior sirve para acelerar la ejecución total de las tareas, ya que es capaz de albergar la ejecución de 8 tareas en momentos de saturación del Grid.

Analizando detalladamente cada intervalo de tiempo señalado en la Figura 3.5, podemos contrastar el funcionamiento teórico del primer modelo con el real. Así, cabe señalar que el intervalo de inicialización del proceso de planificación (A) es despreciable, ya que el metaplanificador es capaz de ponerse en marcha en poco menos de un segundo.

Centrando la atención en los intervalos que afectan a la infraestructura Grid, se puede ver que la ejecución real muestra un comportamiento muy similar a la aproximación teórica. Se puede destacar que el intervalo G_3, encargado de asignar los trabajos al Grid y transferir los ficheros necesarios para que los recursos queden preparados para la ejecución de las tareas, pierde peso en la ejecución real al cobrar más importancia el intervalo de la ejecución de tareas, G_2.

En cuanto al tiempo consumido por el despliegue de las VMs, se puede hacer un análisis exhaustivo para determinar en qué se emplea cada instante de tiempo. En

concreto, el intervalo de detección de la saturación del Grid y el despliegue de las VMs (C_1) se producen las siguientes etapas:

- **Detección de la saturación de la infraestructura Grid**: los primeros instantes de tiempo se emplean en esta etapa, donde se consulta al sistema de información Grid (MDS en este caso) con el fin de encontrar un recurso disponible. El tiempo es depreciable (0.1 segundos).

- **Lanzamiento de la VM**: esta etapa abarca el tiempo transcurrido desde que se lanza la VM mediante el gestor de máquinas virtuales hasta que ésta alcanza el estado *running*. El tiempo consumido ronda los 120 segundos, durante los cuales se copia la VMI desde el *front-end* del clúster a los nodos internos y posteriormente desplegada. Dependiendo de si el nodo donde se va a desplegar la VM está encendido o no, este período puede aumentar.

- **Activación de SSH**: la activación del servicio SSH es necesaria para realizar la posterior fase de transferencia de ficheros de entrada al recurso remoto. El tiempo empleado en ello y en que el metaplanificador se dé cuenta ronda los 30 segundos.

El intervalo que se corresponde con la asignación de los trabajos al Cloud, C_4, está formado por dos etapas destacables, como son:

- **Transferencia de ficheros de entrada**: en esta etapa tiene lugar el envío de los ficheros necesarios para la contextualización y posterior ejecución del contextualizador. Su duración depende del estado de la red en el momento de la transferencia, pero no suele superar los 2 segundos.

- **Contextualización**: la última acción a llevar a cabo sobre una VM para dejarla preparada para la ejecución de la aplicación es la

contextualización. En esta fase, se instalará Tomcat, Ant y Opal, además de opalizar la aplicación, consumiendo unos 80 segundos.

Cabe señalar que, una vez determinado el número de VMs necesarias para la ejecución, éstas son puestas en marcha y contextualizadas en paralelo, para solapar el tiempo de despliegue de cada máquina y reducir el intervalo de tiempo destinado a este fin. Además, una vez arrancadas, las VMs se mantienen en funcionamiento hasta el fin de la ejecución. Es por ello que al hacer uso por segunda vez de la infraestructura Cloud (en el instante 780), el intervalo de tiempo C_1 ya no aparece puesto que las VMs ya están en marcha, y el intervalo C_4 se ve reducido a menos de la mitad debido a que las VMs ya están contextualizadas y preparadas para la ejecución y simplemente se realiza una comprobación de que este hecho se cumpla.

El coste de oportunidad o C_0, es lo suficientemente grande como para que el planificador se plantee la delegación de tareas al Cloud, ya que el tiempo de contextualización de las máquinas virtuales es relativamente pequeño en comparación con el tiempo invertido en la ejecución de los trabajos.

Un aspecto importante a destacar es el tiempo que permanecen ambas infraestructuras a pleno rendimiento (*SAT*), ya que es síntoma de que se están aprovechando al máximo los recursos, y en consecuencia se está llevando a cabo una buena planificación. Además, es obvio que si la ejecución de las tareas se hubiera realizado sobre una sola infraestructura, el tiempo total de ejecución hubiese sido superior al obtenido. Esto se puede extrapolar a casos de uso en los que se requiera ejecutar un número mayor de trabajos, donde los beneficios serían aún mayores.

6.4.3. RESULTADOS: ACCESO A CLOUD PRIVADO Y PÚBLICO

En este apartado se analiza la segunda prueba realizada que involucra los tres tipos de infraestructuras estudiadas: Grid, Cloud privado y Cloud público. Cabe recordar que se han empleado 10 nodos de la infraestructura Grid y 4 VMs de la

infraestructura Cloud privada, al igual que en el caso anterior, pero además se va a disponer de acceso a cuatro recursos proporcionados por el proveedor de Cloud público Amazon EC2. Al igual que en el caso anterior se han ejecutado 30 tareas de similar duración.

Antes de analizar los resultados es imprescindible destacar el tipo de instancias empleadas en Amazon EC2 y lo que ello implica. En esta ejecución se han empleado instancias *micro*, cuyas características se analizaron anteriormente en la Tabla 2.1. La elección de este tipo de instancias, las más sencillas y que menos prestaciones ofrecen, es debida a las restricciones que plantea el plan gratuito [87] ofrecido por Amazon, y empleado en la realización de la tesis de máster, ya que al ser un trabajo con fines académicos no se disponía de presupuesto. Este tipo de instancias ofrecen una pequeña cantidad continua de recursos de CPU y se adaptan a aplicaciones con un rendimiento bajo. En consecuencia, los resultados obtenidos para las 4 tareas recaídas en los recursos del Cloud público quedan afectados por este hecho. Sin embargo, estos mismos resultados sirven para demostrar el comportamiento del modelo 1 bajo tres infraestructuras distintas, siempre teniendo en cuenta que si se dispone de presupuesto, la ejecución en Amazon EC2 será más rápida y podrá albergar la ejecución de un mayor número de tareas, acelerando el tiempo total de ejecución del conjunto de ellas.

La Figura 6.4 muestra los resultados obtenidos empleando una infraestructura híbrida a tres niveles bajo el modelo de detección de saturación. La infraestructura Grid ha llevado a cabo la ejecución de 20 tareas, mientras que las infraestructuras Cloud se han encargado de las 10 restantes (6 la infraestructura Cloud privada y 4 la infraestructura Cloud pública).

Al igual que en el caso anterior, la detección de la saturación de los recursos Grid provoca el despliegue de la infraestructura Cloud privada (instante 31). La saturación de esta última, junto con el hecho de que la infraestructura Grid sigue saturada y quedan tareas por ejecutar, provoca el uso de los recursos proporcionados por el Cloud público (instante 258). Cabe destacar que, en este caso de estudio, realmente las

infraestructuras no se encuentran al máximo de su capacidad, pero sí los nodos a los que tiene acceso el usuario, luego bajo el punto de vista de la ejecución para el usuario, las infraestructuras se encuentran saturadas.

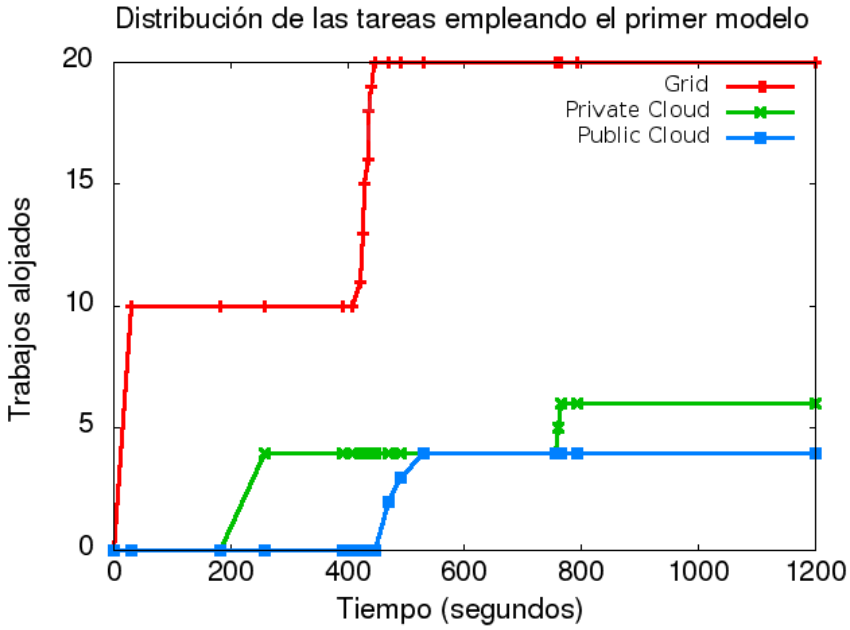

Figura 6.4 - Distribución de las tareas empleando el primer modelo sobre tres infraestructuras.

Teniendo en cuenta los inconvenientes que supone el uso de instancias *micro*, se pueden destacar dos aspectos relevantes inducidos por el uso de la tercera infraestructura. El primero de ellos afecta al tiempo de despliegue de una VM. En la gráfica es algo difícil de apreciar, pero el tiempo de despliegue de una máquina virtual en Amazon EC2 es más rápido que en OpenNebula, entre un 10% y 15%, incluyendo en este período la activación del servicio SSH. En cambio, al emplear instancias *micro*, la ejecución de la contextualización es más lenta en las máquinas de EC2 (en la gráfica, la contextualización de las AMIs de Amazon comienza en el instante 391 y, aunque la ejecución de la contextualización se realiza en paralelo, existe cierto desfase en su finalización debido a que las instancias *micro* funcionan con ciclos de reloj sobrantes de las máquinas de Amazon).

El segundo aspecto importante que la gráfica pone de manifiesto es el incremento del tiempo de ejecución de las tareas que se ejecutan en el Cloud público. Ello es debido de nuevo al tipo de instancia empleada, y hace duplicar e incluso triplicar en algunos casos el tiempo de ejecución de la misma tarea en una infraestructura u otra (aunque en la gráfica no se muestra por facilitar la visión del resto de datos, algunas tareas ejecutadas sobre EC2 llegaron a completarse consumiendo más de 20 minutos). Este inconveniente radica en la falta de presupuesto a la hora de hacer las pruebas, pero es importante destacar que si se dispusiese del mismo, este modelo ayudaría a reducir el tiempo de ejecución global de las tareas, aprovechando debidamente los recursos.

También cabe señalar que el coste de oportunidad (C_0) en el caso del Cloud privado frente al Grid es lo suficientemente amplio como para obtener ventajas en el tiempo de ejecución total de las tareas (al igual que pasaba en el apartado anterior). En el caso del empleo del Cloud público, el C_0 con respecto al Cloud privado es lo suficientemente elevado como para que el metaplanificador considere necesario el despliegue de la infraestructura Cloud pública, ya que el tiempo de despliegue y contextualización de las instancias se ve amortizado por el tiempo de ejecución de las tareas. Frente a la infraestructura Grid, este intervalo es bastante más reducido, debido a la duración de las tareas, pero es suficiente como para plantear el despliegue de la infraestructura Cloud pública. En un caso de estudio donde las tareas tengan una duración elevada, el C_0 pondrá en evidencia las ventajas del uso de una tercera infraestructura.

No se va a entrar en más detalle de análisis puesto que el comportamiento de esta segunda prueba es muy similar a la primera y en ella ya se explicó con detalle cada intervalo de tiempo.

6.5. COMPORTAMIENTO BAJO EL MODELO 3

Con el objetivo de demostrar el comportamiento del modelo 3, que pretende el uso combinado de infraestructuras Grid y Cloud para balancear la carga de los trabajos entre ambas, en este apartado se presentan los resultados tras la ejecución de la aplicación *gBiOhj* bajo la implementación de este modelo. Ello permitirá analizar el comportamiento del planificador que aplica este modelo bajo un entorno real y las principales ventajas que aporta el mismo. Al igual que en el apartado anterior, en primer lugar se definen las pruebas que se van a llevar a cabo para concluir presentando los resultados obtenidos.

6.5.1. DEFINICIÓN DE LAS PRUEBAS

Se pretende analizar el comportamiento de la implementación realizada en dos situaciones: cuando el número de tareas supera al número de recursos disponibles y cuando el número de recursos es suficiente como para albergar todas las tareas de una sola vez. Para ello, se va a disponer de un número fijo de nodos en la infraestructura Grid, y un número variable de máquinas virtuales en la infraestructura Cloud privada, que permitirá analizar el comportamiento del modelo sobre distintos escenarios. El número de tareas también será variable para poder estudiar el comportamiento de la implementación en las dos situaciones expuestas.

En este caso no se va emplear la infraestructura Cloud pública puesto que, aunque se ha desarrollado el soporte necesario para utilizarla, la desventaja que supone el uso de las instancias *micro* frente al resto de recursos computacionales hace que carezca de interés su análisis.

Al igual que en las pruebas con el modelo 1, se asume que cada tarea se va a ejecutar en una CPU, luego podremos albergar a tantas tareas concurrentes como el número de nodos que tenga la infraestructura híbrida. El resto de tareas se irán procesando conforme se vayan liberando los recursos de cómputo. Además, cada VM desplegada contará con una sola CPU y todas las VMs serán iguales. Para el

despliegue de las VMs que albergarán la ejecución de *gBiObj*, se va a emplear la misma VMI descrita en para las pruebas del modelo 1.

A diferencia del caso anterior, esta vez los recursos computacionales estarán desplegados y contextualizados previamente a la ejecución de las tareas, puesto que lo que interesa analizar no es el tiempo de despliegue y contextualización, que se ha podido observar anteriormente, sino el reparto de tareas que realiza el metaplanificador. Este reparto reproduce el comportamiento expuesto anteriormente en la sección 4.3 del capítulo 4.

6.5.2. RESULTADOS

En este apartado se analizan los resultados obtenidos para la ejecución de la aplicación empleada en el caso de estudio en el entorno híbrido descrito anteriormente y bajo las limitaciones y simplificaciones expuestas. La Tabla 6.2 muestra cinco ejecuciones distintas de la aplicación *gBiObj* con un valor de *nliters=2560000* para todas las tareas ejecutadas.

# tareas	**# nodos Grid**	**# VMs**	**Distribución de trabajos (Grid/Cloud)**	**Tiempo de ejecución (segundos)**
20	8	2	(16, 4)	1088,76
	8	4	(16, 4)	856,27
	8	8	(12, 8)	832,59
15	8	8	(8, 7)	538,64
10	8	8	(5, 5)	534,53

Tabla 6.2 - Distribución de trabajos empleando el modelo 3.

Antes de analizar los resultados, cabe recordar la duración aproximada de las tareas con el valor de *nliters* dado, ya que es fundamental para comprender los valores obtenidos. En el caso de una ejecución en bajo la infraestructura Grid, una tarea

consume alrededor de 360 segundos, mientras que si su ejecución se realiza en el Cloud privado, el tiempo consumido ronda los 500 segundos. Así, analizando los resultados obtenidos para un número fijo de 20 tareas, donde el número de recursos entre ambas infraestructuras en conjunto es menor al número de trabajos, se puede destacar la aceleración del tiempo total de ejecución de las tareas conforme el número de recursos disponibles para su ejecución aumenta. Éste es un comportamiento lógico además de predecible, pero que era uno de los objetivos que perseguía la implementación de este modelo.

Las dos primeras filas de la tabla muestran la ejecución de 20 tareas pudiendo acceder, en el primer caso, a 8 recursos Grid y a 2 Cloud; y a 8 recursos Grid y 4 Cloud para el segundo. El reparto de las tareas resulta ser el mismo para ambos casos, pero el tiempo de ejecución disminuye. Ello es debido a que, en el primer escenario, ambas infraestructuras realizan dos ciclos de ejecución de tareas (entendiendo un ciclo de ejecución de tareas como un bloque de ejecución de trabajos que abarca la totalidad de los nodos disponibles en la infraestructura), y el tiempo se ve penalizado por la infraestructura Cloud, menos potente computacionalmente. En el segundo, la infraestructura Cloud es capaz de ejecutar 4 tareas de forma concurrente y, aunque la infraestructura Grid realiza la ejecución del mismo número de tareas que antes, ya no se ve penalizada por el Cloud, que realiza un ciclo menos de ejecución.

La última ejecución con 20 tareas, con acceso a 8 recursos de cada infraestructura, obtiene un tiempo muy similar al anterior caso, en el que sólo habían 4 recursos Cloud disponibles. En este caso, el reparto de tareas es más equitativo que en el anterior, pero al superar el número de tareas al número de recursos resulta necesario realizar un segundo ciclo de ejecución en la infraestructura Grid, puesto que es la que antes libera sus recursos, y el tiempo total de ejecución queda afectado por este hecho. Estos resultados indican que no siempre un número mayor de recursos accesibles acelerará la ejecución de las tareas. Además, aunque el reparto de la carga es más equitativo, la utilización de los recursos disminuye en el segundo caso.

En el caso contrario, cuando el número de recursos disponibles supera al número de tareas a ejecutar, ambas ejecuciones (con 15 y 10 tareas) obtienen un tiempo muy similar, ya que ambos grupos de tareas pueden ejecutarse de forma concurrente en la infraestructura híbrida. Los resultados están influidos de nuevo por la capacidad de cómputo inferior que presentan los recursos Cloud. Este es el mejor caso para poder realizar un reparto equitativo de la carga sin influir de forma negativa en los criterios de utilización y rendimiento.

7. CONCLUSIONES

Para cerrar este documento, en este último capítulo se exponen las conclusiones que la autora extrae tras la realización de la tesina. También se incluye un apartado que repasa las publicaciones científicas que ha dado lugar el trabajo aquí expuesto.

7.1. RESUMEN Y TRABAJO FUTURO

Los objetivos planteados al comienzo del trabajo se han podido llevar a cabo de forma satisfactoria. El principal objetivo, diseño de modelos teóricos de metaplanificación híbrida Grid/Cloud, se ha conseguido mediante el desarrollo de tres modelos de integración de ambos tipos de infraestructuras capaces de minimizar los inconvenientes de ambos entornos impulsando sus potenciales ventajas. El desarrollo de los modelos se ha llevado a cabo teniendo en cuenta el estudio del estado del arte realizado, que ha puesto de manifiesto el trabajo realizado hasta ahora en el ámbito de la computación híbrida. Todo ello ha permitido construir unas herramientas de metaplanificación híbrida capaces de gestionar recursos de tres infraestructuras distintas (Grid, Cloud privado y Cloud público), y de aprovechar al máximo los recursos de que el usuario dispone con el objetivo de lograr el mejor tiempo de ejecución posible. La ampliación del campo de acción de las herramientas de metaplanificación al uso de una infraestructura pública como Amazon EC2, la más importante a nivel de IaaS, ha permitido analizar el soporte que se ofrece actualmente a cualquier usuario, lejos del ámbito científico y universitario, que desea acceder a los servicios que proporciona la tecnología del Cloud Computing.

La arquitectura diseñada permite abstraer al usuario de, prácticamente, todos los detalles de más bajo nivel que implica el uso de las infraestructuras Cloud a nivel de

IaaS. La selección de la VMI apropiada y la contextualización de los recursos se realizan de forma automática y transparente al usuario, quien sólo tiene que proporcionar los requisitos de su aplicación, pero no tiene que conocer detalles concretos como la elección del hipervisor. La abstracción del uso del Grid ya la proporcionaba originalmente GMarte.

Por todo ello, tal y como se ha podido apreciar a lo largo de esta memoria, la primera conclusión que se puede extraer es que las tecnologías de Grid, y Cloud son complementarias y por ello, capaces de coexistir y cooperar a diferentes niveles de abstracción, proporcionando un mayor número de beneficios a sus usuarios.

El estudio de las sobrecargas ha servido para poner de manifiesto los pensamientos que ya se tenían de antemano sobre este tema. Pero lejos de ser un estudio que simplemente demuestra lo evidente, también se han podido observar resultados cuanto menos curiosos, como el caso de las operaciones de lectura, con un rendimiento superior al caso de la máquina física debido al hecho de que al cargar la imagen de la máquina virtual sobre la máquina física, también se carga información a la que se accede posteriormente, evitando tener que acceder a disco en este tipo de operaciones.

El caso de estudio desarrollado empleando una aplicación científica computacionalmente intensa ha servido para demostrar el comportamiento de los modelos desarrollados bajo un entorno real. Tanto el modelo basado en la detección de la saturación (modelo 1) como el modelo que persigue el balanceo de la carga de trabajos (modelo 3) son capaces de llevar a cabo la ejecución del conjunto de las tareas realizando un aprovechamiento de recursos que permite acelerar la ejecución de las mismas. El modelo 2 no se aplicó puesto que las dependencias de la aplicación no eran demasiado complejas y se podían encontrar recursos que se adaptaban a los requerimientos de la tarea.

Bajo la opinión personal de la autora, la realización de este trabajo ha supuesto la conexión y maduración de los conocimientos adquiridos a lo largo del período

formativo del máster. Además, la colaboración con el grupo de investigación del GRyCAP ha supuesto una experiencia enriquecedora en todos los sentidos, tanto académicos como personales. La posibilidad de colaborar codo con codo con los que han sido mis profesores durante la carrera y el máster supone una experiencia inolvidable. La elaboración de artículos científicos ha supuesto todo un reto completado con éxito. La base de la que se partió a la hora de realizar la implementación, ha facilitado la consecución de esta tarea pero aun así, el enfrentamiento a un conjunto de código ajeno hasta adaptarlo a uno mismo también ha supuesto un esfuerzo.

En cuanto al trabajo futuro, siguiendo por el camino teórico de los modelos de uso, se plantea el estudio de nuevos modelos de uso sobre infraestructuras híbridas, teniendo en cuenta las limitaciones económicas y el consumo de energía. Se plantea el desarrollo de nuevos modelos de elasticidad para aprovechar la eficiencia del Cloud, mediante la realización de pronósticos para el aprovisionamiento y repliegue de recursos computacionales, con el objetivo de optimizar el tiempo de ejecución total de las tareas y el presupuesto del usuario. Para ello, se deberán construir algoritmos eficientes de estimación del número de VMs óptimo teniendo en cuenta la elasticidad de las infraestructuras Cloud.

También se puede explorar el campo del *Sky Computing*, un concepto que representa el aprovisionamiento de recursos proporcionado por múltiples proveedores Cloud. En este caso, ya se ha dado el primer paso utilizando APIs de agregación para poder interactuar con distintos proveedores empleando las mismas librerías.

Estas y otras tareas se podrían llevar a cabo continuando los estudios e investigaciones científicas, camino duro pero gratificante, y llevando a cabo una tesis doctoral en el marco de la computación de altas prestaciones.

7.2. PUBLICACIONES CIENTÍFICAS

El trabajo desarrollado en esta tesis de máster ha dado lugar a distintas publicaciones científicas, detalladas a continuación:

- Publicaciones en congresos:

  [1] A. Calatrava, G. Moltó, V. Hernández: *Combining Grid and Cloud Resources for Hybrid Scientific Computing Executions*. En: 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Grecia, pp. 494–501 (2011).

  [2] G. Moltó, A. Calatrava, V. Hernández: *A Service-Oriented Architecture for Scientific Computing on Cloud Infrastructures*. En: VECPAR 2012 (enviado para revisión).

Del 29 de noviembre al 1 de diciembre se celebró el congreso "*3rd IEEE International Conference on Cloud Computing Technology and Science*" (CloudCom 2011 [95]) en Grecia. Este congreso internacional indexado (*Tier C* según [88]), donde se seleccionaron menos de un 25% de los artículos enviados, seleccionó nuestro artículo "*Combining Grid and Cloud Resources for Hybrid Scientific Computations*". Esta publicación resumió los parámetros y modelos de ejecución híbrida desarrollados durante el proyecto, además de la presentación de un caso de estudio que demostraba su funcionalidad.

Además, se envió el artículo titulado "*A Service-Oriented Architecture for Scientific Computing on Cloud Infrastructures*" al congreso indexado VecPar [96] (*International Conference on Vector and Parallel Processing*) (*Tier B* según [88]). Este trabajo resume la arquitectura sobre la que se ha desarrollado la presente tesis de máster, incluyendo las herramientas de metaplanificación desarrolladas. En el caso de que finalmente el trabajo no sea aceptado, se plantea la posibilidad de mejorarlo y enviarlo a otros congresos como Euro-Par [97] (*International Conference on Parallel Processing*), un

congreso indexado cuya edición en 2012 incluye temas relacionados con Grid y Cloud Computing.

APÉNDICE A

EJEMPLO SIMPLE DEL USO DEL METAPLANIFICADOR

En este apéndice de la memoria se muestra un caso de estudio realizado para observar el funcionamiento del metaplanificador desarrollado. En concreto se realiza la ejecución de una aplicación muy sencilla que invoca al comando *sleep* de Unix. Como se puede observar a continuación, se crean 5 tareas a ejecutar entre las tres infraestructuras: Grid, Cloud privado y Cloud Público. La elección de cada una de ellas está delegada en el metaplanificador que implementa el modelo 1, donde la saturación de cada infraestructura es la que marcará la decisión a tomar. Nótese que la elección de cualquier otra aplicación no supondría mayor esfuerzo de programación para el usuario. Seguidamente se muestra el código desarrollado para este caso de estudio:

```
package org.grycap.GMarte.test.Amanda;

import java.util.Vector;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.grycap.GMarte.cloud.AccessCredentials;
import org.grycap.GMarte.resources.TestBed;
import org.grycap.GMarte.tasks.GridTask;
import org.grycap.GMarte.tasks.GridTaskLoR;
import org.grycap.GMarte.tasks.GridTaskLoRApplication;
import org.grycap.GMarte.tasks.GridTaskStudy;
import org.grycap.GMarte.tasks.files.GridFile;
import org.grycap.GMarte.tasks.files.GridInputFileSet;
import org.grycap.GMarte.execution.scheduling.Scheduler;
import
org.grycap.GMarte.execution.scheduling.hybrid.OrchestratorFirstModel;
import
org.grycap.GMarte.execution.scheduling.orchestrator.OrchestratorSchedu
ler;
import org.grycap.GMarte.resources.testbeds.GRyCAPTestBed;
import org.grycap.GMarte.tasks.DeploymentConfiguration;
import org.grycap.GMarte.tasks.GridTaskConfiguration;
import org.grycap.GMarte.tasks.files.GridExecutableFile;
import org.grycap.GMarte.tasks.files.GridOutputFileSet;
```

```
public class TestPlanificadorHibrido {

 static Logger log = Logger.getLogger(TestModelo1Hibrido.class);

 private static GridTaskStudy createTaskStudy (int nTasks){
  GridTaskStudy gts = new GridTaskStudy();
  for (int i = 0 ; i < nTasks; i++){
   // Creamos una tarea para ejecutar la aplicación
      GridTask gt = new GridTask("sleep");

   // Indicamos el fichero ejecutable
      GridExecutableFile gef = new GridExecutableFile("sleep/sleep");
      gef.setIsRemote(true);
      gt.setGridExecutableFile(gef);

   // Indicamos la ruta local donde recibir los resultados
      gt.setLocalDataContainer("C:\\Users\\amcaar\\Work");

   // Y los argumentos para lanzar la aplicación
      gt.setArguments("20");
      gt.setCloudPrivateArguments("20");
      gt.setCloudPublicArguments("20");

   // Requerimientos de la VM.
      GridTaskLoR gridTaskLoR = new GridTaskLoR();
      gridTaskLoR.addRequirement(GridTaskLoR.REQ_OS_NAME, "linux");
      gt.setGridTaskLoR(gridTaskLoR);

   // Comandos auxiliares para poder desplegar la aplicación
       DeploymentConfiguration dc = new DeploymentConfiguration();
      dc.addDeployCommand("tar xzvf sleep.tar.gz");
      GridTaskConfiguration gtc = gt.getGridTaskConfiguration();
      gtc.setDeploymentConfiguration(dc);
      gt.setGridTaskConfiguration(gtc);

   // Asociamos la configuracion de despliegue al fichero
      gef.setDeploymentConfiguration(dc);

   // Especificamos el paquete con los ficheros de entrada para la
      aplicacion
      GridInputFileSet gifs = new GridInputFileSet();
      GridFile gf1 = new GridFile("C:\\Users\\amcaar\\sleep.tar.gz");
      gifs.addGridFile(gf1);
      gt.setGridInputFileSet(gifs);

       //Especificamos los ficheros de salida a recuperar
      GridOutputFileSet gofs = new GridOutputFileSet();
      gofs.addWildCard("*.std");
      gt.setGridOutputFileSet(gofs);

      //Configuración de la tarea
 gt.getGridTaskConfiguration().setUseMagicKeywordToInvestigateFailures
(false);

      //Añadimos la tarea al grupo de tareas
      gts.addGridTask(gt);
  }
  return gts;
 }
```

```
public static void main(String args[]){

  PropertyConfigurator.configure("log4j.properties");

  //Indicamos el host donde se van a lanzar las VMs
  String endpoint = "http://kefren.i3m.upv.es:2633/RPC2";

  //Indicamos las credenciales de acceso a emplear para acceder al
  cloud
  AccessCredentials credentials = new
AccessCredentials("amcaar:3184cy3fdc85913006483deOg0P70fcn32af36bd",
"A38IHKAIOBREfPACQ7AQI6Q", "9rq0E2GB56YVJ316K406LPz-
xObLCYX+3nxaHLVDW", "user_ec2_us_east_keypair",
"C:\\Users\\amcaar\\workspace\\id_rsa-user_us_east_1");

  //Indicamos el n° de nodos accesibles por el usuario
  int nodos = 4;

  //Indicamos el identificador de las VMs desplegadas en ONE (si es -1
  significa que hay que lanzar una VM nueva)
  int ids[] = new int[2];
  ids[0] = 1489;
  ids[1] = -1;

  //Indicamos las IPs de las VMs desplegadas en EC2 (si las hubiere)
  String ips[] = new String[1];
  ips[0] = "50.128.4.67"

  //Se crea el array de tareas (con una sola tarea).
  GridTaskStudy gts = createTaskStudy(5);

  //Se crea el testbed de recursos grid.
  TestBed tbg = new
GRyCAPTestBed().getTestBedFromGridResource("odin.itaca.upv.es");

  //Se añade el planificador a las tareas, indicando el TestBed y el
  grupo de tareas (GridTaskStudy)
  OrchestratorFirstModel sched = new OrchestratorFirstModel(tbg, gts,
credentials, ips, ids, nodos, endpoint);

  //Ponemos en marcha el planificador
  sched.start();

  //Esperamos a que acaben de ejecutar las tareas (este método está
  en la clase "scheduler")
  sched.waitUntilFinished();

  log.info("FINISHED. ABOUT TO EXIT");
  System.exit(0);
 }
}
```

APÉNDICE B

USO DE AMAZON EC2 DESDE LÍNEA DE COMANDOS

En este apéndice de la memoria se pretende introducir al lector al empleo de Amazon EC2, mediante una pequeña guía de uso desarrollada tras la propia experiencia de la autora.

Aunque Amazon ofrece una interfaz web bastante amigable para facilitar su uso, en esta tesis de máster se ha optado por aprender las herramientas que se ofrecen sobre línea de comandos, ya que resulta más fácil posteriormente relacionar cada operación con las operaciones ofrecidas por los APIs de agregación o el propio API Java que ofrece Amazon.

El primer paso a seguir para poder acceder a los servicios que ofrece Amazon EC2, es darse de alta. En la página web oficial [18] se encuentra el enlace para ello. Tras ello, debemos generar un certificado X.509 desde la interfaz de Amazon y guardar en local los dos ficheros generados, con extensión *.cert*. Seguidamente, debemos descargar las herramientas de EC2 para línea de comandos [89]. Una vez descargadas, tendremos que configurar las variables de entorno necesarias para que los comandos funcionen correctamente. En concreto, en una máquina Linux sería necesario configurar las siguientes variables:

```
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk

export EC2_HOME=/home/user/AmazonEC2/ec2-api-tools-1.4.4.1

export EC2_KEY_DIR=/home/user/ AmazonEC2/.ec2

export EC2_PRIVATE_KEY=${EC2_KEY_DIR}/pk-nombre_fichero.pem

export EC2_CERT=${EC2_KEY_DIR}/cert-nombre_fichero.pem

export PATH=$PATH:$EC2_HOME/bin
```

El siguiente paso a seguir es crear un *key pair*. El *key pair* está compuesto por una clave pública y otra privada, y es empleado por Amazon para manejar la seguridad. La clave pública es la que se inocula a la VM, mientras que la privada es la que se descarga el usuario y se utiliza como argumento para conectarse a la VM mediante SSH. Empleando este *key pair* y SSH podremos acceder la primera vez a la máquina virtual que lancemos. Esta es la única forma de acceder por primera vez a la instancia lanzada, ya que en las siguientes podríamos crear un usuario y contraseña si se precisase o seguir accediendo mediante el *key pair*. Resulta necesario destacar que para cada región necesitaremos tener un par de claves distinto, ya que como se ha comentado en secciones anteriores, Amazon está organizado en regiones y éstas son totalmente opacas entre ellas. A partir de este punto, se asume que la región sobre la que vamos a trabajar es la que Amazon toma por defecto (*us_east_1*). En cualquier comando de los que se muestran a continuación podemos utilizar la opción "*--region*" para modificarlo.

El comando para crear un nuevo *key pair* es el siguiente:

```
user@ubuntu:~$ ec2-create-keypair <nombre> --region <región>
```

Como salida obtendremos la clave privada que posteriormente se empleará para acceder a nuestras instancias. La clave pública la guarda Amazon EC2. En cualquier momento podemos consultar qué pares de claves tenemos creados, empleando el comando *ec2-describe-keypairs*.

Una vez tenemos configurado el entorno y generado el par de claves, es momento de buscar la imagen de la máquina que queremos desplegar. Para buscar AMIs podemos proceder de tres formas distintas. La primera de ellas consiste en explorar la web de AMIs de Amazon [90] a través de un navegador web. También podemos realizar la búsqueda empleando la consola gráfica que proporciona Amazon, y por último, podemos consultar el catálogo de imágenes mediante el siguiente comando:

```
user@ubuntu:~$ ec2-describe-images --all
```

El siguiente paso consiste en crear un grupo de seguridad y asociarle los permisos que creamos oportunos. Un grupo de seguridad define un conjunto de reglas de seguridad para limitar las conexiones permitidas a las instancias que situemos dentro del grupo, especificando quién puede acceder y cómo, indicando IPs, puertos y protocolos. Los grupos de seguridad pueden alojar más de una instancia a la vez. Para crear un grupo en EC2 basta con ejecutar el siguiente comando:

```
user@ubuntu:~$ ec2-add-group <nombre_del_grupo>
```

Y seguidamente proporcionar los permisos que se consideren necesarios. Por ejemplo, vamos a permitir las conexiones SSH:

```
user@ubuntu:~$ ec2-authorize quick-start-1 -p tcp -p 22 -s 0.0.0./0
```

Podemos consultar qué grupos tenemos creados hasta el momento con el comando *ec2-describe-group* y eliminarlos, de la siguiente forma:

```
user@ubuntu:~$ ec2-delete-group <nombre_del_grupo>
```

Llegados a este punto es momento de poner en marcha la instancia. Para ello haremos uso del comando *ec2-run-instances* donde podremos indicar la AMI a desplegar, el tipo de instancia que queremos, el número de instancias, el grupo de seguridad, la clave o la región, entre otros:

```
user@ubuntu:~$ ec2-run-instances <id_ami>  --instance-count <nº_instancias>
--instance-type <tipo_instancia> --key <nombre_keypair> --group <nombre_grupo>
```

Este comando nos devolverá la información de la nueva instancia. De entre todos los datos, cabe destacar la importancia del identificador de instancia, que nos servirá para comprobar su estado mediante el siguiente comando:

```
user@ubuntu:~$ ec2-describe-instances  <id_instancia>
```

Así también obtendremos la IP pública de la instancia para que, cuando la máquina alcance el estado *running*, podamos conectarnos a ella empleando SSH con la opción *-i* para indicar el *key pair*:

```
user@ubuntu:~$ ssh -i <ruta_al_keypair>@<ip_instancia>
```

También podemos conectarnos a la máquina a través de su nombre, si la IP pública no la podemos obtener. Este nombre es del estilo a "*ec2-50-16-135-90.compute-1.amazonaws.com*".

Por último, para detener la instancia, emplearemos el comando *ec2-terminate-instances* de la siguiente forma:

```
user@ubuntu:~$ ec2-terminate-instances <id_instancia>
```

Hasta este punto, podemos hacer un uso básico de los servicios ofrecidos por EC2, empleando las máquinas virtuales que ofrece Amazon o que otros usuarios han preparado y compartido con el resto. Pero en ocasiones, como el caso de estudio presentado en esta memoria, interesa contar con una AMI preconfigurada para la ejecución de la aplicación, reduciendo el tiempo de contextualización de las ejecuciones. Para ello, tras desplegar y configurar a nuestro gusto una máquina virtual a partir de una que ya existía, es necesario registrar una nueva imagen en el catálogo de EC2. A continuación se detallan los pasos necesarios para llevar a cabo esta acción.

En primer lugar, tras seleccionar la AMI sobre la que realizaremos los cambios para adaptarla a nuestras necesidades, lanzamos a ejecución una instancia de la misma y nos conectamos a ella, tal y como se ha explicado en este mismo apéndice. Para poder generar la nueva AMI necesitaremos disponer de los certificados X.509 para poder cifrarla y firmarla posteriormente, luego será necesario copiarlos a nuestra instancia:

```
user@ubuntu:~$ scp -i <ruta_al_keypair> <nombre_fichero_cert> ec2-user@<IP_AMI
_desplegada>:<ruta_destino>
```

Desde dentro de nuestra instancia, y tras realizar las modificaciones oportunas, invocaremos el comando *ec2-bundle-vol*, encargado de comprimir el contenido de la misma, cifrar y firmar la nueva AMI a través de nuestro certificado X.509. Nótese que la AMI desplegada deberá disponer de las herramientas de línea de comandos que ofrece Amazon EC2. Este comando suele consumir varios minutos, dependiendo del tamaño de datos que contenga la instancia.

```
ec2_user@ubuntu:~$ ec2-bundle-vol -d <ruta_destino> -k <ruta_pk_X509> -c <ruta_cert_X509> -u <ID_cuenta_Amazon> -s <tamaño_fichero_image> -r <arquitectura> -e <ficheros_a_excluir>
```

Como salida, obtendremos en el directorio destino especificado mediante la opción *-d* una serie de ficheros *image.part.XX*, un fichero *.manifest*, que contiene los metadatos de la nueva AMI, y un fichero *image*, del tamaño especificado mediante la opción *-s*. Todo ello constituye el *bundle* de EC2, que representa a la imagen de máquina virtual. Seguidamente, es necesario subir a Amazon S3, el servicio de almacenamiento de Amazon, todos los ficheros que ha generado la orden anterior. Para ello, desde dentro de nuestra instancia ya modificada y configurada, ejecutaremos:

```
ec2_user@ubuntu:~$ ec2-upload-bundle -b <nombre_bucket> -m <ruta_fichero_manifest> -a <access_key> -s <private_key>
```

Este comando puede consumir varios minutos dependiendo del estado de la red y el tamaño de los ficheros a almacenar. A continuación, hay que registrar la nueva AMI en el catálogo de máquinas virtuales de Amazon EC2, para poder usarla en futuras ocasiones. Para ello, ejecutaremos desde fuera de la instancia el siguiente comando:

```
user@ubuntu:~$ ec2-register <ruta_fichero_manifest_S3>
```

Por defecto, la nueva AMI será privada para el usuario que estemos empleando. Como resultado de este comando obtendremos el identificador de la nueva AMI, que podremos volver a obtener en cualquier momento ejecutando el comando *ec2-describe-images*.

Esta nueva imagen se ha generado con un valor para *root device* de *instance-store*, es decir, que el almacenamiento de los datos que introduzcamos en la instancia será local a la misma, luego, como se detalló en el apartado 2.2.2.2 del capítulo 2 de la presente memoria, los cambios introducidos en la instancia sólo persistirán durante la vida de la misma. Ello puede implicar contratiempos tales como la imposibilidad de lanzar la nueva AMI como instancia micro, puesto que este tipo de instancias deben soportar EBS (*Elastic Block Store*).

Para que la nueva imagen soporte EBS el procedimiento a seguir es distinto al anterior. En primer lugar, debemos partir de una AMI que ya tuviese soporte a EBS, luego habrá que refinar la búsqueda de la imagen sobre la que partir. Aun así, si la imagen que queremos emplear como base no cumple este requisito, podemos crear un nuevo volumen EBS y asociarlo a la instancia desplegada de esa AMI, volcando al nuevo volumen todo el contenido de la instancia.

Una vez tengamos la instancia con el volumen EBS asociado a la misma, y la hayamos configurado a nuestro gusto, es momento de crear un *snapshot* del volumen. Un *snapshot* es una instantánea del volumen en un punto determinado del tiempo, que se convierte en persistente ya que se almacena en S3. Estas instantáneas se pueden registrar como nuevas AMIs en el catálogo de Amazon EC2 para poder emplearlas como punto de partida en futuras ejecuciones. Para ello necesitamos obtener el identificador del volumen, que podremos obtener invocando al comando *ec2-describe-volumes*. Tras este paso, ejecutamos el siguiente comando:

```
user@ubuntu:~$ ec2-create-snapshot <id_volumen_EBS> --description <descripcion>
```

Para comprobar que todo ha funcionado correctamente, podemos ejecutar el comando *ec2-describe-snapshots* que nos devolverá la lista de instantáneas que tenemos creadas con nuestra cuenta. Con ello podemos recuperar también el identificador del nuevo *snapshot*, necesario para poder registrar posteriormente esta instantánea como una nueva AMI.

Para finalizar el proceso de creación de una nueva AMI con soporte a EBS, registraremos el *snapshot* creado en el paso anterior empleando el mismo comando que en el caso de una imagen con almacenamiento local, pero indicando que esta vez se trata de una instantánea:

```
user@ubuntu:~$ ec2-register --snapshot <id_snapshot> --kernel <version_kernel> --name <nombre> --architecture <arquitectura> --root-device-name <ruta_montaje_volumen_EBS>
```

Esta orden nos devolverá el identificador de la nueva AMI creada, si todo ha funcionado correctamente.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Web del WLCG. http://lcg.web.cern.ch/lcg/

[2] I. Foster and C. Kesselman: *The grid in a nutshell*. En: Grid Resource Management: State of the Art and Future Trends, pp. 3-13 (2004).

[3] Ian Foster: *What is the Grid? A Three Point Checklist*. En: GRID Today (2002).

[4] EGEE. http://www.eu-egee.org/

[5] ES-NGI. http://www.e-ciencia.es/ngi/

[6] UNICORE. http://www.unicore.eu/

[7] Globus Toolkit 2. Buscar algun articulo referencia

[8] gLite. http://glite.cern.ch/

[9] Michael Armbrust et al.: *Above the clouds: A Berkeley View of Cloud Computing*. 2009

[10] Definición de Cloud Computing del NIST. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[11] Miller, Michael: *Cloud Computing, Web-Based Applications That Change the Way You Work and Collaborate Online*. Indianapolis, Que, (2008). ISBN: 978-0768686227.

[12] GMail. http://www.google.com/apps/intl/es/business/gmail.html

[13] Google Sites. http://www.google.com/apps/intl/es/business/sites.html

[14] Granneman, S.: *Google Apps Deciphered: Compute in the Cloud to Streamline Your Desktop*. Prentice Hall (2008). ISBN: 978-0-13-700776-9.

[15] Google App Engine. http://code.google.com/intl/es-ES/appengine/

[16] Windows Azure. http://www.microsoft.com/windowsazure/

[17] Van Vliet, J., Paganelli, F.: *Programming Amazon EC2*. O'Reilly (2011). ISBN: 978-1-449-39368-7

[18] Web de Amazon EC2. http://aws.amazon.com/ec2/

[19] Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.:*Capacity leasing in cloud systems using the opennebula engine*. En: Workshop on Cloud Computing and its Applications 2008 (CCA08).

[20] Granneman, S.: *Google Apps Deciphered: Compute in the Cloud to Streamline Your Desktop*. Prentice Hall (2008). ISBN: 978-0-13-700776-9.

[21] OFV Whitepaper. http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf

[22] Red Hat Enterprise Linux. http://www.es.redhat.com/.

[23] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield: *Xen and the Art of Virtualization*. En: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp 164-177, Bolton Landing, NY, USA (2003).

[24] Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: *KVM: The linux virtual machine monitor*. In: OLS '07: The 2007 Ottawa Linux Symposium, pp. 225–230. (2007).

[25] VMware. http://www.vmware.com

[26] VMware Player. http://www.vmware.com/products/player/

[27] Virtual Box. http://www.virtualbox.org/

[28] Virtual PC. http://www.microsoft.com/windows/virtual-pc/

[29] Openstack. http://www.openstack.org/

[30] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L.,Zagorodnov, D.: *The eucalyptus open-source cloud-computing system.* En: Proceedings of 9th IEEE International Symposium on Cluster Computing and the Grid. (2009).

[31] Globus Nimbus. http://www.nimbusproject.org/

[32] H. Kim, Y. el-Khamra, S. Jha and M. Parashar.: *An autonomic approach to integrated HPC Grid and Cloud usage.* En: Fifth IEEE International Conference on e-Science, pp. 366-373 (2009).

[33] H. Kim, Y. el-Khamra, S. Jha and M. Parashar: *Exploring Application and Infrastructure Adaptation on Hybrid Grid-Cloud Infrastrructure.* En: 19th ACM International Symposium on High Performance Distributed Computing, pp 402-412 (2010).

[34] G. V. Evoy, B. Schulze: *Using Clouds to address grid limitations.* En: 6th international workshop on Middleware for grid computing (2008).

[35] H. Kloh, B. Schulze, A. Mury and R. Pinto: *A scheduling model for workflows on Grids and Clouds.* En: Procedings of the 8yh International Workshop on Middleware for Grids, Clouds and e-Science. Nº de dicembre, ACM, p.3 (2010).

[36] I. Brandic, D. Music, S. Dustdar: *Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services.* En: 6th international conference industry session on Grids meets autonomic computing (GMAC '09), (2009).

[37] E. Huedo, R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente: *Architectures for Enhancing Grid Infrastructures with Cloud Computing.* En: Grids Clouds and Virtualization. Computer Communications and Networks. M. Cafaro and G. Aloisio, Eds. Springer London, pp 55-69 (2011).

[38] C. V. Blanco, E. Huedo, R. S. Montero and I. M. Llorente: *Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers.* En: 2009 Workshops at the Grid and Pervasive Computing Conference. IEEE, pp 113-120 (2009).

[39] C. V. Blanco, E. Huedo, R. S. Montero and I. M. Llorente: *On the use of clouds for grid resource provisioning*. En: Journal of Future Generation Computer Systems. Nº 27, pp 600-605 (2011).

[40] T. S. Somasundaram, B. R. Amarnath, et all.: *CARE Resource Broker: A framework for scheduling and supporting virtual resource management*. En: Journal of Future Generation Computer Systems. Nº 26, pp 337-347 (2010).

[41] B. Amedro, F. Baude y F. Huet: *Combining Grid and Cloud Resources by Use of Middleware for SPMD Applications*. En: Conference on Cloud, pp 177-184 (2010).

[42] Proactive. http://proactive.inria.fr/

[43] A. Merzky, K. Stamou and S. Jha: *Application Level Interoperability between Clouds and Grids*. 2009 Workshops at the Grid and Pervasive Computing Conference. IEEE, pp 143-150 (2009).

[44] S. Pallickara, M. Pierce, Q. Dong and C. Kong: *Enabling Large Scale Scientific Computations for Expressed Sequence Tag Sequencing over Grid and Cloud Computing Clústers*. En: Eigth International Conference on Parallel Processing and Applied Mathematics (PPAM 2009). Citeseer (2009).

[45] CometCloud.http://nsfcac.rutgers.edu/CometCloud/CometCloud/CometCloud_Home_Page.html

[46] FutureGrid. https://portal.futuregrid.org/

[47] StratusLab. http://www.stratuslab.org/index.php

[48] CDAC Cloud-Grid integration. http://www.ow2.org/xwiki/bin/download/Events2010AnnualConference/Program/NexusCloud-M-Sriramagiri.pdf

[49] J. M. Alonso, V. Hernández, G. Moltó: *GMarte: Grid middleware to abstract remote task execution*. En: Concurrency and Computation: Practice and Experience, vol. 18, no. 15, pp. 2021-2036 (2006).

[50] G. Moltó, V. Hernández, J. Alonso, A service-oriented WSRF-based architecture for metascheduling on computational Grids, Future Generation Computer Systems 24 (4) (2008) 317-328.

[51] G. Moltó: *Computación de altas prestaciones sobre entornos Grid en aplicaciones biomédicas: simulación de la actividad eléctrica cardiaca y diseño de proteínas*. Tesis Doctoral(2007).http://riunet.upv.es/bitstream/handle/10251/1831/tesisUPV2679.pdf?sequence=1

[52] J.V. Carrión, G. Moltó, C. De Alfonso, M. Caballer, and V. Hernández: *A Generic Catalog and Repository Service for Virtual Machine Images*. En: 2nd International ICST Conference on Cloud Computing (CloudComp 2010).

[53] A. Calatrava: *Desarrollo de interfaces de alto nivel para la interacción con gestores de máquinas virtuales en infraestructuras de tipo Cloud*. Proyecto Final de Carrera (2010).http://riunet.upv.es/bitstream/handle/10251/8607/Desarrollo%20de%20Interfaces%20de%20Alto%20Nivel%20para%20la%20Interacci%C3%B3n%20con%20Gestores%20de%20M%C3%A1quinas%20Virtuales%20en%20.pdf?sequence=1

[54] Krishnan, S, L Clementi, J Ren, Philip Papadopoulos, and Wilfred Li. *Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service*. En 2009 IEEE Congress on Services (2009).

[55] Java. http://java.com/es/

[56] LibCloud. http://libcloud.apache.org/

[57] Phyton. http://www.python.org/

[58] DeltaCloud. http://incubator.apache.org/deltacloud/index.html

[59] Ruby. http://www.ruby-lang.org/es/

[60] GoGrid. http://www.gogrid.com/

[61] Fog. http://fog.io/1.0.0/index.html

[62] Jclouds. http://www.jclouds.org/

[63] Azure Storage Service. http://www.microsoft.com/windowsazure/features/storage/

[64] Dasein Cloud API. http://dasein-cloud.sourceforge.net/

[65] Simple Cloud API. http://simplecloud.org/

[66] Nirvanix. http://www.nirvanix.com/

[67] RightScale API. http://support.rightscale.com/15References/RightScale_API_Reference_Guide

[68] SPEC. http://www.spec.org/benchmarks.html#cpu

[69] SPECvirt 2010. http://www.spec.org/virt_sc2010/

[70] Test de Linpack. http://www.top500.org/project/linpack/

[71] TOP500. http://www.top500.org/

[72] BYTEmark. http://www.tux.org/~mayer/linux/bmark.html

[73] Documentación sobre BYTEmark. http://www.tux.org/~mayer/linux/byte/bdoc.pdf

[74] UnixBench. http://code.google.com/p/byte-unixbench/

[75] Whetstone. http://en.wikipedia.org/wiki/Whetstone_(benchmark)

[76] Dhrystone. http://es.wikipedia.org/wiki/Dhrystone

[77] IOzone. http://www.iozone.org/

[78] Bonnie. http://www.textuality.com/bonnie/

[79] Bonnie++. http://www.coker.com.au/bonnie++/

[80] B. Wolman, T. B. Olson: *IOBENCH: a system independent IO benchmark*. In: ACM SIGARCH Computer Architecture News (1989).

[81] Lista de benchmarks de E/S más importantes. http://www.mcs.anl.gov/~thakur/pio-benchmarks.html

[82] FLASH I/O. http://www.ucolick.org/~zingale/flash_benchmark_io/

[83] IOR. http://sourceforge.net/projects/ior-sio/

[84] Guía de uso de IOzone
http://www.iozone.org/docs/IOzone_msword_98.pdf

[85] C. de Alfonso, M. Caballer and V. Hernández: *Efficient power management in High Performance Computer clusters*. En: proceedings of the International Conference on Green Computing 2010 (ICGreen 2010).

[86] G. Moltó, M. Suárez, P. Tortosa, J. M. Alonso, V. Hernández y A. Jaramillo: *Protein design based on parallel dimensional reduction*. En: Journal of chemical information and modeling, vol. 49, no. 5, pp. 1261–71 (2009).

[87] Plan gratuito de Amazon. http://aws.amazon.com/es/free/

[88] Clasificación CORE. http://core.edu.au/

[89] EC2 API Tools. http://aws.amazon.com/developertools/351

[90] Catálogo de AMIs de EC2. http://aws.amazon.com/amis

[91] Amazon S3. http://aws.amazon.com/es/s3/

[92] Amazon EBS. http://aws.amazon.com/es/ebs/

[93] Amazon SimpleDB. http://aws.amazon.com/es/simpledb/

[94] K. Keahey, M. Tsugawa, A. Matsunaga and J. Fortes: *Sky Computing*. En: IEEE Internet Computing, vol. 13, pp. 43-51 (2009).

[95] CloudCom 2011. http://2011.cloudcom.org/

[96] VecPar 2012. http://nkl.cc.u-tokyo.ac.jp/VECPAR2012/

[97] Euro-Par 2012. http://europar2012.cti.gr/home

[98] MSC Nastran. http://www.mscsoftware.com/products/cae-tools/msc-nastran.aspx

[99] Xianghua Xu, Feng Zhou y Jian Wan Yucheng Jiang: *Quantifying Performance Properties of Virtual Machine*. En: International Symposium on Information Science and Engieering (2008).

[100] Jianhua Che, Qinming He, Qinghua Gao y Dawei Huang: *Performance Measuring and Comparing of Virtual Machine Monitors.* En: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (2008).

[101] Jianhua Che, Yong Yu, Congcong Shi y Weimin Lin: *A Synthetical Performance Evaluation of OpenVZ, Xen and KVM.* En: IEEE Asia-Pacific Services Computing Conference (2010).

[102] Todd Deshane, Zachary Shepherd, Jeanna Matthews, et al.: *Quantitative Comparison of Xen and KVM.* En: Xen Summit (2008). Disponible: http://www.mulix.org/pubs/integrated/Deshane-XenSummit08.pdf

[103] Nathan Regola y Jean-Christophe Ducom: *Recommendations for Virtualization Technologies in High Performance Computing.* En: 2nd IEEE International Conference on Cloud Computing Technology and Science (2010).

[104] Victor Delgado: *Exploring the limits of Clouds Computing.* (2010) http://upcommons.upc.edu/pfc/bitstream/2099.1/13421/1/VDelgado_thesis.pdf