UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Arquitectura De Computadores y Automática

EJECUCIÓN ADAPTATIVA EN GRIDS COMPUTACIONALES

MEMORIA PRESENTADA PARA OPTAR AL GRADO DE DOCTOR POR

Eduardo Huedo Cuesta

Bajo la dirección de los Doctores:

Ignacio Martín Llorente
Rubén Santiago Montero

Madrid, 2004

ISBN: 84-669-2588-0

Ejecución Adaptativa en Grids Computacionales

Tesis Doctoral

Eduardo Huedo Cuesta

Madrid, Octubre de 2004

Dpto. de Arquitectura de Computadores y Automática

Universidad Complutense de Madrid

A María Pilar
y a mi familia.

Agradecimientos

Me gustaría agradecer la colaboración y el apoyo de todos los que han contribuido a la realización de esta Tesis y sin los cuales no hubiera sido capaz de terminarla.

En primer lugar, a mis directores de Tesis y compañeros de fatigas, Ignacio Martín Llorente y Rubén Santiago Montero, por pensar siempre en mí para sus proyectos y por haber trabajado tanto como yo en que esta investigación saliera adelante.

En segundo lugar, a Juan Pérez Mercader, Director del Centro de Astrobiología, por confiar en mí y darme la oportunidad de realizar este trabajo en su centro. Y a Luis Vázquez, Director del Laboratorio de Computación Avanzada, por su apoyo incondicional al Grid y a la línea de investigación que iniciamos en el centro. También al Instituto Nacional de Técnica Aeroespacial y al Ministerio de Ciencia y Tecnología, por confiar en nosotros y financiar nuestra investigación. Y a la gente de IRISGrid: Manuel Delfino, Jesús Marco, Vicente Hernández, Nacho López... por sus esfuerzos para crear una infraestructura Grid en España.

A toda la gente que trabaja en el Centro de Astrobiología, por crear un ambiente tan agradable. En especial, a la gente del Laboratorio de Computación Avanzada. A Mª Paz, mi compañera de despacho y casi vecina en Hilarión Eslava 9, por ser tan afable. A Antonio Giaquinta, por su eficacia consiguiendo recursos. A Kai Neuffer, por su espíritu de servicio y su continua disposición a colaborar. A Fernando Camps, por sus ganas de aprender. A Alain Lepinette, por ayudarme con babieca y no enfadarse demasiado cada vez que se "desbocaba". Y a Mauro López, por la ilusión con la que ha comenzado.

También al resto del Grupo de Arquitectura de Sistemas Distribuidos y Seguridad: Rafael Moreno, Teresa Higuera, José Herrera y Antonio Fuentes. Y al Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid.

Por supuesto, a mis padres, Eduardo y Gloria, por apoyarme y aconsejarme desde el primer día de mi vida. A mis hermanos: Gloria, Ana, Cristina y José Luis, con los que siempre he estado "muy unido". Y a mis otros hermanos: Chiqui, Fran, Amores, Josean, Luis, Paco, Carlos, Ángel... por estar siempre ahí.

Y a Mariapi, por comprenderme y animarme durante estos largos años de trabajo y viajes, por perdonarme después de haberme ido a Austria sin ella, por todas las veces que no he ido a recogerla a la biblioteca ni al salir del trabajo... Espero que, a partir de ahora, nuestros destinos se vayan acercando también geográficamente. Te prometí que citaría tu trabajo de fin de carrera [1]. Promesa cumplida.

Índice general

Prefacio

La necesidad de aprovechar los recursos disponibles en los sistemas informáticos conectados a Internet y de simplificar su utilización ha dado lugar a una nueva forma de tecnología de la información conocida como *Grid Computing*. Esta nueva tecnología es análoga a las redes de suministro eléctrico (*power grids*), de donde recibe su nombre: la idea es ofrecer un único punto de acceso a un conjunto de recursos distribuidos geográficamente (superordenadores, clústeres, almacenamiento, fuentes de información, instrumentos, personal...). De este modo, los sistemas distribuidos se pueden emplear como un único sistema virtual en aplicaciones intensivas en datos, con gran demanda computacional o que requieren la colaboración entre múltiples recursos de diversos tipos.

A pesar del esfuerzo realizado durante los últimos años, el desarrollo y ejecución de aplicaciones en el Grid continúa requiriendo un gran nivel de experiencia debido a su complejidad. El desarrollador o usuario es responsable de programar o realizar manualmente todas las etapas de planificación de trabajos para obtener alguna funcionalidad: descubrimiento y selección de recursos; y preparación, envío, monitorización, migración y finalización de trabajos. Además, se deben considerar todas las circunstancias que pueden darse en un entorno altamente heterogéneo y dinámico como es el Grid.

El objetivo principal de esta Tesis es el estudio y análisis de las diferentes técnicas de planificación y ejecución de trabajos en Grids que proporcionan una mayor eficiencia, fiabilidad y facilidad de uso. De entre todas las técnicas existentes, quizá las más interesante sea la ejecución adaptativa de trabajos.

El aspecto fundamental de la ejecución adaptativa es el reconocimiento de las condiciones cambiantes, tanto de los recursos del Grid como de las demandas de la aplicación. Para conseguir tal funcionalidad, proponemos un modelo de aplicación consciente del Grid, que incluye funcionalidad auto-adaptativa, y un agente de envío, que proporciona los mecanismos de ejecución necesarios para adaptar la ejecución de la aplicación. La aplicación debe estar equipada con la funcionalidad necesaria para soportar las circunstancias de replanificación iniciadas por la aplicación, mientras que el agente debe estar continuamente vigilando la ocurrencia de circunstancias de replanificación iniciadas por el Grid o por la aplicación.

Las técnicas estudiadas se han incluido en una nueva herramienta experimental basada en Globus, que permite una ejecución de trabajos más sencilla y eficiente en entornos Grid dinámicos, a la manera "enviar y olvidar". La adaptación a las condiciones cambiantes

se consigue implementando una replanificación automática de las aplicaciones cuando se producen degradaciones en el rendimiento, descubrimientos de "mejores" recursos, cambios en los requisitos o preferencias de la aplicación, cancelaciones o suspensiones del trabajo, fallos, etc. La replanificación de un trabajo puede conducir a su migración a otro recurso más apropiado.

La viabilidad de las técnicas desarrolladas en este trabajo se demuestra con la ejecución de varios tipos de aplicaciones, a saber: aplicaciones de alta productividad de Dinámica de Fluidos Computacional (análisis de la dinámica de un fluido sobre una placa delgada) y Bioinformática (predicción de la estructura y propiedades termodinámicas de secuencias de proteínas); y los *NAS Grid Benchmarks*, que incluyen aplicaciones con flujos de trabajo complejos. Los resultados obtenidos demostrarán el impacto decisivo que tiene la capacidad de adaptar la ejecución del trabajo en el rendimiento y robustez que pueden alcanzarse.

Preface

The need to take advantage of the available resources in the computer systems connected to Internet and to simplify their utilization, has led to a new way of information technology known as *Grid Computing*. This new technology is similar to electric power grids: the aim is to provide a single access point to a set of geographically distributed resources (supercomputers, clusters, storage, information sources, scientific instruments, staff...). This way, distributed systems can be used as a single virtual system for data-intensive, compute-intensive o collaboration-intensive applications.

In spite of the great research effort made in the last years, application development and execution in the Grid continue requiring a high level of expertise due to its heterogeneous and dynamic nature. The developer or user is responsible for programming or manually performing all the job submission stages in order to achieve any functionality: resource discovery and selection; and job preparation, submission, monitoring, migration and termination. Moreover, all the circumstances that can arise in a highly heterogeneous and dynamic environment like the Grid must be considered.

The main objective of this Thesis is the study and analysis of different techniques for job scheduling and execution in Grids that provide a greater efficiency, reliability and easy of use. Among all the existent techniques, the most interesting is maybe the adaptive execution of jobs.

The fundamental aspect of adaptive execution is the recognition of changing conditions of both Grid resources and application demands. In order to achieve such functionality, we propose a Grid-aware application model, which includes self-adapting functionality, and a submission agent that provides the runtime mechanisms needed to adapt the execution of the application. The application must be equipped with the functionality needed to support the application-initiated migration circumstances, while the agent is continuously watching the occurrence of Grid- and application-initiated migration circumstances.

The proposed techniques has been included in a new experimental tool based on Globus, which allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "submit & forget" fashion. Adaptation to changing conditions is achieved by dynamic rescheduling of jobs: once the job is initially allocated, it is rescheduled when a performance degradation is detected, a "better" resource is discovered, the application demands change, the job is cancelled or suspended, a failure is detected... Job rescheduling can lead to its migration to a more suitable resource.

The feasibility of the techniques developed in this work has been demonstrated with the execution of several kinds of applications, namely: high throughput applications in Computational Fluid Dynamics (analysis of fluid dynamics over a flat plate) and Bioinformatics (prediction of protein structure and thermodynamics properties); and the *NAS Grid Benchmarks*, which include applications with complex work-flows. The results obtained show the decisive impact of the capability to adapt job execution in the performance and reliability that can be achieved.

1. Introducción

El objetivo de este capítulo es proporcionar una visión global de la tendencia actual de las diferentes tecnologías que permiten aprovechar de forma conjunta los recursos disponibles en sistemas interconectados por red. En concreto, nos centraremos en la tecnología Grid emergente y en los desafíos a los que tiene que enfrentarse y que han motivado esta investigación.

1.1. Tecnologías de Computación en Red

Los siguientes modelos de computación en red aportan mecanismos para aprovechar al máximo los recursos distribuidos, que generalmente se encuentran infrautilizados:

- *Cluster Computing*: Consiste en el diseño de un *cluster* dedicado de equipos como alternativa a la adquisición de un equipo multiprocesador [2]. Su ventaja es la mejor relación coste/rendimiento. Sus inconvenientes son su dificultad de programación y mantenimiento. Los *clusters* suelen estar gestionados por *software* de planificación como MOSIX [3] de la Universidad de Israel o PBS [4] desarrollado por NASA y comercializado por Altair Engineering.

- *Intranet Computing*: Se trata de unir la potencia computacional desaprovechada en los recursos distribuidos en una red dentro de un único dominio de administración. Su principal ventaja es que, para determinados tipos de aplicaciones, puede proporcionar rendimientos semejantes a los ofrecidos por sistemas de alto rendimiento con un coste económico casi nulo. Las herramientas SGE [5] de Sun Microsystems, LSF [6] de Platform Computing o Condor [7] de la Universidad de Wisconsin permiten la gestión oportunista de los recursos, es decir, la utilización de los recursos de una *intranet* cuando éstos se encuentran ociosos. Existen empresas como GridSystems, Avaki, Entropia o United Devices que comercializan *software* de *Intranet Computing* específico para aplicaciones paramétricas.

- *Internet Computing*: Consiste en aprovechar la potencia de los recursos distribuidos en Internet siguiendo el modelo cliente/servidor o mediante redes de pares (*Peer-to-Peer*, P2P) [8]. Actualmente, casi todas estas herramientas se limitan a la ejecución de aplicaciones paramétricas. Su ventaja es el gran rendimiento que se puede obtener, dado el gran número de recursos computacionales a los que se tiene acceso. Sus principales inconvenientes son debidos al bajo ancho de banda y a la escasa seguridad de Internet [9]. Avaki, Entropia [10] o United Devices mantienen versiones de sus herramientas que permiten su uso en Internet. Probablemente, el ejemplo más típico de *Internet Computing* es el proyecto SETI@home [11, 12].

El uso de las tecnologías descritas anteriormente posibilita el aprovechamiento eficiente de los recursos dentro de una misma organización. Algunas de ellas, como SGE, LSF o Condor, permiten incluso unir diferentes departamentos u organizaciones, pero con la condición de que sea su *software* el que gestione los recursos internos. Sin embargo, ninguna de estas tecnologías permite unir dominios de administración diferentes manteniendo la política de seguridad de cada centro y las herramientas de planificación ya en uso. Por otro lado, los interfaces y protocolos básicos que utilizan las herramientas anteriores son propietarios (cerrados), de propósito específico y no estándar.

1.2. La Filosofía Grid

Las tecnologías descritas en la sección anterior son casos especiales de un nuevo paradigma de computación distribuida que en poco tiempo va a revolucionar no sólo la computación de altas prestaciones, sino Internet en general. Esta nueva tendencia supone un cambio radical en la colaboración de sistemas conectados a Internet y, en particular, en la computación de altas prestaciones debido a su enorme potencial respecto al intercambio y gestión de recursos. Es importante resaltar que **la tecnología Grid no pretende sustituir a las tecnologías anteriores**, ya que su ámbito de aplicación es diferente. El objetivo de la tecnología Grid es unir de forma segura los recursos de diferentes dominios de administración respetando su autonomía (políticas de seguridad, herramientas de gestión internas, condiciones de uso, etc.). Además, su objetivo es unir todo tipo de recursos y no únicamente capacidad de cálculo y almacenamiento.

Una de las primeras definiciones de Grid apareció en [13]:

A computational Grid is a hardware and software infrastructure that provi-

> *des dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.*

Esta definición se centra en el aspecto computacional, ya que la tecnología Grid (o al menos las primeras herramientas) estaba pensada inicialmente para acceder a grandes superordenadores con el fin de obtener una gran potencia computacional. Por tanto, la tecnología Grid debería proporcionar el acceso a recursos computacionales avanzados (*high-end*) de forma que se pudiera confiar en su funcionamiento (*dependable*), con un interfaz uniforme (*consistent*), desde cualquier sitio (*pervasive*) y con un coste asequible (*inexpensive*).

Una definición posterior apareció en [14]:

> *The Grid problem is defined as flexible, secure and coordinated resource sharing among dynamic collections of individuals, institutions and resources known as virtual organizations.*

En este caso se centra en la compartición de recursos (de todo tipo) entre colecciones dinámicas de recursos, individuos e instituciones conocidas como organizaciones virtuales (*Virtual Organization*, VO). Eso sí, de forma flexible, segura y coordinada.

La definición definitiva apareció en [15], e impone tres requisitos que debe cumplir un sistema para poder ser considerado un Grid:

> *A Grid is a system that...*
>
> 1. *...coodinates resources that are not subject to a centralized control...*
> 2. *...using standard, open, general-purpose protocols and interfaces...*
> 3. *...to deliver nontrivial qualities of services.*

Para que un sistema se considere un Grid, debe carecer de un control centralizado, debe estar basado en protocolos e interfaces estándares, abiertos y de propósito general, y debe proporcionar calidades de servicio no triviales. Cabe resaltar que esta definición ha recibido numerosas críticas, principalmente del sector empresarial, que aboga por un enfoque más centralizado y sin tanta necesidad de estandarización [16], sin embargo, esta definición goza de total respaldo por parte del sector académico y de investigación.

Un sistema de gestión de *clusters*, como SGE, LSF o PBS, instalado en un computador paralelo o sobre una red local, puede proporcionar garantías de calidad de servicio y,

por tanto, constituye un potente recurso para el Grid. Sin embargo, tal sistema no es un Grid por sí mismo, debido al control centralizado de los componentes que gestiona: tiene conocimiento completo del estado del sistema y de las peticiones de los usuarios, y control completo sobre los componentes individuales. Además, estos productos tampoco cumplirían el segundo criterio, ya que utilizan interfaces y protocolos propietarios. A una escala diferente, la *Web* tampoco es, *todavía*, un Grid: sus protocolos abiertos y de propósito general proporcionan acceso a recursos distribuidos, pero no el uso coordinado de esos recursos para generar calidades de servicio interesantes.

En cuanto al proceso de estandarización, el GGF (*Global Grid Forum*) [17] es el organismo encargado de crear los estándares de servicios y protocolos necesarios para crear la infraestructura o tecnología Grid. El trabajo del GGF se realiza dentro de sus diferentes grupos de trabajo o investigación. Un grupo de trabajo (*Working Group*, WG) se centra normalmente en una característica o tecnología muy específica con la intención de desarrollar uno o más documentos específicos generalmente con el objetivo de proporcionar especificaciones, directrices o recomendaciones. Un grupo de investigación (*Research Group*, RG) se suele centrar a más largo plazo, con la intención de explorar un área donde el desarrollo de especificaciones podría ser prematuro. Los diferentes grupos de trabajo o investigación del GGF se dividen en las siguientes áreas:

- Aplicaciones y entornos de programación
- Arquitectura
- Datos
- Sistemas de información y rendimiento
- Redes de pares (*Peer-to-Peer*, P2P)
- Planificación y gestión de recursos
- Seguridad

El Grid no es una idea nueva. El concepto de usar múltiples recursos distribuidos para trabajar cooperativamente en un única aplicación ha estado rondando durante varias décadas. A finales de los años 70 se trabajó en sistemas operativos en red (*networked operating systems*) y a finales de los años 80 se comenzaron a estudiar los sistemas operativos distribuidos (*distributed operating systems*). Poco después entraron en juego los campos

de la computación heterogénea (*heterogeneous computing*) y la computación paralela distribuida (*parallel distributed computing*), que unidos dieron lugar a la meta-computación (*metacomputing*). No hay muchas diferencias entre el concepto actual de Grid y el de los antiguos trabajos sobre sistemas distribuidos. Quizá la diferencia principal es el enfoque diferente de la tecnología Grid, que se centra en la autonomía de las organizaciones, en la heterogeneidad de los recursos (no sólo computacionales) y en los beneficios, en términos de rendimiento y facilidad de uso, para el usuario final.

En [18], se enumeran las diez cuestiones principales que debe resolver la tecnología Grid para ser considerada un éxito. El trabajo presentado en esta Tesis trata de contestar a algunas de estas preguntas:

1. ¿Por qué los Grids no tienen todavía una funcionalidad básica?

   Moraleja: Antes de tener un Grid satisfactorio, debemos tener un Grid completamente funcional.

2. ¿Por qué no hay más desarrolladores y usuarios para el Grid?

   Moraleja: Hasta que no se proporcione un mejor conjunto de servicios básicos para los desarrolladores, los Grids no serán utilizados en todo su potencial.

3. ¿Dónde están las herramientas para ayudar al desarrollo de aplicaciones para el Grid?

   Moraleja: El Grid necesita desesperadamente herramientas que faciliten su uso.

4. ¿Cómo hacer que los Grids sean seguros?

   Moraleja: Sin una infraestructura de seguridad, los usuarios no aprovecharán el Grid.

5. ¿Cómo se definen los interfaces y definiciones estándar para el Grid?

   Moraleja: La falta de estándares continúa dificultando la interoperatibilidad que el Grid necesita.

6. ¿Cómo se puede gestionar la variabilidad del Grid?

   Moraleja: El comportamiento cambiante del Grid es un hecho y debe ser tratado.

7. ¿Qué modelos de coste se necesitan en el Grid?

   Moraleja: El coste es un concepto necesario para el Grid.

8. ¿Dónde están las herramientas de gestión de sistemas Grid?

   Moraleja: Para que el Grid tenga el mayor número de recursos, su administración debe ser sencilla y rápida.

9. ¿Cuáles son los beneficios de compartir recursos en el Grid?

   Moraleja: Los beneficios se derivarán de poner freno al egoísmo de los usuarios y propietarios de recursos.

10. ¿Cómo se puede financiar la investigación y desarrollo necesarios para el Grid?

    Moraleja: Sin soporte para trabajo básico en funcionalidad, estándares e ingeniería del *software*, el Grid nunca estará a la altura de su potencial.

A pesar de que existen otras tecnologías Grid como Legion [19], Polder [20] o MOL [21], la mayoría de los proyectos Grid actuales están basados en los servicios y protocolos proporcionados por el proyecto (o alianza) Globus [22]. La tecnología Globus ha sido seleccionada como estándar *de facto* por las 12 compañías más importantes del sector de computación de altas prestaciones (Compaq, Cray, SGI, Sun Microsystems, Entropia, IBM, Microsoft, Platform Computing y Veridian en Estados Unidos; y Fujitsu, Hitachi y NEC en Japón). El conjunto de herramientas de Globus (*Globus Toolkit*, GT) es una colección de componentes *software* abiertos diseñados para soportar el desarrollo de aplicaciones sobre entornos distribuidos de tipo Grid. Cada componente proporciona un servicio básico como autenticación, asignación de recursos, descubrimiento y monitorización de recursos, comunicación, detección de fallos y acceso remoto a datos. La tecnología Globus se describe con mayor profundidad en la siguiente sección.

1.3. La Tecnología Globus

En esta sección se muestra el conjunto de herramientas, librerías, servicios y protocolos de Globus [23]. Nos centraremos en la versión 2 (*Globus Toolkit 2*, GT2), en la que están basados la mayoría de los proyectos Grid actuales. También hablaremos brevemente de la versión 3 (*Globus Toolkit 3*, GT3) al final de la sección.

El conjunto de herramientas de Globus (*Globus Toolkit*, GT) [23] es un *software* libre y de código abierto cuyo desarrollo está liderado por diversos grupos en *Argonne National Laboratory*, *University of Southern California* y *University of Chicago*.

Globus permite la construcción de entornos Grid a gran escala y el desarrollo de aplicaciones basadas en el Grid. Consta de tres pilares básicos: gestión de recursos, servicios de información y gestión de datos. Estos tres pilares están construidos sobre una infraestructura de seguridad común, tal y como se ilustra en la figura 1.1.

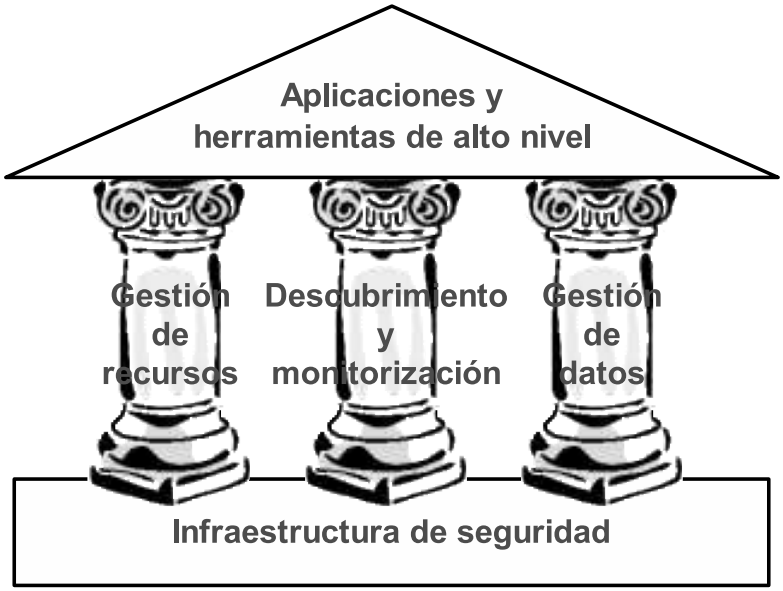

FIGURA 1.1: Componentes de Globus.

1.3.1. Infraestructura de Seguridad

Las principales motivaciones detrás de la infraestructura de seguridad de Globus (*Globus Security Infrastructure*, GSI) [24] son:

- La necesidad de comunicación segura (autenticada y quizás confidencial) entre los elementos de un Grid.
- La necesidad de soportar la seguridad a través de fronteras organizacionales, que imposibilita el uso de un sistema de seguridad gestionado de forma centralizada.
- La necesidad de proporcionar registro único (*single sign-on*) para los usuarios del Grid, incluyendo la delegación de credenciales para computaciones que requieren múltiples recursos y organizaciones.

GSI usa criptografía de clave pública (*Public Key Infrastructure*, PKI), también denominada criptografía de clave asimétrica, como base para su funcionalidad. Muchos de los términos y conceptos usados en esta descripción provienen de su uso en criptografía de clave pública [25].

GSI está basado en estándares o recomendaciones internacionales, como los certificados X.509 [26] y el protocolo de comunicaciones TLS (*Transport Layer Security*), descrito en el RFC 2246 [27] y antes conocido como SSL (*Secure Sockets Layer*). Se han añadido extensiones a estos estándares para proporcionar delegación y registro único, como se verá más adelante. La implementación de GSI en Globus se adhiere al GSS-API (*Generic Security Service API*), que es un API estándar para sistemas de seguridad promovido por el IETF (*Internet Engineering Task Force*) y descrito en los RFCs 2743 [28] y 2744 [29]. Además, se han propuesto extensiones a este estándar para soportar la delegación de credenciales [30].

Posiblemente, éste es el componente de Globus con más éxito. Puede ser usado independientemente del resto de componentes, incluso en entornos no relacionados con el Grid. A continuación se describen los componentes y funcionalidades más importantes de la infraestructura de seguridad de Globus.

Certificados

Cada usuario y servicio del Grid es identificado por medio de un certificado. Un certificado GSI incluye cuatro datos principales:

- El nombre del sujeto (*subject name*), que identifica a la persona u objeto representado por el certificado.
- La clave pública perteneciente al sujeto.
- La identidad de la autoridad de certificación (*Certification Authority*, CA) que firma el certificado para hacer constar que tanto la clave pública como la identidad pertenecen al sujeto.
- La firma digital de la CA.

Nótese que se necesita una tercera parte, la CA, para certificar el enlace entre la clave pública y el sujeto del certificado. Para confiar en el certificado y sus contenidos, se debe confiar en el certificado de la CA. El enlace entre la CA y su certificado debe establecerse por medio de métodos no criptográficos, de otra forma el sistema no sería fiable.

Los certificados GSI se codifican en el formato X.509 [26], un formato estándar establecido por el IETF en el RFC 3280 [31]. Estos certificados pueden compartirse con otro *software* basado en criptografía de clave pública, incluyendo los navegadores y gestores de correo comerciales de Microsoft y Netscape.

Los certificados X.509 se especifican usando la notación ASN.1 (*Abstract Syntax Notation number One*) [32] y se codifican en formato PEM (*Privacy Enhanced Mail*), descrito en el RFC 1421 [33]. En la figura 1.2 se muestra un certificado de usuario usado en el banco de pruebas IRISGrid (ver sección A.3 en los apéndices), en formato X.509, donde se pueden observar los campos de información comentados anteriormente, junto con otros como el periodo de validez, los algoritmos de cifrado, las versiones del formato utilizadas o las extensiones al estándar. La parte legible del certificado es la especificación ASN.1 y la parte ilegible es la misma especificación codificada en formato PEM.

Autenticación Mutua

Antes de que la autenticación mutua pueda ocurrir, cada parte involucrada debe primero confiar en las CAs que firmaron los certificados del resto de las partes. En la práctica, esto significa que cada parte debe tener copias de los certificados de las CAs y deben confiar en que esos certificados realmente pertenecen a las CAs.

Para autenticarse mutuamente, el primer elemento (A) establece una conexión con el segundo elemento (B). Para iniciar el proceso de autenticación, A le da a B su certificado. El certificado le dice a B quién afirma ser A (es decir, su identidad), cuál es su clave pública y qué CA usa para certificar su identidad. B deberá primero asegurarse de que el certificado es válido, comprobando la firma digital de la CA.

Una vez que B ha comprobado el certificado de A, B debe asegurarse de que A es realmente el elemento identificado en el certificado. Para ello, B genera un mensaje aleatorio y se lo envía a A para que lo encripte. A encripta el mensaje usando su clave privada y se lo devuelve a B. B desencripta el mensaje usando la clave pública de A. Si el mensaje resultante es igual que el mensaje aleatorio original, entonces B sabe que A es quien dice ser. Ahora que B confía en la identidad de A, la misma operación debe ocurrir en sentido inverso.

Confidencialidad e Integridad de la Comunicación

GSI puede usarse fácilmente para establecer una clave compartida (también llamada simétrica) para encriptar la comunicación si se desea que ésta sea confidencial. Por defecto,

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 26 (0x1a)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ES, O=irisgrid, CN=irisgrid CA
        Validity
            Not Before: Nov 14 09:53:38 2003 GMT
            Not After : Nov 13 09:53:38 2006 GMT
        Subject: C=ES, O=irisgrid, OU=LCASAT-CAB, \
                    CN=Eduardo Huedo Cuesta/Email=huedoce@inta.es
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:da:89:a2:52:c4:46:b3:be:8b:7d:ad:79:b7:d1:
                    86:de:82:87:c4:2b:8f:13:41:63:7a:21:f2:e9:ff:
                    91:3f:76:d1:ae:6d:db:56:bb:ff:18:03:0f:f4:9b:
                    ce:4f:c7:a2:b4:7e:24:bf:28:ba:18:cd:b1:4c:c9:
                    52:2b:18:9b:c3:47:b4:2d:68:51:cb:99:0b:97:eb:
                    91:f0:96:57:8f:38:a4:78:87:e0:83:74:97:2c:ca:
                    4c:02:fb:50:b6:96:47:d4:6d:41:1b:de:e2:af:d6:
                    80:fc:b0:c3:71:19:c0:93:9a:c5:b6:fa:08:25:a1:
                    b9:1a:a7:28:a4:af:45:f8:53
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Netscape Cert Type:
                SSL Client, SSL Server, S/MIME, Object Signing
    Signature Algorithm: md5WithRSAEncryption
        83:2f:15:ab:33:94:7f:d7:9d:48:4d:8c:e9:e2:5b:e4:5b:bf:
        a6:02:97:30:14:60:31:8c:61:1c:2f:a9:db:bc:a3:bd:a9:83:
        42:7c:9e:2d:9f:27:8c:a3:a6:82:61:83:f1:0d:bf:3e:18:7a:
        f6:e0:d1:7c:5a:81:d2:54:e9:3f:42:1c:7b:a7:07:18:a6:b6:
        c8:5a:fb:49:7a:30:16:b9:aa:de:7e:b2:c3:fa:c5:80:b9:23:
        5d:b5:3b:da:70:e0:f7:08:5f:6e:8b:05:8f:a2:6a:e4:4a:69:
        c1:83:a0:a3:b2:e6:b7:2d:eb:bf:6e:b7:99:49:37:ee:e9:21:
        4f:74
-----BEGIN CERTIFICATE-----
MIICNTCCAZ6gAwIBAgIBGjANBgkqhkiG9w0BAQQFADA2MQswCQYDVQQGEwJFUzER
MA8GA1UEChMIaXJpc2dyaWQxFDASBgNVBAMTC2lyaXNncmlkIENBMB4XDTAzMTEx
NDA5NTMzOFoXDTA2MTExMzA5NTMzOFowdDELMAkGA1UEBhMCRVMxETAPBgNVBAoT
CGlyaXNncmlkMRMwEQYDVQQLEwpMQ0FTQVQtQ0FCMRowGwYDVQQDExRFZHVhcmRv
IEh1ZWRvIEN1ZXN0YTEeMBwGCSqGSIb3DQEJARYPaHVlZG9jZUBpbnRhLmVzMIGf
MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDaiaJSxEazvot9rXm30YbegofEK48T
QWN6IfLp/5E/dtGubdtWu/8YAw/0m85Px6KOfiS/KLoYzbFMyVIrGJvDR7QtaFHL
mQuX65HwllePOKR4h+CDdJcsykwC+1C2lkfUbUEb3uKv1oD8sMNxGcCTmsW2+ggl
obkapyikr0X4UwIDAQABoxUwEzARBglghkgBhvhCAQEEBAMCBPAwDQYJKoZIhvcN
AQEEBQADgYEAgy8VqzOUf9edSE2M6eJb5Fu/pgKXMBRgMYxhHC+p27yjvamDQnye
LZ8njKOmgmGD8Q2/Phh69uDRfFqBOlTpPOIce6cHGKa2yFr7SXowFrmq3n6yw/rF
gLkjXbU72nDg9whfbosFj6Jq5EppwYOgo7Lmty3rv263mUk37ukhT3Q=
-----END CERTIFICATE-----
```

Figura 1.2: Certificado de usuario de IRISGrid

GSI no establece comunicación confidencial (es decir, encriptada) entre las partes, debido a la sobrecarga que introduce.

GSI sí proporciona por defecto (aunque puede desactivarse si se desea) integridad en la comunicación, de forma que se protege la información ante manipulaciones externas. Para ello, el emisor realiza un resumen (*digest*) del mensaje (ver RFC 1421 [33]) que encriptará con su clave privada. La integridad en la comunicación introduce algo de sobrecarga en la comunicación, pero no tanto como la encriptación.

Delegación de Certificados

GSI proporciona la capacidad de delegación: una extensión del protocolo TLS estándar que reduce el número de veces que el usuario debe introducir su contraseña. Si una computación en el Grid requiere que se usen múltiples recursos (requiriendo cada uno de ellos autenticación mutua), o si es necesario tener agentes (locales o remotos) que soliciten servicios en nombre del usuario, se puede eliminar la necesidad de reintroducir la contraseña del usuario creando un certificado delegado (*proxy certificate*).

Un certificado delegado consiste en un nuevo certificado (con una nueva clave pública) y una nueva clave privada. El nuevo certificado contiene la identidad del propietario, modificada para indicar que es delegado, y está firmado por el propietario en lugar de por la CA. Además, los certificado delegados tienen un tiempo de vida limitado (12 horas por defecto).

La clave privada del certificado delegado debe mantenerse segura, pero como éste no es válido por mucho tiempo, no es necesario mantenerla tan segura como la clave privada del propietario. Por tanto, es posible almacenar la clave privada del certificado delegado en el sistema de almacenamiento local sin encriptar, siempre que los permisos del fichero eviten que cualquiera pueda usarla fácilmente. Una vez que se crea y almacena el certificado delegado, el usuario puede usarlo junto con su clave pública para autenticarse mutuamente sin tener que introducir una contraseña.

El proceso de autenticación mutua varía ligeramente para soportar la delegación, ya que hay que enviar tanto el certificado delegado como el certificado del propietario y comprobar que el certificado delegado está firmado con la clave privada del propietario, y éste a su vez con la clave privada de la CA. Se establece así una cadena de confianza desde la CA hasta el certificado delegado a través del usuario.

Es posible tener más de un nivel de delegación. En ese caso, tendremos dos tipos de certificados delegados: completos (*full proxy*) y limitados (*limited proxy*). Los certificados

delegados completos pueden crear nuevos certificados delegados (limitados o completos), mientras que los limitados no pueden hacerlo. Algunos servicios, como GRAM, requieren ser accedidos con un certificado delegado completo, mientras que otros, como GASS o GridFTP, aceptan certificados limitados. Dado que GRAM usa un certificado delegado limitado para identificar a cada trabajo, los trabajos enviados no pueden enviar a su vez nuevos trabajos y, sin embargo, sí que pueden acceder a ficheros remotos mediante GASS o GridFTP.

GSI es el único *software* que soporta las extensiones de delegación de TLS. El proyecto Globus [22] ha trabajado activamente, junto con el GGF y el IETF, para establecer los certificados delegados como una extensión estándar de TLS, especificada en el RFC 3820 [34], de forma que puedan ser usados con cualquier otro *software* basado en TLS.

Como ejemplos, la delegación se utiliza para proporcionar registro único (*single sign-on*) a los recursos del Grid, para que un trabajo pueda acceder a un fichero remoto, para realizar transferencias entre terceros con GridFTP, para proporcionar comunicación segura entre subtrabajos co-asignados a varios recursos, etc.

1.3.2. Gestión de Recursos

El gestor de asignación de recursos de Globus (*Globus Resource Allocation Manager*, GRAM) [35] es el módulo que proporciona la ejecución remota, así como el control y monitorización de la misma. GRAM simplifica el uso de los sistemas remotos proporcionando un interfaz uniforme para la ejecución de trabajos sobre distintos gestores de recursos. Cuando un cliente envía un trabajo, la petición se envía al equipo remoto, donde es gestionada por un demonio guardián (*gatekeeper*). Entonces, el *gatekeeper* crea un gestor del trabajo (*job manager*) que inicia, controla y monitoriza el trabajo. Cuando el trabajo termina, el *job manager* envía la información de estado al cliente e igualmente termina.

GRAM utiliza un sistema de acceso global a almacenamiento secundario (*Global Access to Secondary Storage*, GASS) [36] para realizar la transferencia de ficheros de entrada y salida antes y después, respectivamente, del envío de un trabajo. Además, GASS dispone de un sistema de *cache* que permite la reutilización de los ficheros por varios trabajos.

La figura 1.3 muestra una vista conceptual de GRAM. Contiene los siguientes elementos:

- Cliente GRAM: En este caso se trata del comando `globusrun`, que usa el API de cliente GRAM para enviar un trabajo y (opcionalmente) arranca un servidor GASS para transferir el ejecutable y los ficheros de entrada y salida estándar.

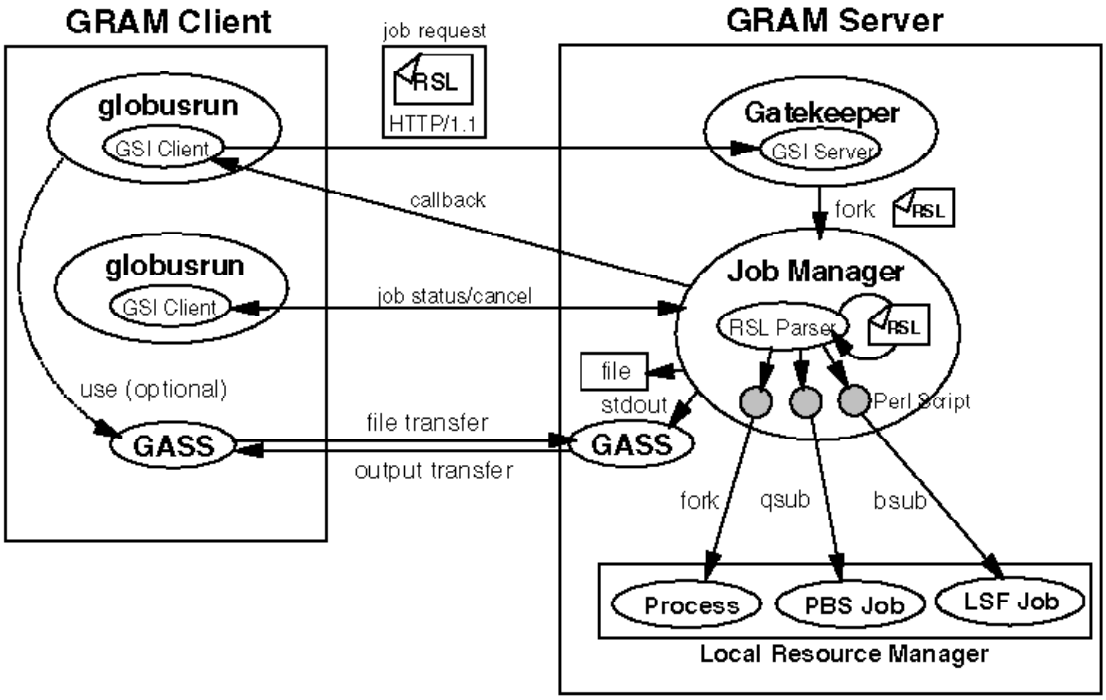

FIGURA 1.3: Componentes de GRAM [37].

- Lenguaje de especificación de recursos (*Resource Specification Language*, RSL): Es el lenguaje usado por los clientes para describir los recursos que se solicitan en una petición de envío de trabajo.

- Demonio guardián (*gatekeeper*): Crea un canal de comunicación segura entre clientes y servidores GRAM. Es similar al demonio de Internet (`inetd`) en términos de funcionalidad, sin embargo, el *gatekeeper* proporciona mecanismos de autenticación y autorización. Se comunica con el cliente GRAM, lo autentica y ve si está autorizado para enviar trabajos. Si es así, crea un *job manager* delegándole la autoridad para comunicarse con el cliente.

- Gestor del trabajo (*job manager*): Se crea como parte del proceso de petición de trabajos y proporciona los interfaces que controlan la asignación para cada gestor de recursos local, como PBS, LSF o SGE. Sus funciones son: procesar el RSL, transferir el ejecutable y los ficheros de entrada desde los clientes usando GASS, asignar la petición de trabajo al gestor de recursos local, enviar información sobre los cambios de estado del trabajo a los clientes (mediante *callbacks*), recibir peticiones de estado y cancelación de los clientes, y enviar los resultados de salida a los clientes usando

GASS.

- Servidor GASS: GRAM usa GASS para transferir ficheros entre el cliente y el recurso remoto.

Existe también un co-asignador (*Dynamically Updated Runtime On-line Co-allocator*, DUROC) [38] que se utiliza para asignar a la vez varios recursos, gestionados por GRAM, de forma coordinada.

1.3.3. Servicios de Información

El descubrimiento, caracterización y monitorización de recursos, servicios y trabajos son problemas desafiantes debido a la considerable diversidad, elevado número, comportamiento dinámico y distribución geográfica de las entidades en las que un usuario podría estar interesado. Consecuentemente, los servicios de información son una parte vital de cualquier infraestructura Grid, proporcionando los servicios fundamentales para el descubrimiento y la monitorización y, por tanto, para planificar y adaptar el comportamiento de las aplicaciones.

El servicio de descubrimiento y monitorización (*Monitoring and Discovery Service*, MDS) [39], antes conocido como servicio de directorio para meta-computación (*Metacomputing Directory Service*, MDS), proporciona acceso uniforme, flexible, escalable y eficiente a información estática y dinámica de recursos. MDS usa LDAP (*Lightweight Directory Access Protocol*), descrito en el RFC 2251 [40], como interfaz común a tal información.

LDAP es un protocolo cliente/servidor para almacenar información y responder a consultas. LDAP no es una base de datos, sino un protocolo para acceso a bases de datos y directorios. LDAP tiene dos tipos de comunicación: cliente a servidor y servidor a servidor. Las comunicaciones cliente a servidor básicas permiten a las aplicaciones de usuario contactar con un servidor LDAP para crear, recuperar (la función principal de MDS), modificar y borrar datos mediante los comandos LDAP estándar. Las comunicaciones servidor a servidor definen cómo los múltiples servidores de una red comparten los contenidos de un árbol de información LDAP y cómo actualizan y replican la información entre ellos.

LDAP se diseñó para contener pequeños registros de información en una estructura jerárquica. La estructura básica de LDAP es un simple árbol de información, que se asemeja mucho a la de un árbol de directorios en un sistema de ficheros. Comenzado en el nodo raíz, contiene una vista jerárquica de todos sus datos y proporciona un sistema de búsqueda en árbol. Este árbol se denomina árbol de información de directorio (*Directory Information*

Tree, DIT). Los subárboles del DIT contienen toda la información en un servidor LDAP determinado; además, los subárboles puede estar distribuidos o replicados.

Normalmente, se utiliza el formato LDIF (*LDAP Data Interchange Format*), descrito en el RFC 2849 [41], para definir los registros del DIT en forma de fichero de texto, que describe los distintos elementos dentro de un registro y la forma en que éste está organizado.

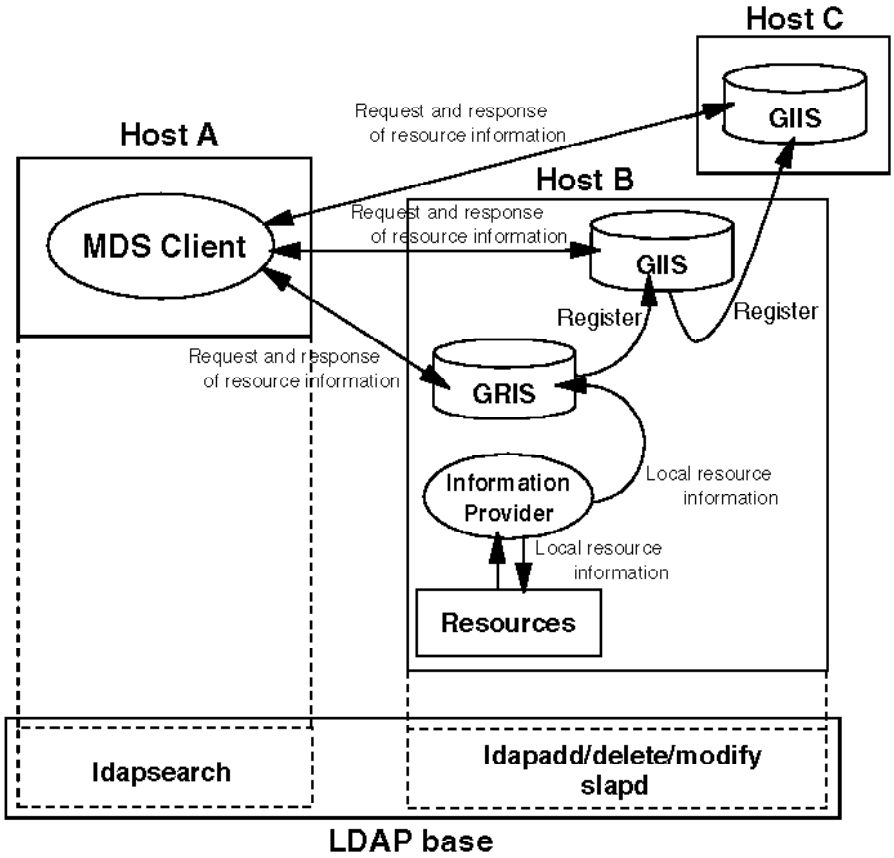

FIGURA 1.4: Componentes de MDS [37].

La figura 1.4 muestra una vista conceptual de MDS. Contiene los siguientes elementos:

- Cliente MDS: En este caso se trata del comando `ldapsearch`, que es un cliente genérico para el protocolo LDAP.

- Proveedor de información (*information provider*): Traduce las propiedades y el estado del recurso local al formato definido en el esquema y en los ficheros de configuración, usando el formato LDIF.

- Servicio de información de recursos del Grid (*Grid Resource Information Service*, GRIS): Es el depósito de la información local de cada recurso, derivada de los proveedores de información. La información local mantenida en el GRIS se actualiza cuando es solicitada y tiene un tiempo de vida (*Time-To-Live*, TTL) determinado.

- Servicio de información de índice del Grid (*Grid Index Information Service*, GIIS): Es un depósito que contiene índices a información de recursos registrada por los servidores GRIS y otros servidores GIIS. Se puede ver como un servidor de información a nivel del Grid o de una organización virtual. GIIS tiene una estructura jerárquica, similar a la del sistema DNS. Un servidor GRIS es capaz de registrar su información en uno o más servidores GIIS.

1.3.4. Gestión de Datos

Además de GASS, mencionado en la sección 1.3.2, Globus incluye componentes para la transferencia y gestión de datos en entornos Grid de alto rendimiento [42, 43, 44]. Por ejemplo, GridFTP [45, 46] proporciona transferencias de datos eficientes, seguras y confiables entre los nodos del Grid. La palabra GridFTP puede referirse a un protocolo, un servidor o un conjunto de herramientas.

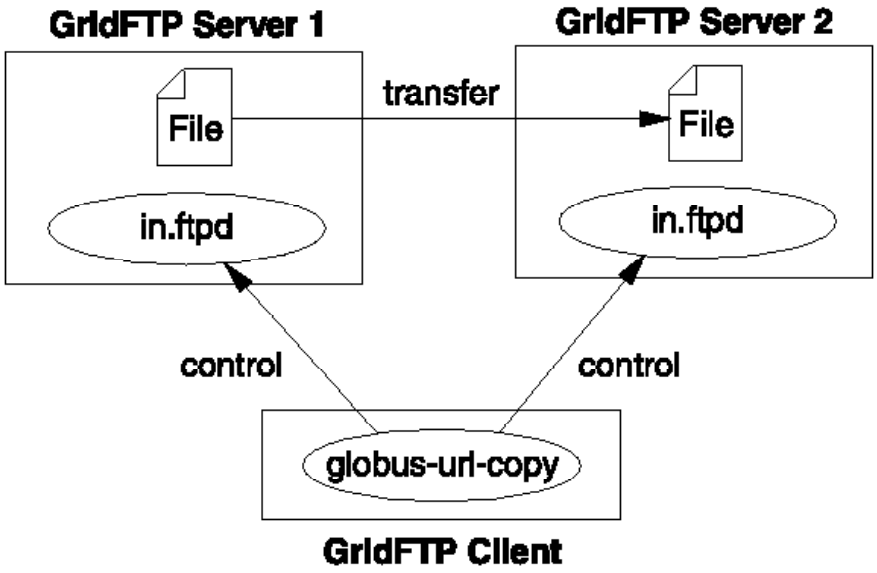

FIGURA 1.5: Transferencia entre terceros de GridFTP [37].

Se intenta que el protocolo GridFTP sea usado en todas las transferencias de datos del

Grid. Se basa en FTP (*File Transfer Protocol*), definido en el RFC 959 [47], extendiendo el protocolo estándar con funcionalidades como transferencias multicanal, auto-ajuste, segmentación de comandos, transferencia entre terceros y seguridad.

El conjunto de herramientas de Globus (GT) proporciona el servidor y el cliente GridFTP, que están implementados por el demonio `in.ftpd` y el comando `globus-url-copy`, respectivamente. Ambos componentes soportan la mayoría de las funcionalidades definidas en el protocolo GridFTP.

Globus también proporciona un servicio de catálogo de réplicas (*Globus Replica Catalog*) para registrar la localización de las copias de un fichero. Asimismo, proporciona un mecanismo, por encima de GridFTP y del catálogo de réplicas, para la gestión de réplicas (*Globus Replica Management*), que soporta operaciones de publicación, copia y borrado de réplicas de manera consistente y fiable [43]. En la actualidad se está integrando un servicio de localización de réplicas (*Replica Location Service*, RLS) desarrollado en el proyecto EDG.

1.3.5. Arquitectura de Servicios Grid

La nueva arquitectura abierta de servicios Grid (*Open Grid Services Architecture*, OGSA) [48, 49] muestra una clara convergencia hacia una arquitectura de sistemas Grid basados en servicios, conceptos y tecnologías *Web*, como la utilizada en el campo del *e-Business*. Esta evolución representa una gran oportunidad para lograr una amplia aceptación y difusión de la tecnología Grid, que puede extenderse, al igual que lo hizo la tecnología *Web*, desde su ámbito original, en el área de la computación científica, al de las aplicaciones comerciales.

Por otro lado, la infraestructura abierta de servicios Grid (*Open Grid Services Infrastructure*, OGSI) [50] se refiere a la infraestructura básica sobre la que se construye OGSA. En su núcleo está la especificación de servicio Grid (*Grid Service Specification*), que define los interfaces y comportamientos estándar de un servicio Grid, construido sobre una base de servicios *Web*.

La versión 3 del conjunto de herramientas de Globus (GT3) es una implementación de código abierto de la primera versión de la especificación OGSI, un bloque fundamental en el marco de trabajo OGSA. La figura 1.6 muestra la relación entre OGSA, OGSI y GT3 [51]. La tabla 1.1 muestra la correspondencia entre los componentes de GT2, estudiados en las secciones anteriores, y los servicios de GT3 [37].

Se están realizando esfuerzos en el GGF para documentar "buenas prácticas", guías de

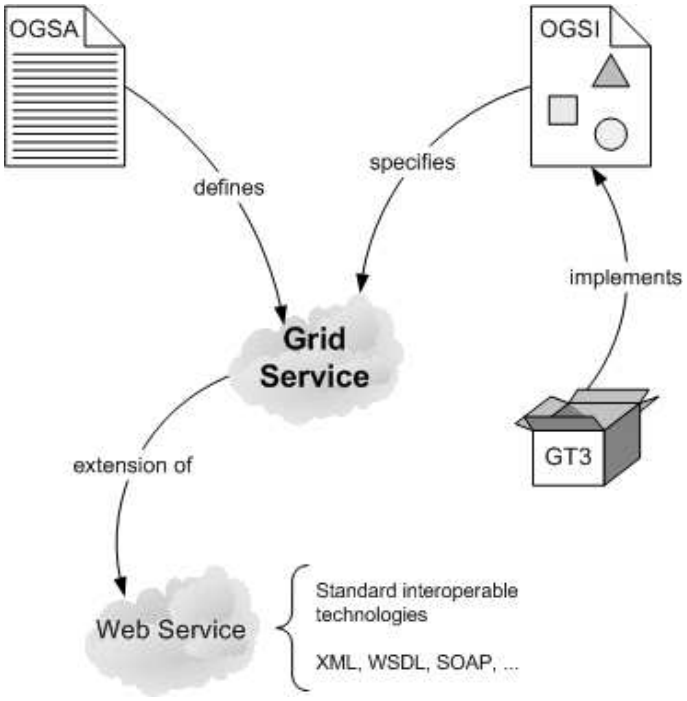

FIGURA 1.6: Relación entre OGSA, OGSI y GT3 [51].

implementación y estándares para tecnologías Grid. Entre los grupos de trabajo (*Working Group*, WG) del GGF relacionados con OGSA se incluyen los siguientes:

- *Open Grid Services Architecture Working Group* (OGSA-WG)
- *Open Grid Services Infrastructure Working Group* (OGSI-WG)
- *Open Grid Service Architecture Security Working Group* (OGSA-SEC-WG)
- *Database Access and Integration Services Working Group* (DAIS-WG)

Dado que OGSA está construido sobre los servicios *Web*, es muy posible que se incorporen especificaciones incluidas en el W3C (*World Wide Web Consortium*), IETF, OASIS (*Organization for the Advancement of Structured Information Standards*) y otras organizaciones de estandarización.

Recientemente, ha aparecido un nuevo marco de trabajo para modelar recursos mediante servicios *Web*, el WSRF (*Web Services Resource Framework*) [52, 53], que se perfila como sustituto del actual OGSI en la próxima versión de Globus (*Globus Toolkit 4*, GT4). Esta nueva infraestructura aprovecha desarrollos recientes en la arquitectura de servicios

TABLA 1.1: Correspondencia entre los componentes de GT2 y los servicios de GT3 [37].

Componentes de GT2	**Servicios de GT3**
GRAM *gatekeeper*	MMJFS (*Master Managed Job Factory Service*)
GRAM *job manager*	MJS (*Managed Job Service*)
MDS GIIS	*Index services*
MDS GRIS	SDE (*Service Data Element*) in MMJFS
GridFTP *server*	GridFTP *server*
GRAM *reporter*	SDE en MJS
`globus-url-copy`	RFT (*Reliable File Transfer*)

Web, como el direccionamiento de servicios *Web* (*WS-Addressing*), y elimina ciertas complicaciones introducidas por OGSI, como la necesidad de utilizar contenedores de servicios Grid específicos por encima de los contenedores de servicios *Web* estándar existentes en los servidores de aplicaciones. Este cambio en la infraestructura no afecta a la arquitectura OGSA, que continúa completamente vigente.

1.4. Motivación de la Investigación Desarrollada

La tecnología Grid no pretende competir con las tecnologías existentes dentro de un centro de investigación o empresa. Su objetivo es interconectar de forma débilmente acoplada recursos en Internet gestionados por diferentes tecnologías y políticas de seguridad, hoy en día incompatibles. Este nuevo paradigma requiere investigación y desarrollo sobre nuevos sistemas de ejecución capaces de reaccionar dinámicamente a las condiciones altamente variables de este entorno.

Desde su aparición en el año 1997, la tecnología Globus se ha convertido paulatinamente en el estándar de facto para la computación distribuida en Grids. Los componentes de Globus ofrecen la infraestructura básica necesaria para el desarrollo y ejecución de aplicaciones distribuidas. Estos componentes, ya sea de forma independiente o conjunta, facilitan el acceso transparente y seguro a recursos distribuidos geográficamente en múltiples dominios de administración, además de servir como soporte básico para implementar las fases de la planificación de trabajos en Grids, a saber: descubrimiento y selección de recursos; y preparación, envío, monitorización, migración y finalización de trabajos. Sin embargo, la explotación de forma agregada de recursos distribuidos geográficamente está lejos de ser sencilla ya que, a pesar de la relativa madurez de Globus, el usuario es responsable de

realizar manualmente las acciones de planificación de trabajos detalladas anteriormente. Además, Globus no ofrece ningún soporte para la migración de trabajos, y por lo tanto para la ejecución adaptativa necesaria en Grids dinámicos.

El objetivo principal de esta investigación es el estudio y desarrollo de técnicas relacionadas con el envío, la planificación y la gestión de trabajos en Grids computacionales heterogéneos y dinámicos. A continuación se describen las características dinámicas de un Grid y los requisitos de las nuevas aplicaciones auto-adaptativas. De este análisis se extraen las hipótesis de partida de la esta investigación.

1.4.1. Características Dinámicas del Grid

Existen diferentes modelos de Grid, desde entornos homogéneos y dedicados con alto grado de centralización de los servicios a modelos altamente heterogéneos y desacoplados. Como hipótesis de partida, a continuación describimos las cuatro características de nuestro modelo de Grid [54] que determinan la forma en la que debe realizarse la planificación y ejecución de trabajos sobre los recursos que lo constituyen:

- **Autonomía**: Los recursos del Grid está distribuidos geográficamente a través de múltiples dominios de administración y pertenecen a distintas organizaciones. Es necesario que se reconozca la autonomía de los propietarios de los recursos así como sus gestores de recursos locales y políticas de seguridad.

- **Heterogeneidad**: Un Grid supone una multiplicidad de recursos que son heterogéneos en naturaleza y abarcan un vasto rango de tecnologías (*clusters*, multiprocesadores, equipos de escritorio, sistemas de almacenamiento masivo...).

- **Escalabilidad**: Un Grid podría crecer desde unos pocos recursos integrados hasta millones. Esto suscita el problema de la degradación potencial del rendimiento a medida que el tamaño de un Grid se incrementa. Consecuentemente, las aplicaciones que requieren un gran número de recursos distribuidos geográficamente deben ser diseñadas para ser tolerantes a la latencia y al ancho de banda.

- **Dinamismo**: Tanto los gestores de recursos como las aplicaciones deben adaptar su comportamiento dinámicamente y usar los recursos y servicios disponibles de una manera eficiente y efectiva.

Estas características generales hacen que los entornos Grid presenten condiciones altamente variables de forma impredecible, que podemos clasificar del siguiente modo:

- **Alta frecuencia de fallos**: En un Grid los fallos, tanto de los recursos como de la red de interconexión, son la regla más que la excepción. De hecho, con tantos recursos la probabilidad de que alguno falle es bastante alta.

- **Disponibilidad dinámica de los recursos**: Los recursos compartidos en un Grid no pertenecen al mismo dominio de administración, de forma que una vez enviado el trabajo su propietario pierde el control del mismo en favor del administrador o sistema gestor del recurso, quien libremente puede cancelar o suspender su ejecución. Además, continuamente se están añadiendo o quitando recursos del Grid.

- **Carga dinámica de los recursos**: Los recursos de un Grid son explotados por usuarios internos además de por otros usuarios del Grid, de forma que los recursos que inicialmente se mostraban ociosos pueden saturarse durante la ejecución del trabajo y, viceversa, recursos con una alta carga de trabajo inicial pueden quedar desocupados.

- **Coste dinámico de los recursos**: El coste de los recursos puede variar en función de la hora del día o de la carga del propio recurso [55].

Por tanto, para obtener un grado razonable de rendimiento y tolerancia a fallos en la aplicación, los trabajos deben ser capaces de replanificarse y migrar a través de los recursos de un Grid adaptándose de acuerdo a las características, disponibilidad, carga y coste de sus recursos. La migración es la clave para la ejecución adaptativa de trabajos en un entorno Grid dinámico. Un trabajo puede ser abortado y migrado cuando se excede un límite de migración establecido en términos de métricas sobre el estado del sistema. De hecho, ésta es la aproximación seguida por la mayoría de herramientas de gestión de recursos distribuidos a nivel local o departamental descritas en la sección 1.1.

1.4.2. Aplicaciones Auto-Adaptativas

Debido a la naturaleza heterogénea y dinámica del Grid, el usuario final debe establecer los requisitos que deben cumplir los recursos destino y los criterios para clasificar los recursos que los cumplan. Usualmente, los requisitos se basan en atributos estáticos o poco dinámicos de los recursos (sistema operativo, arquitectura, disponibilidad de *software*...) mientras que los criterios de clasificación se basan, además, en atributos más dinámicos (coste, espacio en disco, carga de trabajo, memoria libre...).

Tradicionalmente, los requisitos de la aplicación se definen en el envío y se supone que permanecen invariables durante toda la vida del trabajo. Por otro lado, el rendimiento es monitorizado normalmente en términos de información del estado global del sistema. Sin embargo, la tecnología Grid emergente ha dado lugar a una nueva generación de aplicaciones que cuentan con su propia habilidad para adaptar su ejecución a las condiciones cambiantes del Grid [56, 57]. Este nuevo tipo de aplicaciones auto-adaptativas se caracterizan por:

- **Elección dinámica de recursos**: Las aplicaciones auto-adaptativas son capaces de tomar decisiones sobre su propia selección de recursos a medida que evoluciona su ejecución, esto es, generan dinámicamente sus requisitos y criterios de clasificación. Por ejemplo, los métodos numéricos de refinamiento adaptativo de malla, aumentan sistemáticamente el número de nodos de la malla computacional en aquellas regiones que precisan una mayor resolución. En este sentido, la cantidad de memoria física necesaria para albergar la simulación no se conoce a priori. Este tipo de aplicaciones podrían generar dinámicamente sus requisitos de memoria, asegurándose siempre de que se ejecutan en el recurso adecuado.

- **Generación de un perfil de rendimiento**: Las aplicaciones auto-adaptativas pueden generar un registro de rendimiento en función de métricas intrínsecas de la aplicación, para poder así detectar un deterioro en el aprovechamiento del recurso. Por ejemplo, el registro de rendimiento de un método iterativo podría incluir el tiempo empleado en cada iteración, mientras que un código de física de partículas podría mostrar el número de colisiones calculadas por segundo.

1.4.3. Eventos de Replanificación

En relación al alto grado de variabilidad de los recursos de un Grid y las características auto-adaptativas de las aplicaciones discutidas previamente, en esta investigación hemos considerado las siguientes causas de replanificación:

- Replanificación originada por el dinamismo del Grid:
  - Se descubre un recurso "mejor" que el actual (migración oportunista).
  - El recurso remoto (*hardware*, *software* o conexión de red) falla (recuperación ante fallos).

  - El trabajo es cancelado o suspendido.

- Replanificación originada por la auto-adaptación de las aplicaciones:

  - Se detecta un deterioro del rendimiento (violación de contrato de rendimiento).
  - La aplicación varía sus demandas de recursos (auto-migración).

La replanificación de un trabajo puede conducir a su migración si se considera que ésta es factible y merece la pena. Para evaluar la posibilidad de realizar una migración se debe tener en cuenta el motivo de la replanificación, el estado de los recursos candidatos y del recurso actual, el estado del resto de trabajos, etc.

1.5. Conclusiones

Nuestra hipótesis de partida es que el Grid únicamente se extenderá si se mantiene un modelo altamente desacoplado, semejante al usado hoy día en Internet, por ejemplo, para la gestión de los paquetes. El envío, la planificación y la gestión de trabajos en un entorno tan complejo como el Grid continúa siendo un desafiante tema de investigación:

- El usuario debe ser un experto y realizar manualmente cada una de las fases de planificación necesarias para enviar un trabajo.
- Debido a la naturaleza dinámica del Grid, el rendimiento obtenido y la tolerancia a fallos son actualmente muy bajos.
- Los usuarios requieren soporte para poder desarrollar aplicaciones auto-adaptativas.

A continuación, se describe la estructuración en capítulos del resto de esta memoria. En el capítulo 2 se enumeran los pasos necesarios para realizar la planificación y ejecución de trabajos en Grids y se presentan las técnicas utilizadas para tolerar el dinamismo del Grid y la auto-adaptación de las aplicaciones.

En el capítulo 3 se presenta el principal resultado de la investigación realizada, que constituye el desarrollo de una tecnología que, usando los servicios básicos ofrecidos por Globus, permite a los usuarios lanzar trabajos y olvidarse de los detalles de implementación y las características heterogéneas y dinámicas de un Grid. Se pretende contribuir a que el acceso a los recursos remotos por parte de los científicos para cubrir sus necesidades de cálculo sea tan sencillo, flexible y fiable como lo es el uso de la corriente eléctrica.

Las herramientas *software* generadas en los proyectos de investigación en desarrollo hoy día para realizar estas acciones [54] o bien no soportan migración y auto-adaptación de trabajos o bien están basadas en complejas arquitecturas de servicios que, desde nuestro punto de vista, son poco realistas y no son fieles a la filosofía orientada a usuario del Grid.

En el capítulo 5 se presentan los resultados de aplicar la tecnología desarrollada en esta investigación al campo de la Bioinformática. En concreto, se muestra el proceso de adaptación de una aplicación existente, que predice la estructura y propiedades termodinámicas de secuencias de proteínas, para su ejecución en el Grid con la ayuda de nuestra herramienta experimental.

En el capítulo 6 se presenta un campo de investigación en auge, que es la evaluación del rendimiento en Grids. En ese capítulo, se evalúa un banco de pruebas basado en Globus usando un conjunto de *benchmarks* para el Grid junto con una serie de métricas propuestas.

Finalmente, en el capítulo 7 se presentan las conclusiones y principales aportaciones de esta investigación, y se adelantan las principales líneas de trabajo futuras.

Se han incluido también dos apéndices a esta memoria. El primero de ellos muestra en detalle los bancos de prueba utilizados en los distintos experimentos que se presentan, mientras que el segundo es un manual de referencia de la herramienta experimental desarrollada.

2. Planificación y Ejecución de Trabajos en Grids

En este capítulo se enumeran los pasos necesarios para realizar la planificación y ejecución de trabajos en Grids. Además, se presentan las técnicas utilizadas para tolerar el dinamismo del Grid y la auto-adaptación de las aplicaciones. Por último, se hace un repaso de los proyectos de investigación relacionados.

2.1. Introducción

La planificación en Grid (también denominada super-planificación o meta-planificación) se ha definido [13] como el proceso de planificar trabajos sobre recursos pertenecientes a múltiples dominios de administración basándose en una política definida en términos de requisitos de los trabajos, productividad del sistema, tiempo de respuesta de la aplicación, restricciones de presupuesto, fechas tope, etc.

En este contexto, un *recurso* se define como algo que se comparte y que puede o necesita ser planificado (sistema computacional, enlace de red, espacio en disco...), mientras que un *trabajo* se define como una tarea que necesita un recurso para su realización (ejecución de una aplicación, transferencia de datos, almacenamiento de un fichero...).

Como ya se discutió en la sección 1.4.1, un Grid se caracteriza por cuatro aspectos fundamentales [54]:

1. Autonomía.
2. Heterogeneidad.
3. Escalabilidad.
4. Dinamismo.

Estas características determinan completamente la forma en la que debe hacerse la planificación y ejecución en Grids. Por ejemplo, la escalabilidad y la autonomía de los

múltiples dominios de administración hacen poco deseable la existencia de un único intermediario (*broker*) de recursos centralizado, con total control sobre las peticiones de los clientes y el estado de los recursos. Por otra parte, las características dinámicas de los recursos (en términos de disponibilidad, capacidad y coste) hacen esencial la habilidad de adaptar la planificación y ejecución de los trabajos a las condiciones cambiantes. Además, la heterogeneidad de los recursos implica un grado más de dificultad.

2.2. Fases de la Planificación y Ejecución de Trabajos en Grids

Hasta hace muy poco, el planificador de Grid más común era el propio usuario. En las siguientes secciones, se describen los pasos que debe seguir un usuario o herramienta para planificar trabajos en un Grid según el GGF [58, 59, 60].

Estos pasos de planificación no tienen por qué realizarse en el orden indicado. De hecho, en nuestro trabajo hemos modificado ligeramente el orden y agrupación de las etapas de planificación descritas en la sección anterior para aportar mayor flexibilidad. En el capítulo 3 se comentarán estas etapas en detalle y se describirá la implementación realizada en la herramienta Grid*W*ay.

2.2.1. Fase 1. Descubrimiento de Recursos

El descubrimiento de recursos implica que el planificador debe seleccionar un conjunto de recursos que serán investigados con mayor detalle en la siguiente fase, la de recopilación de información. Al inicio de esta fase, el conjunto potencial de recursos es el conjunto vacío, y al final, es algún subconjunto que ha pasado un requisito mínimo de viabilidad. La mayoría de planificadores hacen esto en tres pasos: filtro de autorización, conocimiento de los requisitos del trabajo y filtro para que se cumplan los requisitos mínimos del trabajo.

Paso 1. Filtro de Autorización

En un Grid, no todos los usuarios tienen acceso a todos los recursos, ya que normalmente la autorización se realiza localmente y, por tanto, depende de las políticas de seguridad existentes en cada organización. Generalmente, se asume que el planificador conoce o tiene la forma de saber a qué recursos tiene el usuario acceso en términos de servicios básicos.

Al final de este paso, el planificador tendrá una lista de máquinas o recursos a los que el usuario tiene acceso.

Paso 2. Definición de los Requisitos de la Aplicación

Para proceder al descubrimiento de recursos, se deben especificar un conjunto mínimo de requisitos del trabajo para que el planificador pueda filtrar posteriormente el conjunto de recursos que no sean factibles (ver paso 3). El conjunto de posibles requisitos del trabajo puede ser muy amplio, y varía significativamente entre trabajos. Puede incluir detalles estáticos, como el sistema operativo o la arquitectura para la que está disponible o es más apropiado un ejecutable. También es posible especificar detalles dinámicos, por ejemplo, una cantidad mínima de memoria libre, necesidades de conectividad, almacenamiento temporal necesario, etc. Cuantos más detalles se incluyan, mejor se podrá realizar el emparejamiento entre trabajo y recurso.

Uno de los métodos más utilizados para realizar el emparejamiento entre trabajo y recurso es el *matchmaking* de Condor [61], donde los trabajos y recursos se describen mediante anuncios clasificados (*Classified Advertisements*, ClassAds) escritos en un lenguaje formal, simple pero potente, que describe sus atributos y políticas de asignación.

Paso 3. Filtro de Requisitos Mínimos

Dado un conjunto de recursos a los que el usuario tiene acceso y el conjunto de requisitos mínimos que tiene el trabajo, el tercer paso en la fase de descubrimiento de recursos consiste en filtrar los recursos que no cumplen los requisitos mínimos del trabajo. El planificador normalmente realiza este paso recorriendo la lista de recursos y eliminando aquéllos que no cumplen los requisitos del trabajo que se conozcan. También se podría combinar con la recogida de información más detallada de cada recurso (paso 4), de hecho así es como la mayoría de los sistemas propuestos lo hacen. Sin embargo, al hacerlo por separado, el planificador puede eliminar los recursos inapropiados en esta fase para simplificar la recogida de información de la fase siguiente.

2.2.2. Fase 2. Selección de Recursos

Dado un grupo de posibles recursos (o conjuntos de recursos), los cuales cumplen los requisitos mínimos del trabajo, se debe seleccionar un único recurso (o conjunto de recursos)

para planificar el trabajo en él. Esto se realiza normalmente es dos pasos: recogida de información y toma de decisión.

Paso 4. Recopilación de Información

Para hacer el mejor emparejamiento posible entre trabajo y recurso, el planificador necesita recoger información dinámica acerca de los recursos. Dependiendo de la aplicación y del recurso en cuestión, la información necesaria puede ser diferente. El planificador debe conocer la carga en las distintas máquinas o el estado del sistema de colas. Además, también juegan un papel importante las características físicas y los requisitos de *software* – ¿tiene la máquina el compilador necesario?, ¿tiene espacio de disco suficiente para los datos?, etc. También hay cuestiones acerca de la localización y la conectividad – ¿está la máquina suficientemente cerca de donde se almacenan los datos? [62]. Todos estos problemas se multiplican en el caso de selección de múltiples recursos [63]. La realización de reservas adelantadas (paso 6) puede ayudar en esta fase, aunque no es estrictamente necesaria.

Paso 5. Selección del Sistema

Una vez recogida la información en el paso anterior, en este paso el planificador toma una decisión sobre a qué recurso (o conjunto de recursos) debe enviar el trabajo. Esto puede hacerse de diferentes maneras, por ejemplo se puede asignar a cada recurso un valor numérico, o rango, que estime la idoneidad de los recursos para ejecutar el trabajo. El rango se puede calcular usando modelos de rendimiento más o menos complicados, como se verá en el capítulo 4. Se podría realizar además una selección especulativa, donde un trabajo se envía a múltiples recursos y una vez que comienza a ejecutarse, o incluso una vez que termina de hacerlo, en alguno de ellos se cancela el resto de envíos.

2.2.3. Fase 3. Ejecución del Trabajo

La fase de ejecución del trabajo implica varios pasos, de los cuales pocos han sido definidos de una manera uniforme entre los recursos, aunque iniciativas como GRAM [35] o DRMAA [64], que se comentarán más adelante, suponen una gran ayuda.

Paso 6. Reserva Adelantada (Opcional)

Puede darse el caso de que para hacer mejor uso de un sistema dado o para proporcionar garantías de funcionamiento (QoS), la totalidad o parte de los recursos tengan que ser

reservados por adelantado. Dependiendo del recurso, esto puede ser más o menos fácil de hacer y podría hacerse de manera automática o mediante intervención humana. Además, las reservas podrían expirar, quizás con un coste asociado.

Paso 7. Envío del Trabajo

Una vez que se han elegido los recursos, la aplicación debe enviarse. Esto puede ser tan sencillo como ejecutar un único comando o tan complicado como ejecutar una serie de guiones, y puede o no incluir la preparación del sistema o la transferencia de ficheros (ver paso 8).

Paso 8. Tareas de Preparación

El paso de preparación puede implicar la configuración del sistema, la transferencia de ficheros, la solicitud de una reserva u otras acciones necesarias para preparar el entorno de trabajo en el recurso antes de ejecutar la aplicación.

Paso 9. Monitorización del Progreso

Dependiendo de la aplicación y de su tiempo de ejecución, el planificador podría monitorizar el progreso de la aplicación y posiblemente cambiar de opinión acerca de dónde o cómo se está ejecutando, por lo que podría realizarse una migración del trabajo, es decir, volver de nuevo al paso 4.

Paso 10. Tareas de Finalización

Cuando el trabajo termina, el planificador debe reparar en ello y, en ese momento, podría proceder a recuperar determinados ficheros del recurso remoto para realizar un análisis de los resultados, limpiar el entorno de ejecución, eliminar ajustes temporales, etc.

2.3. Técnicas Adaptativas para Planificar y Ejecutar Trabajos en Grids

Debido a las condiciones cambiantes del Grid y a las necesidades dinámicas de las aplicaciones (ver secciones 1.4.1 y 1.4.2), es necesario establecer técnicas adaptativas para la planificación y ejecución de los trabajos.

2.3.1. Planificación Adaptativa

La única forma de obtener planificaciones fiables es considerando las características dinámicas de los recursos disponibles en el Grid [65, 66, 67, 68]. En general, la planificación adaptativa (*adaptive scheduling*) puede considerar, entre otros, factores como disponibilidad, rendimiento, carga, proximidad (en términos de ancho de banda y latencia de los enlaces de interconexión, como se verá en el capítulo 4), etc. escalados apropiadamente de acuerdo a las necesidades de la aplicación. El planificador debe recoger periódicamente información del Grid para planificar adaptativamente las tareas pendientes, de acuerdo a las demandas de la aplicación y al estado de los recursos del Grid, como se explicará en la sección 3.4.

2.3.2. Ejecución Adaptativa

Para obtener un grado razonable de rendimiento y tolerancia a fallos en las aplicaciones, los trabajos deben ser capaces de migrar entre los recursos del Grid adaptándose a eventos generados dinámicamente tanto por el Grid como por la aplicación en ejecución [69, 70, 71]. Esto es lo que se conoce como ejecución adaptativa (*adaptive execution*). El planificador debe evaluar cada evento de replanificación para decidir si la migración es factible y merece la pena, como se verá en la sección 3.5. Algunas razones, como la cancelación o el fallo del trabajo, hacen que se inicie inmediatamente el proceso de migración. Otras razones, como el descubrimiento de un nuevo recurso, hacen que el proceso de migración se inicie sólo si el nuevo recurso descubierto es "mejor" que el actual. En este caso, el tiempo de finalización [71, 69] y el coste de transferencia de ficheros de entrada y reinicio [62] deben ser también considerados, como se verá en el capítulo 4.

2.4. Planificación de Aplicaciones de Alta Productividad

En esta sección se describe una política de planificación para aplicaciones de alta productividad y, en concreto, las de barrido de parámetros. Se presenta un algoritmo de planificación que combina: (i) planificación adaptativa, para reflejar las características dinámicas del Grid; (ii) ejecución adaptativa, para migrar trabajos en ejecución a "mejores" recursos y proporcionar tolerancia a fallos; (iii) reutilización de ficheros comunes entre tareas, para reducir la sobrecarga por transferencia de ficheros.

El Grid constituye una plataforma muy apropiada para ejecutar aplicaciones de barrido de parámetros (*Parameter Sweep Application*, PSA) [72, 73, 74], que aparecen de manera natural en muchos campos científicos y de ingeniería como Bioinformática, Dinámica de Fluidos Computacional (*Computational Fluid Dynamics*, CFD), Física de Partículas, etc. Este tipo de aplicaciones consiste en la ejecución de un gran número de tareas, cada una de las cuales realiza un cálculo sobre un subconjunto de valores de un parámetro. En este trabajo, consideraremos una PSA como un conjunto de tareas independientes, que potencialmente comparten ficheros comunes (por ejemplo, el ejecutable o algunos ficheros de entrada). A pesar de la estructura relativamente simple de este tipo de aplicaciones, su ejecución eficiente en Grids computacionales conlleva problemas desafiantes, principalmente debido a la propia naturaleza del Grid.

La planificación adaptativa en Grids (ver sección 2.3.1) ha sido estudiada ampliamente y es generalmente aceptada como la solución para el dinamismo del Grid [65, 66, 68, 74, 75]. Trabajos previos han demostrado claramente el factor crítico de la información dinámica recogida del Grid para generar planificaciones fiables.

Adicionalmente, la ejecución adaptativa en Grids (ver sección 2.3.2) proporciona la capacidad de migrar trabajos en ejecución a recursos más apropiados, basándose en eventos generados dinámicamente tanto por el Grid como por las aplicaciones en ejecución, y puede contribuir también a mejorar el rendimiento y tolerancia a fallos obtenidos por las aplicaciones en un Grid [70, 71, 69, 76, 75].

Probablemente, uno de los factores más relevantes cuando se planifican múltiples tareas es el impacto de los tiempos de transferencia de ficheros. Para ejecutar eficientemente este tipo de aplicaciones, la planificación debe potenciar la reutilización de los ficheros compartidos entre tareas. Se han propuesto varias heurísticas para planificar aplicaciones de barrido de parámetros considerando requisitos de E/S de ficheros [67]. El planificador puede aprovechar la compartición de ficheros mediante el uso de la *cache* de GASS o el solapamiento de las fases de preparación y finalización de trabajos (ver sección 3.5) de tareas diferentes enviadas simultánea o consecutivamente al mismo recurso. Esto es especialmente importante no sólo para reducir la sobrecarga de la transferencia de ficheros, sino también para impedir la saturación del servidor de ficheros donde se almacenan, que puede ocurrir con PSAs a gran escala.

Estos tres aspectos –planificación adaptativa, ejecución adaptativa y reutilización de ficheros compartidos– deben combinarse para elaborar una política de planificación en Grid para PSAs. La figura 2.1 muestra la estructura general del algoritmo de planificación

desarrollado. Este algoritmo combina las consideraciones anteriores para ejecutar eficientemente aplicaciones de barrido de parámetros en Grids computacionales.

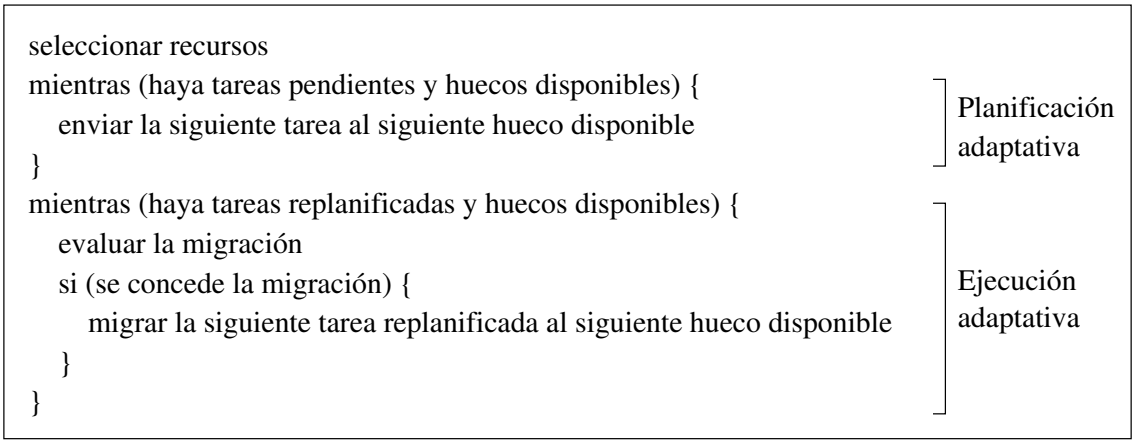

FIGURA 2.1: Algoritmo de planificación para aplicaciones de alta productividad.

2.5. Investigación Relacionada

2.5.1. Condor/G

Condor [7] es un sistema de gestión de recursos (*Resource Management System*, RMS) para trabajos intensivos en computación. Condor proporciona gestión de trabajos, política de planificación, esquema de prioridades y monitorización y gestión de recursos. Los usuarios envían sus trabajos a Condor y éste posteriormente elige cuándo y dónde ejecutarlos (según la política), monitoriza su progreso y finalmente informa al usuario de su finalización.

La arquitectura innovadora de Condor y sus mecanismos únicos le permiten funcionar bien en entornos en los que un RMS tradicional flaquearía –áreas como computación de alta productividad (*High Throughput Computing*, HTC) sostenida y computación oportunista. El objetivo de un entorno de computación de alta productividad es proporcionar grandes cantidades de potencia computacional tolerante a fallos sobre períodos de tiempo prolongados, utilizando de manera efectiva todos los recursos disponibles en la red. El objetivo de la computación oportunista es utilizar recursos en cualquier momento que estén disponibles, sin requerir el 100 % de disponibilidad. Los dos objetivos se acoplan de una forma natural, ya que la computación de alta productividad se consigue más fácilmente a través de principios oportunistas. Algunos de los mecanismos innovadores de Condor son el

lenguaje ClassAds para la definición de trabajos y recursos, el mecanismo de *matchmaking* para realizar el emparejamiento entre trabajos y recursos [61], la migración de trabajos y las llamadas al sistema remotas.

Condor/G [77] representa el matrimonio de tecnologías de los proyectos Globus y Condor. De Globus proviene el uso de protocolos para comunicaciones seguras entre dominios y el acceso estandarizado a una variedad de sistemas de gestión de recursos remotos. De Condor provienen cuestiones más cercanas al usuario como el envío de trabajos, la asignación de recursos, la recuperación de errores y la creación de un entorno de ejecución amigable. El resultado es muy beneficioso para el usuario final, que es capaz de utilizar grandes colecciones de recursos, que atraviesan múltiples dominios de administración, como si pertenecieran al dominio personal del usuario (ver figura 2.2).

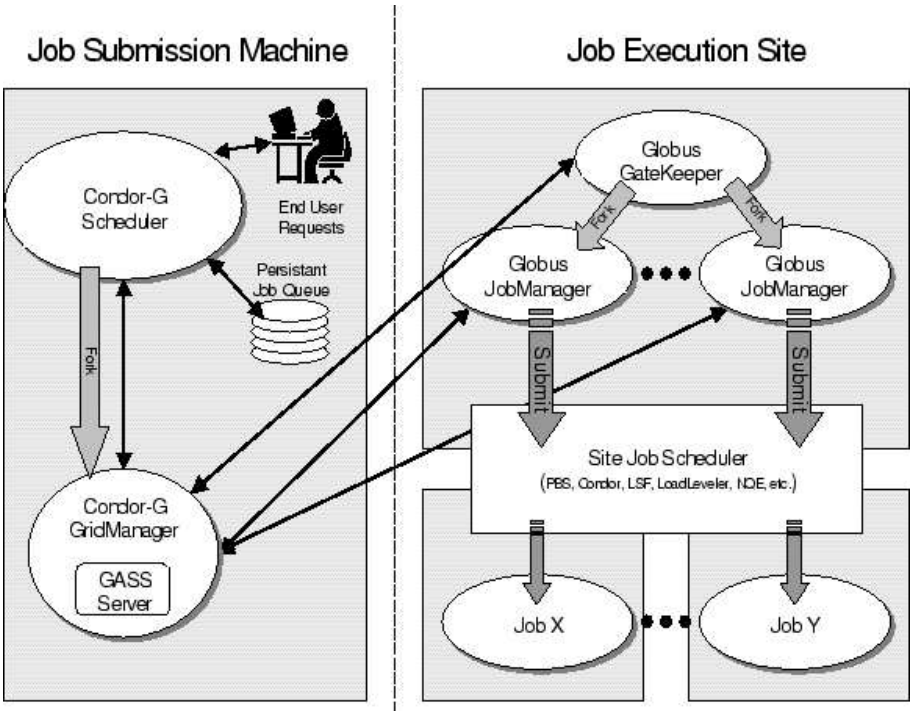

FIGURA 2.2: Ejecución remota de trabajos mediante Condor/G en recursos gestionados con Globus [77].

La tecnología de Condor puede existir tanto en los clientes (*front-ends*) como en los servidores (*back-ends*) de un entorno *middleware*. Condor/G se puede usar como un servicio confiable de gestión de trabajos, mientras que Condor se puede usar como un servicio de gestión de recursos (un generador del Grid), y finalmente los servicios de Globus se pueden usar como un puente entre ambos. Como ejemplo, el componente *Grid Resource Broker*

del proyecto EDG (ver siguiente sección) usa Condor/G como servicio de envío confiable de trabajos.

2.5.2. European Data Grid (EDG)

El proyecto DataGrid europeo (*European Data Grid*, EDG) [78], liderado por el CERN (*Conseil Européen pour la Recherche Nucléaire*) y financiado por la Unión Europea, tiene el objetivo de establecer un Grid computacional intensivo en datos para el análisis de datos provenientes de la exploración científica. La principal aplicación impulsora del proyecto DataGrid es el acelerador LHC (*Large Hadron Collider*), que operará en el CERN desde el 2005 hasta el 2015, aproximadamente. El LHC representa un salto adelante en cuanto a energía, densidad y frecuencia de colisión de los haces de partículas. Este salto es necesario para producir algunos ejemplos de partículas no descubiertas previamente, como el bosón de Higgs o tal vez quarks y leptones super-simétricos. El LHC presentará algunos desafíos en términos de computación y gestión de datos. El principal objetivo de la iniciativa DataGrid es desarrollar y probar la infraestructura que permitirá la creación de colaboratorios[1] científicos, donde los investigadores y científicos realizarán sus actividades sin tener en cuenta su localización geográfica. Esos colaboratorios permitirán interacciones personales, así como la compartición de datos e instrumentos, de forma global. El proyecto está diseñando y desarrollando soluciones de *software* y bancos de prueba (*testbeds*) escalables para manejar muchos Peta-bytes de datos distribuidos, decenas de miles de recursos de computación y miles de usuarios simultáneos de múltiples instituciones.

El proyecto EDG se divide en doce paquetes de trabajo (*Work Package*, WP) distribuidos en cuatro grupos de trabajo (*Working Group*, WG), como se muestra en la figura 2.3:

- *Middleware* de datos y computación: Cada uno de estos WPs desarrollarán partes específicas y bien definidas del *middleware* del Grid. Cada uno de ellos puede verse como un pequeño proyecto en sí mismo.

    1. Gestión de la carga de trabajo.
    2. Gestión de datos.
    3. Servicios de monitorización.
    4. Gestión de la infraestructura.
    5. Gestión de almacenamiento masivo.

[1] del inglés *collaboratory* (*collaboration* + *laboratory*)

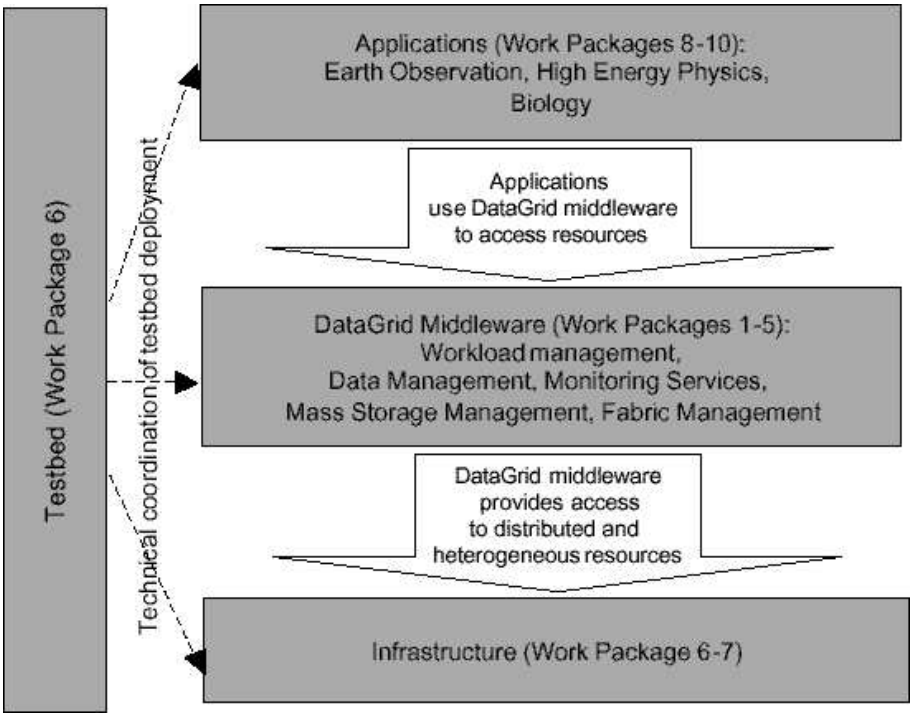

FIGURA 2.3: Estructura de los WPs del proyecto EDG [78].

- Banco de pruebas e infraestructura: Para el éxito del proyecto, es necesaria una infraestructura internacional con calidad de producción. En estos WPs, se cotejarán todos los desarrollos de los WPs 1 a 5 y se integrarán en sucesivas distribuciones de *software*. También recogerán la realimentación obtenida por los experimentos de aplicación y se la transmitirán a los desarrolladores, de forma que enlazarán el desarrollo, las pruebas y las experiencias de los usuarios.

  6. Banco de pruebas y demostraciones.
  7. Servicios de red.

- Aplicaciones: Estos WPs proporcionarán experimentos con aplicaciones que probarán y realimentarán los WPs de desarrollo de *middleware*, a través de los WPs del banco de pruebas.

  8. Aplicaciones de Física de altas energías (*High Energy Physics*, HEP).
  9. Aplicaciones de observación de la Tierra (*Earth Observation Science*, EOS).
  10. Aplicaciones de Biología.

- Gestión: Estos WPs asegurarán una diseminación activa de los resultados del proyecto y una gestión profesional.

  11. Explotación y diseminación.
  12. Gestión del proyecto.

El trabajo hace hincapié en posibilitar el proceso distribuido de aplicaciones intensivas en datos en el área de la Física de altas energías, la observación de la Tierra y la Biología. El principal desafío que encara el proyecto es proveer la forma de compartir ingentes cantidades de datos distribuidos a través de la infraestructura de red actual. El proyecto EDG descansa sobre las tecnologías Grid emergentes, especialmente Globus, que se espera que posibiliten la creación de un entorno computacional a gran escala consistente en colecciones de ficheros, bases de datos, computadores, instrumentos científicos y dispositivos.

En cuanto a la planificación de trabajos (WP1), ésta se basa en el mecanismo de *matchmaking* de Condor [61] y el uso del lenguaje ClassAds para especificación de recursos y trabajos. El componente *Grid Resource Broker* intenta emparejar cada trabajo con el recurso más adecuado para él, según las características de cada uno, teniendo en cuenta los siguientes factores [79]:

- Localización de los ficheros de entrada y salida.
- Autorización para usar un recurso.
- Asignación de recursos para el usuario o grupo al que el usuario pertenece.
- Requisitos y preferencias del usuario.
- Estado de los recursos disponibles.

Actualmente, el proyecto EGEE (*Enabling Grids for E-science in Europe*) [80] supone la continuación del proyecto EDG (junto con sus sucesores LCG y CrossGrid) y aspira a integrar los esfuerzos sobre Grid existentes a nivel nacional, regional y temático para crear una infraestructura de Grid europea al servicio de la investigación. Esta infraestructura se construirá sobre GÉANT, la red de investigación europea, y explotará la experiencia sobre Grid obtenida con anteriores proyectos financiados por la Unión Europea, como el proyecto EDG, e iniciativas nacionales sobre Grid, como e-Science en Reino Unido e IRISGrid en España (ver sección A.3 en los apéndices).

2.5.3. Nimrod-G

Nimrod-G [66] es un intermediario (*broker*) de recursos del Grid que realiza la gestión y planificación de recursos Grid a nivel mundial para aplicaciones paramétricas. Consiste en cuatro componentes clave: motor de granja de tareas (*Task Farming Engine*, TFE), planificador, despachador y agentes (ver figura 2.4). El motor de granja de tareas programable y persistente permite el uso de planificadores definidos por el usuario y aplicaciones personalizadas o entornos de resolución de problemas (*Problem Solving Environment*, PSE) en lugar de los componentes por defecto. El despachador usa los servicios de Globus para desplegar agentes Nimrod-G en recursos remotos, que gestionan la ejecución de los trabajos asignados. El sistema de gestión local inicia la ejecución del agente Nimrod-G, que interacciona con el servidor de E/S ejecutándose en la máquina del usuario para obtener el guión de tarea que le ha asignado el planificador y ejecuta los comandos Nimrod especificados en el guión. El planificador tiene la habilidad de alquilar recursos y servicios Grid, dependiendo de su capacidad, coste y disponibilidad, guiándose por los requisitos de calidad de servicio (*Quality of Service*, QoS) del usuario. Nimrod-G proporciona descubrimiento, selección y planificación de recursos y ejecución transparente de trabajos en recursos remotos. El usuario puede establecer la fecha tope (*deadline*) para la se necesitan los resultados y el intermediario Nimrod-G intenta encontrar los recursos computacionales más asequibles disponibles en el Grid y usarlos de forma que se satisfaga esa fecha tope y el coste de la computación sea mínimo.

Específicamente, Nimrod-G soporta fechas tope y restricciones de presupuesto para hacer optimizaciones de planificación, y además gestiona la oferta y demanda de recursos en el Grid usando un conjunto de servicios distribuidos de economía computacional y comercio de recursos (*GRid Architecture for Computational Economy*, GRACE) [81, 82]. Nimrod-G soporta cuatro algoritmos de planificación de fecha tope y presupuesto restringido (*Deadline and Constrained Budget*, DCB) diferentes:

- optimización del coste: usa todos los recursos más asequibles para asegurarse de que la fecha tope se puede cumplir y el coste computacional se minimiza.
- optimización del tiempo: usa todos los recursos posibles para procesar trabajos en paralelo tan pronto como sea posible.
- optimización de coste y tiempo: es similar a la estrategia de optimización del coste, pero si hay múltiples recursos con el mismo coste, aplica la estrategia de optimización del tiempo mientras planifica trabajos en ellos.

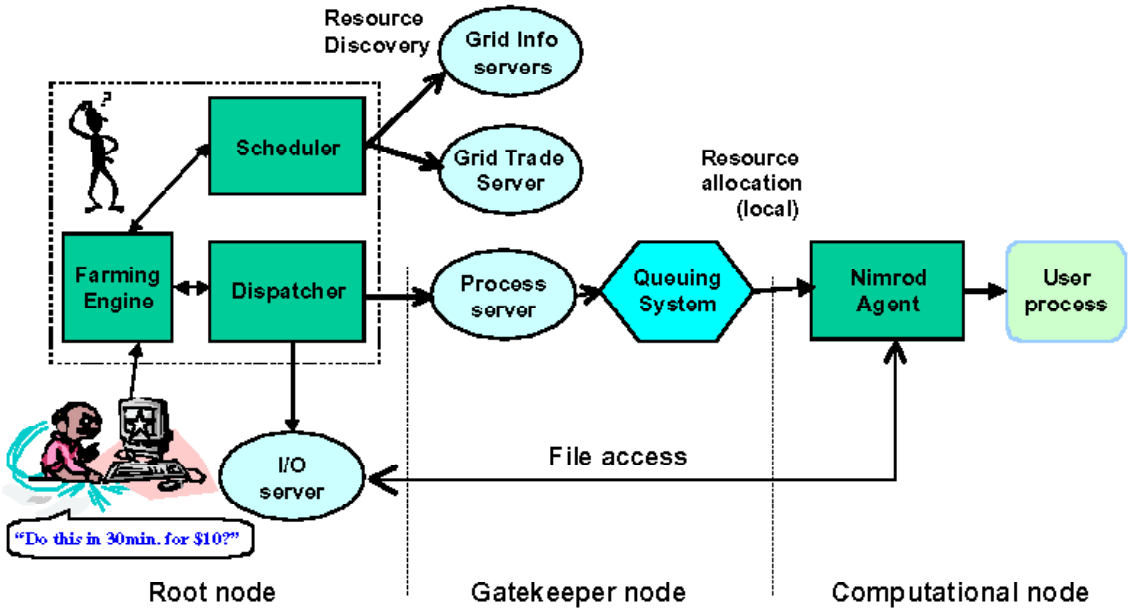

FIGURA 2.4: Arquitectura de Nimrod-G [66].

- optimización conservadora del tiempo: es similar a la estrategia de optimización del tiempo, pero garantiza que cada trabajo no procesado dispone de un presupuesto mínimo para serlo.

Nimrod-G ha sido utilizado para resolver aplicaciones de computación intensiva en datos, como la simulación de la calibración de una cámara de ionización y la modelización de moléculas para diseño de fármacos [83].

2.5.4. Application-Level Scheduler (AppLeS)

Iniciado en 1996, los objetivos del proyecto AppLeS [84, 65] han sido investigar la planificación adaptativa en computación Grid y llevar los resultados de la investigación a aplicaciones reales para validar la eficacia de su aproximación y extraer del Grid rendimiento para el usuario final. Estos objetivos se han conseguido por medio de una aproximación que incorpora información estática y dinámica de recursos, predicciones de rendimiento, información específica del usuario y de la aplicación, y técnicas de planificación para adaptar la ejecución de la aplicación al vuelo. Cada aplicación se integra con un agente de planificación personalizado que monitoriza el rendimiento disponible de los recursos y genera dinámicamente una planificación para la aplicación. Los pasos seguidos por el agente AppLeS se muestran en la figura 2.5.

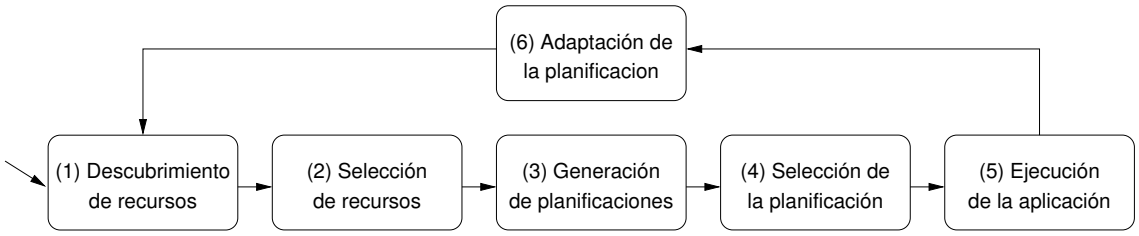

FIGURA 2.5: Pasos en la metodología de AppLeS [65].

El proyecto AppLeS ha desarrollado plantillas (*AppLeS Templates*) que expresan características comunes de varias aplicaciones similares, pero no idénticas. Cada plantilla se ha desarrollado para hospedar una clase de aplicaciones similares estructuralmente. Hasta la fecha, se han desarrollado dos plantillas: APST (*AppLeS Parameter Sweep Template*) [85] para aplicaciones de barrido de parámetros, y AMWAT (*AppLeS Master-Worker Application Template*) [86] para aplicaciones del tipo jefe/trabajador. Adicionalmente, se ha desarrollado AppLeS para superordenadores (*Supercomputing AppLeS*, SA), para planificar trabajos amoldables (esto es, trabajos paralelos que pueden ejecutarse con diferentes particionados) en superordenadores paralelos.

2.5.5. Grid Application Development System (GrADS)

El objetivo del proyecto GrADS [87, 88] es simplificar la computación distribuida heterogénea, de la misma manera que la tecnología *Web* simplificó la compartición de información a través de Internet. Para ello, el proyecto explora los problemas científicos y técnicos que deben resolverse para hacer más fácil a los usuarios el desarrollo, ejecución y optimización de aplicaciones en el Grid. El *software* desarrollado en el proyecto GrADS, GrADSoft [89], está basado en muchos de los principios de AppLeS y generaliza su metodología.

La figura 2.6 presenta la nueva estructura de desarrollo de programas sugerida por el proyecto GrADS. En lo que se refieren como arquitectura GrADSoft, los pasos discretos de creación, compilación, ejecución y análisis "post-mortem" de la aplicación se reemplazan por un proceso continuo de adaptación de las aplicaciones al entorno Grid cambiante y a la instancia específica del problema. Dos conceptos clave son críticos para el funcionamiento de este sistema: primero, una aplicación debe ser encapsulada como un programa de objetos configurables (*configurable object program*), que puede ser rápidamente optimizado para su ejecución en una colección específica de recursos del Grid; y segundo, el sistema descansa sobre contratos de rendimiento (*performance contracts*) que especifican el rendimiento

esperado de los módulos como una función de los recursos disponibles [90].

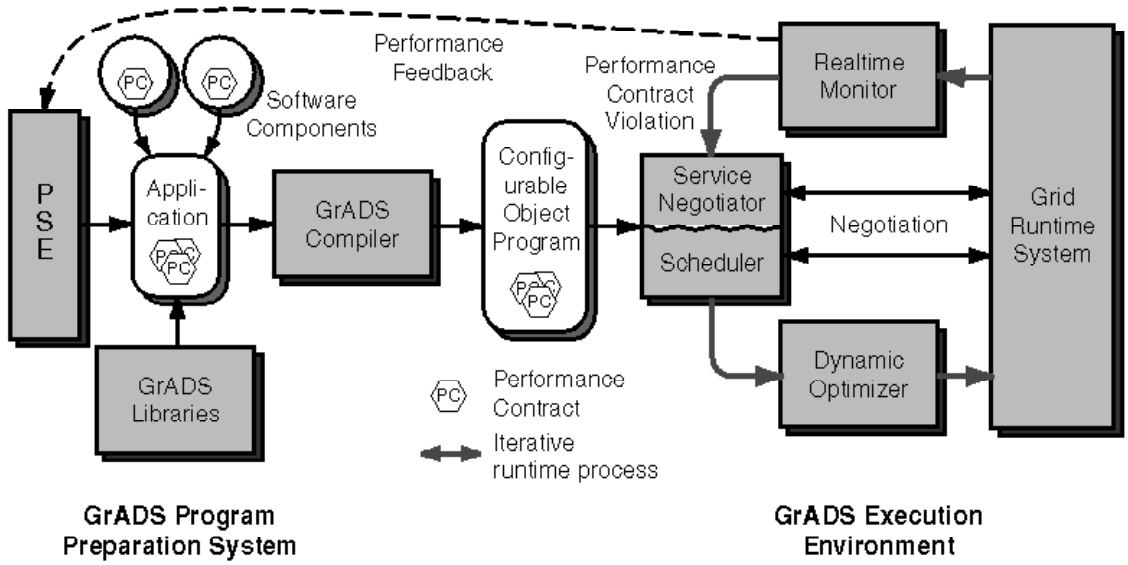

FIGURA 2.6: Arquitectura de preparación y ejecución de programas de GrADS [88].

La parte izquierda de la figura 2.6 muestra las herramientas usadas para la preparación de las aplicaciones (*GrADS Program Preparation System*). Se espera que la mayoría de desarrolladores de aplicaciones usen entornos de resolución de problemas (PSE) de alto nivel para ensamblar aplicaciones Grid a partir de una serie de componentes de un dominio específico.

La parte derecha de la figura 2.6 muestra el entorno de ejecución (*GrADS Execution Environment*). Cuando una aplicación se envía al entorno de ejecución, la infraestructura GrADSoft debe primero determinar qué recursos hay disponibles y asegurar un subconjunto apropiado para la aplicación. Usando anotaciones de los contratos de rendimiento y resultados del análisis del compilador, los negociadores del servicio intermedian en la asignación y planificación de los componentes del módulo sobre los recursos del Grid. Después, la infraestructura invoca el optimizador dinámico para adaptar el programa y obtener así un buen rendimiento con los recursos disponibles. En este paso, se insertan también sensores y actuadores para ayudar al sistema de monitorización del rendimiento a controlar la ejecución de la aplicación.

Durante la ejecución de la aplicación, un monitor de tiempo real rastrea el comportamiento del programa y valida el comportamiento observado frente a las garantías de los contratos de rendimiento. Si se viola un contrato de rendimiento, el monitor responderá interrumpiendo la ejecución a través de un actuador, llevando a varias acciones posibles. El

actuador puede invocar el optimizador dinámico con más información (de la monitorización del rendimiento) para mejorar el comportamiento del programa en el actual contexto de ejecución, o puede negociar un nuevo contexto de ejecución donde el ejecutable existente satisfaga con más probabilidad el viejo contrato, o bien puede hacer las dos cosas. Las predicciones dinámicas del rendimiento de los recursos y de la capacidad del Grid son usadas para reducir la sobrecarga de la renegociación. El objetivo de este sistema de lazo cerrado es que la ejecución de la aplicación proceda de forma fiable, cumpliendo las especificaciones de sus contratos de rendimiento en el entorno Grid cambiante.

La metodología de GrADS se ha utilizado, entre otros, en el proyecto ScaLAPACK (*Scalable Linear Algebra PACKage*) [91] para crear aplicaciones numéricas auto-adaptativas (*Self-Adaptive Numerical Software*, SANS) [92]. También se ha utilizado en el proyecto Cactus [93], que incorpora mecanismos adaptativos de selección de recursos que permiten a las aplicaciones migrar a mejores recursos [68].

2.5.6. GridLab

Dos aspectos importantes de la tecnología Grid, ignorados durante mucho tiempo, forman la base del proyecto GridLab [94, 95], que trata de construir componentes para aplicaciones Grid (al igual que MatLab lo hace para las matemáticas) y bancos de prueba realistas para su desarrollo.

1. **Co-desarrollo de infraestructura y aplicaciones**: Se propone un programa balanceado con co-desarrollo de un rango de aplicaciones Grid basadas en Cactus [93, 96] (un marco de trabajo para aplicaciones Grid) y Triana [97] (una herramienta de flujo de datos usada en investigación de ondas gravitacionales), junto con desarrollo de infraestructura, trabajando en bancos de prueba transatlánticos de superordenadores y *clusters* variados. Esta aproximación práctica asegura que el *software* desarrollado permitirá el uso sencillo y eficiente de los recursos Grid en un entorno real. Se mantendrán y actualizarán los bancos de pruebas mediante el despliegue de nueva infraestructura y tecnologías de aplicaciones de gran escala conforme se vayan desarrollando. Se realizarán prototipos de todos los entregables (*deliverables*) inmediatamente y continuamente se realizarán pruebas de campo por parte de las distintas comunidades de usuarios. Centrarse en marcos de trabajo para aplicaciones específicas, permitirá crear inmediatamente aplicaciones Grid que funcionan para ganar experiencia en el desarrollo de componentes más genéricos durante el proyecto.

2. **Computación Grid dinámica**: Se desarrollarán capacidades para que códigos de simulación y visualización sean conscientes del entorno Grid cambiante, y para que puedan explotar completamente los recursos dinámicos para nuevos e innovadores escenarios de aplicación. Por ejemplo, las propias aplicaciones tendrán la capacidad de migrar de sitio a sitio durante su ejecución, tanto completamente como en parte, para iniciar tareas relacionadas, o para adquirir o liberar recursos adicionales demandados por las disponibilidades cambiantes de los recursos del Grid o por las necesidades de las mismas aplicaciones.

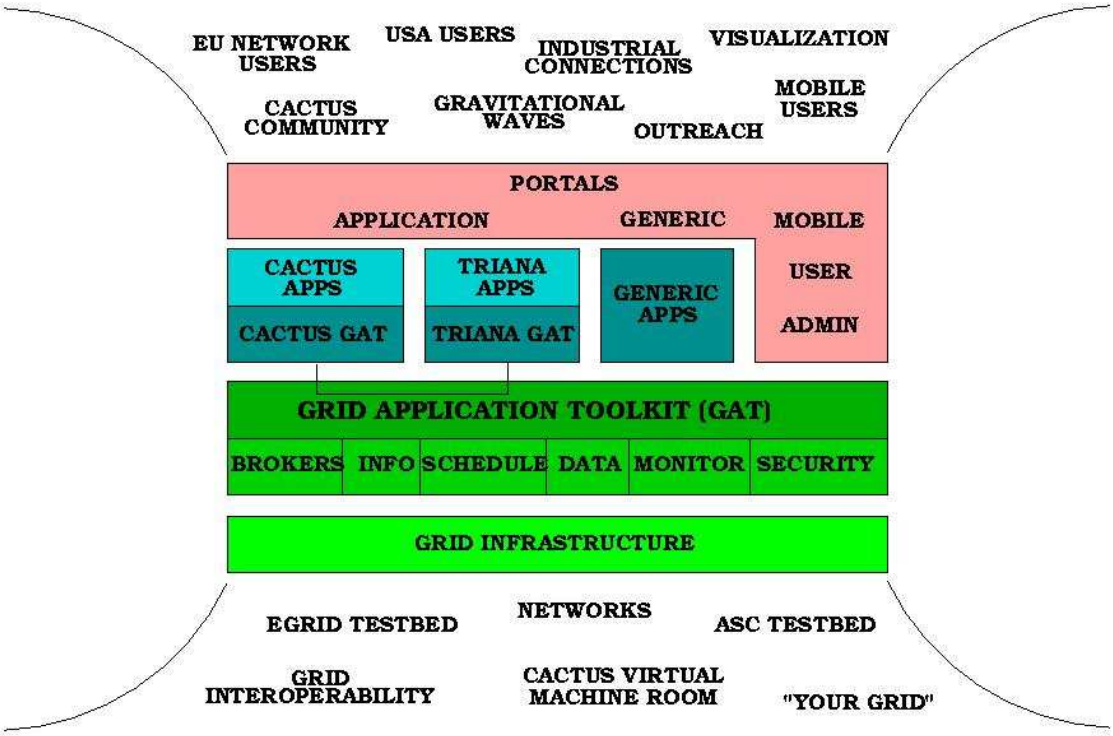

Figura 2.7: Arquitectura de GridLab [94].

Estos elementos se unificarán para el desarrollo de tecnologías innovadoras y prácticas de computación Grid, que serán entonces rápida y sencillamente adoptadas y explotadas por aplicaciones de muchos y variados campos de ciencia e ingeniería, como se muestra en la figura 2.7. Los objetivos clave específicos del proyecto son:

- **Diseñar y desarrollar un conjunto de herramientas para aplicaciones Grid (*Grid Application Toolkit*, GAT)**, para proporcionar funcionalidad básica fácil de usar a través de un conjunto de APIs genéricos cuidadosamente construido tanto

para códigos de simulación como para *software* de Grid. GAT contendrá módulos independientes para tratar diferentes aspectos de la programación en Grid, incluyendo simulación, rendimiento y monitorización del Grid, intermediación y selección de recursos, seguridad, notificación, colaboración, gestión de datos, visualización remota e interacción remota con las aplicaciones.

- **Mejorar aplicaciones reales para el Grid**, implementando nuevos escenarios de simulación dinámicos usando GAT. Tanto Cactus como Triana se extenderán para integrar y explotar los elementos de GAT, haciendo la computación en Grid fácilmente explotable por un amplio rango de aplicaciones. Sus aplicaciones guiadas por simulación e intensivas en cálculo son totalmente diferentes de las aplicaciones altamente guiadas por datos de otros proyectos Grid (por ejemplo, EDG, GriPhyN y EuroGrid).

- **Desarrollar y probar la infraestructura y aplicaciones Grid en bancos de prueba reales**, construidos enlazando colecciones heterogéneas de super-computadores y otros recursos a lo largo de Europa y Estados Unidos, usando y extendiendo los bancos de prueba existentes. Se asegurará la interoperatibilidad entre los diferentes bancos de pruebas, usando también bancos de pruebas en producción en Estados Unidos, guiando la conectividad internacional de redes de alta velocidad. Las pruebas será llevadas a cabo por el proyecto y por varias comunidades de usuarios grandes y estrechamente relacionadas, incluyendo una red de astrofísica europea y varias colaboraciones multidisciplinares financiadas por Estado Unidos.

El proyecto GridLab está dividido en 14 paquetes de trabajo (WP) junto con un comité técnico, responsable de la arquitectura general del proyecto y de la coordinación para asegurar la interoperatibilidad de los WPs individuales:

1. *Grid Application Toolkit* (GAT).
2. *Cactus Grid Application Toolkit* (CGAT).
3. *Triana Grid Application Toolkit* (TGAT).
4. Portales al Grid.
5. Gestión del banco de pruebas.
6. Seguridad.

7. Componentes adaptativos.
8. Manipulación de datos y visualización.
9. Gestión de recursos.
10. Servicios de información.
11. Monitorización.
12. Acceso para usuarios móviles.
13. Explotación y diseminación.
14. Gestión del proyecto.

2.6. Conclusiones

En este capítulo se han descrito los pasos a seguir en la planificación y ejecución de trabajos en Grids. Además, se han introducido las dos técnicas más importantes utilizadas para adaptar la planificación y ejecución de los trabajos. En el siguiente capítulo se describirá la implementación de estas técnicas en un prototipo experimental.

El objetivo de la investigación presentada en esta Tesis es similar al de otros proyectos sobre Grid: simplificar la computación distribuida y heterogénea. Sin embargo, las directrices que han guiado el diseño de nuestro prototipo han sido muy diferentes:

1. Amplio rango de aplicación: no debe estar ligado a una clase específica de aplicación generada por un entorno de programación dado.
2. Fácil despliegue: no debe requerir la instalación de servicios Grid específicos en los recursos del banco de pruebas.
3. Reutilización de las aplicaciones existentes: no debe requerir necesariamente cambios en el código de las aplicaciones.
4. Extensión y adaptación de la funcionalidad: debe poder ser extendido o modificado para comunicarse con los servicios Grid existentes en un banco de pruebas dado.

TABLA 2.1: Tabla comparativa de los proyectos relacionados más importantes.

Proyecto	**Planificación adaptativa**	**Ejecución adaptativa**	**Requisitos**			
			1	**2**	**3**	**4**
Condor/G	No	No	Sí	Sí	Sí	No
EDG	Sí	No	Sí	No	Sí	No
Nimrod-G	Sí	No	Sí	Sí	Sí	No
AppLeS	Sí	No	Sí	Sí	Sí†	No
GrADS	Sí	Sí	No	No	No	No
GridLab⋆	Sí	Sí	?	?	?	?
Grid*W*ay	Sí	Sí	Sí	Sí	Sí	Sí

† Sólo si se usan las plantillas.
⋆ No se conoce ningún prototipo completo.

La tabla 2.1 muestra una matriz en la que los distintos proyectos mencionados en la sección anterior se hacen corresponder con las funcionalidades de planificación y ejecución adaptativa y los requisitos que han guiado el diseño de la herramienta Grid*W*ay.

Condor/G no proporciona ni planificación ni ejecución adaptativa. EDG añade planificación adaptativa a Condor/G, pero sigue un enfoque centralizado con requisitos muy restrictivos para su despliegue. Nimrod-G y AppLeS sólo proporcionan planificación adaptativa, siendo estos proyectos los que más han contribuido al desarrollo de esta técnica. GrADS generaliza la metodología de AppLeS, proporcionando ejecución adaptativa, pero exige la utilización de un entorno de preparación de programas específico para desarrollar sus aplicaciones. Por último, GridLab también proporciona planificación y ejecución adaptativa, pero aún no se conocía ningún prototipo completo implementado cuando se terminó de escribir esta Tesis.

Como se ve en la tabla 2.1, las herramientas *software* generadas en los proyectos de investigación en desarrollo hoy día [54] o bien no soportan migración y auto-adaptación de trabajos o bien están basadas en complejas arquitecturas de servicios que, desde nuestro punto de vista, son poco realistas y no son fieles a la filosofía orientada a usuario del Grid.

Es importante remarcar que la herramienta experimental Grid*W*ay no requiere la instalación de nuevo *software* en los recursos del Grid. La herramienta Grid*W*ay es completamente funcional en cualquier banco de pruebas de Grid basado en Globus. Creemos que ésta es una ventaja importante debido a posibles problemas socio-políticos, ya que la cooperación entre distintos centros de investigación, administradores y usuarios es siempre complicada.

3. Prototipo para Planificación y Ejecución Adaptativa en Grids

En este capítulo se describen las principales características y el funcionamiento básico de Grid*W*ay [98, 76], un prototipo experimental que proporciona planificación y ejecución adaptativa en entornos Grid. Además, se demuestran las funcionalidades de adaptación del prototipo con la ejecución de un código de dinámica de fluidos computacional.

3.1. Introducción

A pesar del esfuerzo realizado durante los últimos años, el desarrollo y ejecución de aplicaciones en el Grid continúa requiriendo un gran nivel de experiencia debido a su complejidad. El desarrollador o usuario es responsable de programar o realizar manualmente todas las etapas de planificación de trabajos para obtener alguna funcionalidad: descubrimiento y selección de recursos, y preparación, envío, monitorización, migración y finalización de trabajos [58]. Además, debe considerar todas las circunstancias que pueden darse en un entorno tan heterogéneo y dinámico como el Grid.

Como parte de esta investigación, hemos desarrollado una nueva herramienta experimental basada en Globus que permite una ejecución de trabajos más sencilla y eficiente en entornos Grid dinámicos, a la manera "enviar y olvidar" (*submit & forget*). La adaptación a las condiciones cambiantes se consigue implementando una replanificación automática de las aplicaciones cuando se producen degradaciones del rendimiento, descubrimientos de "mejores" recursos, cambios de requisitos, decisiones del propietario del recurso, fallos, etc. Esta replanificación puede conducir a la migración de la aplicación a otro recurso más apropiado.

El aspecto fundamental de la ejecución adaptativa es el reconocimiento de las condiciones cambiantes, tanto de los recursos del Grid (ver sección 1.4.1) como de las demandas de la aplicación (ver sección 1.4.2). Para conseguir tal funcionalidad, proponemos un modelo

de aplicación consciente del Grid (*Grid-aware application model*), que incluye funcionalidad auto-adaptativa, y un agente de envío, que proporciona los mecanismos necesarios para adaptar la ejecución de la aplicación. La aplicación debe estar equipada con la funcionalidad necesaria para soportar las circunstancias de replanificación iniciadas por la aplicación, mientras que el agente está continuamente vigilando la ocurrencia de circunstancias de replanificación iniciadas por el Grid o por la aplicación.

3.2. Modelo de Aplicación

La figura 3.1 muestra el modelo estándar de una aplicación que procesa unos datos almacenados en unos ficheros de entrada y genera unos resultados que son almacenados en unos ficheros de salida. Además, es común que la aplicación genere periódicamente ficheros de reinicio para proporcionar tolerancia a fallos.

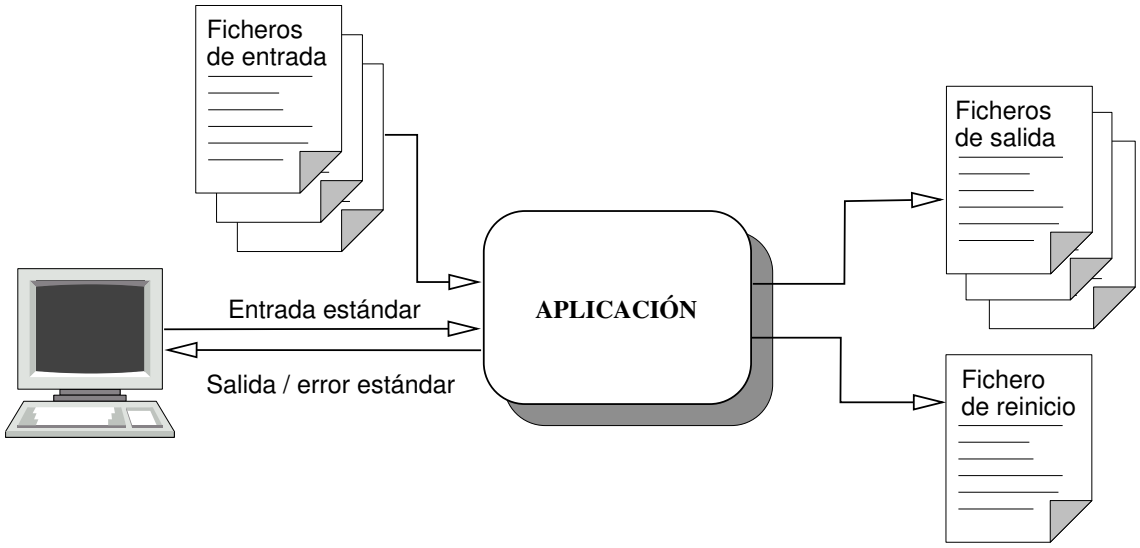

FIGURA 3.1: Modelo de aplicación estándar.

La figura 3.2 muestra un modelo de aplicación que extiende el modelo de aplicación estándar para proporcionar mayores funcionalidades en un entorno Grid.

Para adaptar la ejecución de un trabajo a sus demandas dinámicas, la aplicación debe especificar sus necesidades mediante una expresión de requisitos (`requirement expression`). El usuario podría definir un conjunto inicial de requisitos, que podrían ser modificados dinámicamente cuando más, o incluso menos, recursos fueran necesarios. En Grid*Way*, la expresión de requisitos se especifica como un filtro de búsqueda LDAP, especificados en el

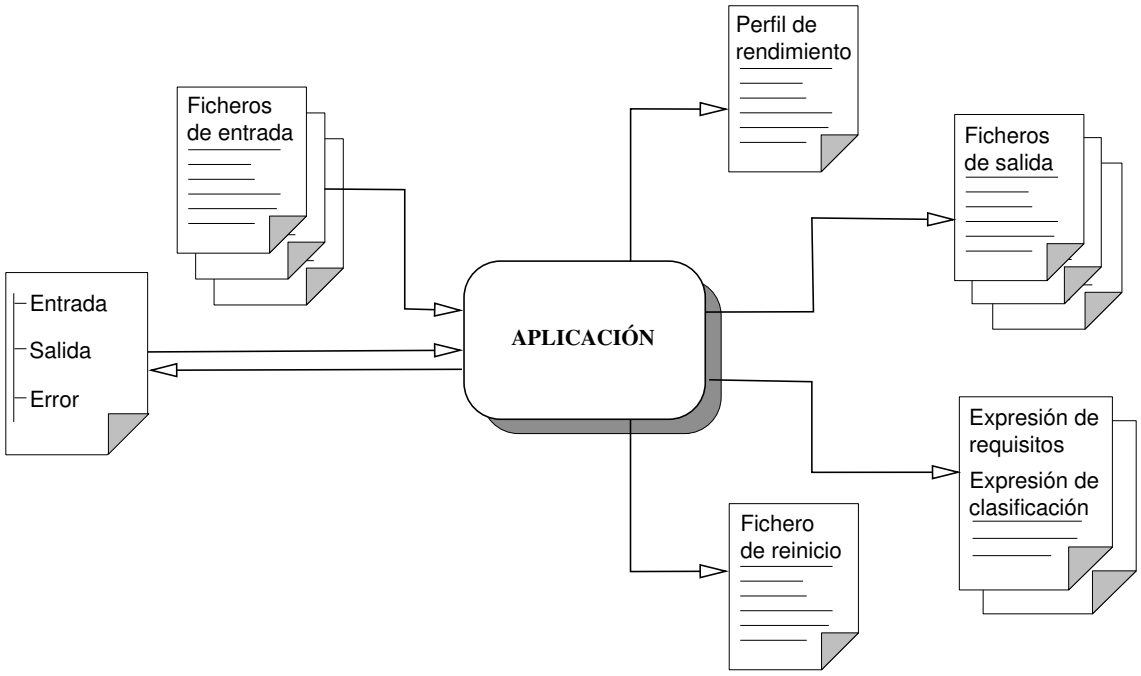

FIGURA 3.2: Modelo de aplicación consciente del Grid.

RFC 2254 [99].

Además, para priorizar los recursos que cumplen los requisitos de acuerdo con sus necesidades, la aplicación debe especificar sus preferencias por medio de una expresión de clasificación (`ranking expression`) que asigna un valor numérico –o rango (*rank*)– a cada recurso. El rango asignado a cada recurso indica la idoneidad de ese recurso para ejecutar un trabajo determinado. Por ejemplo, una aplicación intensiva en cálculo asignaría un rango mayor a aquellos equipos con procesadores más rápidos y más descargados, mientras que una aplicación intensiva en datos podría beneficiar a aquellos equipos más cercanos a los datos que se necesitan. En Grid*W*ay, la expresión de clasificación se especifica como un guión que recibe la información de monitorización de los recursos como variables.

Debido a la alta tasa de fallos del Grid y a la replanificación dinámica, es muy conveniente que la aplicación genere ficheros de reinicio (`restart files`) de forma que se pueda reiniciar la ejecución a partir de un punto dado. Si no se proporcionan estos ficheros, el trabajo tendría que reiniciarse desde el principio. Debe implementarse un *checkpointing* a nivel de usuario gestionado por el programador, ya que actualmente el *checkpointing* a nivel de sistema no es posible entre recursos heterogéneos. Es conveniente generar ficheros de reinicio que sean independientes de la arquitectura, usando formatos ASCII o HDF (*Hierarchical Data Format*) [100], para no limitar el conjunto potencial de recursos candidatos

para la migración.

Para detectar degradaciones del rendimiento, es conveniente que la aplicación almacene un perfil de rendimiento (`performance profile`) con información de monitorización en términos de métricas intrínsecas a la aplicación. Por ejemplo, para un código iterativo podría escribirse el tiempo consumido en cada iteración del algoritmo.

3.3. Arquitectura de Grid *W*ay

El núcleo de la herramienta Grid *W*ay es un agente de envío (*submission agent*) que realiza automáticamente los pasos necesarios para planificar y ejecutar un trabajo en un entorno Grid (ver sección 2.2) y vigila la ejecución correcta y eficiente del mismo.

La arquitectura del *submission agent* se muestra en la figura 3.3. El usuario interactúa con la herramienta a través de un gestor de peticiones (*request manager*), el cual reenvía sus peticiones (*submit*, *kill*, *stop*, *resume*...) al gestor de despacho (*dispatch manager*). El *dispatch manager* se despierta periódicamente en cada intervalo de planificación e intenta enviar los trabajos pendientes a recursos del Grid y es también responsable de decidir si la migración de los trabajos replanficados es factible y merece la pena. Una vez que un trabajo es asignado a un recurso, se arrancan un gestor de envío (*submission manager*) y un monitor del rendimiento (*performance monitor*) para vigilar su ejecución correcta y eficiente.

La flexibilidad de la herramienta está garantizada mediante un interfaz de programación (*Application Program Interface*, API) bien definido para cada componente del agente de envío. En el caso del cliente, se ha implementado el estándar DRMAA (ver sección 6.4) por encima del interfaz propio de Grid *W*ay. Además, la herramienta ha sido diseñada para ser modular y, por tanto, permitir la extensión y mejora de sus capacidades.

Los siguientes módulos (*plug-ins*) se pueden establecer para cada trabajo individualmente y proporcionan la capacidad de auto-adaptación de las aplicaciones:

- selector de recursos (*resource selector*): Actúa como un intermediario de recursos (*resource broker*) para construir una lista priorizada de recursos candidatos, siguiendo las requisitos y preferencias proporcionados inicialmente por el usuario o dinámicamente por la aplicación.

- evaluador del rendimiento (*performance evaluator*): Usado para evaluar periódicamente el rendimiento de la aplicación.

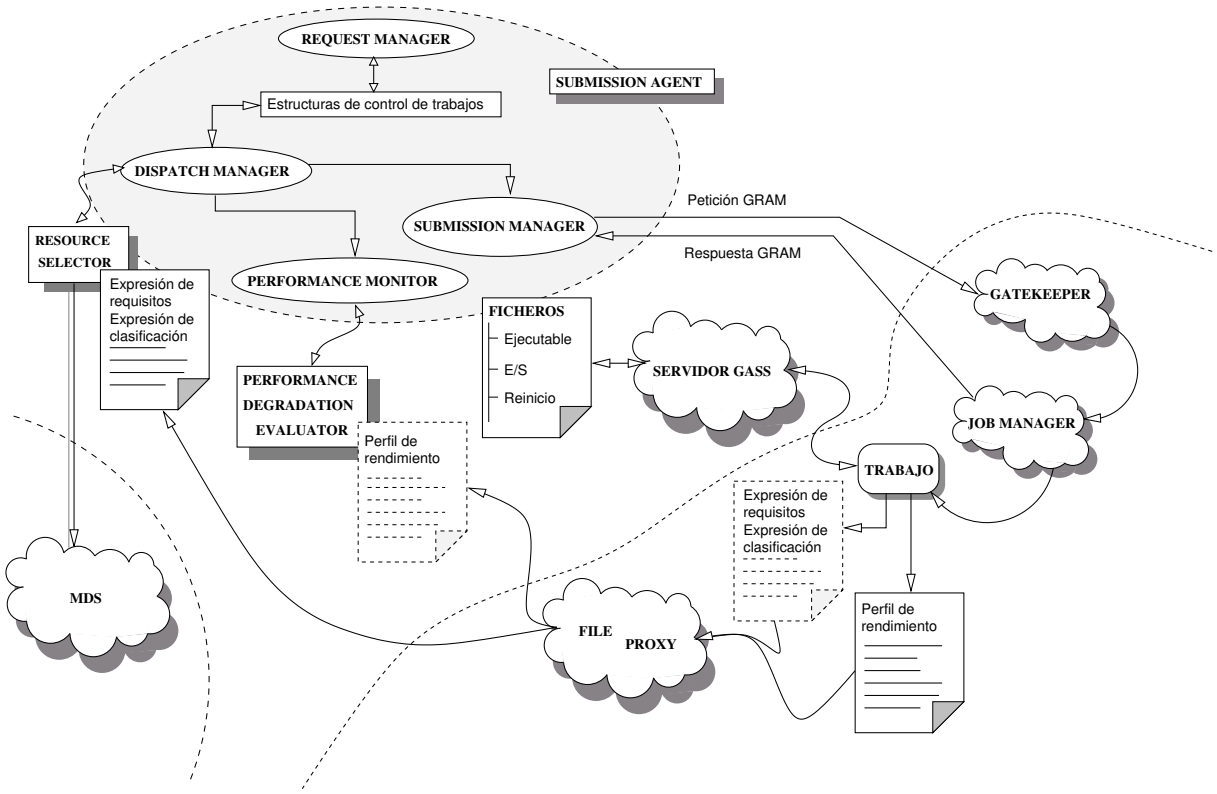

FIGURA 3.3: Arquitectura de la herramienta Grid*Way*.

La ejecución de los trabajos se realiza en tres fases mediante los siguientes módulos:

- prólogo (*prologue*): Prepara el sistema remoto y realiza la transferencia de los ficheros de entrada.
- envoltorio (*wrapper*): Ejecuta el trabajo y devuelve su código de salida.
- epílogo (*epilogue*): Realiza la transferencia de los ficheros de salida y limpia el sistema remoto.

3.4. Planificación de Trabajos

Las tareas de planificación se realizan en el estado "PENDING" del *dispatch manager* (ver figura 3.4). El *dispatch manager* planifica los trabajos haciendo uso del módulo *resource selector*. La planificación es adaptativa (*adaptive scheduling*, ver sección 2.3.1), es decir, se realiza en función de los recursos disponibles y de los trabajos pendientes, en ejecución y ya terminados en cada momento.

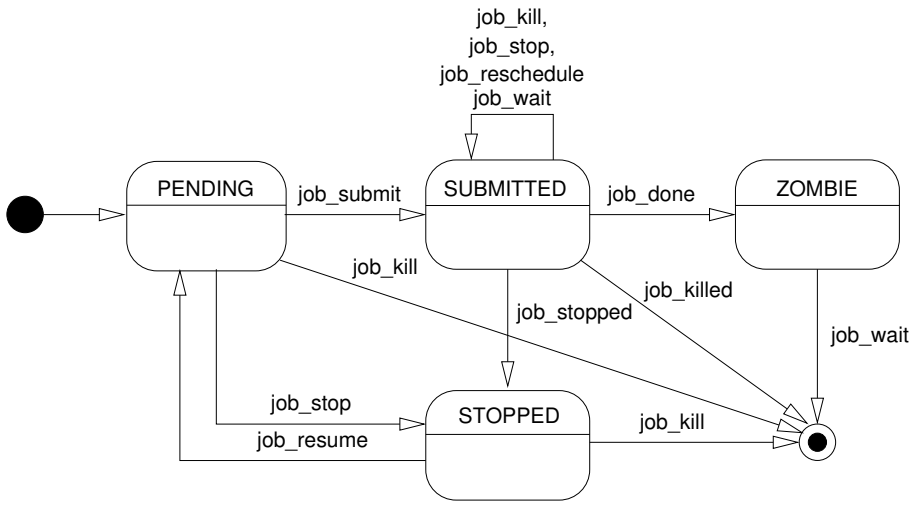

FIGURA 3.4: Diagrama de estados del *dispatch manager*.

Se planifican trabajos individuales y en array. En el caso de trabajos en array, se ejecuta una vez el *resource selector* y se envían tareas a los equipos con nodos disponibles, haciendo lo que se conoce como inundación de tareas (*task flooding*, ver sección 2.4).

El usuario tiene la posibilidad de parar (*stop*) y continuar (*resume*) trabajos, así como de solicitar la replanificación (*reschedule*) o cancelación (*kill*) de los mismos.

3.4.1. Descubrimiento y Selección de Recursos

Quizás la parte más importante de la planificación en Grids es la selección de recursos, que a su vez depende completamente de la información obtenida del Grid. Debido a la naturaleza heterogénea y dinámica del Grid, el usuario final debe establecer los requisitos que deben cumplir los recursos candidatos en el proceso de descubrimiento y los criterios para clasificar esos recursos en el proceso de selección.

Los atributos necesarios para el descubrimiento y la selección de los recursos son recogidos de los servicios de información disponibles en el Grid, típicamente, de MDS. Normalmente, el descubrimiento se basa sólo en atributos –casi– estáticos (sistema operativo, arquitectura, tamaño de memoria...) tomados del GIIS, mientras que la selección se basa en atributos más dinámicos (espacio en disco, carga de trabajo, memoria libre...) tomados del GRIS.

El proceso de descubrimiento y selección de la herramienta Grid*W*ay se muestra en la figura 3.5. Inicialmente, se descubren los recursos de ejecución disponibles accediendo al servidor GIIS y, aquellos recursos que no cumplen los requisitos proporcionados por

medio de una expresión de requisitos (`requirement expression`), son filtrados. En este paso, se realiza una prueba de autenticación (mediante una prueba de conexión, *ping*, al *gatekeeper* de GRAM) para garantizar que el usuario tiene acceso al recurso remoto. Entonces, se recogen los atributos dinámicos de cada recurso candidato de su servidor GRIS local. Esta información es usada por la expresión de clasificación (`ranking expression`) para asignar un rango a cada recurso candidato. La expresión de clasificación puede incluir modelos de rendimiento, como se verá en el capítulo 4. Finalmente, la lista resultante de recursos candidatos priorizados es usada para despachar los trabajos, realizándose la selección de los recursos en función de no sólo el rango asignado los recursos candidatos, sino también del número de nodos que tengan disponibles, el rango del recurso actual (en caso de replanificación), el estado del resto de trabajos planificados, la causa de replanificación (si la hay), etc.

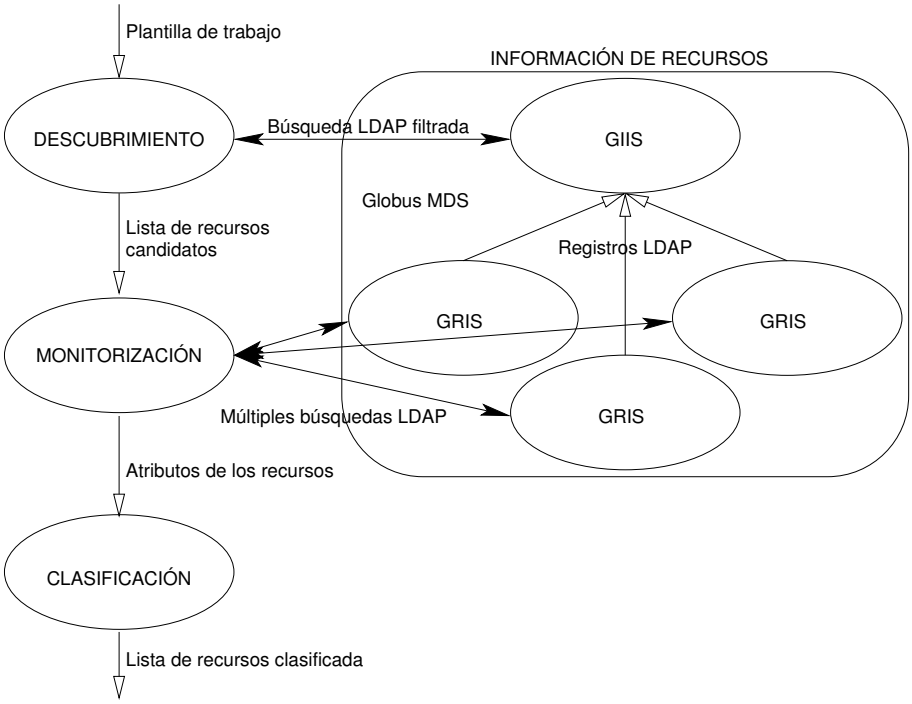

FIGURA 3.5: Descubrimiento y selección de recursos en la herramienta Grid*W*ay.

Para reducir la sobrecarga en la recuperación de información, la información de los servidores GIIS y GRIS es almacenada localmente en una *cache* y actualizada independientemente para determinar separadamente cada cuánto se examina el banco de pruebas para encontrar nuevos recursos y la frecuencia de monitorización de los recursos. En los experimentos siguientes, hemos establecido el tiempo máximo de almacenamiento en *cache*

a 5 minutos para el servidor GIIS y a 30 segundos para el GRIS.

3.5. Ejecución de Trabajos

Las acciones de ejecución de trabajos se realizan en el estado "SUBMITTED" del *dispatch manager* (ver figura 3.4). La ejecución se realiza en tres fases –preparación, envío y finalización– que se describirán en las siguientes subsecciones. Esta estrategia presenta las siguientes ventajas frente a la ejecución en un solo paso:

- Es válido para sistemas cerrados, donde los nodos de cálculo no tienen conexión directa a Internet.
- Permite planificar separadamente las fases de transferencia y de cálculo.
- Permite un mejor ajuste de los parámetros de definición de trabajos (lenguaje RSL), como `maxcputime`.
- Proporciona un método sencillo y eficiente de implementar la migración de trabajos.

El envío de cada fase se realiza usando un gestor de envío específico para Globus (*Globus submission manager*), que utiliza GRAM. Este gestor de envío específico para Globus se podría cambiar por cualquier otro que utilizara SSH, Legion, Condor/G, etc.

Para proporcionar ejecución adaptativa (ver sección 2.3.2), los trabajos, una vez enviados, son replanificados dinámicamente cuando se dan determinadas causas. Como ya se discutió en la sección 1.4.3, los trabajos enviados se replanifican bajo las siguientes circunstancias:

- Debido al dinamismo del Grid:
  - Periódicamente, para descubrir "mejores" recursos (migración oportunista, que se verá en el capítulo 4).
  - Cuando el recurso remoto (*software*, *hardware* o conexión de red) falla.
  - Cuando el trabajo es cancelado o suspendido.
- Debido a la auto-adaptación de las aplicaciones:
  - Cuando se detecta una degradación del rendimiento.

- Cuando la demanda de recursos de la aplicación cambia (auto-migración).

A estas causas hay que sumar la posibilidad de que el usuario solicite una replanificación cuando lo crea conveniente.

Durante el proceso de replanificación, el *dispatch manager* invoca al *resource selector* y, dependiendo de los recursos disponibles, la razón de replanificación y otros factores, puede decidir iniciar la migración del trabajo a un nuevo recurso seleccionado. Por ejemplo, si la replanificación fue para descubrir recursos mejores, la migración sólo se iniciará si durante la selección de recursos se descubrió un recurso mejor que el actual, es decir, con mayor rango (como se discutió en la sección 3.4.1). Sin embargo, si la replanificación fue debida a un fallo, el trabajo migrará siempre que haya un recurso disponible [69]. El proceso de migración se verá en la sección 3.5.4.

3.5.1. Preparación

Las acciones de preparación se realizan en el estado "Prologuing" del *submission manager* (ver figura 3.6), y se inician con el envío del módulo *prologue* al recurso remoto. El módulo *prologue* se encarga de preparar el sistema remoto, creando un directorio específico para el trabajo, y de transferir al mismo los ficheros de entrada (*file stage-in*), que pueden ser locales o remotos (especificados como una URL GASS o GridFTP).

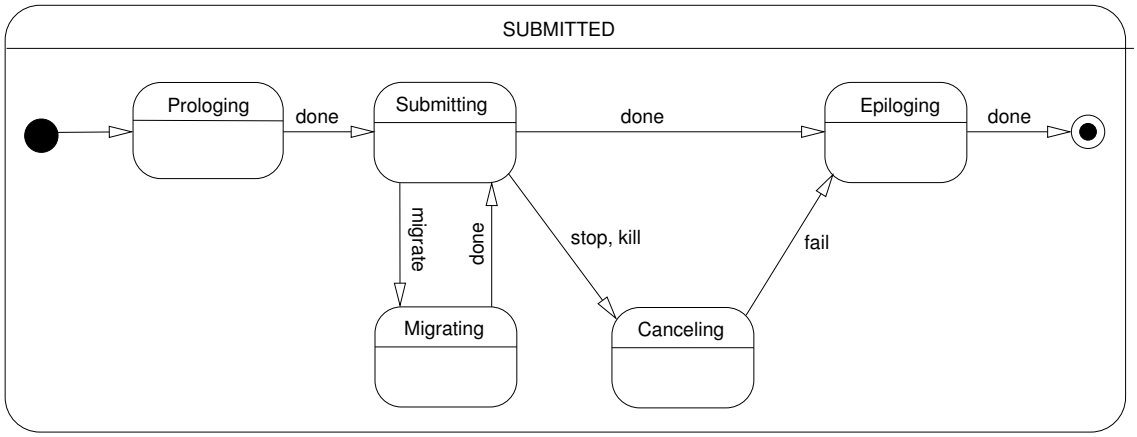

FIGURA 3.6: Diagrama de estados del *submission manager*.

Algunos ficheros de entrada se pueden definir como ficheros compartidos. En ese caso, los ficheros son almacenados en la *cache* de GASS, de forma que pueden ser reutilizados por todos los trabajos que se envíen al mismo recurso. También pueden transferirse ficheros

comprimidos, que serán descomprimidos en el recurso remoto. El uso de ficheros compartidos y comprimidos reduce enormemente los tiempos de transferencia, especialmente en aplicaciones de alta productividad, como se vio en la sección 2.4.

También sería posible desarrollar un módulo *prologue* que implementara un mecanismo de selección y diseminación de réplicas de entrada usando los servicios proporcionados por el catálogo de réplicas de Globus [101].

3.5.2. Envío y Monitorización

Las acciones de envío y monitorización se realizan en el estado "Submitting" del *submission manager* (ver figura 3.6), y se inician con el envío del módulo *wrapper* al recurso remoto. El módulo *wrapper* se encarga de ejecutar la aplicación y de obtener su código de salida.

Para monitorizar la ejecución correcta del trabajo, el *submission manager* utilizan los siguientes mecanismos:

- El *job manager* de GRAM notifica fallos en el envío como *callbacks*. Este tipo de fallos incluye fallos de conexión, de autenticación, de autorización, de interpretación del RSL, de expiración de credenciales, etc.

- El *job manager* de GRAM se comprueba periódicamente en cada intervalo de consulta (*polling interval*). Si el *job manager* no responde, se comprueba el *gatekeeper*. Si el *gatekeeper* responde, se inicia un nuevo *job manager* para que siga vigilando el trabajo. Si el *gatekeeper* no responde es porque ha ocurrido un fallo en el recurso o en su enlace de red. Esta es la aproximación seguida en Condor/G [77].

- El código de salida capturado se puede usar para determinar si el trabajo fue ejecutado con éxito o no. Si no se pudo establecer el código de salida correctamente es porque el trabajo falló o fue cancelado prematuramente.

Cuando se detecta un fallo no recuperable, el *submission manager* reintenta la ejecución del *prologue*, *wrapper* o *epilogue* un número de veces especificado por el usuario y, cuando no quedan más reintentos, realiza la acción elegida por el usuario entre dos posibilidades: parar el trabajo para continuarlo después manualmente, o replanificarlo automáticamente.

Para monitorizar la ejecución eficiente del trabajo, el *performance monitor* solicita replanificaciones del trabajo:

- periódicamente, para detectar "mejores" recursos.
- cuando se detecta una degradación de rendimiento.
- cuando se supera el tiempo máximo de suspensión.

Para detectar degradaciones del rendimiento, el *performance monitor* utiliza el módulo *performance evaluator*, que procesa el perfil de rendimiento, generado por la aplicación durante su ejecución, y detecta degradaciones del rendimiento en términos de métricas intrínsecas a la aplicación. Por ejemplo, para un código iterativo puede establecerse un tiempo máximo por iteración. Además, el *submission manager* lleva la cuenta del tiempo que el trabajo pasa suspendido o pendiente de ejecutarse, de modo que el *performance monitor* solicita una replanificación del trabajo cuando este tiempo sobrepasa un límite máximo.

La transferencia de ficheros generados o actualizados dinámicamente por la aplicación en tiempo de ejecución (ficheros dinámicos, como el perfil de rendimiento o el fichero con la expresión de requisitos) no es posible siguiendo un modelo de servidor inverso, donde el servidor de ficheros (GASS o GridFTP) se arranca en la máquina cliente. Los sistemas cerrados impiden a los trabajos ejecutándose en los nodos de cálculo "privados" actualizar ficheros en la máquina cliente "pública". Este problema se ha resuelto usando un intermediario de ficheros (*file proxy*) en el nodo frontal (*front-end*) del sistema remoto. De esta forma, los trabajos en ejecución actualizan sus ficheros dinámicos localmente dentro del *cluster*, por ejemplo por medio de NFS, que son accesibles a la máquina cliente a través del *file proxy* remoto (ver figura 3.7).

3.5.3. Finalización

Las acciones de finalización se realizan en el estado "Epiloguing" del *submission manager* (ver figura 3.6), y se inician con el envío del módulo *epilogue* al recurso remoto. El módulo *epilogue* se encarga de transferir al cliente los ficheros de salida (*file stage-out*) y de limpiar el sistema remoto, es decir, eliminar los ficheros almacenados en el directorio remoto y eliminar de la *cache* del GASS los ficheros compartidos.

También sería posible desarrollar un módulo *epilogue* que implementara un mecanismo de diseminación de réplicas de salida usando los servicios proporcionados por el catálogo de réplicas de Globus [101].

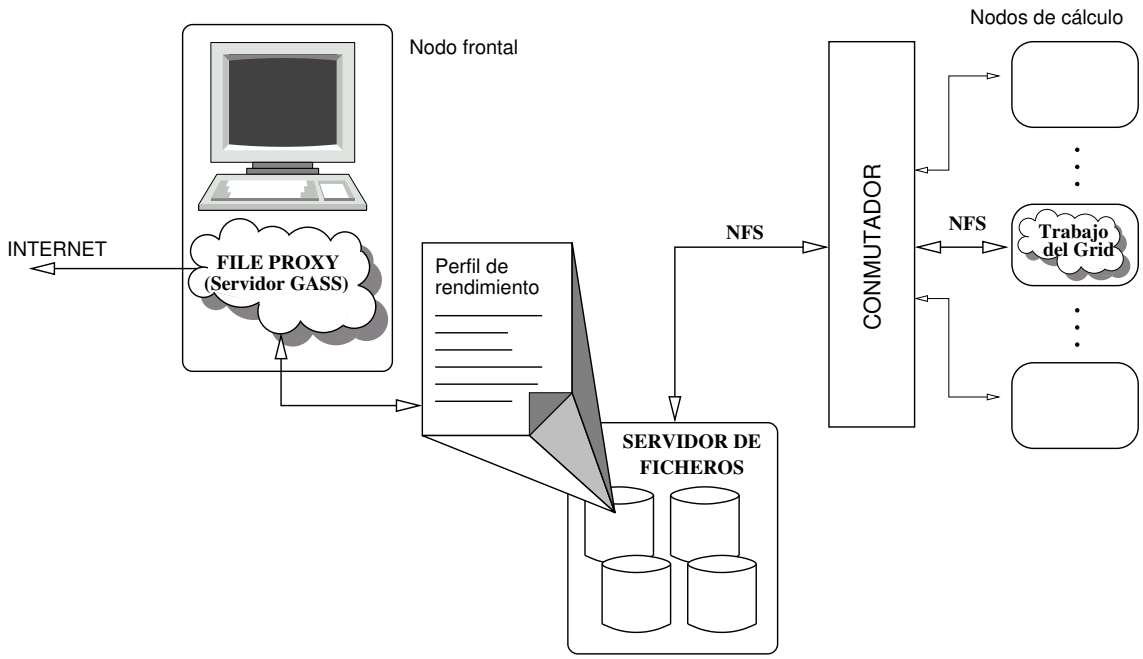

FIGURA 3.7: Acceso a ficheros dinámicos en sistemas cerrados por medio de un *file proxy*.

3.5.4. Migración de Trabajos

Las acciones de migración de trabajos se realizan en el estado "Migrating" del *submission manager* (ver figura 3.6). Como se ve en la figura 3.8, la migración consiste en la cancelación del trabajo (si es que todavía sigue ejecutándose), la preparación del nuevo sistema seleccionado (transfiriendo los ficheros de reinicio), la limpieza del sistema antiguo y el envío del trabajo al nuevo sistema.

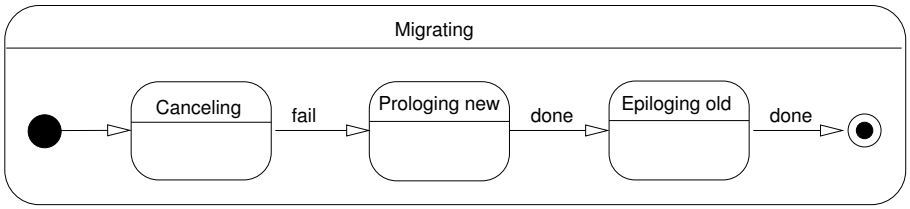

FIGURA 3.8: Diagrama de estados del *submission manager* durante una migración.

3.6. Experimentos sobre Ejecución Adaptativa

En esta sección se demuestran las posibilidades de migración del prototipo Grid *W*ay [69] con la ejecución de un código de dinámica de fluidos computacional CFD. La aplicación

resuelve las ecuaciones incompresibles de Navier-Stokes en 3D para simular la capa límite sobre una placa delgada. El núcleo numérico consiste en un algoritmo iterativo multimalla robusto, caracterizado por un perfil de ejecución intensivo en cálculo [102, 103].

Los ficheros del experimento consisten en el ejecutable (0.5MB) y la malla computacional (0.5MB), proporcionados para todas las arquitecturas de los recursos del banco pruebas, y algunos ficheros de parámetros (1K) que describen la simulación numérica a realizar. El nombre final de los ficheros del ejecutable y la malla computacional se obtiene resolviendo la variable `GW_ARCH` en tiempo de ejecución para el recurso seleccionado. Además, la aplicación genera ficheros de reinicio (*checkpoint*) (0.5MB) en cada iteración del método multimalla.

La aplicación no impone, inicialmente, ningún requisito a los recursos, sin embargo, dado que la aplicación es una simulación intensiva en cálculo, la expresión de clasificación utilizada beneficia a aquellos recursos con mayor rendimiento de procesador y menor carga de trabajo. La expresión es la siguiente:

$$rango = \begin{cases} FLOPS & \text{si} \quad CPU_{15} \geq 1; \\ FLOPS \cdot CPU_{15} & \text{si} \quad CPU_{15} < 1. \end{cases} \tag{3.1}$$

donde $FLOPS$ es el rendimiento pico alcanzable por el procesador del recurso y CPU_{15} es la carga promedio en los últimos 15 minutos. Cabe destacar que, en el caso de *clusters* heterogéneos, $FLOPS$ es el rendimiento promedio de todos los nodos de cálculo. Sin embargo, en la literatura se proponen otras alternativas. Por ejemplo, en el proyecto NorduGrid [104] se sigue una aproximación más conservadora, igualando $FLOPS$ al rendimiento del nodo más lento.

Los escenarios descritos en las siguientes secciones fueron creados artificialmente, sin embargo, representan situaciones reales que pueden ocurrir en un entorno Grid. Los experimentos se han realizado en el banco de pruebas TRGP, que se describe en la sección A.1 de los apéndices.

3.6.1. Descubrimiento de un Recurso Mejor

En este caso, el intervalo de descubrimiento se ha establecido deliberadamente a un valor bajo (60 segundos) para reaccionar rápidamente ante apariciones de nuevos recursos. El perfil de ejecución de la aplicación se presenta en la figura 3.9. Inicialmente, sólo los recursos de ICASE están disponibles para envío de trabajos, ya que sciclone había sido

desconectado por razones de mantenimiento. El *resource selector* elige urchin para ejecutar el trabajo y los ficheros son transferidos (preparación y envío en el periodo del segundo 0 al 34). El trabajo empieza a ejecutarse (segundo 34). El periodo de descubrimiento expira (segundo 120) y el *resource selector* encuentra que sciclone presenta mayor rango que el recurso original (periodo del segundo 120 al 142). El proceso de migración se inicia entonces (cancelación, preparación, finalización y envío en el periodo del segundo 142 al 236). Finalmente el trabajo completa su ejecución en sciclone.

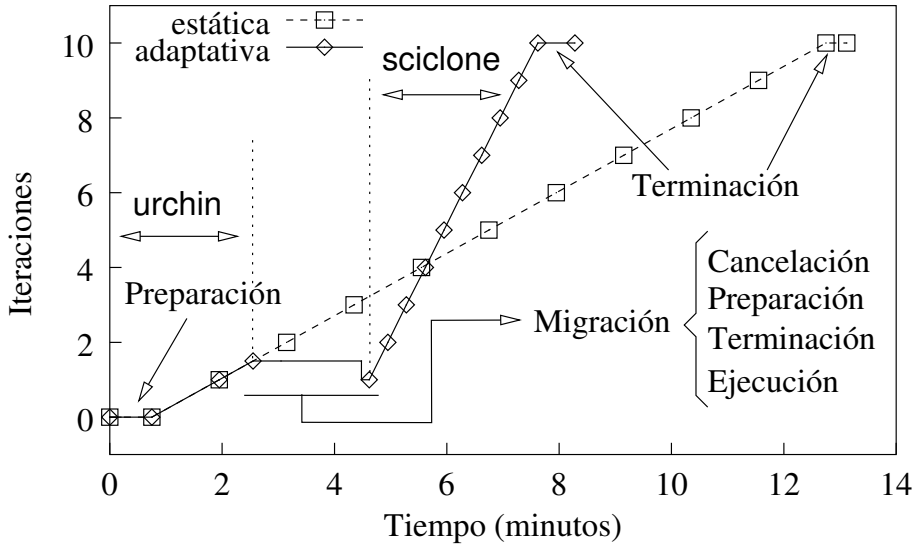

FIGURA 3.9: Perfil de ejecución de la aplicación cuando se descubre un recurso mejor.

La figura 3.9 muestra que el tiempo de ejecución total es un 42 % menor cuando el trabajo es migrado. Esta mejora podría ser incluso mayor para tiempos de ejecución más largos, ya que el tiempo de migración (95 segundos) representa el 20 % del tiempo total de ejecución. Es interesante remarcar que la mejora obtenida mediante la migración del trabajo depende enormemente de la cantidad de trabajo computacional ya realizado por la aplicación y la sobrecarga inducida por la migración [71, 62]. También es interesante recordar que la herramienta también permite al usuario forzar la replanificación de un trabajo para detectar nuevos recursos.

3.6.2. Detección de una Degradación del Rendimiento

El *resource selector* encuentra a whale como mejor recurso, y se envía el trabajo (preparación y envío en el periodo del segundo 0 al 34). Sin embargo, whale se satura con una carga de trabajo intensiva en cálculo (segundo 34). Como resultado, el *performance*

evaluator detecta una degradación de rendimiento cuando el tiempo de iteración excede el tiempo máximo de iteración establecido (40 segundos). El trabajo se migra a sciclone (cancelación, preparación, finalización y envío en el periodo del segundo 209 al 304), donde continúa ejecutándose desde el último contexto guardado. El perfil de ejecución para esta situación se muestra en la figura 3.10. En este caso, el tiempo total de ejecución es un 35 % menor cuando el trabajo es migrado. El coste de la migración (95 segundos) es alrededor del 21 % del tiempo total de ejecución.

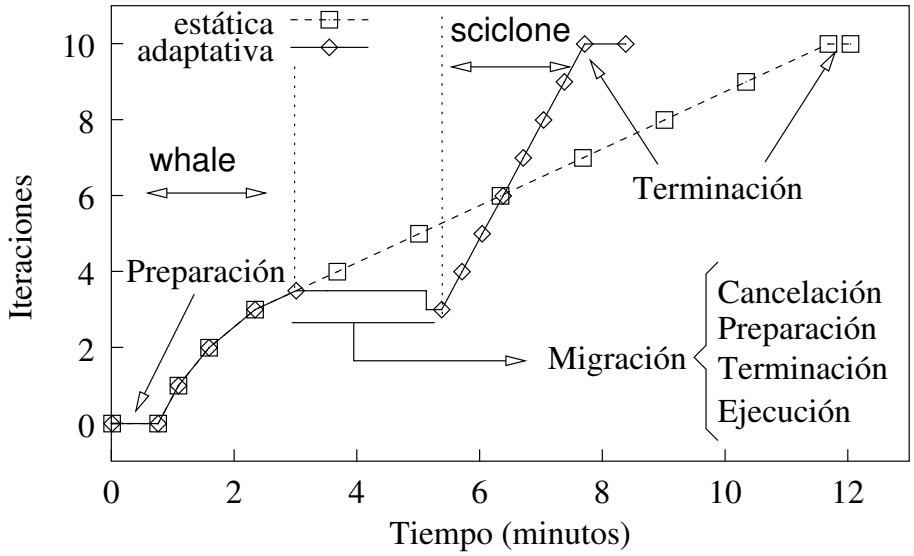

FIGURA 3.10: Perfil de ejecución de la aplicación cuando se aplica una carga de trabajo artificial.

3.6.3. Detección de un Fallo

El *resource selector* encuentra a whale como mejor recurso, se transfieren los ficheros y se envía el trabajo (preparación y envío en el periodo del segundo 0 al 45). Al principio, el trabajo se ejecuta normalmente. Sin embargo, whale es desconectado (segundo 125). Después de 50 segundos, se detecta el fallo de conexión y el trabajo es migrado a sciclone (preparación y envío en el periodo del segundo 175 al 250). La aplicación es ejecutada de nuevo desde el principio debido a que la máquina local no tiene acceso a los ficheros de reinicio generados en whale. Finalmente, el trabajo completa su ejecución en sciclone. El perfil de ejecución de la aplicación se muestra en la figura 3.11.

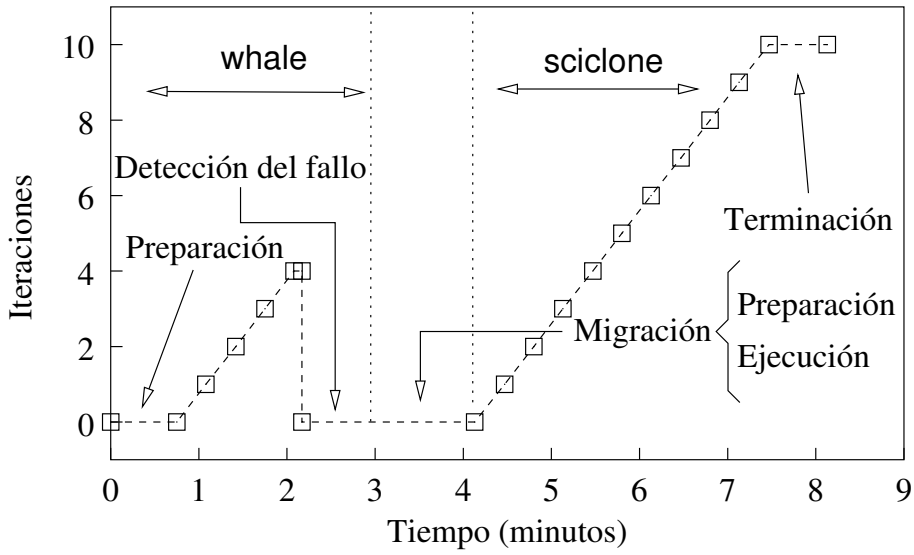

FIGURA 3.11: Perfil de ejecución de la aplicación cuando el recurso remoto falla.

3.6.4. Superación del Tiempo Máximo de Suspensión

Inicialmente, se selecciona sciclone para la ejecución, se transfieren los ficheros y se envía el trabajo (preparación y envío en el periodo del segundo 0 al 51). El trabajo se pone en espera explícitamente mediante la ejecución del comando PBS `qhold` en el sistema remoto. El trabajo es replanificado tan pronto como el tiempo máximo de suspensión establecido (40 segundos) es excedido. El *resource selector* selecciona whale como siguiente recurso, y el proceso de migración es entonces iniciado (cancelación, preparación, finalización y envío en el periodo del segundo 102 al 165). Finalmente, el trabajo completa su ejecución en whale. El perfil de ejecución de la aplicación se muestra en la figura 3.12.

3.6.5. Cambio de Requisitos

El método multimalla del código CFD es un algoritmo multimalla completo (*Full MultiGrid*, FMG). Los cálculos se inician en la malla más gruesa y, una vez que el sistema discreto es resuelto, la solución se transfiere al siguiente nivel más fino. La solución prolongada es entonces usada como solución aproximada del método multimalla en ese nivel. Este procedimiento se repite hasta que se alcanza el nivel más fino. El trabajo en ejecución solicita una mayor cantidad de memoria RAM, de 512MB a 2GB, antes de procesar la malla más fina para asegurarse de tener suficiente memoria principal disponible. El trabajo cambia el fichero dinámico con la expresión de requisitos (`requirement expression`) para

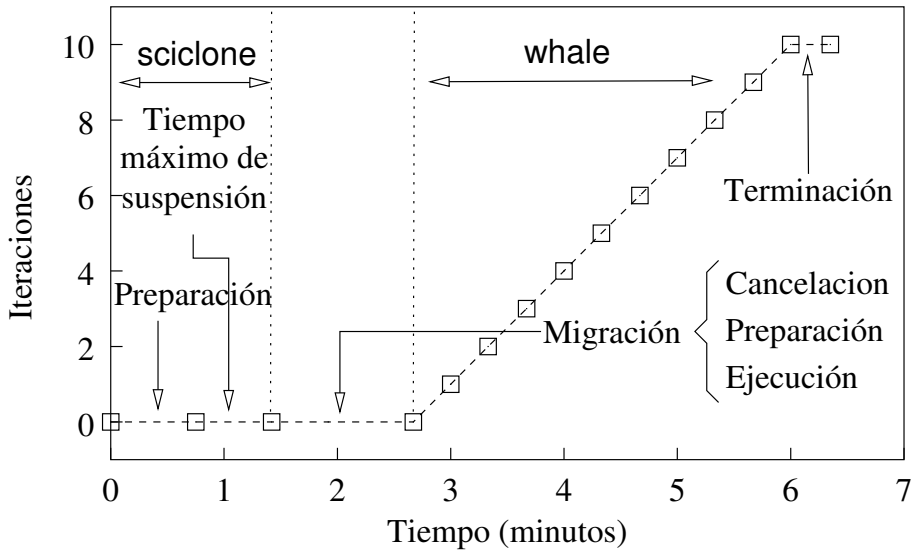

FIGURA 3.12: Perfil de ejecución de la aplicación cuando se excede el tiempo máximo de suspensión.

indicar:

```
(Mds-Memory-Ram-sizeMB>=2048)
```

y solicita una replanificación. Como resultado de esta actualización, el trabajo es migrado de urchin, donde fue inicialmente enviado y que no cumple los nuevos requisitos, a whale, que sí los cumple.

3.7. Experimentos sobre Planificación Adaptativa

Consideremos ahora una aplicación de barrido de parámetros consistente en 200 tareas independientes. Cada tarea calcula el flujo sobre una placa delgada para un número de Reynolds diferente, desde 10^2 hasta 10^4. Para este experimento se utilizó el banco de pruebas UCM-CAB, cuyas principales características están resumidas en la sección A.2 de los apéndices.

El tiempo total de ejecución de la aplicación paramétrica fue de 8778 segundos (2 horas, 26 minutos y 18 segundos), con un tiempo medio de respuesta por trabajo de 43 segundos. La figura 3.13 muestra el tiempo medio de respuesta por trabajo en cada uno de los recursos del banco de pruebas UCM-CAB; las barras de error representan la desviación estándar del tiempo de respuesta. Estos tiempos incluyen la sobrecarga inducida por el *middleware* de Globus. Además, el tiempo de ejecución medio en babieca incluye el tiempo de espera en la

cola del sistema PBS. Comparada con la ejecución en el recurso más rápido del banco de pruebas (pegasus, con 62 segundos por trabajo), estos resultados representan una reducción del 30 % en el tiempo total de ejecución.

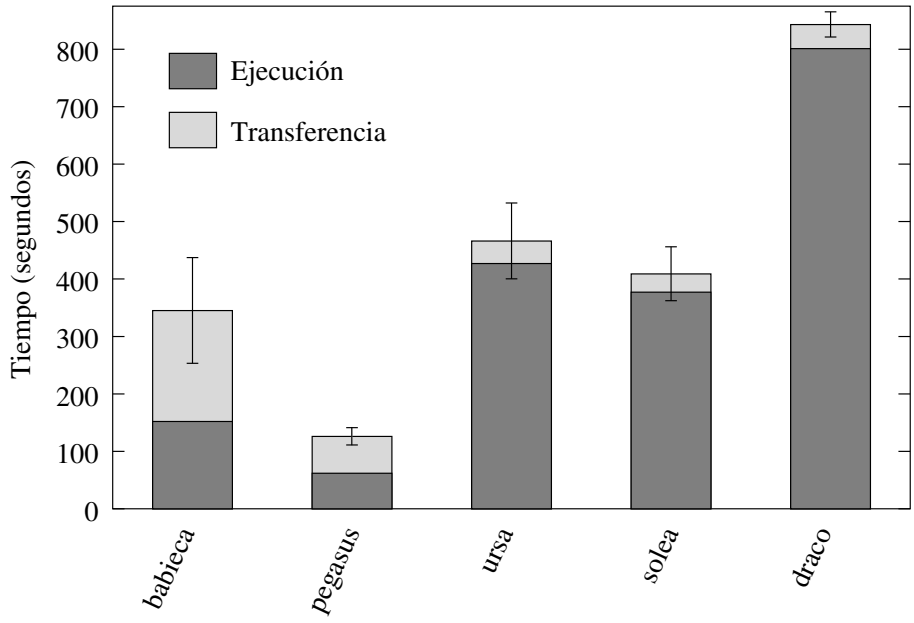

FIGURA 3.13: Tiempo medio de respuesta y desviación estándar para la aplicación de barrido de parámetros en cada recurso del banco de pruebas.

Evaluemos ahora la planificación realizada por la herramienta Grid *Way* comparada con la planificación óptima realizada de forma estática. Esta comparación debe verse sólo como una referencia, cuyo objetivo es establecer un límite máximo de rendimiento y destacar la importancia de la planificación adaptativa en un entorno Grid. La planificación óptima minimizaría el tiempo total de ejecución de la aplicación (*application makespan*) [67, 105], es decir, el tiempo hasta que todos los resultados de la aplicación están disponibles para el usuario:

$$\begin{aligned} &\textbf{Minimizar} \\ &\qquad \text{máx}\{N_i \overline{T}_i\} \; \forall i \in GR \\ &\textbf{sujeto a} \\ &\qquad \sum_{i \in GR} N_i - 200 = 0; \\ &\qquad 0 \leq N_i \leq 200 \; \forall i \in GR, \end{aligned} \tag{3.2}$$

donde N_i es el número de trabajos ejecutados en el recurso i, $\overline{T}_i$ es el tiempo medio de respuesta en el recurso i y GR es el conjunto de todos los recursos del Grid ($GR = \{pegasus, draco, ursa, solea, babieca\}$).

La figura 3.14 muestra el número de trabajos planificados en cada recurso por la herramienta Grid*W*ay y la solución al problema de optimización para minimizar la expresión 3.2. El tiempo total de ejecución para la planificación óptima sería, en teoría, de 7285 segundos (2 horas, 1 minuto y 25 segundos), con un tiempo medio de respuesta por trabajo de 36 segundos, es decir, un 17 % mejor que la planificación realizada por la herramienta Grid*W*ay.

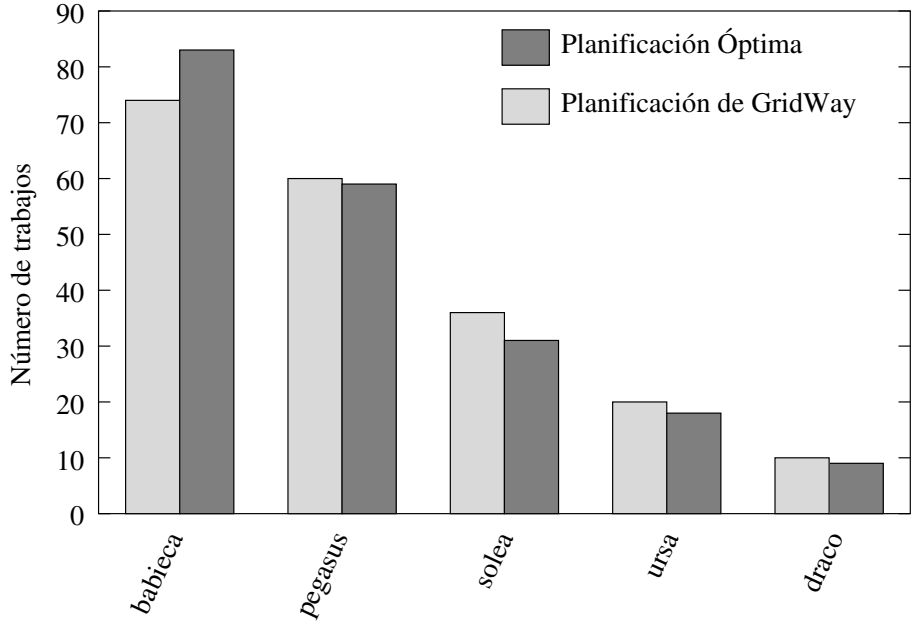

FIGURA 3.14: Número de trabajos planificados en cada recurso para la planificación óptima y la realizada por la herramienta Grid*W*ay.

La principal diferencia entre ambas planificaciones es el mayor número de trabajos asignados a babieca por la planificación óptima. Sin embargo, el 14 % de los trabajos enviados a babieca fallaron, por lo que tuvieron que ser replanificados dinámicamente a otros recursos disponibles en el banco de pruebas. La figura 3.15 muestra el tiempo de respuesta dinámico de los trabajos durante la ejecución de la aplicación de barrido de parámetros. Como podría esperarse, los fallos experimentados en babieca incrementaron el tiempo medio de respuesta por trabajo, así como el tiempo total de ejecución.

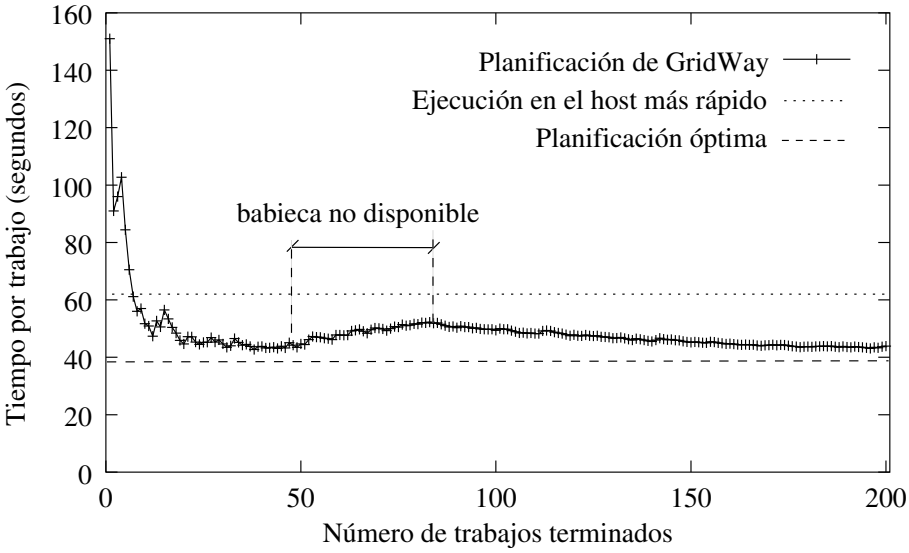

FIGURA 3.15: Tiempo de respuesta dinámico de los trabajos en la ejecución de la aplicación paramétrica.

3.8. Conclusiones

En este capítulo se han descrito las principales características y el funcionamiento del prototipo Grid*W*ay para planificación y ejecución adaptativa en entornos Grid. Se ha mostrado la capacidad de Grid*W*ay para mejorar el rendimiento de aplicaciones individuales y aplicaciones de alta productividad, en especial las de barrido de parámetros.

Grid*W*ay consigue la ejecución robusta y eficiente de las aplicaciones combinando: **planificación adaptativa**, para reflejar las características dinámicas del Grid; **ejecución adaptativa**, para migrar trabajos en ejecución a "mejores" recursos y proporcionar tolerancia a fallos; y **reutilización de ficheros** comunes entre tareas, para reducir la sobrecarga por transferencia de ficheros.

Tanto la tolerancia a fallos como el tiempo de respuesta mejoran cuando la aplicación es ejecutada a través de Grid*W*ay. Estos resultados experimentales son prometedores, ya que muestran cómo estas aplicaciones pueden aprovechar de manera eficiente los recursos computacionales altamente distribuidos proporcionados por un Grid.

4. Selección de Recursos para Migración Oportunista

Este capítulo se centra en la migración oportunista de trabajos cuando un recurso aparece disponible en el Grid, bien porque se añade en ese momento al Grid, o bien porque una aplicación completa su ejecución y lo deja libre. En esta situación, el rendimiento del nuevo recurso, el tiempo de ejecución restante de la aplicación y también la proximidad –en términos de ancho de banda y latencia– del nuevo recurso a los datos necesarios, se convierten en factores críticos para decidir si la migración es factible y merece la pena. Discutiremos una extensión de la herramienta Grid*W*ay para considerar todos estos factores en las etapas de selección de recursos y migración de trabajos, de forma que se mejore el tiempo de respuesta de las aplicaciones individuales [62, 106]. Los beneficios del nuevo selector de recursos se demostrarán con la ejecución adaptativa de un código de dinámica de fluidos computacional (CFD).

4.1. Introducción

Los Grids son por naturaleza entornos altamente dinámicos y heterogéneos, y éste es especialmente el caso del rendimiento de los enlaces de interconexión entre los recursos del Grid. Por tanto, la selección de recursos en el Grid debe tener en cuenta la proximidad de los recursos computacionales a los datos necesarios para reducir el coste de transferencia de los ficheros [62, 107]. Este hecho es especialmente importante en el caso de ejecución adaptativa de trabajos, ya que la migración requiere la transferencia de grandes ficheros de reinicio entre los recursos de ejecución.

Trabajos previos sobre migración oportunista [71, 68] han demostrado claramente la relevancia de considerar la cantidad de trabajo computacional ya realizado por la aplicación, la necesidad de una métrica para medir la ganancia en rendimiento obtenida por la migración y el factor crítico de la información dinámica acerca de la carga de los recursos del Grid. Sin embargo, estos trabajos previos no consideran la proximidad de los recursos computacionales a los datos necesarios y, por tanto, la potencial ganancia en rendimiento

puede verse sustancialmente reducida por la sobrecarga inducida por la migración.

4.2. Estimación del Rendimiento de la Red de Interconexión

Se pueden adoptar diferentes estrategias para obtener los atributos de rendimiento de la red (ancho de banda y latencia entre recursos) dependiendo de los servicios disponibles en cada banco de pruebas. Por ejemplo, el servicio MDS de Globus se puede configurar para proporcionar ese tipo de información accediendo al NWS (*Network Weather Service*) o activando la publicación de las estadísticas del servicio GridFTP (ver subsecciones siguientes). Alternativamente, el usuario final puede proporcionar sus propios programas de prueba de la red (por ejemplo, usando la herramienta `Iperf` [108, 109]) o tablas estáticas.

4.2.1. Network Weather Service (NWS)

NWS [110, 111] es un sistema distribuido que monitoriza periódicamente y predice dinámicamente el rendimiento que diversos recursos computacionales y de red pueden ofrecer durante un intervalo de tiempo dado. El servicio opera un conjunto distribuido de sensores de rendimiento (monitores de red, de procesador, etc.), de los cuales recoge lecturas de las condiciones instantáneas. Posteriormente, se usan modelos numéricos para generar predicciones sobre qué condiciones se darán en un marco de tiempo dado. Esta funcionalidad es análoga a la de los sistemas de predicción meteorológica (*weather forecasting*), de ahí el nombre dado a este sistema. La figura 4.1 muestra la arquitectura de NWS.

NWS se ha desarrollado para ser usado por planificadores dinámicos y para proporcionar calidad del servicio (QoS) estadística en sistemas distribuidos. La metodología de planificación de AppLeS (ver sección 2.5.4) hace un uso extensivo de sus servicios y se han desarrollado prototipos para Globus y la arquitectura de sistemas de información del Grid (*Grid Information System*, GIS) del GGF.

Actualmente, el sistema incluye sensores para rendimiento TCP/IP (ancho de banda y latencia) de extremo a extremo, porcentaje de CPU libre y memoria no paginada disponible. Sin embargo, se está trabajando en un interfaz de sensores, que permitirá que se configuren nuevos sensores internos en el sistema.

El conjunto actual de métodos de predicción soportados trata las medidas sucesivas de cada monitor como una serie temporal. Los métodos iniciales se dividen en tres categorías:

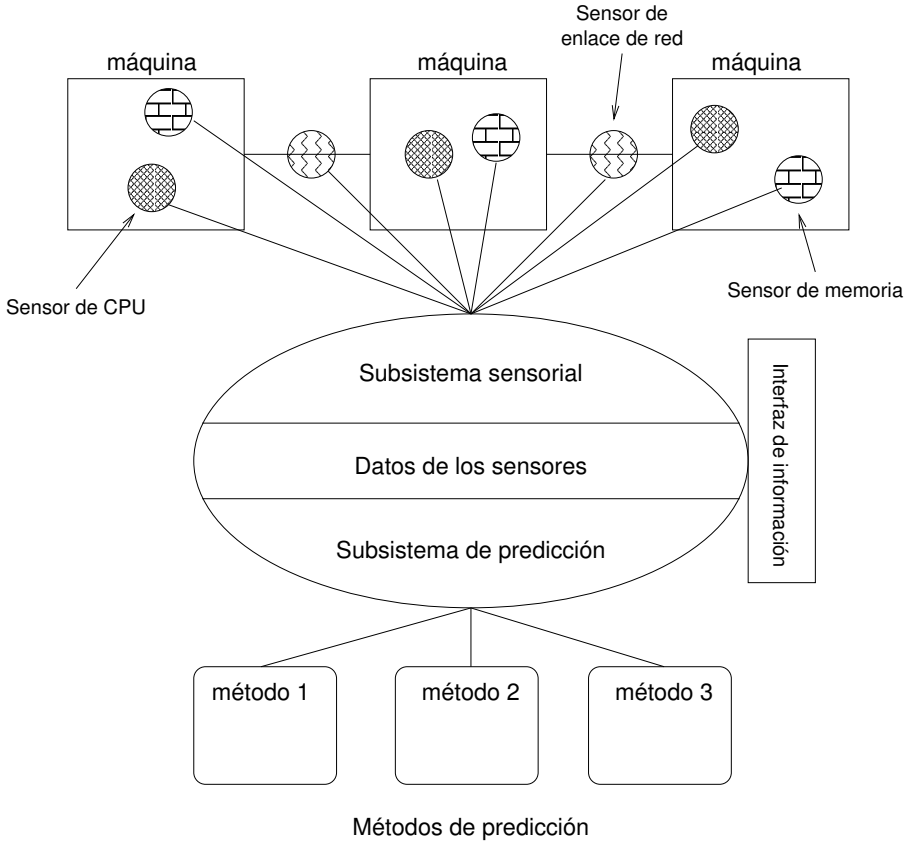

FIGURA 4.1: Arquitectura de NWS [111].

- Métodos basados en la media, que usan una estimación de la media de la muestra como predicción.
- Métodos basados en la mediana, que usan una estimación de la mediana, lo cual es muy útil si la secuencia de medidas presenta valores atípicos (*outliers*) asimétricos que ocurren aleatoriamente.
- Métodos auto-regresivos, que modelan mejor el tráfico agregado de paquetes en Internet.

El sistema lleva cuenta de la precisión de todos los predictores (usando el error de la predicción como medida de precisión), y usa aquél que exhibe el menor error acumulativo en un momento dado para generar una predicción. De esta forma, NWS identifica automáticamente la mejor técnica de predicción para cada recurso. Es más, conforme se vayan desarrollando nuevos métodos de predicción, podrán ser usados automáticamente para predecir el rendimiento de los recursos para los cuales sean más precisos.

4.2.2. GridFTP Reporter

En otro trabajo [112], se ha instrumentado el servidor GridFTP para registrar el rendimiento alcanzado por cada transferencia de datos de extremo a extremo. Esta información, junto con meta-datos que indican la naturaleza de la transferencia, sirve como entrada para unos predictores. Las entradas del registro incluyen la dirección origen, nombre del fichero, tamaño, número de flujos paralelos, tamaño del *buffer* TCP, tiempos de inicio y fin, tiempo total consumido por la transferencia, ancho de banda agregado alcanzado por la transferencia, naturaleza de la operación (lectura o escritura) y volumen lógico de origen y destino de la transferencia.

El uso de transferencias de datos reales para recoger información de rendimiento, en lugar de pruebas sintéticas, proporciona datos más precisos y no impone una sobrecarga adicional. La desventaja es que no se tiene control sobre los intervalos en los que se recogen los datos. Esta situación puede limitar las técnicas predictivas que se aplican a los datos. Sin embargo, el sistema podría extenderse para realizar pruebas de transferencia de ficheros a intervalos regulares.

La simple recogida de datos no es suficiente para tomar una decisión sobre, por ejemplo, selección de réplicas. En la mayoría de los casos, es necesaria una predicción del comportamiento futuro, no un mero recitado del comportamiento pasado. Por este motivo, se usan técnicas de predicción similares a las de NWS, realizando filtrados independientes del contexto (por ejemplo, tomando sólo las últimas cinco medidas) y filtrados dependientes del contexto (por ejemplo, eligiendo sólo los datos para transferencias de tamaño similar). Además, en lugar de elegir una única técnica de predicción, se evalúa un número de ellas y se elige la más apropiada al vuelo, como se hace en NWS.

Para el acceso a los datos de monitorización de GridFTP, se utiliza un proveedor de información MDS que accede a los datos registrados para anunciar un conjunto de medidas recientes así como un resumen estadístico del resto de los datos. Para generar información estadística de las transferencias, se utiliza un guión de servidor (*back-end script*) de LDAP para filtrar la información del registro junto con un nuevo esquema MDS para publicar estos datos. La figura 4.2 muestra un fragmento de la salida del proveedor de información de GridFTP.

```
...
dn: "147.96.80.227, hostname=ursa.dacya.ucm.es, o=DACYA-UCM, o=Grid"
cn: "147.96.80.227"
hostname: "ursa.dacya.ucm.es"
gridftpurl: "gsiftp://ursa.dacya.ucm.es:2811"
minrdbandwidth: 1462K
maxrdbandwidth: 12800K
avgrdbandwidth: 6062K
avgrdbandwidthtenmbrange: 5714K
...
```

FIGURA 4.2: Fragmento de la salida del proveedor de información de GridFTP.

4.3. Selección de Recursos Considerando la Proximidad a los Datos

El nuevo proceso de selección de recursos considera tanto el rendimiento dinámico como la proximidad dinámica de los recursos computacionales. En particular, los siguientes términos son considerados en la etapa de selección de recursos:

- El tiempo computacional estimado en el recurso candidato cuando el trabajo se envía desde el cliente o migra desde otro recurso, para reducir el tiempo de ejecución.
- La proximidad entre el recurso candidato y el cliente, para reducir el coste de envío y monitorización del trabajo y de la transferencia de ficheros almacenados en el cliente.
- La proximidad entre el recurso candidato y un servidor de ficheros remoto, para reducir el coste de transferencias cuando algunos ficheros de entrada o salida se almacenan en ese servidor.
- La proximidad entre el recurso candidato y el actual recurso, para reducir la sobrecarga de la migración, ya que los ficheros de reinicio se transfieren entre ellos.

4.3.1. Modelo de Rendimiento

Para reflejar todas las circunstancias previamente descritas, cada recurso candidato (h_n) se clasifica usando un modelo de rendimiento que trata de estimar el tiempo de respuesta (T_{wall}) cuando el trabajo es enviado o migrado a ese recurso en un instante determinado (t_n):

$$Rank(h_n, t_n) = -T_{wall}(h_n, t_n). \tag{4.1}$$

En este caso, podemos asumir que el tiempo total de envío se puede dividir en:

$$T_{wall}(h_n, t_n) = T_{exe}(h_n, t_n) + T_{xfr}(h_n, t_n), \tag{4.2}$$

o simplemente en:

$$T_{wall}(h_n, t_n) = T_{exe}(h_n, t_n), \tag{4.3}$$

donde T_{exe} es el tiempo de ejecución estimado y T_{xfr} es el tiempo de transferencia estimado, como se indica más abajo.

Considerando primero la ejecución en un solo recurso, el tiempo computacional de una aplicación intensiva en procesador en el recurso h en el instante t puede ser estimado por:

$$T^s_{exe}(h,t) = \begin{cases} \frac{Op}{FLOPS} & \text{if} \quad CPU(t) \geq 1; \\ \frac{Op}{FLOPS \cdot CPU(t)} & \text{if} \quad CPU(t) < 1, \end{cases} \tag{4.4}$$

donde $FLOPS$ es el rendimiento pico alcanzable por el procesador del recurso, $CPU(t)$ es el porcentaje de CPU libre en el instante t, tal y como la proporciona el esquema MDS por defecto (en el atributo `Mds-Cpu-Free-1minX100`), y Op es el número de operaciones en coma flotante que realiza la aplicación. En el caso de *clusters* con dedicación exclusiva de los nodos, se considera que $CPU(t)$ es igual a 1 para cada nodo disponible en el *cluster*.

Sin embargo, la expresión anterior no es acertada cuando el trabajo ha estado ejecutándose en varios recursos y después migra a uno nuevo. En esta situación se debe considerar también la cantidad de trabajo computacional ya realizado. Supongamos que una aplicación ha estado ejecutándose en los recursos $h_0 \ldots h_{n-1}$ en los instantes $t_0 \ldots t_{n-1}$ y entonces migra al recurso h_n en el instante t_n. En ese caso, el tiempo de ejecución podría estimarse por:

$$T_{exe}(h_n, t_n) = \sum_{i=0}^{n-1} t^i_{exe} + \left(1 - \sum_{i=0}^{n-1} \frac{t^i_{exe}}{T^s_{exe}(h_i, t_i)}\right) T^s_{exe}(h_n, t_n), \tag{4.5}$$

donde $T^s_{exe}(h,t)$ se calcula usando la expresión 4.4, y t^i_{exe} es el tiempo que el trabajo ha estado ejecutándose en el recurso h_i, que es medido por la herramienta Grid$W$ay. Nótese que cuando n es 0, es decir, cuando no hay migración sino que se trata del primer envío, las expresiones 4.4 y 4.5 son equivalentes.

De forma similar, la siguiente expresión estima el tiempo de transferencia:

$$T_{xfr}(h_n, t_n) = \sum_{i=0}^{n-1} t^i_{xfr} + \sum_j \frac{Data_{h_n,j}}{bw(h_n, j, t_n)} \quad j = client, file\ server, exec\ host, \tag{4.6}$$

donde $bw(h_1, h_2, t)$ es el ancho de banda entre los recursos h_1 y h_2 en el instante t, $Data_{h_1,h_2}$ es el tamaño de los ficheros a transferir entre ellos, y t^i_{xfr} es el tiempo que duró la transferencia en el recurso h_i, que es medido por la herramienta Grid *Way*.

Cuando el trabajo se replanifica para descubrir nuevos recursos, el *dispatch manager* elige el recurso con mayor rango (menor tiempo total de envío estimado) para migrar el trabajo sólo si tiene mayor rango que el actual recurso donde fue enviado el trabajo, de otra forma la migración es rechazada. Por tanto, el trabajo será migrado sólo si se considera que la migración es provechosa. Sin embargo, cuando el trabajo es replanificado debido a un fallo o una petición del usuario, el *dispatch manager* siempre concede la migración al candidato con mayor rango, incluso si tiene un rango menor que el actual recurso de ejecución.

4.4. Migración Oportunista de Trabajos

Para implementar la migración oportunista de trabajos, el *dispatch manager* se despierta en cada intervalo de descubrimiento (*discovering interval*) e intenta encontrar un recurso mejor para cada trabajo activando el proceso de selección de recursos. Para evaluar los beneficios de la migración de trabajos desde el actual recurso (h_{n-1}) a cada recurso candidato (h_n), definimos la ganancia de migración (G_m) como:

$$G_m = \frac{T_{wall}(h_{n-1}, t_{n-1}) - T_{wall}(h_n, t_n)}{T_{wall}(h_{n-1}, t_{n-1})}, \tag{4.7}$$

donde $T_{wall}(h_{n-1}, t_{n-1})$ es el tiempo total estimado de envío en el recurso donde se está ejecutando el trabajo, y $T_{wall}(h_n, t_n)$ es el tiempo total estimado de envío cuando el trabajo migra al nuevo recurso candidato. La migración sólo se concede si la ganancia de migración es mayor que un límite definido por el usuario, de otra forma se rechaza. Nótese que, aunque el límite de migración es fijo para un trabajo dado, la ganancia de migración se calcula dinámicamente para tener en cuenta la sobrecarga dinámica de la transferencia de datos, el rendimiento dinámico de los recursos y el progreso de la aplicación. En los experimentos presentados, la ganancia de migración se ha fijado a un 10 %.

4.5. Experimentos

El comportamiento de la estrategia de selección de recursos descrita anteriormente se demuestra con la ejecución de un código CFD, que resuelve las ecuaciones incompresibles de Navier-Stokes en 3D usando un método multimalla iterativo [102]. Los experimentos se han realizado en el banco de pruebas de la UCM, cuyas características se resumen en la sección A.2. En los siguientes experimentos, la máquina cliente (*client*) es ursa, que almacena un fichero de entrada con los parámetros de la simulación, y el servidor de ficheros (*file server*) es cepheus, que almacena el ejecutable y la malla computacional. El fichero de salida con los campos de velocidad y presión es transferido de vuelta al cliente, ursa, para realizar un post-proceso de los resultados. La tabla 4.1 muestra las máquinas disponibles en el banco de pruebas, el rendimiento pico de su procesador (MFLOPS), y el ancho de banda (MB/s) entre ellas y los recursos involucrados en el experimento.

TABLA 4.1: Máquinas disponibles en el banco de pruebas de la UCM, rendimiento pico de su procesador y ancho de banda entre ellas y los recursos involucrados en el experimento (*client*=ursa, *file server*=cepheus y *exec host*=draco).

Nombre completo	Rendimiento pico	Ancho de banda		
		client	*file server*	*exec host*
ursa.dacya.ucm.es	330	∞	0.4	0.4
draco.dacya.ucm.es	175	0.4	0.4	∞
columba.dacya.ucm.es	225	0.4	0.4	0.4
cepheus.dacya.ucm.es	325	0.4	∞	0.4
solea.quim.ucm.es	350	0.2	0.2	0.2

Se imponen dos requisitos a los recursos computacionales: un mínimo de 128MB de memoria principal, suficiente para alojar la simulación CFD, y un porcentaje de CPU libre mayor del 90 %, para evitar migraciones oscilantes. Inicialmente, columba y solea se encuentran ejecutando una carga de trabajo artificial, por lo que la aplicación es enviada a draco (*exec host*), puesto que es el único recurso que cumple los requisitos previos. A continuación, evaluaremos las estrategias de replanificación basadas en las expresiones 4.2 y 4.3, cuando termina la carga de trabajo artificial ejecutándose en columba y solea en diferentes puntos de ejecución (iteraciones) de la aplicación ejecutándose en draco.

Supongamos primero que la aplicación se replanifica considerando sólo el tiempo de ejecución (expresión 4.3), como se muestra en la figura 4.3. En este caso, el tiempo de

transferencia de ficheros no se considera, por lo que la migración a cualquiera de los recursos presenta una ganancia de migración. El *dispatch manager* considerará factible la migración al recurso con mayor rango, solea, hasta que se alcance la octava iteración ($T_{wall} = 302$).

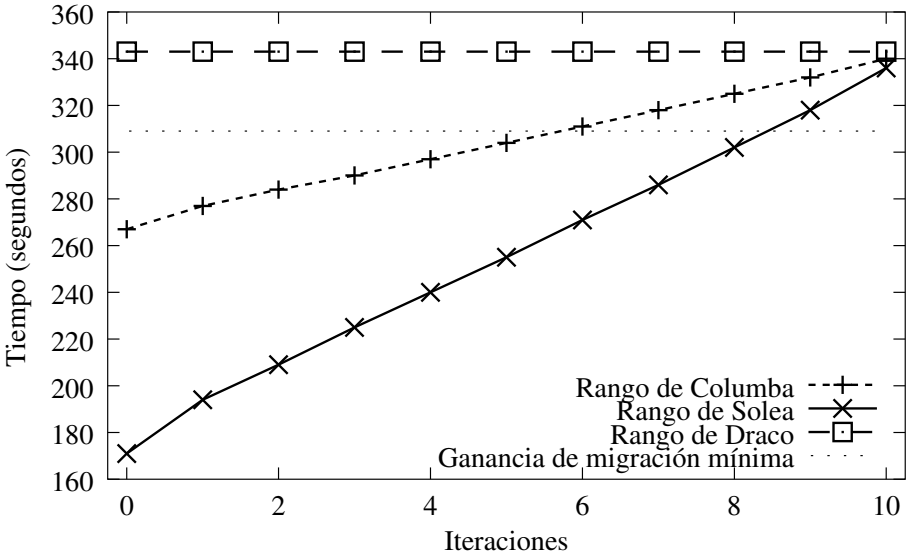

FIGURA 4.3: Tiempos de envío estimados (rangos) de la aplicación cuando migra desde draco a diferentes recursos en diferentes puntos de ejecución, usando la expresión 4.3.

La figura 4.4 muestra los rangos dinámicos de solea y columba cuando la aplicación se replanifica usando la expresión 4.2. En esta situación, la migración a solea será concedida sólo hasta que se alcance la segunda iteración ($T_{wall} = 325$). Nótese que, a partir de la quinta iteración, la ganancia en rendimiento ofrecida por solea y columba no es suficiente para compensar la sobrecarga inducida por la migración. Es más, a partir de la sexta iteración, el recurso candidato con mayor rango es columba (el más cercano), aunque presenta un rendimiento de procesador peor que solea, ya que la proximidad a los datos se va haciendo más importante conforme la aplicación progresa y queda menos trabajo computacional por hacer.

La figura 4.5 muestra los tiempos de ejecución medidos en draco sin migración y cuando la aplicación es realmente migrada a solea y a columba en distintas iteraciones. Estos resultados experimentales muestran claramente que la replanificación basada sólo en el rendimiento de los recursos y en el progreso de la aplicación (expresión 4.3) puede no producir beneficios de rendimiento. En particular, replanificando el trabajo basándose en la expresión 4.2 se obtiene una ganancia de rendimiento del 13 % (12 % predicho). Esta estrategia de selección de recursos rechazará migraciones a partir de la tercera iteración, evitando pér-

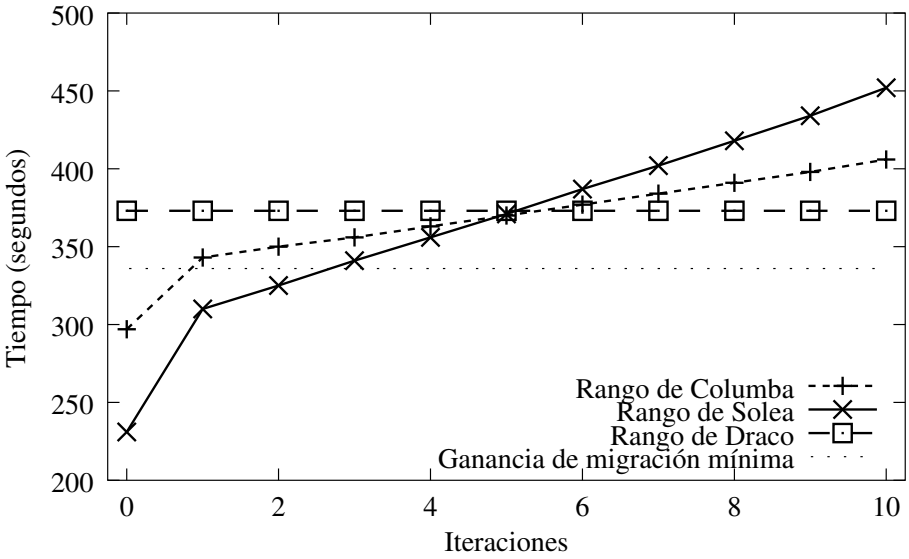

FIGURA 4.4: Tiempos de envío estimados (rangos) de la aplicación cuando migra desde draco a diferentes recursos en diferentes puntos de ejecución, usando la expresión 4.2.

didas de rendimiento de hasta el 15 %, que ocurrirían con la estrategia de replanificación basada sólo en el rendimiento. Nótese, que esta pérdida de rendimiento podría ser siempre evitada con un valor pesimista de G_m. Sin embargo, el uso de la expresión 4.2 permite un valor más agresivo del límite de la ganancia de migración, y por tanto una mejora en el tiempo de respuesta de la aplicación.

4.6. Conclusiones

En este capítulo, se ha analizado la importancia de considerar la proximidad de los recursos en el proceso de selección. En el caso de migración oportunista, la calidad de la red de interconexión tiene un impacto decisivo en la sobrecarga inducida por la migración del trabajo. De esta forma, es al menos tan importante considerar la proximidad del recurso a los datos necesitados, como considerar sus características de rendimiento. Se espera que la proximidad de los recursos sea incluso más importante para tamaños de fichero mayores y redes más heterogéneas. Es interesante hacer notar que la arquitectura modular y descentralizada de la herramienta Grid*W*ay garantiza la escalabilidad de la estrategia de selección de recursos, así como el rango de aplicación, ya que no está especializado para un conjunto específico de aplicaciones.

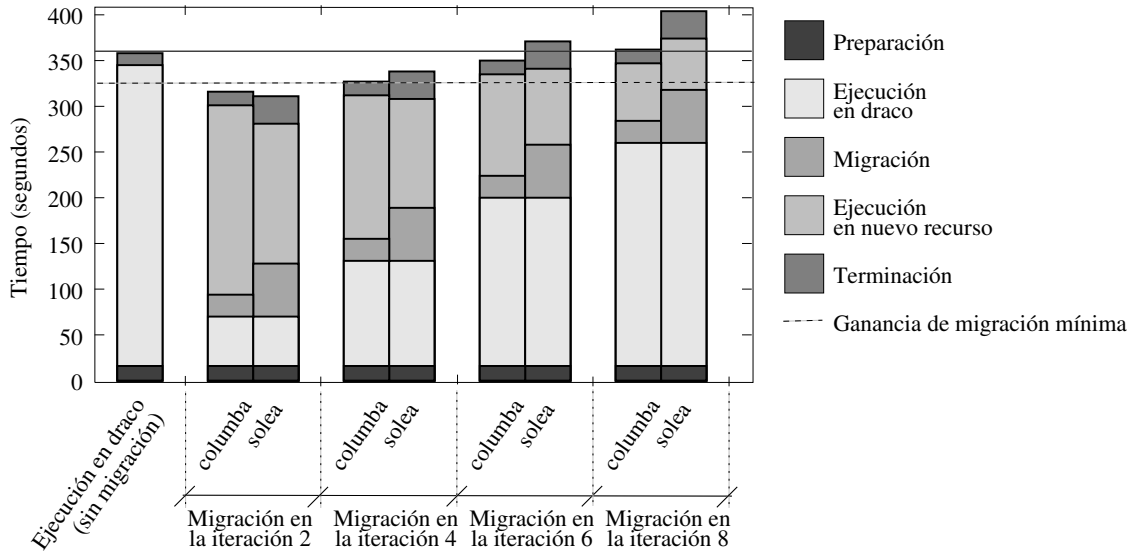

FIGURA 4.5: Perfil de ejecución de la aplicación cuando es ejecutada en draco, y cuando migra desde draco a solea o a columba en diferentes puntos de ejecución.

5. Aplicación a la Bioinformática

En este capítulo se muestran algunas experiencias obtenidas aplicando la tecnología Grid a la Bioinformática. Para ello, se ha adaptado una aplicación existente, que predice la estructura y propiedades termodinámicas de secuencias de proteínas, para su ejecución en el Grid con la ayuda de la herramienta Grid*W*ay [113, 114, 115].

5.1. Introducción

La Bioinformática, que se basa en la gestión y análisis de grandes cantidades de datos biológicos, podría beneficiarse enormemente de la idoneidad del Grid para ejecutar aplicaciones de alta productividad (ver sección 2.4). Es más, las colecciones de datos biológicos están creciendo muy rápidamente debido a la proliferación de procesos de experimentación de alta productividad y de organizaciones dedicadas a la Biotecnología. Por tanto, el análisis de los datos generados por estas nuevas técnicas de laboratorio pronto será posible únicamente a través de la computación Grid de alta productividad.

Uno de los principales desafíos en Biología Computacional trata sobre el análisis de la ingente cantidad de secuencias de proteínas proporcionadas por proyectos de Genómica a un ritmo cada vez mayor. La estructura de una proteína está codificada en su secuencia de aminoácidos, pero intentar descifrarla se ha convertido en un problema muy complicado, que todavía espera una solución completa. No obstante, en algunos casos, particularmente cuando se conocen proteínas homólogas estructuralmente, los métodos computacionales pueden ser bastante fiables.

5.2. Descripción de la Aplicación

En la Unidad de Bioinformática del Centro de Astrobiología (CAB), se ha desarrollado un algoritmo de predicción de estructuras de proteínas usando una función de energía

libre efectiva. Para cada estructura en el banco de datos de proteínas (*Protein Data Bank*, PDB), el algoritmo construye el alineamiento con la secuencia objetivo que maximiza una puntuación basada en una función de energía libre [116] más un término que penaliza los huecos en el alineamiento. Estos huecos representan residuos que son eliminados de la secuencia o de la estructura de forma que ambas encajen. Esto es motivado por el hecho de que durante la evolución se insertan o eliminan aminoácidos de las secuencias de proteínas y, por tanto, se va estropeando el alineamiento perfecto y sin huecos que tienen dos secuencias cuando se originan a partir de un antecesor común. La introducción de huecos incrementa enormemente el espacio de estructuras candidatas para la predicción de estructuras.

Para acelerar el análisis y reducir los datos necesarios, los ficheros de la PDB son preprocesados para extraer las matrices de contacto, que proporcionan una representación reducida de las estructuras de proteínas. El algoritmo es entonces aplicado dos veces, la primera vez como una búsqueda rápida para seleccionar las 200 mejores estructuras candidatas, y la segunda con parámetros que proporcionan una búsqueda más precisa del alineamiento óptimo.

El método ha sido probado en la quinta ronda del concurso CASP (*Critical Assessment of techniques for protein Structure Prediction*) [117]. Aunque es menos eficiente que los métodos basados en homología para reconocer proteínas relacionadas distantemente, cuando un pariente cercano de la estructura objetivo se encuentra en la PDB, incluso con muy poca similitud en la secuencia, el algoritmo la reconoce y produce un buen alineamiento entre la secuencia y la estructura. En estos casos, el algoritmo puede usarse para estimar parámetros termodinámicos de la secuencia objetivo, como la energía libre de plegamiento y el hueco de energía normalizado, y de esta forma ha servido para confirmar resultados previos sobre la eficiencia de plegamiento de proteínas en diferentes bacterias [118].

Se prevén dos estudios a gran escala aplicando el método arriba descrito. En ambos casos será necesario construir el alineamiento entre secuencia y estructura para un gran conjunto de secuencias, para lo que la tecnología Grid será esencial. Un estudio consiste en predecir la estructura y propiedades termodinámicas de todas las proteínas contenidas en un genoma completo. Esta aplicación todavía necesita algunas mejoras en el método, que falla si en la PDB no existe una estructura estrechamente relacionada con la objetivo. Actualmente, se está trabajando en la mejora de la función de energía y en la extensión del espacio de estructuras candidatas evaluadas.

En otro estudio, se aplica el algoritmo de predicción de estructuras a un gran número de familias de proteínas ortólogas, esto es, proteínas que realizan la misma función pero

en organismos diferentes, extendiendo así un estudio realizado previamente en el que se demostró que la eficiencia de plegamiento es menor en proteínas de bacterias intracelulares que en sus parientes de vida libre [118]. Si se conoce una estructura representativa de una proteína del conjunto, se espera que el algoritmo la reconozca como el mejor modelo estructural para cada secuencia, lo que permitirá estimar sus propiedades termodinámicas. Los resultados biológicos de este estudio comparativo de varias familias de proteínas han mostrado que existe una correlación entre características del genoma, como su tamaño y contenido de bases del ADN, y la termodinámica del plegamiento de proteínas [119].

5.3. Cambios en la Aplicación para su Ejecución en el Grid

Hemos modificado la aplicación para que proporcione un fichero de reinicio (`restart file`) y un perfil de rendimiento (`performance profile`). El fichero de reinicio, que es independiente de la arquitectura, almacena las mejores proteínas candidatas encontradas hasta el momento y la siguiente proteína en la PDB a analizar. El perfil de rendimiento almacena el tiempo dedicado en cada iteración del algoritmo, donde una iteración consiste en el análisis de un número determinado de secuencias.

Inicialmente, la aplicación no impone ningún requisito a los recursos, por lo que la expresión de requisitos (`requirement expression`) es nula. La expresión de clasificación (`ranking expression`) usa un modelo de rendimiento para estimar el tiempo de respuesta del trabajo como la suma del tiempo de ejecución y de transferencia, derivado del rendimiento y proximidad de los recursos candidatos (como se discutió en el capítulo 4).

El módulo de selección de recursos (*resource selector*) consiste en un archivo de comandos que consulta al MDS [39] para encontrar recursos de ejecución potenciales (ver sección 3.4.1). El módulo de evaluación del rendimiento (*performance evaluator*) es otro archivo de comandos que analiza el perfil de rendimiento y detecta caídas del rendimiento cuando el último tiempo de iteración es mayor que un límite dado.

El experimento completo es enviado como un trabajo en *array*, donde cada secuencia es analizada en una tarea independiente, especificando toda la información necesaria en un fichero de plantilla de trabajo (`job template`).

Los ficheros del experimento consisten en: el ejecutable (0.5MB) proporcionado para todas las arquitecturas de recursos en el banco de pruebas, los ficheros de la PDB (12.2MB) compartidos y comprimidos para reducir el tiempo de transferencia (ver sección 3.5.1),

Tabla 5.1: El banco de pruebas UCM-CAB.

Nombre	Modelo	Nodos	Vel. (MHz)	SO	Mem. (MB)	VO	Job mgr.
ursa	Sun Blade 100	1	500	Solaris 8	256		fork
draco	Sun Ultra 1	1	167	Solaris 8	128	DACYA	fork
pegasus	Intel Pentium 4	1	2400	Linux 2.4	1024		fork
solea	Sun Enterprise 250	2	296	Solaris 8	256	QUIM	fork
babieca	Compaq Alpha DS10	5	466	Linux 2.2	256	CAB	PBS

algunos ficheros de parámetros (1KB) y el fichero con la secuencia a analizar (1KB). El nombre final del ejecutable se obtiene resolviendo la variable `GW_ARCH` en tiempo de ejecución para el recurso seleccionado, y el nombre final con la secuencia a analizar, con la variable `GW_TASK_ID` para el trabajo actual.

5.4. Experimentos sobre Planificación Adaptativa

Hemos realizado los experimentos en el banco de pruebas experimental UCM-CAB, resumido en la tabla 5.1 y en la sección A.2 de los apéndices.

El algoritmo de predicción de estructuras y propiedades termodinámicas se ha aplicado a 88 secuencias de la enzima *Triosa Fosfato Isomerasa* (ver figura 5.1) expresada en distintos organismos. Una enzima es un tipo especial de proteína que actúa como catalizador de determinadas reacciones bioquímicas.

La figura 5.2 muestra la productividad obtenida cuando todas las máquinas del banco de pruebas estaban operativas. El tiempo total del experimento fue de 7,15 horas (7 horas y 9 minutos) y la productividad media fue de 12,30 trabajos/hora, lo que supone un tiempo medio de respuesta por trabajo de 4,88 minutos (4 minutos y 53 segundos).

5.4.1. Tolerancia a Fallos

La figura 5.3 muestra la productividad obtenida cuando babieca estuvo temporalmente desconectada por razones de mantenimiento. El tiempo total del experimento fue de 7,31 horas (sólo 9,6 minutos más que el experimento anterior) y la productividad media fue de 12,04 trabajos/hora, lo que supone un tiempo de respuesta medio por trabajo de 4,98 minutos. La productividad media bajó de 12,71 a 10,61 trabajos/hora durante el periodo

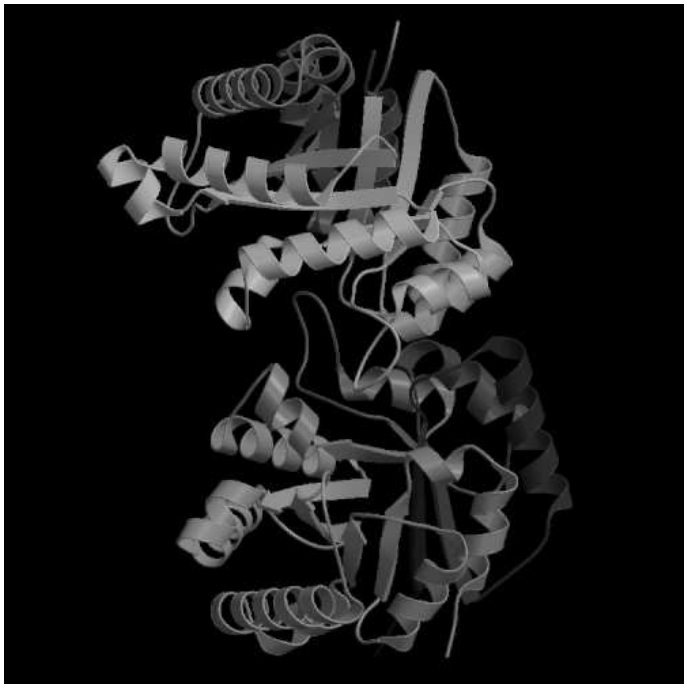

FIGURA 5.1: Estructura de la enzima *Triosa Fosfato Isomerasa*.

en que babieca estuvo desconectada, pero cuando volvió a estar operativa, la productividad comenzó a incrementarse hasta los 12,04 trabajos/hora.

5.4.2. Mejora del Rendimiento

La figura 5.4 muestra la productividad obtenida cuando pegasus fue descubierto a mitad del experimento, ya que hasta ese momento estaba fuera de servicio. El tiempo total del experimento fue de 8,65 horas, y la productividad media fue de 10,17 trabajos/hora, lo que supone un tiempo de respuesta medio por trabajo de 5,9 minutos. Antes de descubrir pegasus, la productividad media era sólo de 8,31 trabajos/hora, sin embargo, una vez descubierto, la productividad se incrementó hasta los 10,17 trabajos/hora.

5.5. Experimentos sobre Ejecución Adaptativa

En esta sección se muestran experimentos sobre ejecución adaptativa, donde el algoritmo sólo se aplica a cinco secuencias para poder mostrar el perfil de ejecución con mayor detalle. En este caso, nos hemos centrado en las capacidades de auto-adaptación de la aplicación (ver sección 1.4.2), es decir, la aparición de una degradación en el rendimiento

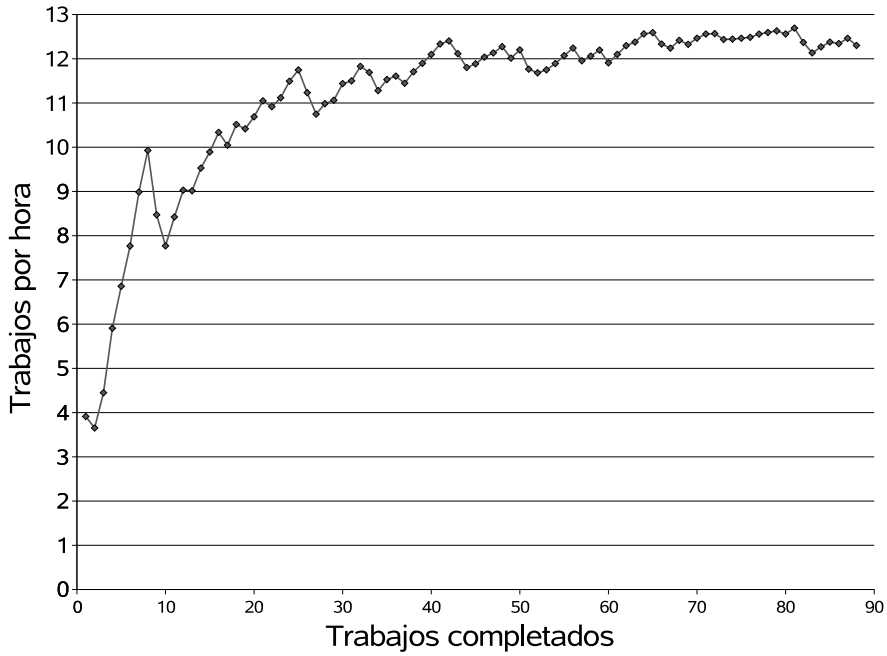

FIGURA 5.2: Productividad cuando todas las máquinas del banco de pruebas estaban operativas.

obtenido por la aplicación y la solicitud de una auto-migración debido a un cambio en los requisitos de la misma.

5.5.1. Detección de una Degradación del Rendimiento

Consideremos primero un experimento consistente en cinco tareas, cada una de las cuales aplica el algoritmo de predicción de estructuras a una secuencia diferente de la cadena épsilon de la enzima *ATP Sintasa* (ver figura 5.5) presente en organismos diferentes. Poco después de enviar el experimento, pegasus se saturó con una aplicación intensiva en cálculo.

La figura 5.6 muestra el perfil de ejecución en esta situación, junto con la carga en pegasus que causó la degradación del rendimiento y el progreso de la tarea 0 obtenido de su perfil de rendimiento. Inicialmente, se asignó una tarea a pegasus y las cuatro restantes, a babieca. Cuando el *performance evaluator* detectó la degradación del rendimiento, se solicitó la migración del trabajo. Dado que existía un hueco disponible en babieca, el trabajo migró a esa máquina aunque presentaba un menor rendimiento. A pesar de la sobrecarga inducida por la migración del trabajo (el 6 % del tiempo total de ejecución) el trabajo 0

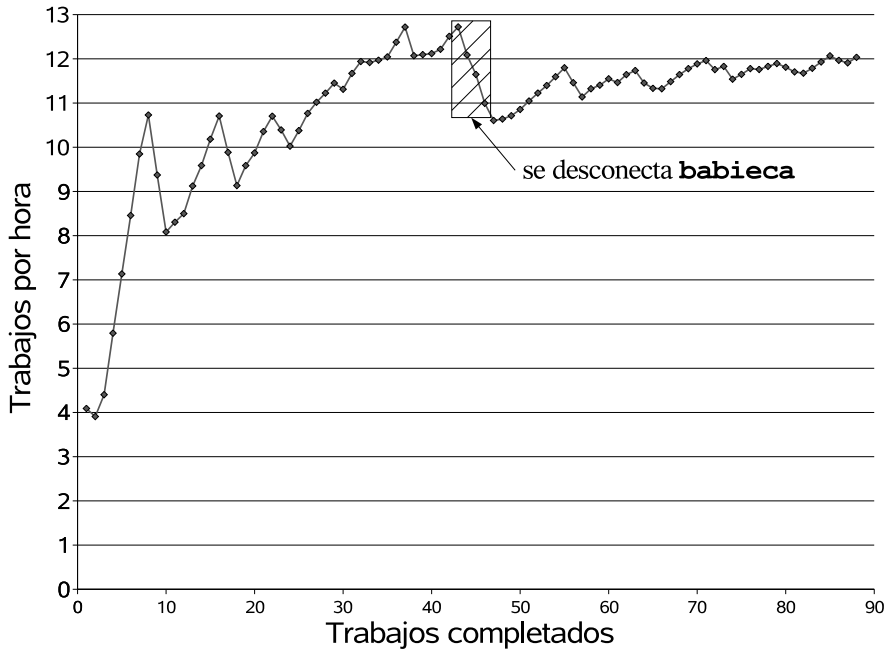

FIGURA 5.3: Productividad cuando babieca estuvo temporalmente desconectada.

terminó antes que el resto, debido al mejor rendimiento ofrecido por pegasus antes de su saturación.

5.5.2. Cambio Obligatorio en los Requisitos

En el siguiente experimento, hemos aplicado el algoritmo de predicción de estructuras a cinco secuencias de la enzima *Triosa Fosfato Isomerasa*, que son considerablemente más largas que las usadas en el caso anterior (ver figuras 5.1 y 5.5), presentes en distintos organismos.

Como ya se mencionó en la sección 5.2, la aplicación se divide en dos fases diferentes. Primero, se realiza un análisis somero para obtener las 200 mejores proteínas candidatas y, entonces, se realiza otro análisis más exhaustivo para obtener las 20 mejores proteínas candidatas de las 200 obtenidas en la primera fase. Como el análisis de la segunda fase realiza un alineamiento de secuencias más preciso y la secuencia objetivo es bastante larga, éste necesita más memoria que el análisis de la primera fase. Por tanto, la aplicación cambia sus requisitos de recurso antes de comenzar la segunda fase para asegurarse de que tiene suficiente memoria (512MB). Esto se realiza modificando el fichero dinámico que contiene la expresión de requisitos del trabajo (ver sección 3.5.2). El único recurso que cumple los

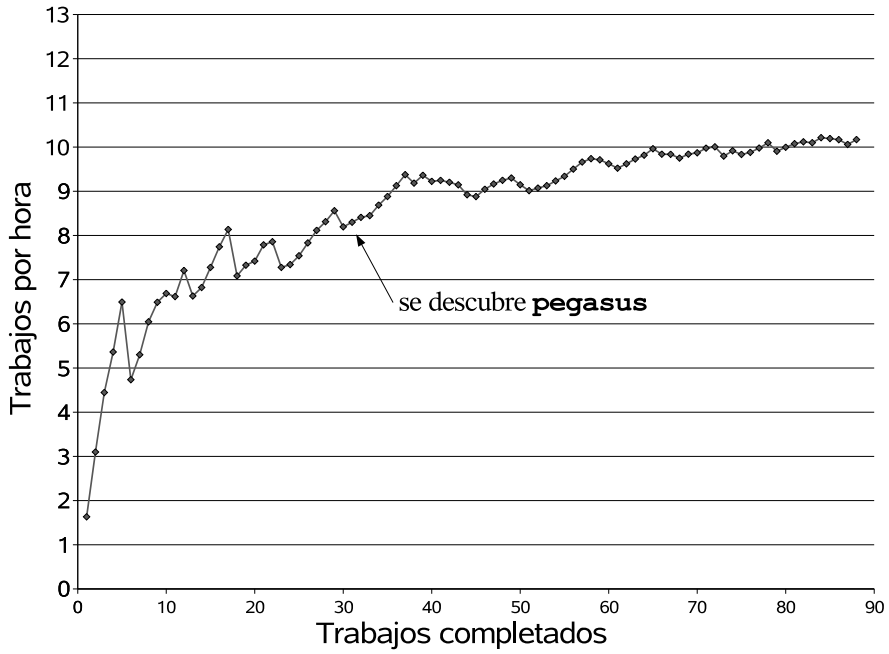

FIGURA 5.4: Productividad cuando pegasus fue descubierto a mitad del experimento.

requisitos de la segunda fase es pegasus.

La figura 5.7 muestra el perfil de ejecución en esta situación. El trabajo 0 comienza a ejecutarse en pegasus, mientras que el resto lo hace en babieca. Cuando el trabajo 0 completa su ejecución, Grid*W*ay detecta que pegasus se ha quedado libre y migra allí el trabajo 1, ya que presenta un rango mayor (es un caso de migración oportunista, como se vio en el capítulo 4). Después de eso, los trabajos del 2 al 4 solicitan una auto-migración ya que han cambiado sus requisitos para iniciar la segunda fase del análisis de proteínas y su recurso actual, babieca, no los cumple. Los trabajos 0 y 1 también cambiaron sus requisitos antes de iniciar la segunda fase, sin embargo, su recurso de ejecución en ese momento, pegasus, sí los cumplía, por lo que pudieron continuar con su ejecución si solicitar una auto-migración. Como pegasus estaba ocupado con el trabajo 1, los trabajos del 2 al 4 tuvieron que esperar a que estuviera disponible. Esos trabajos fueron enviados consecutivamente a pegasus (ver figura 5.7) para completar la segunda fase del análisis.

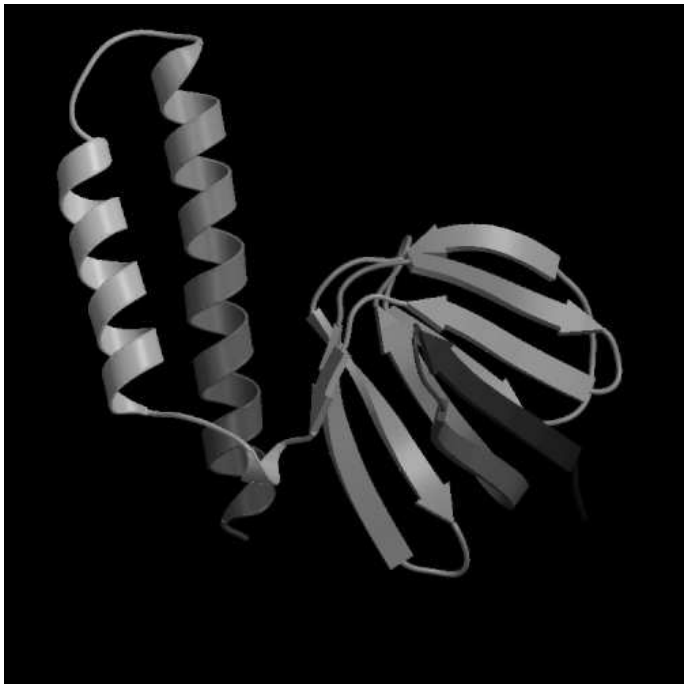

FIGURA 5.5: Estructura de la cadena épsilon de la enzima *ATP sintasa*.

5.6. Conclusiones

En este capítulo hemos mostrado una manera efectiva de proporcionar planificación y ejecución adaptativa en Grids en un campo con gran demanda de recursos computacionales y de almacenamiento como es la Bioinformática. Una de las ventajas de la herramienta Grid*W*ay es que no requiere necesariamente cambios en el código fuente de las aplicaciones. No obstante, con unos cambios mínimos, e incluso deseables en determinados marcos de trabajo, las aplicaciones pueden beneficiarse de las características de auto-adaptación que proporciona la herramienta Grid*W*ay.

En el ámbito de la aplicación, estos resultados son prometedores, ya que muestran el potencial del Grid para el estudio de grandes números de secuencias de proteínas, y sugieren la posible aplicación de estos métodos al conjunto completo de proteínas existente en un genoma microbiano. Esto último todavía requiere ciertas mejoras en el algoritmo de predicción de estructuras.

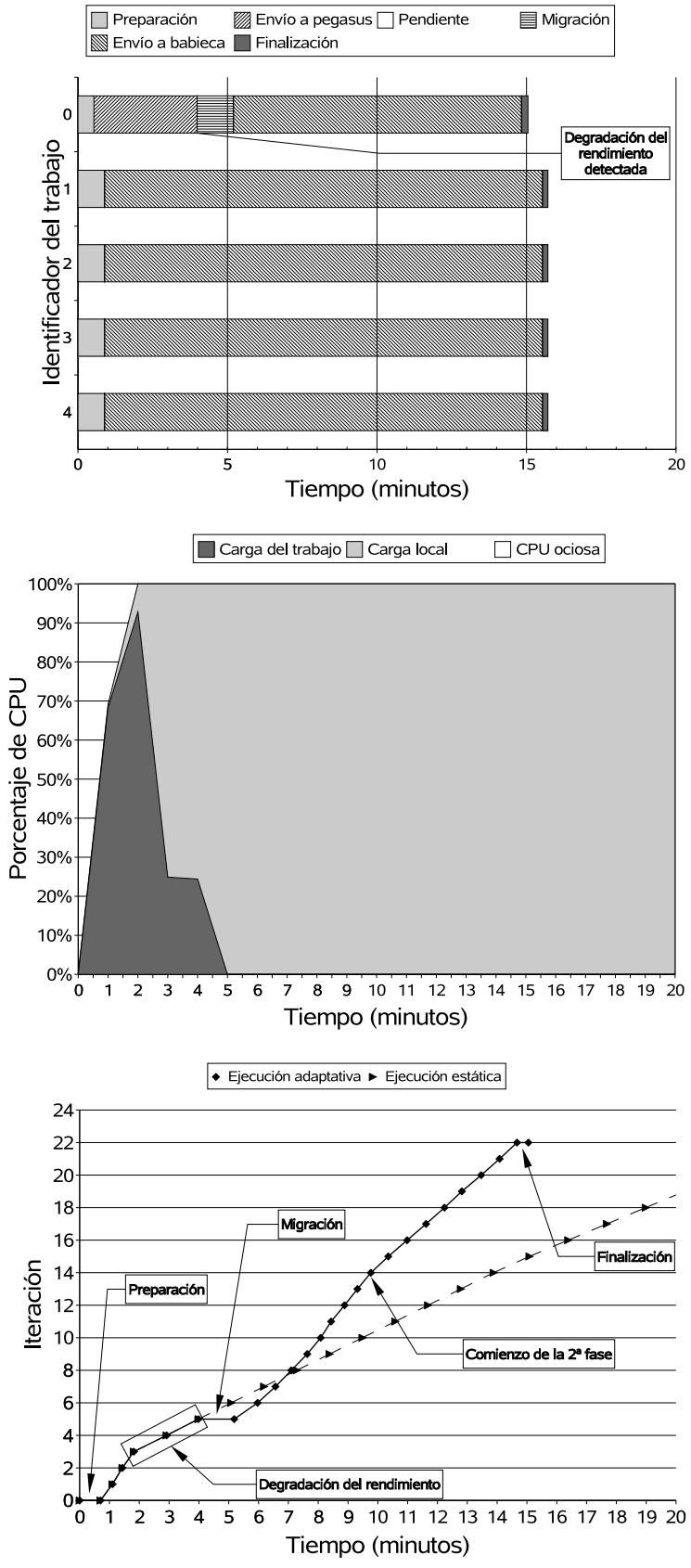

FIGURA 5.6: Perfil de ejecución (arriba), carga en pegasus (medio) y progreso de la tarea 0 (abajo) cuando se detecta una degradación del rendimiento.

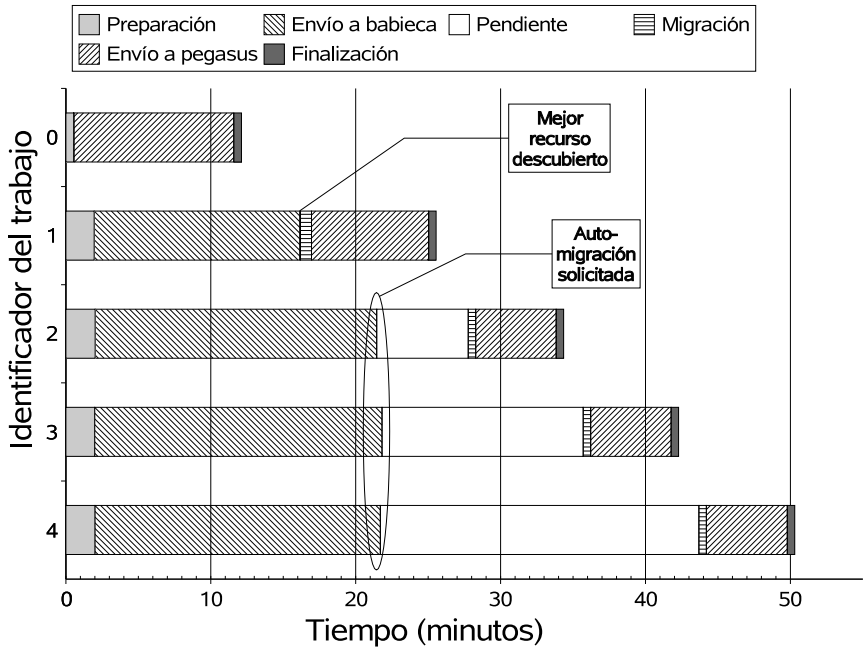

FIGURA 5.7: Perfil de ejecución cuando ocurre un cambio obligatorio en los requisitos.

6. Aplicación a la Evaluación del Rendimiento en Grids

Para evaluar la funcionalidad, fiabilidad y rendimiento proporcionadas por un entorno Grid, ajustar su configuración e incluso comparar cuantitativamente implementaciones alternativas, debe definirse un conjunto apropiado de métricas y programas de prueba (*benchmarks*). En este capítulo, se evalúa un banco de pruebas basado en Globus usando los *benchmarks* de NAS (*NASA Advanced Supercomputing division*) para el Grid (*NAS Grid Benchmarks*, NGB) junto con una serie de métricas propuestas. La especificación de lápiz y papel de este conjunto de *benchmarks* para Grid computacionales ha sido implementada usando el API de aplicaciones para gestión de recursos distribuidos (*Distributed Resource Management Application API*, DRMAA), soportado por la herramienta Grid*W*ay [120, 121]. Los resultados experimentales muestran que los NGB pueden ser una herramienta muy valiosa, primero, para comprobar la funcionalidad de un entorno Grid, y segundo, para analizar su rendimiento con el fin de ajustar sus componentes.

6.1. Introducción

El uso eficiente de las actuales infraestructuras Grid emergentes sólo puede conseguirse mediante la definición de un conjunto apropiado de métricas de rendimiento y una familia de *benchmarks* capaz de capturar las capacidades de los Grids para realizar computaciones distribuidas. El establecimiento de tal conjunto de métricas es una tarea difícil debido a que los Grid son entornos dinámicos y heterogéneos, lo que produce una pobre repetibilidad de las medidas. Es más, esas métricas deberían reflejar la interacción entre las distintas capas que forman un entorno Grid.

El grupo de investigación sobre evaluación (*benchmarking*) del Grid (*Grid Benchmarking Research Group*, GBRG) [122] propone crear un conjunto de *benchmarks* representativos para el Grid [123], que plasme escenarios de uso desafiantes con especial énfasis en el uso de grandes cantidades de datos. Estos *benchmarks* ayudarían a comprender cómo las aplicaciones reales usan el Grid. Servirían para validar el *middleware* de Grid y estimar

la calidad de servicio alcanzada, por lo que podrían guiar los procesos de desarrollo de aplicaciones y diseño de bancos de pruebas.

La suite NGB [124] ha sido la primera especificación de *benchmarks* para el Grid, la cual define un conjunto de grafos de flujo de datos que representan aplicaciones ejecutadas típicamente en el Grid. La especificación NGB sugiere el uso del tiempo de respuesta por trabajo como métrica cuantitativa básica de rendimiento. Sin embargo, otras métricas como el uso de recursos o los tiempos de transferencia de datos entre tareas se identifican como útiles para propósitos de diagnóstico. Otras métricas cualitativas, como seguridad y tolerancia a fallos, son aludidas como cruciales para conseguir un Grid satisfactorio [125].

El objetivo de este capítulo es, primero, proponer un conjunto de métricas con el objetivo de evaluar las capacidades de un entorno Grid, y segundo, mostrar cómo los NGB pueden ser una valiosa herramienta para probar la funcionalidad de un entorno Grid y para analizar su rendimiento a fin de ajustar sus componentes. Además, se demostrará que un banco de pruebas Grid basado en Globus [23] y Grid*W*ay proporciona la funcionalidad necesaria para ejecutar los NGB. La implementación de los NGB se ha realizado usando el estándar DRMAA [64], siendo Grid*W*ay una de las primeras herramientas en adoptar este estándar.

Las capacidades del Grid se suelen proporcionar mediante tres capas que forman típicamente un Grid computacional: servicios locales (nodos de cálculo, sistemas operativos, planificadores locales...), *middleware* de Grid (envío de trabajos, servicios de información, gestión de datos...) y herramientas de alto nivel (intermediarios de recursos, metaplanificadores, portales de computación...). Siguiendo esta estructura en capas, primero se describe una visión acerca de la evaluación del Grid y se presentan las métricas de rendimiento usadas en este trabajo. Después, se detallan las principales características de la suite NGB y se describe el estándar DRMAA. Finalmente, se discute cómo NGB puede ser una valiosa herramienta para los desarrolladores del Grid, permitiendo ajustar algunos parámetros de la ejecución, desde estrategias de planificación a parámetros del *middleware*.

6.2. Evaluación de un Entorno Grid

Dada la complejidad del proceso de desarrollo y ejecución de aplicaciones en el Grid, la funcionalidad debe ser considerada como una métrica valiosa, y los *benchmarks* del Grid deben reflejar la habilidad del entorno para ejecutar aplicaciones distribuidas de manera desatendida. La suite NGB entra dentro de esta categoría, y la capacidad para ejecutarla

constituye un *benchmark* apropiado para probar la funcionalidad del entorno.

Por otro lado, los trabajos del *benchmark* deben ser expresados usando interfaces estándar de alto nivel. Como ocurrió con otros estándares como MPI (*Message Passing Interface*) para paso de mensajes o OpenMP (*Open Multi-Processing*) para multiproceso, es previsible que DRMAA sea progresivamente adoptado por la mayoría de sistemas de gestión de recursos distribuidos (*Distributed Resource Management System*, DRMS).

Los *benchmarks* para el Grid deben proporcionar herramientas y métricas para medir y ajustar el rendimiento del Grid. Esto puede ser de gran ayuda para comparar cuantitativamente implementaciones alternativas o para ajustar una configuración básica. En este caso, el tiempo de respuesta (T_{wall}) es la métrica más importante para hacer la evaluación de un Grid, como se indica en la especificación de los NGB [124]. Sin embargo, otras métricas como el uso medio de recursos, los tiempos de ejecución o el tiempo de sobrecarga del Grid son apropiadas para propósitos de ajuste cuando es posible repetir las pruebas. En la sección 6.5, serán consideradas las siguientes métricas junto con el tiempo de respuesta por trabajo:

- Tiempo de reacción (T_r): Tiempo entre el envío de un trabajo y el comienzo de la fase de transferencia de ficheros de entrada en el servidor de ejecución. El tiempo medio de reacción proporciona información acerca de la sobrecarga inducida por el planificador y por el *middleware* del Grid, y puede usarse para ajustar algunos parámetros de sus componentes.

- Tiempos de transferencia (T_{xfr}) y ejecución (T_{exe}): Estas métricas son útiles para evaluar el impacto de las estrategias de movimiento de datos (reutilización de ficheros, selección y diseminación de réplicas...), el rendimiento de un recurso individual o la influencia de la red de interconexión. La desviación estándar en la media de estas métricas se puede usar como un indicador del dinamismo y heterogeneidad del entorno. Los resultados NGB presentados en la sección 6.5 también incluyen los tiempos de transferencia de datos entre tareas (aristas del grafo) y los tiempos de ejecución de cada tarea (nodos del grafo).

- Uso medio de recursos (U), definido como:

$$U = \frac{1}{n} \cdot \frac{\sum_{i=1}^{n} T_{exe}^{i}}{T_{wall}},$$

donde n es el número de recursos del Grid utilizados durante la ejecución del *bench-*

mark, T^i_{exe} es el tiempo de ejecución en cada recurso y T_{wall} es el tiempo de respuesta del *benchmark*. El uso medio de los recursos puede ser útil para determinar la eficiencia del algoritmo de planificación, y su desviación estándar puede usarse como medida del balanceo de la carga.

- Productividad (P): Aunque un *benchmark* que mida la productividad debe ser intrusivo, podría ser importante analizar el comportamiento del entorno durante la ejecución fuera de línea de grandes aplicaciones de computación de alta productividad (HTC). Como es usual, la productividad se define como el número de tareas o instancias de los NGB completadas por unidad de tiempo.

Los Grids son además difíciles de aprovechar eficientemente debido a su naturaleza heterogénea y a las condiciones cambiantes e impredecibles que presentan. La planificación y ejecución adaptativa son algunas de las técnicas propuestas en la literatura [65, 126, 82, 68, 71, 70] para conseguir un grado razonable de rendimiento y tolerancia a fallos en la aplicación. Por tanto, los *benchmarks* apropiados para el Grid deben también ayudar a determinar la fiabilidad y adaptación dinámica del entorno Grid. Esas métricas son difíciles de cuantificar. En este contexto, los *benchmarks* para el Grid deben proporcionar los comportamientos esperados o tasas de fallos cuando un nodo se sobrecarga o es desconectado a fin de medir su fiabilidad. Es interesante hacer notar que la actual especificación de los NGB no incluye ningún procedimiento para reproducir los estos escenarios.

6.3. Los NAS Grid Benchmarks

Los *NAS Grid Benchmarks* (NGB) [124, 125] se presentan como un grafo de flujo de datos que encapsula una instancia de un código de los *NAS Parallel Benchmarks* (NPB) [127] en cada nodo del grafo, que se *comunica* con otros nodos mediante el envío y recepción de datos de inicialización. Los NGB se centran en Grids computacionales, que se usan principalmente para ejecutar trabajos intensivos en cálculo que potencialmente procesan grandes conjuntos de datos.

Cada *benchmark* consiste en la ejecución de varios códigos NPB que simbolizan cálculos científicos (resolutor de flujo SP, BT y LU), post-proceso (suavizador de datos MG) y visualización (analizador espectral FT). Como NPB, NGB especifica distintas clases o tamaños de problema, en términos de número de tareas, tamaño de malla computacional o iteraciones.

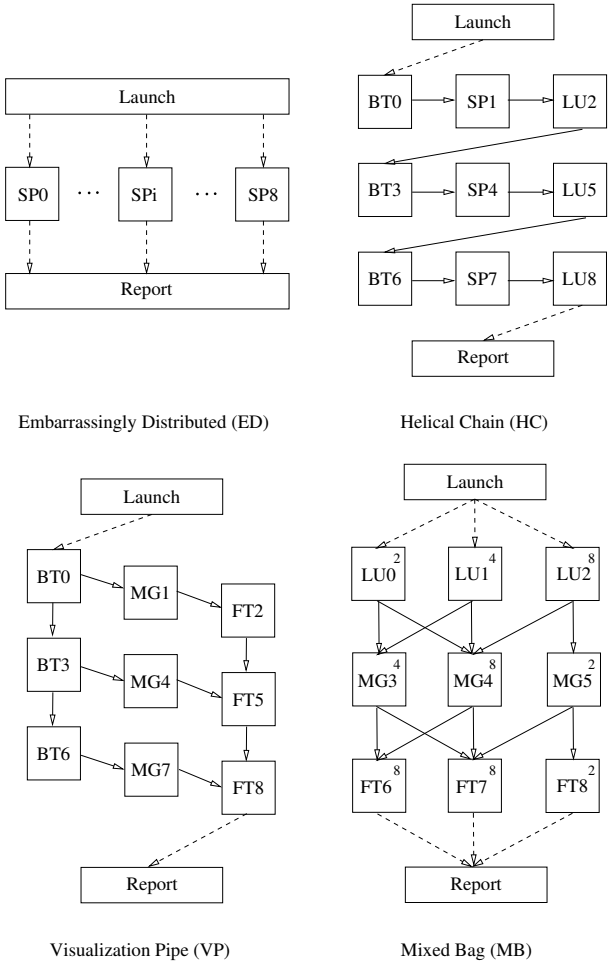

FIGURA 6.1: Las cuatro familias de los NGB.

La figura 6.1 muestra las cuatro familias definidas en los NGB:

- *Embarrassingly Distributed* (ED) representa aplicaciones de barrido de parámetros (PSA, ver sección 2.4), que constituyen múltiples ejecuciones independientes del mismo programa, pero con diferentes parámetros de entrada.

- *Helical Chain* (HC) representa largas cadenas de procesos repetitivos, como ocurre al dividir largas simulaciones en series de tareas o al construir procesos compuestos como tuberías (*pipelines*) computacionales.

- *Visualization Pipe* (VP) representa cadenas de procesos compuestos, como los que aparecen al visualizar las soluciones de un fluido conforme la simulación progresa.

- *Mixed Bag* (MB) de nuevo consiste en una secuencia de cálculo, post-proceso y visua-

lización, pero ahora el énfasis se pone en introducir asimetría. El tamaño y cantidad de ficheros recibidos por cada nodo del *benchmark* es muy variable, así como el trabajo computacional que tiene asignado, ya que también varía el número de iteraciones a realizar (mostrado, de manera relativa, en la esquina superior derecha de cada nodo).

6.4. El Estándar DRMAA

Uno de los aspectos más importantes de la computación Grid es su potencial habilidad para ejecutar trabajos distribuidos que se comunican. Este tipo de computaciones distribuidas pueden ser expresadas fácilmente con el estándar DRMAA [120, 121]. La especificación DRMAA [64] constituye un interfaz homogéneo y portable a diferentes DRMS para gestionar el envío, monitorización y control de trabajos, y para la recuperación del estado de los trabajos finalizados.

Aunque DRMAA podría servir de interfaz con DRMS a diferentes niveles, por ejemplo, al nivel de *intranet* con *Sun Grid Engine* (SGE) o Condor, en este contexto sólo se considera su aplicación a nivel de Grid. De esta forma, el DRMS (Grid*W*ay en nuestro caso) interactúa con el gestor local de trabajos (PBS, Condor, SGE...) a través del *middleware* de Grid (Globus). Este esquema de desarrollo y ejecución con DRMAA, Grid*W*ay y Globus se representa en la figura 6.2.

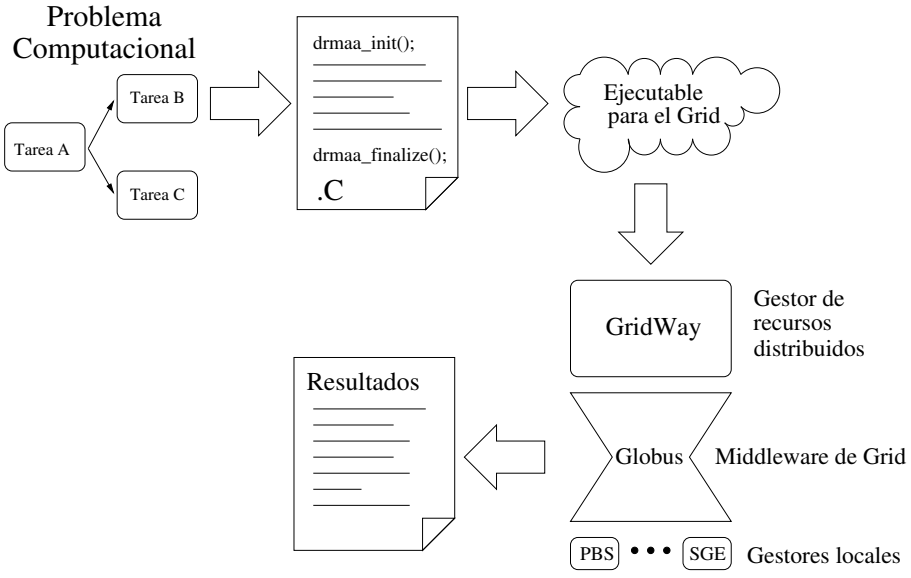

FIGURA 6.2: Ciclo de desarrollo y ejecución usando el interfaz DRMAA, el DRMS Grid*W*ay y el *middleware* Globus.

En la siguiente lista se describen las rutinas del interfaz DRMAA implementadas en la herramienta Grid*W*ay:

- Rutinas de inicialización y finalización: `drmaa_init` y `drmaa_exit`. Estas rutinas inician y terminan la sesión DRMAA.

- Rutinas de definición y manipulación de trabajos: `drmaa_allocate_job_template`, `drmaa_delete_job_template` y `drmaa_set_attribute`. Estas rutinas permiten la manipulación de las entidades de definición –o plantillas (*templates*)– de trabajos, para establecer parámetros tales como el fichero ejecutable, sus argumentos o los flujos de salida estándar.

- Rutinas de envío de trabajos: `drmaa_run_job` y `drmaa_run_bulk_jobs`. Estas rutinas se usan para enviar trabajos de manera individual o en conjunto (*bulk jobs*).

- Rutinas de control y monitorización de trabajos: `drmaa_control`, `drmaa_job_ps`, `drmaa_wait` y `drmaa_synchronize`. Estas rutinas se utilizan para controlar (matar, suspender, reanudar...) y sincronizar trabajos así como para monitorizar su estado.

Grid*W*ay tiene soporte nativo para conjuntos de trabajos (*bulk jobs*) o, lo que es lo mismo, trabajos en *array*. En la herramienta Grid*W*ay, un trabajo en *array* se define como un grupo de n trabajos que comparten la misma plantilla de definición de trabajos (*job template*), pero que tienen un identificador de trabajo ($TASK\_ID$) diferente, de 0 a $n-1$, que puede usarse para controlar el comportamiento de cada trabajo individual.

El interfaz DRMAA (ver [64] para una descripción detallada del API para lenguaje C) incluye más rutinas en algunas de las categorías anteriores así como rutinas auxiliares que proporcionan representación textual de los errores, no implementadas en la versión actual. Todas las funciones implementadas en la herramienta Grid*W*ay son seguras entre diferentes hilos de ejecución (*thread-safe*).

6.5. Ejecución de NGB con Grid*W*ay Usando DRMAA

En las siguientes subsecciones, analizaremos la habilidad y rendimiento de la herramienta Grid*W*ay para ejecutar la suite NGB, cuyo objetivo no es medir el rendimiento del *hardware* de Grid subyacente, sino la funcionalidad, fiabilidad y rendimiento del entorno Grid completo. No obstante, una clara comprensión de la configuración *hardware* de los

recursos del Grid ayudará al análisis de los resultados subsecuentes. Las principales características del banco de pruebas utilizado para ejecutar los NGB se resumen en la tabla 6.1 y también en la sección A.2 de los apéndices.

TABLA 6.1: Características de las máquinas del banco de pruebas experimental UCM-CAB.

Nombre	VO	Procesadores	SO	Velocidad	Memoria	DRMS
hydrus	DACYA	1×Pentium 4	Linux 2.4	2,5GHz	512MB	fork
cygnus		1×Pentium 4	Linux 2.4	2,5GHz	512MB	fork
pegasus		1×Pentium 4	Linux 2.4	2,4GHz	1GB	fork
aquila		1×Pentium III	Linux 2.4	700MHz	128MB	fork
cepheus		1×Pentium III	Linux 2.4	600MHz	256MB	fork
babieca	CAB	30×Alpha EV6	Linux 2.2	450MHz	256MB	PBS

Los problemas NGB considerados en esta sección pertenecen a las clases W y A, como se define en [124]. Los *benchmarks* de clase A son ejecutados sólo en aquellos recursos con al menos 300MB de memoria principal (hydrus, cygnus y pegasus) para impedir que se active el intercambio de memoria (*memory swapping*). Esto se implementa imponiendo este requisito en la fase de descubrimiento de recursos (ver sección 3.4.1). En los experimentos siguientes, se usará aquila como cliente, por lo que almacenará el ejecutable y los ficheros de entrada y recibirá los ficheros de salida.

6.5.1. Embarrassingly Distributed (ED)

El *benchmark* ED consiste en la ejecución de una serie de tareas independientes. Cada una de ellas consiste en la ejecución del resolutor de flujo SP [127] con un parámetro de inicialización distinto para el campo de flujos. Las aplicaciones de barrido de parámetros (PSA) como ésta pueden ser expresadas directamente con el interfaz DRMAA como un conjunto de trabajos (*bulk jobs*). El código DRMAA para este experimento se muestra en la figura 6.3.

Como ya se discutió en la sección 2.4, a pesar de su estructura relativamente simple, la ejecución eficiente de PSAs en Grids computacionales conlleva problemas desafiantes. En general, la ejecución eficiente de este tipo de aplicaciones debe combinar los siguientes elementos: **planificación adaptativa**, para reflejar las características dinámicas del Grid; **ejecución adaptativa**, para migrar trabajos en ejecución a "mejores" recursos y proporcionar tolerancia a fallos; y **reutilización de ficheros** comunes entre tareas, para reducir

```
// Inicialización
jt= SP;

drmaa_init(contact, err);

// Envío de todos los trabajos simultáneamente
drmaa_run_bulk_jobs(job_ids, jt, 0, num_jobs, 1, err);
drmaa_synchronize(job_ids, timeout, 1, err);

drmaa_exit(err);
```

FIGURA 6.3: Implementación del *benchmark* ED usando DRMAA.

la sobrecarga por transferencia de ficheros.

La figura 6.4 muestra los resultados de la ejecución del *benchmark* ED. Cada uno de los recursos apropiados encontrados en el Grid ejecuta consecutivamente tres tareas SP, que resultan en un tiempo de respuesta de 18.88 minutos, lo que supone una productividad de 0.5 trabajos por minuto. El tiempo medio de ejecución para cada tarea es de 4.25 minutos y el de transferencia, de 18 segundos. Los tiempos de transferencia con pegasus son notablemente mayores (casi tres veces) que los de hydrus o cygnus, a pesar de que todos estos recursos están en la misma red local. Esto es debido a la versión 2.2 de Globus instalada en pegasus, que tiene un periodo de muestreo de 30 segundos para el *job manager* de GRAM, mientras que la versión 2.4 instalada en el resto de máquinas tiene un periodo de muestreo de 10 segundos.

En este caso, el uso medio de recursos durante la ejecución del *benchmark* ED fue de un 67.61 %, el cual podría mejorarse reduciendo el hueco entre tareas enviadas consecutivamente al mismo servidor, reduciendo así el tiempo de reacción. Esta demora es debida principalmente a la frecuencia de actualización del servidor GRIS de MDS y al módulo *resource selector* de Grid*W*ay. Como se vio en la sección 3.4.1, durante el proceso de descubrimiento de recursos de Grid*W*ay, se consulta el servidor GIIS de MDS para encontrar recursos disponibles que cumplan los requisitos del trabajo, y después se consulta el servidor GRIS para monitorizar el estado de los recursos candidatos. La disponibilidad de un recurso se mide en términos de su carga de trabajo o del número de huecos (*slots*) en el sistema de colas, por lo tanto, Grid*W*ay espera a enviar el siguiente trabajo hasta que la carga de trabajo publicada en el GRIS del recurso desciende por debajo de un límite predeterminado o aumenta el número de huecos en el sistema de colas.

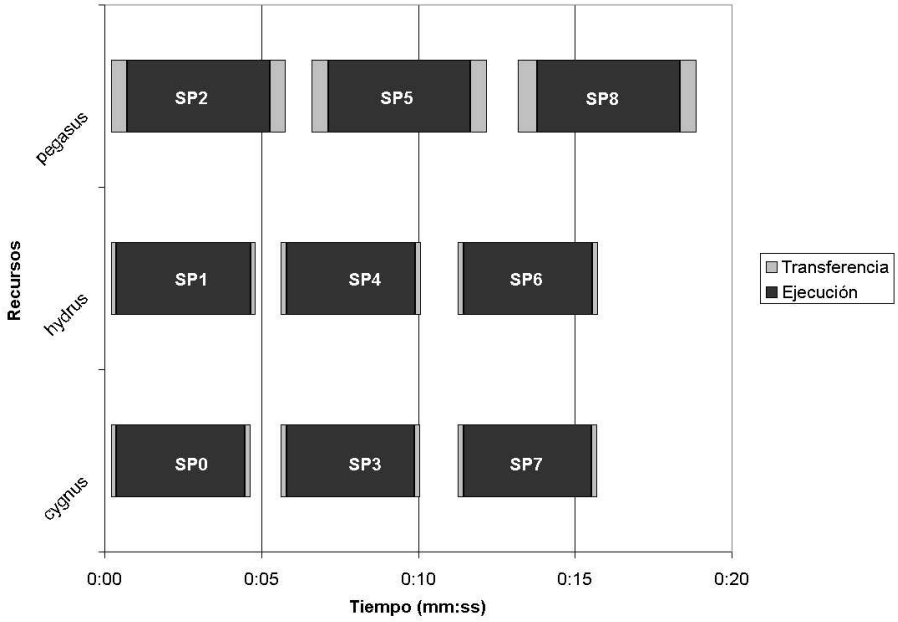

FIGURA 6.4: Perfil de ejecución del *benchmark* ED de clase A.

6.5.2. Helical Chain (HC)

El *benchmark* HC consiste en una secuencia de trabajos que modela una tubería (*pipeline*) computacional o largas simulaciones que se dividen en distintas tareas. Cada trabajo de la secuencia utiliza la solución calculada de su predecesor para inicializarse. Considerando estas dependencias, cada trabajo de la cadena puede ser planificado por Grid*W*ay una vez que el trabajo anterior haya finalizado. El código DRMAA para este caso se muestra en la figura 6.5.

Los resultados del *benchmark* HC (clase A) para esta estrategia de planificación se muestran en la figura 6.6. El tiempo de respuesta es de 17,56 minutos, con un uso medio de recursos del 20,21 %. El retraso de MDS en publicar la información de estado de los recursos se traduce en una planificación oscilante de los trabajos. Esta planificación reduce claramente el rendimiento obtenido comparado con el tiempo de respuesta óptimo[1] de 6,3 minutos.

No obstante, este tipo de aplicaciones puede ser ejecutado a través de la herramienta Grid*W*ay como un solo trabajo. Los ficheros de salida de cada tarea de la cadena pueden ser gestionados por la herramienta como ficheros de reinicio. De esta forma, la aplicación puede aprovecharse de las capacidades proporcionadas por la herramienta Grid*W*ay para

[1]Perteneciente a la ejecución secuencial en la máquina más rápida del Grid.

```
// Inicialización
jobs[0].jt = BT;
jobs[1].jt = SP;
jobs[2].jt = LU;
jobs[3].jt = BT;
jobs[4].jt = SP;
jobs[5].jt = LU;
jobs[6].jt = BT;
jobs[7].jt = SP;
jobs[8].jt = LU;

drmaa_init(contact, err);

// Envío de todos los trabajos consecutivamente
for (i = 0; i<9; i++) {
  drmaa_run_job(job_id, jobs[i].jt, err);
  drmaa_wait(job_id, &stat, timeout, rusage, err);
}

drmaa_exit(err);
```

FIGURA 6.5: Implementación del *benchmark* HC de clase A usando DRMAA.

la ejecución de aplicaciones auto-adaptativas (ver sección 1.4.2):

- La aplicación puede cambiar progresivamente sus requisitos de recurso dependiendo de la tarea de la cadena que se vaya a ejecutar. Por tanto, la aplicación no tiene que imponer el conjunto de requisitos más restrictivo en el inicio, ya que limitaría las oportunidades de la aplicación para comenzar su ejecución
- La aplicación puede generar un perfil de rendimiento para proporcionar información de monitorización en términos de métricas intrínsecas a la aplicación (por ejemplo, el tiempo para realizar cada tarea de la cadena). Este perfil de rendimiento podría usarse para guiar la planificación del trabajo. Por tanto, la aplicación podría migrar a otra máquina cuando alguno de los recursos (memoria, espacio en disco, CPU libre...) escaseara.
- La aplicación puede ser migrada cuando se encuentra un recurso "mejor" en el Grid. En este caso, tanto el tiempo de finalización como los costes de transferencia de ficheros deben ser considerados para evaluar si la migración es provechosa, como se vio en el capítulo 4.

Cuando se envía el *benchmark* HC de clase A como un solo trabajo, el uso medio de recursos aumenta hasta el 91 %, ya que las nueve tareas de la misma cadena se planifican

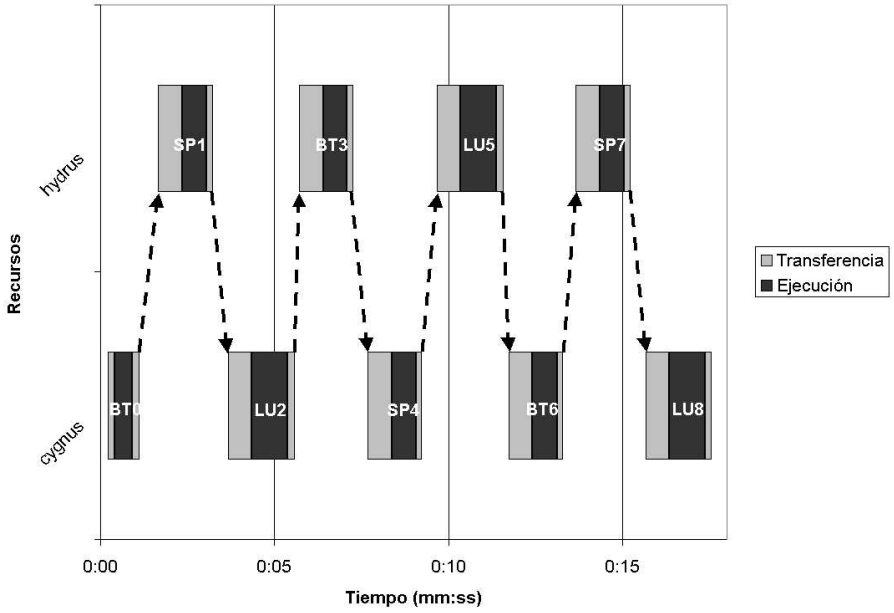

FIGURA 6.6: Perfil de ejecución del *benchmark* HC de clase A.

al mismo recurso (cygnus). En este caso, el tiempo de respuesta es de 7 minutos y el tiempo medio de ejecución se reduce a 6,4 minutos. Esto supone un decremento en el tiempo de respuesta del 60 % comparado con la primera estrategia de planificación, y un incremento de sólo el 11 % comparado con el caso óptimo.

La figura 6.7 muestra los resultados para cuatro instancias del *benchmark* HC de clase W, cada una ejecutada como un solo trabajo. Como consecuencia, el uso medio de recursos es de un 63,27 % en el caso del trabajo 0, ya que la clase W presenta una peor tasa de cálculo/comunicación que la clase A. En este caso, el tiempo medio de respuesta para las cuatro instancias fue de 2,4 minutos.

Consideremos ahora con más detalle la ejecución de la instancia etiquetada como trabajo 3 en la figura 6.7, que es asignado inicialmente a cepheus. Tras un minuto y 38 segundos, la instancia ejecutándose en cygnus (trabajo 0) completa su ejecución. Cuando Grid*Way* detecta que este recurso con mayor rango está disponible, inicia una migración del trabajo. El proceso de migración (incluyendo la transferencia del fichero de salida BT3 desde cepheus a cygnus) representa un 10,91 % del tiempo total de ejecución del trabajo. Sin embargo, el tiempo de respuesta del trabajo 3 es un 22,63 % menor cuando es migrado, esto es, ejecutado adaptativamente, que cuando es ejecutado estáticamente. La figura 6.8 muestra el perfil de ejecución del trabajo 3 (sin incluir los tiempos de transferencia) cuando es

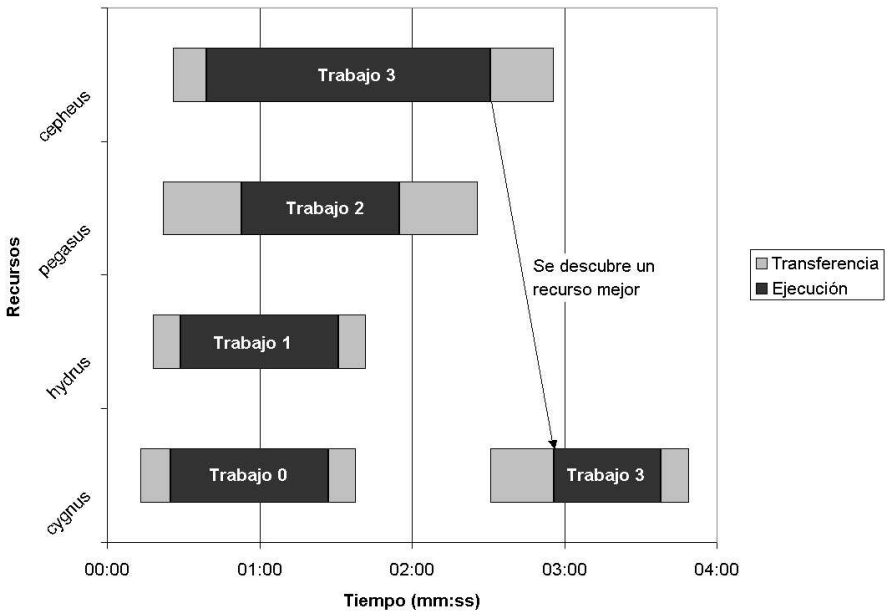

FIGURA 6.7: Perfil de ejecución de cuatro instancias del *benchmark* HC.W implementado como una aplicación auto-adaptativa.

ejecutado estática y adaptativamente.

6.5.3. Visualization Pipe (VP) y Mixed Bag (MB)

Los *benchmarks* VP y HC son combinaciones de los *benchmarks* ED (completamente paralelo) y HC (completamente secuencial) descritos anteriormente. Exhiben algo de paralelismo, que debe ser explotado, pero limitado por las dependencias entre los trabajos. En el caso de VP, el paralelismo está incluso más limitado debido al bajo ancho del canal (sólo 3, para todas las clases) y los largos tiempo de llenado y vaciado del mismo (por ejemplo, con clase A sólo se consigue el paralelismo completo una vez).

Este tipo de aplicaciones podría serializarse y ejecutarse de forma adaptativa como en el caso anterior. Sin embargo, lo más apropiado sería implementarlas como una aplicación de flujo de trabajo (*workflow*) para explotar el paralelismo que exhiben.

Dado que GridWay no soporta directamente la ejecución de flujos de trabajo, hemos desarrollado un motor de flujos de trabajo (*workflow engine*) aprovechando el interfaz de programación DRMAA (ver figura 6.9). El algoritmo utilizado sigue un enfoque voraz (*greedy*), aunque se podrían utilizar distintas políticas para priorizar los trabajos enviados en cada momento, por ejemplo, aquellos trabajos con un conjunto de requisitos más restric-

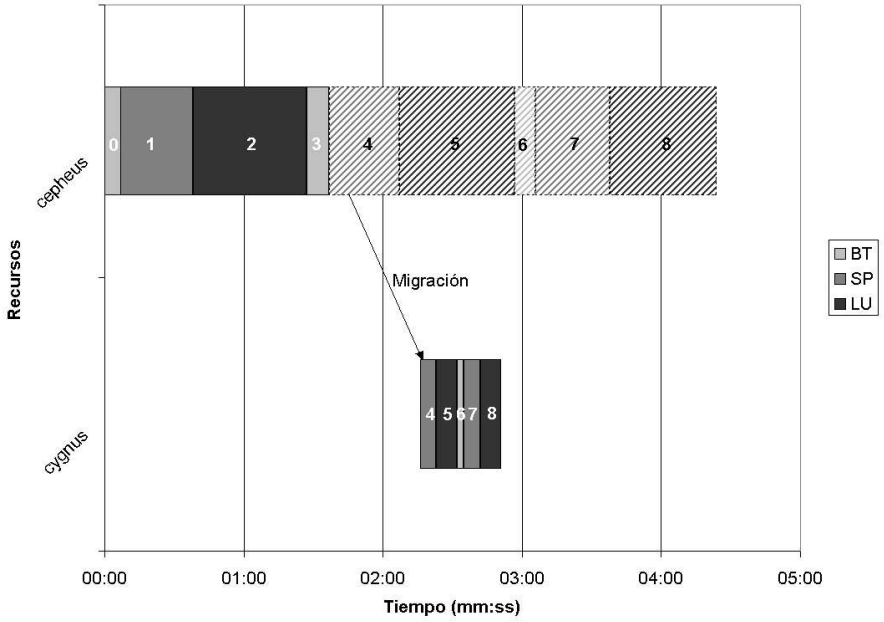

FIGURA 6.8: Perfil de ejecución detallado del trabajo 3 con el *benchmark* HC.W implementado como una aplicación auto-adaptativa.

tivo, con mayor trabajo computacional o con un mayor número de trabajo que dependan de ellos. Existen otras iniciativas, como Pegasus [128], que intentan mapear el flujo de trabajo completo, definido en términos de componentes a nivel de aplicación, a un conjunto de recursos disponibles en el Grid.

La figura 6.10 muestra la inicialización del *workflow engine* para el *benchmark* VP de clase A, mientras que la figura 6.11 muestra los resultados obtenidos con este *benchmark*. Las líneas discontinuas representan dependencias entre los trabajos y las líneas más gruesas representan el camino crítico que determina el tiempo de respuesta del *benchmark* completo. En este caso, el tiempo de respuesta es de 21,68 minutos, con un uso medio de recursos del 35,25 %. La suma de los tiempos de ejecución y transferencia es de 22,93 y 8,1 minutos, respectivamente.

En cuanto al *benchmark* MB, la figura 6.12 muestra la inicialización del *workflow engine* para la clase A, mientras que la figura 6.13 muestra los resultados obtenidos. De nuevo, las líneas discontinuas representan dependencias entre los trabajos y las líneas más gruesas representan el camino crítico que determina el tiempo de respuesta del *benchmark* completo. En este caso, el tiempo de respuesta es de 16,8 minutos, con un uso medio de recursos del 45,7 %. La suma de los tiempos de ejecución y transferencia son de 23,03 y 9,7 minutos, respectivamente. Los experimentos con el *benchmark* MB se realizaron unas

```
drmaa_init(contact, err);

// Mientras queden trabajos por ejecutar...
while (there_are_jobs_left(jobs)) {

  // Enviar trabajos con dependencias resueltas
  for (i = 0; i<num_jobs; i++)
    if (is_job_ready(jobs, i))
      drmaa_run_job(jobs[i].id, jobs[i].jt, err);

  // Esperar a que termine cualquier trabajo
  job_id = "DRMAA_JOB_IDS_SESSION_ANY";
  drmaa_wait(job_id, &stat, timeout, rusage, err);
  set_job_done(jobs, job_id);
}

drmaa_exit(err);
```

FIGURA 6.9: Implementación del *workflow engine* usando DRMAA.

```
// Inicialización
num_jobs = 9;
jobs[0].jt = BT; jobs[0].dep = "";
jobs[1].jt = MG; jobs[1].dep = "0";
jobs[2].jt = FT; jobs[2].dep = "1";
jobs[3].jt = BT; jobs[3].dep = "0";
jobs[4].jt = MG; jobs[4].dep = "3";
jobs[5].jt = FT; jobs[5].dep = "2 4";
jobs[6].jt = BT; jobs[6].dep = "3";
jobs[7].jt = MG; jobs[7].dep = "6";
jobs[8].jt = FT; jobs[8].dep = "5 7";
```

FIGURA 6.10: Inicialización del *workflow engine* para el *benchmark* VP de clase A.

semanas después que con el resto de *benchmarks*, teniendo ya pegasus la versión 2.4 de Globus instalada, por eso el tiempo mínimo de transferencia es algo superior a 10 segundos en todas las máquinas.

En las figuras 6.11 y 6.13 se pueden observar las diferencias entre los *benchmarks* VP y MB. Ambos exhiben algo de paralelismo, como lo demuestra que la suma de los tiempos de ejecución es mayor que el tiempo de respuesta, que podría incrementarse si se aumentara el ancho del canal (limitado a tres trabajos) y se redujera la sobrecarga inducida por el Grid (planificación, transferencia de ficheros, envío de trabajos...). El paralelismo obtenido por el *benchmark* VP es muy pobre, debido a las fases de llenado y vaciado del canal, siendo la suma de los tiempos de ejecución sólo un 4,57 % mayor que el tiempo de respuesta.

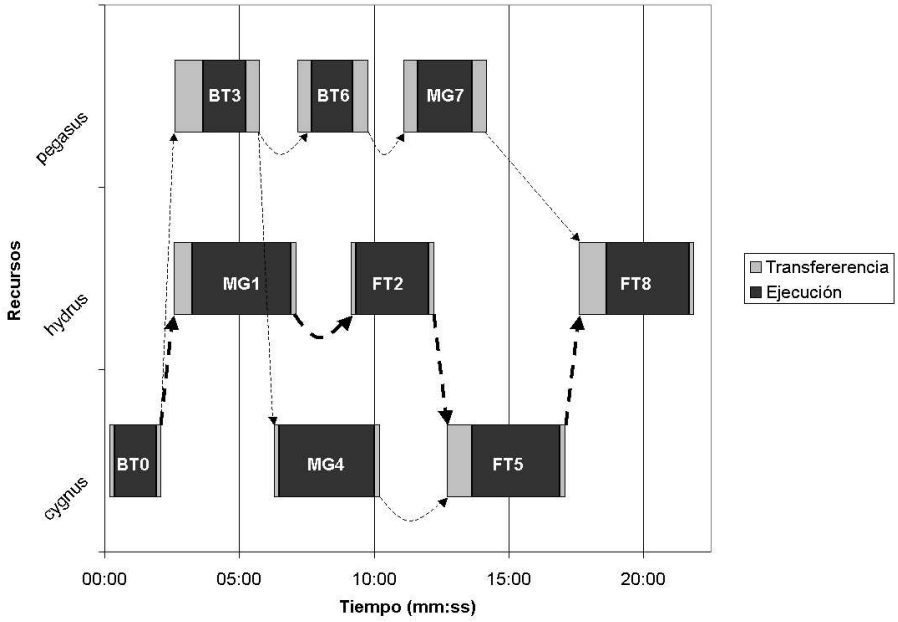

FIGURA 6.11: Perfil de ejecución del *benchmark* VP de clase A.

En cambio, el paralelismo alcanzado por el *benchmark* MB es considerable, teniendo una suma de los tiempos de ejecución un 27,06 % mayor que el tiempo de respuesta. De hecho, la suma de los tiempos de ejecución en ambos *benchmarks* son muy similares (22,93 para VP y 23,03 para MB), sin embargo, el tiempo de respuesta es mucho menor en el caso del *benchmark* MB (un 23,21 % menor que el del *benchmark* VP), debido a que exhibe un mayor grado de paralelismo que posibilita un mejor aprovechamiento de los recursos (un 34,93 % para VP, frente a un 45,7 % para MB).

6.6. Conclusiones

En este capítulo hemos analizado la funcionalidad, fiabilidad y rendimiento de un entorno Grid basado en la herramienta Grid*W*ay y el conjunto de herramientas de Globus. Este entorno ha sido evaluado por medio de la ejecución de los NGB y la definición de un conjunto apropiado de métricas de rendimiento. La funcionalidad del banco de pruebas ha sido demostrada mediante su habilidad para ejecutar la suite NGB usando el interfaz DRMAA. El uso de interfaces estándar permite la comparación entre distintas implementaciones Grid, dado que ni NGB ni DRMAA están ligados a ningún *middleware* de Grid específico.

```
// Inicialización
num_jobs = 9;
jobs[0].jt = LU; jobs[0].dep = "";
jobs[1].jt = LU; jobs[1].dep = "";
jobs[2].jt = LU; jobs[2].dep = "";
jobs[3].jt = MG; jobs[3].dep = "0 1";
jobs[4].jt = MG; jobs[4].dep = "0 1 2";
jobs[5].jt = MG; jobs[5].dep = "2";
jobs[6].jt = FT; jobs[6].dep = "3 4";
jobs[7].jt = FT; jobs[7].dep = "3 4 5";
jobs[8].jt = FT; jobs[8].dep = "5";
```

FIGURA 6.12: Inicialización del *workflow engine* para el *benchmark* MB de clase A.

Los resultados presentados en este capítulo muestran que la suite NGB es una valiosa herramienta para explorar el comportamiento y ajustar el rendimiento ofrecido por cada capa del entorno Grid. Por ejemplo, se ha mostrado la influencia de varios parámetros del conjunto de herramientas de Globus, como la frecuencia de actualización del GRIS y el tiempo de muestreo del *job manager* de GRAM; así como el impacto del planificador del Grid, en este caso Grid*Way*.

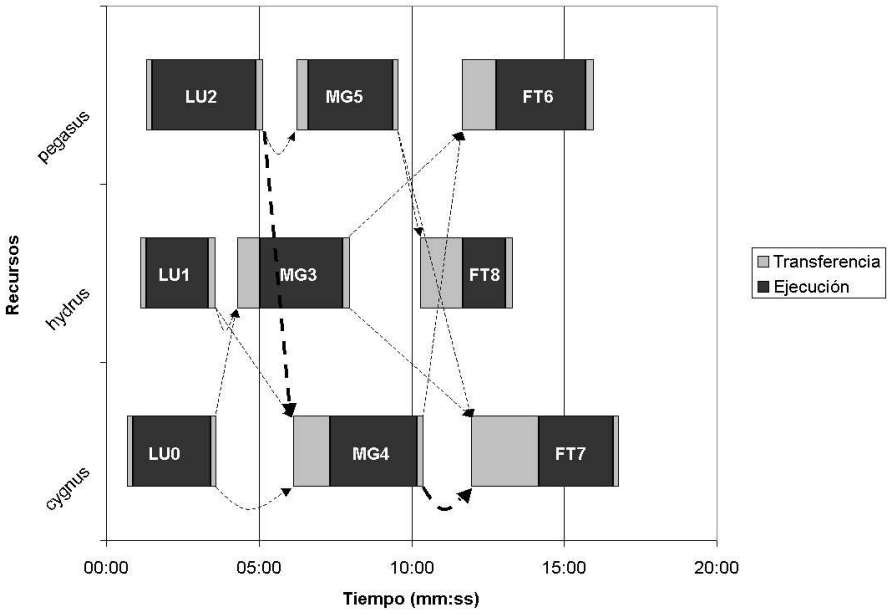

FIGURA 6.13: Perfil de ejecución del *benchmark* MB de clase A.

7. Conclusiones y Principales Aportaciones

El prototipo de investigación Grid*W*ay realiza todas las fases de planificación de trabajos contactando con los servicios Grid disponibles y a continuación vela por la ejecución correcta y eficiente de los trabajos sobre los recursos distribuidos. La herramienta implementa tolerancia a fallos de red o sistemas remotos, planificación de múltiples aplicaciones, monitorización por medio de métricas proporcionadas por la aplicación en ejecución y gestión de migración cuando los requisitos de la aplicación cambian, se detecta un fallo en la red o sistema remoto, se detecta una degradación de rendimiento o se descubre un recurso que ofrece mejores prestaciones.

Actualmente no existen herramientas de planificación con características semejantes. Gracias a esta herramienta se han podido abordar interesantes problemáticas relacionadas con la planificación de trabajos sobre entornos heterogéneos y dinámicos. A continuación describimos algunos resultados de la investigación realizada.

7.1. Planificación y Ejecución Adaptativa

La planificación adaptativa de trabajos es el primer paso para hacer frente a las características dinámicas del Grid. La herramienta Grid*W*ay realiza la planificación de trabajos evaluando periódicamente los recursos disponibles, sus características y las preferencias de los trabajos pendientes. Este componente es muy importante en la ejecución de aplicaciones de alta productividad (mayor número de trabajos que número de recursos), como por ejemplo aplicaciones paramétricas.

Además, con el fin de obtener un grado razonable de rendimiento y tolerancia a fallos, un trabajo debe ser capaz de migrar entre los recursos del Grid. La ejecución adaptativa del entorno Grid*W*ay gestiona la migración de trabajos debida tanto a eventos del propio Grid, como la variación del rendimiento, disponibilidad y coste de los recursos; como al cambio de perfil de requisitos y rango de la aplicación.

La investigación sobre técnicas de planificación y ejecución adaptativa en Grids com-

putacionales ha dado lugar a las siguientes publicaciones:

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *An Experimental Framework for Executing Applications in Dynamic Grid Environments*. Publicado por *NASA Scientific and Technical Information (STI) Program Office* en *NASA/CR-2002-211960 – ICASE Report No. 2002-43*, noviembre de 2002.

- Rubén S. Montero, Eduardo Huedo e Ignacio M. Llorente: *The Grid*W*ay Approach for Job Submission and Management on Grids*. En *iAstro Workshop on Distributed processing, transfer, retrieval, fusion and display of images and signals: High resolution and low resolution in data and information Grids*, Granada, febrero de 2003.

- Rubén S. Montero, Eduardo Huedo e Ignacio M. Llorente: *Experiences about Job Migration on a Dynamic Grid Environment*. En *5th Parallel Computing Conference (ParCo 2003)*, Dresde (Alemania), septiembre de 2003. Publicado por *Elsevier Science* en *Advances in Parallel Computing*, vol. 13, ISBN 0-444-51689-1, octubre de 2004.

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *A Framework for Adaptive Execution in Grids*. Publicado por *Wiley* en *Journal of Software – Practice and Experience*, vol. 34, núm. 7, págs. 631-651, ISSN 0038-0644, 2004.

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *The GridWay Framework for Adaptive Scheduling and Execution on Grids*. En *1st Workshop on Adaptive Grid Middleware (AGridM 2003)*, junto con *12th International Conference on Parallel Architectures and Compilation Techniques (PACT 2003)*, Nueva Orleans (USA), octubre de 2003. Pendiente de publicación por *Nova Science* en *Parallel and Distributed Computing Practices*, ISSN 1097-2803, 2004.

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Adaptive Scheduling and Execution on Computational Grids*. Pendiente de publicación por *Kluwer* en *Journal of Supercomputing*, ISSN 0920-8542, 2004.

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications*. En *12th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP 2004)*, La Coruña, febrero de 2004. Publicado por *IEEE Computer Society Press*, págs. 28–33, ISBN 0-7695-2083-9, 2004.

7.2. Selección de Recursos

Se ha analizado la relevancia de la proximidad de los recursos en el proceso de selección con el fin de disminuir el tiempo de transferencia de ficheros. En caso de migración oportunista, la calidad de la interconexión al recurso con los datos tiene un impacto decisivo en la sobrecarga inducida por la migración. La herramienta Grid*W*ay ha sido extendida para considerar también la proximidad dinámica de los recursos en el proceso de selección y decidir si la migración es posible y además merece la pena. El modelo de rendimiento usado en el nuevo proceso de selección considera tanto el rendimiento dinámico como la proximidad dinámica de los recursos computacionales.

La investigación en selección de recursos ha dado lugar a las siguientes publicaciones:

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Experiences on Grid Resource Selection Considering Resource Proximity*. En *1st European Across Grids Conference*, Santiago de Compostela, febrero de 2003. Publicado por *Springer-Verlag* en *Lecture Notes in Computer Science*, vol. 2970, págs. 1–8, ISBN 3-540-21048-2, ISSN 0302-9743, 2004.

- Rubén S. Montero, Eduardo Huedo e Ignacio M. Llorente: *Grid Resource Selection for Opportunistic Job Migration*. En *9th International Conference on Parallel and Distributed Computing (Euro-Par 2003)*, Klagenfurt (Austria), agosto de 2003. Publicado por *Springer-Verlag* en *Lecture Notes in Computer Science*, vol. 2790, págs. 366–373, ISBN 3-540-40788-X, ISSN 0302-9743, 2003.

- Antonio Fuentes, Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *A Grid Scheduling Algorithm Considering Dynamic Interconnecting Network Quality*. En *3rd Cracow Grid Workshop (CGW'03)*, Cracovia (Polonia), octubre de 2003. Publicado por *Academic Computer Centre CYFRONET AGH*, págs. 151–158, ISBN 83-915141-3-7, 2004.

- Antonio Fuentes, Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Impacto del Ancho de Banda en la Planificación Dinámica de Tareas en Grids Computacionales*. En *Jornadas Técnicas de RedIRIS (JT2003)*, Palma de Mallorca, noviembre de 2003. Pendiente de publicación por *RedIRIS* en *Boletín de RedIRIS*, ISSN 1139-207X, 2004.

7.3. Aplicación a la Bioinformática

Se ha descrito la aplicación de un algoritmo de predicción de estructuras a un gran número de familias de proteínas ortólogas, que realizan la misma función pero en organismos diferentes, extendiendo así un estudio realizado previamente en el que se demostró que la eficiencia de plegamiento es menor en proteínas de bacterias intracelulares que en sus parientes de vida libre. Si se conoce una estructura representativa de una proteína del conjunto, se espera que el algoritmo la reconozca como el mejor modelo estructural para cada secuencia, lo que permitirá estimar sus propiedades termodinámicas. Se han mostrado los beneficios en rendimiento y tolerancia a fallos de la planificación adaptativa de las aplicaciones bioinformáticas sobre el Grid UCM-CAB.

Actualmente, los científicos del Centro de Astrobiología cuentan con una plataforma muy potente para realizar sus simulaciones. Cabe destacar que los resultados de nuestra investigación han permitido ajustar la herramienta Grid*W*ay para que ejecute de forma altamente eficiente una aplicación de alta productividad en Bioinformática. Los resultados muestran el enorme potencial del Grid para este tipo de aplicaciones.

La investigación en aplicaciones de Bioinformática ha dado lugar a las siguientes publicaciones:

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Adaptive Scheduling of a High-Throughput Bioinformatics Application on the Grid*. En 5^{th} *International Conference on Parallel Processing and Applied Mathematics (PPAM 2003)*, dentro de *Applications Grid Workshop (AGW'03)*, Chestocova (Polonia), septiembre de 2003. Publicado por *Springer-Verlag* en *Lecture Notes in Computer Science*, vol. 3019, págs. 840–847, ISBN 3-540-21946-3, ISSN 0302-9743, 2004.

- Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Adaptive Grid Scheduling of a Bioinformatics Application*. En *XIV Jornadas de Paralelismo*, Leganés, septiembre de 2003. Publicado por *Universidad Carlos III de Madrid*, págs. 85–88, ISBN 84-89315-34-5, 2003.

- Eduardo Huedo, Ugo Bastolla, Rubén S. Montero e Ignacio M. Llorente: *Computational Proteomics on the Grid*. Publicado por *Ohmsha* en *New Generation Computing*, número especial sobre *Grid Systems for Life Sciences*, vol. 22, núm. 2, págs. 191-192, ISSN 0288-3635, 2004.

- Eduardo Huedo, Ugo Bastolla, Rubén S. Montero e Ignacio M. Llorente: *A Framework*

for Protein Structure Prediction on the Grid. Pendiente de publicación por *Ohmsha* en *New Generation Computing*, ISSN 0288-3635, 2004.

7.4. Aplicación a la Evaluación del Rendimiento en Grids

Se ha analizado la funcionalidad, fiabilidad y rendimiento de un entorno Grid basado en la herramienta Grid*W*ay y el conjunto de herramientas de Globus. Este entorno ha sido evaluado por medio de la ejecución de los NGB y la definición de un conjunto apropiado de métricas de rendimiento. La funcionalidad del banco de pruebas ha sido demostrada mediante su habilidad para ejecutar la suite NGB usando el interfaz DRMAA. Los resultados presentados muestran que la suite NGB es una valiosa herramienta para explorar el comportamiento y ajustar el rendimiento ofrecido por cada capa del entorno Grid.

La investigación sobre evaluación del rendimiento en Grids computacionales ha dado lugar a las siguientes publicaciones:

- Óscar San José, Luis M. Suárez, Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Resource Performance Management on Computational Grids*. En *2^{nd} International Symposium on Parallel and Distributed Computing (ISPDC 2003)*, junto con *6^{th} Information Society Multi-conference (IS 2003)*, Liubliana (Eslovenia), octubre de 2003. Publicado por *IEEE Computer Society Press*, págs. 215–221, ISBN 0-7695-2069-3, 2004.

- Óscar San José, Luis M. Suárez, Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Preserving Resource Performance on Computational Grids*. En *XIV Jornadas de Paralelismo*, Leganés, septiembre de 2003. Publicado por *Universidad Carlos III de Madrid*, págs. 101–106, ISBN 84-89315-34-5, 2003.

- José Herrera, Eduardo Huedo, Rubén S. Montero e Ignacio M. Llorente: *Execution of Typical Scientific Applications on Globus-based Grids*. En *3^{rd} International Symposium on Parallel and Distributed Computing (ISPDC 2004)*, Cork (Irlanda), julio de 2004. Pendiente de publicación por *IEEE Computer Society Press*, 2004.

- José Herrera, Rubén S. Montero, Eduardo Huedo e Ignacio M. Llorente: *Developing Grid-Aware Applications with DRMAA on Globus-based Grids*. En *10^{th} International Conference on Parallel and Distributed Computing (Euro-Par 2004)*, Pisa (Italia), agosto-septiembre de 2004. Publicado por *Springer-Verlag* en *Lecture Notes in Computer Science*, vol. 3149, págs. 429–435, ISBN 3-540-22924-8, ISSN 0302-9743, 2004.

- José Herrera, Rubén S. Montero, Eduardo Huedo e Ignacio M. Llorente: *DRMAA Implementation within the Grid*W*ay Framework*. En *12th Global Grid Forum (GGF 12)*, Bruselas (Bélgica), septiembre de 2004.

7.5. Trabajo Futuro

Las siguientes líneas resumen nuestro trabajo futuro y en el que ya nos encontramos invirtiendo nuestros esfuerzos:

7.5.1. Mejoras en la Herramienta Grid *W*ay

Nos proponemos mejorar las siguientes características de la herramienta:

- Robustez: Con el fin de conseguir la mayor difusión de la herramienta, nos encontramos realizando pruebas y ajustes en el código que aseguren un funcionamiento eficiente en condiciones de saturación.
- Fiabilidad: Pretendemos realizar investigación sobre tolerancia a fallos correspondiente al cliente (expiración de credenciales y fallo del agente de envío) y calidad de servicio en Grids computacionales. Los resultados de la investigación serán incluidos como mejoras en futuras versiones de la herramienta.
- Facilidad de uso: Estamos desarrollando un interfaz de programación y un conjunto de comandos para la especificación de problemas en terminología Grid. Mejoraremos la implementación del estándar DRMAA, en desarrollo por el grupo de trabajo DRMAA-WG [129] del GGF.

Por otro lado, en tecnología nunca debemos perder de vista la evolución del entorno. Las próximas versiones de Globus (GT3 y GT4) apuestan por servicios Grid basados en servicios *Web*. La implementación de la herramienta Grid *W*ay como servicio Grid representa una gran oportunidad para lograr una amplia aceptación y difusión de nuestra tecnología.

7.5.2. Investigación en Grids Computacionales

En relación con la gestión de trabajos en Grid computacionales, estamos analizando el comportamiento en el Grid de planificadores que busquen mejorar la productividad global de todos los trabajos en lugar de intentar minimizar el tiempo de ejecución de cada

trabajo de forma independiente. También queremos dar soporte para aplicaciones paralelas con MPICH-G2 [130], incluyendo selección múltiple de recursos para co-asignación [38] y adaptación parcial de los subtrabajos. De nuevo, esta investigación es posible gracias a la herramienta Grid*W*ay, y los resultados de la misma se añadirán a futuras versiones de la herramienta.

Otro aspecto que creemos muy interesante es la definición de una metodología de *benchmarking* consensuada que incluya un conjunto de criterios, métricas y programas de prueba que permitan evaluar la funcionalidad, fiabilidad y rendimiento de un entorno Grid, con el fin de ajustar su configuración base e incluso comparar entre diferentes alternativas de implementación de forma cuantitativa.

7.5.3. Integración en Grids de Datos

Existen numerosas iniciativas internacionales relacionadas con Grids de datos. La diferencia con el Grid computacional consiste principalmente en que los datos de entrada de la aplicación se encuentran distribuidos entre los diferentes nodos, ya que exceden la capacidad de almacenamiento de un único centro. Para realizar la gestión de la información se usa sistema directorio que relaciona nombres lógicos con la ubicación de las réplicas que mantienen sus contenidos.

En este tipo de Grids no sólo se debe realizar la planificación en función de la potencia de cálculo de los recursos, sino además se debe considerar la distancia (en términos de conexión de red, como ancho de banda y latencia) del nodo de ejecución a las diferentes réplicas, así como su tamaño. El usuario final debe poder especificar los ficheros de entrada y salida de su aplicación por medio de nombres lógicos. Por otro lado, es también muy importante, debido al tamaño de los ficheros involucrados, que las transferencias se hagan siempre entre recursos remotos sin pasar por el sistema cliente.

Actualmente, se están aplicando las mismas ideas presentadas en el capítulo 4 para desarrollar un selector de recursos de almacenamiento (*storage resource selector*) que considere la proximidad a un conjunto de réplicas de ficheros pertenecientes a una colección lógica. El proceso de selección de recursos de almacenamiento sería equivalente al presentado en la sección 3.4.1, aunque el proceso de descubrimiento se realizaría accediendo a un catálogo de réplicas en lugar de al GIIS. La clasificación de los recursos se basaría en el ancho de banda entre el recurso computacional seleccionado y los recursos de almacenamiento candidatos, junto con los valores recogidos del servidor GRIS de MDS. También se están desarrollando unos módulos de preparación y finalización (*prologue* y *epilogue*) que

implementen un mecanismo de diseminación de réplicas de entrada y de salida usando los servicios proporcionados por el catálogo de réplicas de Globus [101].

Además, estamos trabajando en la integración de bases datos en un sistema Grid de modo que se puedan realizar operaciones sobre la base de datos siguiendo los protocolos Globus. Los resultados de esta investigación serán de gran utilidad para poder portar al Grid aplicaciones que consistan en la realización de operaciones sobre información distribuida en diferentes nodos.

7.5.4. Aplicaciones

Un estudio pendiente en Bioinformática consiste en predecir la estructura y propiedades termodinámicas de todas las proteínas contenidas en un genoma completo. Esta aplicación todavía necesita algunas mejoras en el método, que falla si en la PDB no existe una estructura estrechamente relacionada con la objetivo. Actualmente, se está trabajando en la mejora de la función de energía y en la extensión del espacio de estructuras candidatas evaluadas. Tan pronto como el algoritmo esté preparado, se utilizará la infraestructura Grid y nuestras herramientas para llevar a cabo los experimentos.

Estamos también adaptando otras aplicaciones de interés en Astrobiología al Grid:

- Tenemos en periodo de prueba un *software* de simulación de impactos de meteoritos sobre la superficie de Marte cuyo fin es extraer patrones que permitan averiguar qué zonas de la superficie marciana estuvieron cubiertas de agua [131].

- En breve nos involucraremos en la realización de un Grid de datos que permita a la comunidad científica internacional poder realizar operaciones cruzadas con las bases de datos de los instrumentos de la sonda *Mars Express* de la ESA (*European Space Agency*).

Otro aspecto que nos parece muy interesante es analizar cómo, desde el punto de vista de una aplicación, se puede explotar de la forma más eficiente un entorno Grid dinámico. Para ello, estamos usando como caso de prueba un algoritmo genético en el que en cada sincronización se puede adaptar el particionado a los recursos disponibles en el Grid siguiendo la misma aproximación adaptativa que usa el protocolo TCP para regular el número de paquetes a enviar entre sincronizaciones consecutivas.

A. Bancos de Pruebas

En este apéndice se describen los bancos de pruebas utilizados en los distintos experimentos mostrados en esta Tesis.

A.1. El Banco de Pruebas TRGP

El proyecto TRGP (*Tidewater Research Grid Partnership*) [132] empezó en el verano de 2001 con el objetivo de fomentar el desarrollo y uso de la computación en Grid para diferentes aplicaciones de ciencia e ingeniería.

Los miembros del TRGP a finales de agosto de 2002, cuando se tomaron los resultados experimentales que se muestran en esta Tesis, incluían ICASE (*Institute for Computer Applications in Science and Engineering*) y WM (*College of William & Mary*), como se muestra en la figura A.1. Posteriormente, se unió el banco de pruebas experimental del grupo de *Arquitectura de Sistemas Distribuidos y Seguridad del Departamento de Arquitectura de Computadores y Automática* en la *Universidad Complutense de Madrid*.

Como se ve en la tabla A.1, el Grid TRGP es altamente heterogéneo. Consiste en sistemas con diferentes arquitecturas (*Intel* y *UltraSPARC*), sistemas operativos (*Solaris* y *Linux)*, sistemas de gestión de recursos (PBS y fork) y redes de interconexión (varios tipos de red *Ethernet*, *Myrinet*, *Giganet* y enlaces públicos). Además, la carga de trabajo es muy dinámica, ya que los recursos son principalmente explotados por los usuarios internos.

El banco de pruebas TRGP está formado por los siguientes recursos de computación:

- sciclone [133] es un *cluster* con 108 nodos *Solaris Sun UltraSPARC* con un total de 160 procesadores, 54 GB de memoria, 1,6 TB de disco y 115 GFLOPS de rendimiento pico en coma flotante. La tabla A.2 resume las principales características de su arquitectura, mientras que la figura A.2 muestra un diagrama de la misma.

- coral [134] es un *cluster* con 68 nodos *Linux Intel Pentium* con un total de 103 procesadores, 55 GB de memoria, 1,9 TB de disco y 89 GFLOPS de rendimiento pico en

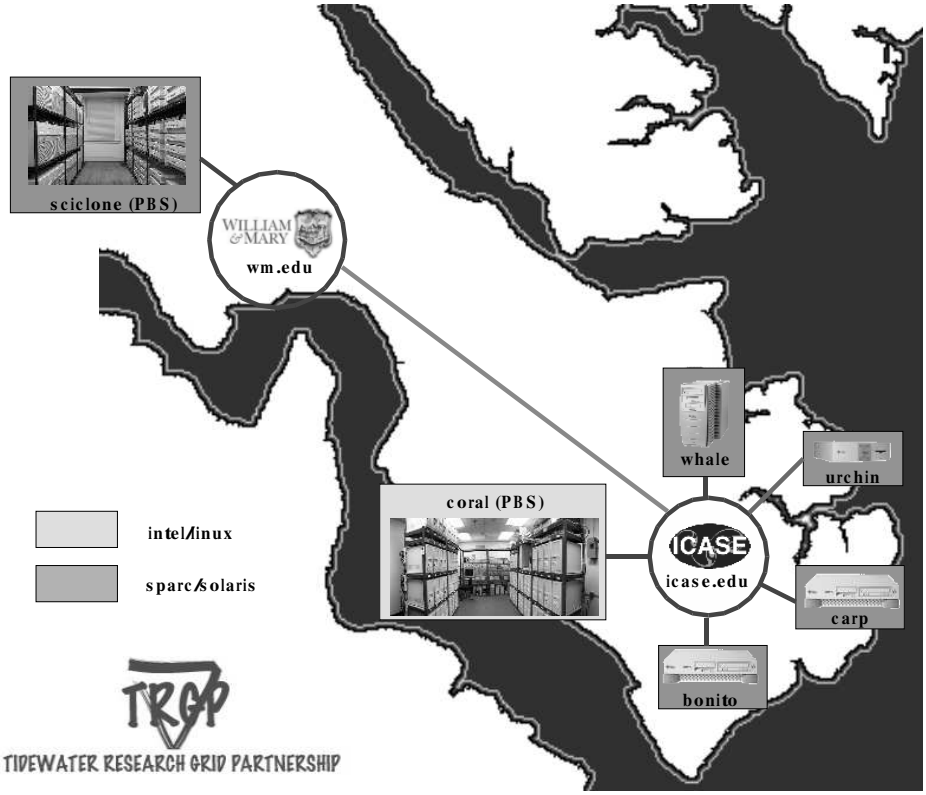

FIGURA A.1: Representación esquemática del TRGP.

TABLA A.1: Características de los recursos del banco de pruebas TRGP.

Nombre	**VO**	**Arquitectura**	**Rendimiento total**	**Procs.**	**OS**	**Mem. total**	**DRMS**
sciclone	WM	Sun UltraSPARC	115GFLOPS	160	Solaris	54GB	PBS
coral	ICASE	Intel Pentium	89GFLOPS	103	Linux	56GB	PBS
whale		Sun UltraSPARC	1,8GFLOPS	2	Solaris	4GB	fork
urchin		Sun UltraSPARC	672MFLOPS	2	Solaris	1GB	fork
carp		Sun UltraSPARC	900MFLOPS	1	Solaris	256MB	fork
tetra		Sun UltraSPARC	800MFLOPS	1	Solaris	256MB	fork
bonito		Sun UltraSPARC	720MFLOPS	1	Solaris	256MB	fork

coma flotante. La tabla A.3 resume las principales características de su arquitectura, mientras que la figura A.3 muestra un diagrama de la misma.

- whale, urchin, carp, tetra y bonito son cinco estaciones de trabajo *Solaris Sun UltraSPARC* con un total de 7 procesadores, 5,7 GB de memoria, 0,2 TB de disco y 4,9 GFLOPS de rendimiento pico en coma flotante. La tabla A.4 resume sus principales características.

El Grid completo reúne 181 nodos con 271 procesadores, 116 GB de memoria, 3,4 TB de disco y 209 GFLOPS de rendimiento pico. La interconexión entre ambas instituciones se realiza mediante una red pública no dedicada. Las estaciones de trabajo y coral en ICASE están conectados mediante una red *Fast Ethernet* conmutada (*switched*). La figura A.4 muestra la configuración del sistema de información jerárquico del Grid TRGP.

Actualmente, el proyecto está parado ya que ICASE cerró en diciembre de 2002, aunque es posible que el proyecto sea retomado en el futuro por el recientemente creado NIA (*NASA Institute of Aerospace*).

A.2. El Banco de Pruebas UCM-CAB

Hasta mayo de 2003, el banco de pruebas experimental UCM-CAB consistía en tres organizaciones virtuales (VO). Las VOs QUIM y DACYA estaban conectadas a través de la red metropolitana (*Metropolitan Area Network*, MAN) de la (*Universidad Complutense de Madrid*, UCM) y ambas estaban conectadas a la VO del (*Centro de Astrobiología*, CAB) a través de una red de área extensa (*Wide Area Network*, WAN) perteneciente a RedIRIS, la red española de investigación. La tabla A.5 muestra un resumen de las características de los recursos de este banco de pruebas durante este primer periodo.

Posteriormente, el número de organizaciones virtuales se redujo a dos, DACYA y CAB. La tabla A.6 muestra un resumen de las características de los recursos de este banco de pruebas desde ese momento hasta la actualidad.

Las VOs DACYA y CAB forman parte a su vez del banco de pruebas nacional IRIS-Grid (que se describirá en la siguiente sección), como VOs DACYA-UCM y LCASAT-CAB, respectivamente. El LCASAT (*Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas*) es quien opera y gestiona el banco de pruebas de Grid del Centro de Astrobiología.

TABLA A.2: Características del *cluster* sciclone.

Subcluster	**Nodos**	**Procesador**	**Memoria por nodo**		**Conexión**
			RAM	**HD**	
typhoon	64	1× 333MHz/2MB US-IIi (Sun Ultra 5)	256MB	9,1GB	Fast Ether.
tornado	32	2× 360MHz/4MB US-II (Sun Ultra 60)	512MB	18,2GB	Fast Ether. Myrinet
hurricane	4	4× 450MHz/4MB US-II (Sun Enterp. 420R)	4GB	18,2GB	Gigabit Eth. Myrinet
	1	2× 360MHz/4MB US-II (Sun Ultra 60)	512MB	18,2GB	
Servidor frontal	1	2× 400MHz/4MB US-II (Sun Ultra 60)	1GB	91GB	Gigabit Eth. Myrinet
gulfstream (Servidor de ficheros)	6	2× 360MHz/4MB US-II (Sun Ultra 60)	512MB	45,5 ó 63,7GB	Gigabit Eth. Myrinet
Conexión entre *clusters*	• Dos conmutadores *Fast Ethernet* para los nodos de typhoon • Conmutador *Fast Ethernet* para los nodos de tornado • Conmutador *Gigabit Ethernet* para los nodos de gulfstream • Conmutador *Gigabit Ethernet* para los nodos de hurricane • Conmutador *Gigabit Ethernet* para interconectar typhoon, tornado y gulfstream • Tres enlaces *Gigabit Ethernet* para interconectar gulfstream y hurricane • Conmutador *Myrinet* para interconectar tornado, gulfstream y hurricane				
OS	Sun Solaris 7				
Servicios GRAM	Fork Job Manager PBS Job Manager				

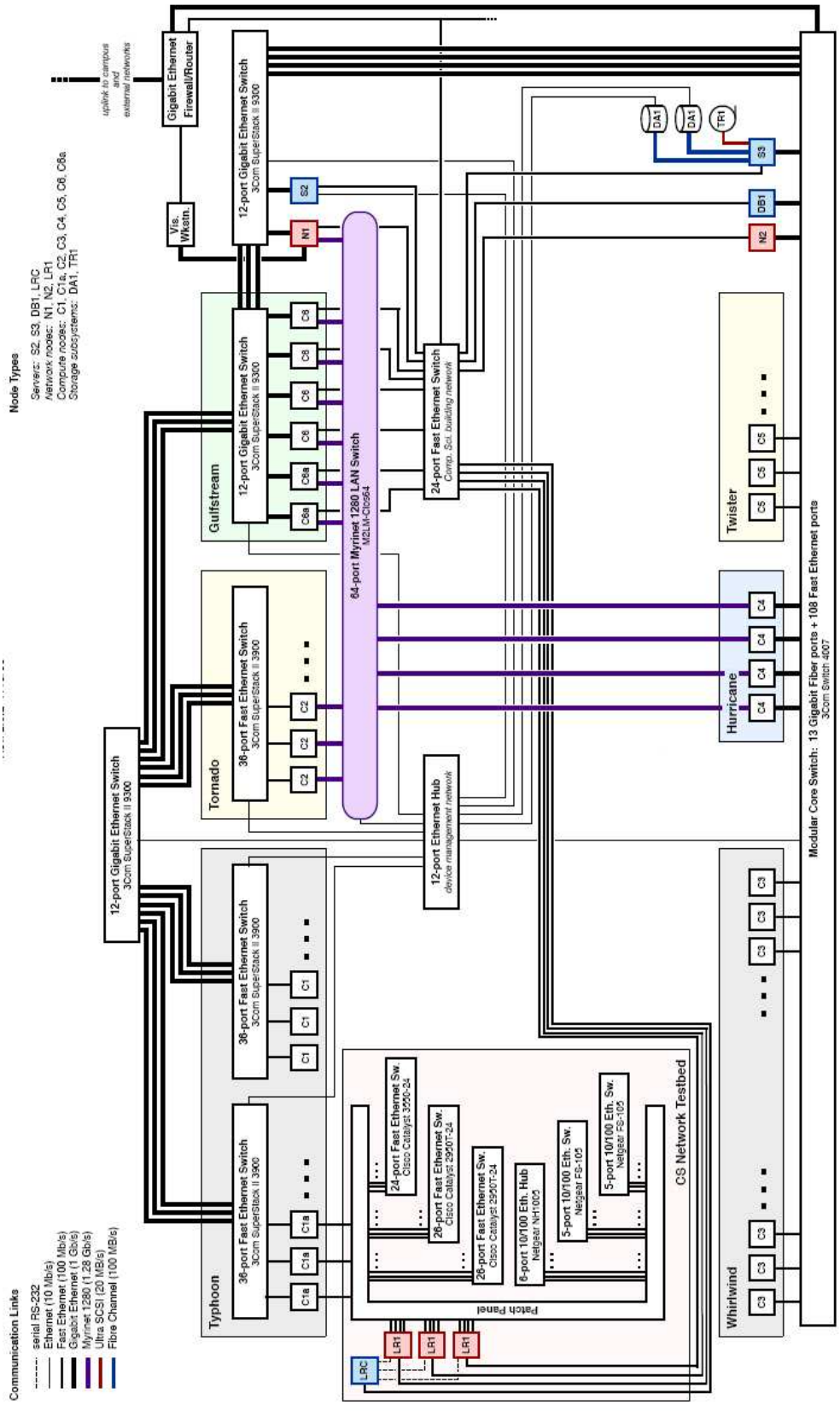

FIGURA A.2: Arquitectura del *cluster* sciclone.

TABLA A.3: Características del *cluster* coral.

Subcluster	**Nodos**	**Procesador**	**Memoria por nodo**		**Conexión**
			RAM	**HD**	
Fase I	8	1× 400MHz/0,5MB Pentium II	384MB	6,5GB	Fast Ether.
Fase II	16	2× 500MHz/0,5MB Pentium III	512MB	14,4GB	Fast Ether. Giganet
Fase III	16	2× 800MHz/0,5MB Pentium III	1GB	30GB	Fast Ether. Giganet
Fase IV	24	1× 1,7GHz/0,25MB Pentium 4	1 ó 2GB	6,5 ó 20GB	Fast Ether.
Servidor frontal	1	2× 400MHz/0,5MB Pentium II	512MB	76GB	Gigabit Eth.
Servidor de ficheros	2	2× 500MHz/0,5MB Pentium III	512MB	108GB	Gigabit Eth.
	1	2× 400MHz/0,5MB Pentium II	384MB	640GB 18GB	
Conexión entre *clusters*	• Conmutadores *Fast Ethernet* y *Giganet* compartidos entre los nodos de las fases II y III • Conmutador *Fast Ethernet* compartido entre los nodos de las fases I y IV • Doble enlace *Gigabit Ethernet* entre las fases II/III y las fases I/IV • Conmutador *Gigabit Ethernet* para conectar las fases II/III, las fases I/IV, el servidor frontal y el servidor de ficheros				
OS	Linux Red Hat 7.2				
Servicios GRAM	Fork Job Manager PBS Job Manager				

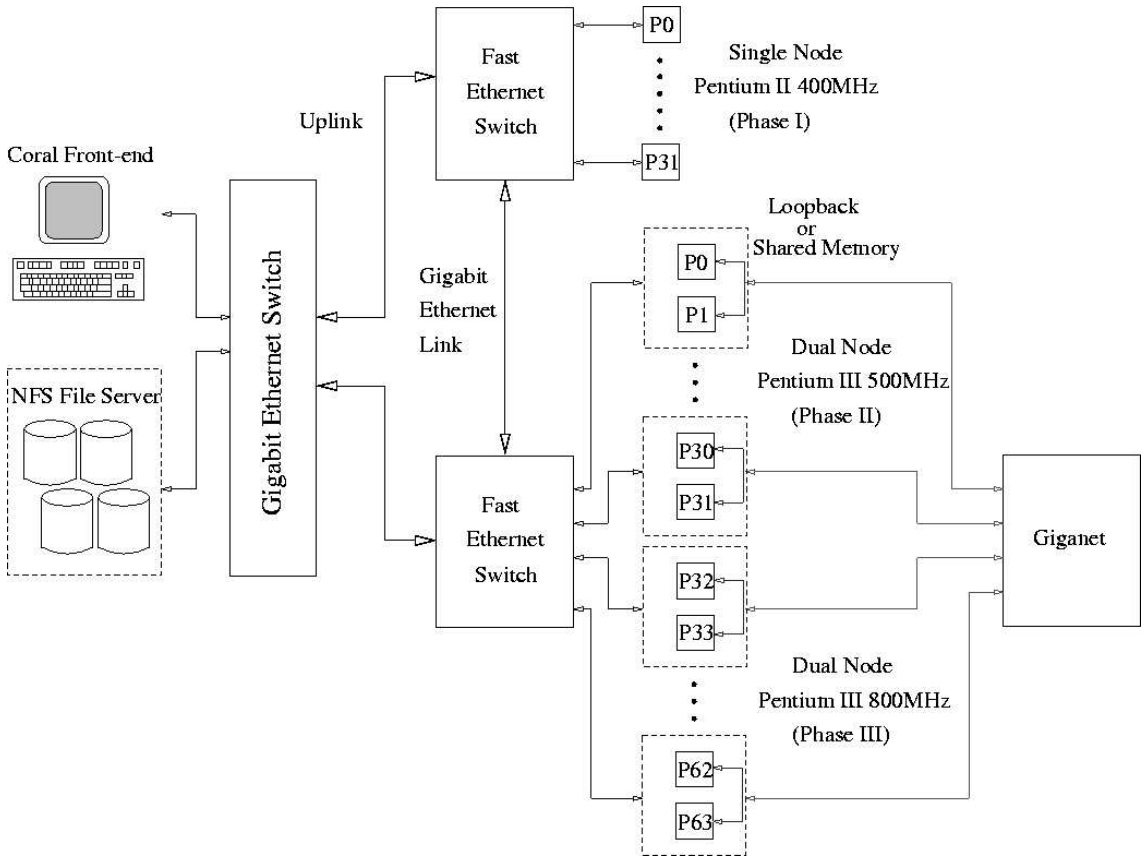

FIGURA A.3: Arquitectura del *cluster* coral.

TABLA A.4: Características de las estaciones de trabajo de ICASE.

Estación	**OS**	**Servicio GRAM**	**Procesador**	**Memoria por nodo**	
				RAM	**HD**
whale	Solaris 7	Fork	2× 450MHz/4MB US-II (Sun Ultra 80)	4GB	40GB
urchin	Solaris 7	Fork	2× 168MHz/0,5MB US-I (Sun Ultra 2)	1GB	150GB
carp	Solaris 7	Fork	1× 450MHz/0,2MB US-IIi (Sun Ultra 5)	256MB	8,1GB
tetra	Solaris 7	Fork	1× 400MHz/0,2MB US-IIi (Sun Ultra 5)	256MB	19GB
bonito	Solaris 7	Fork	1× 360MHz/0,2MB US-IIi (Sun Ultra 5)	256MB	8,1GB

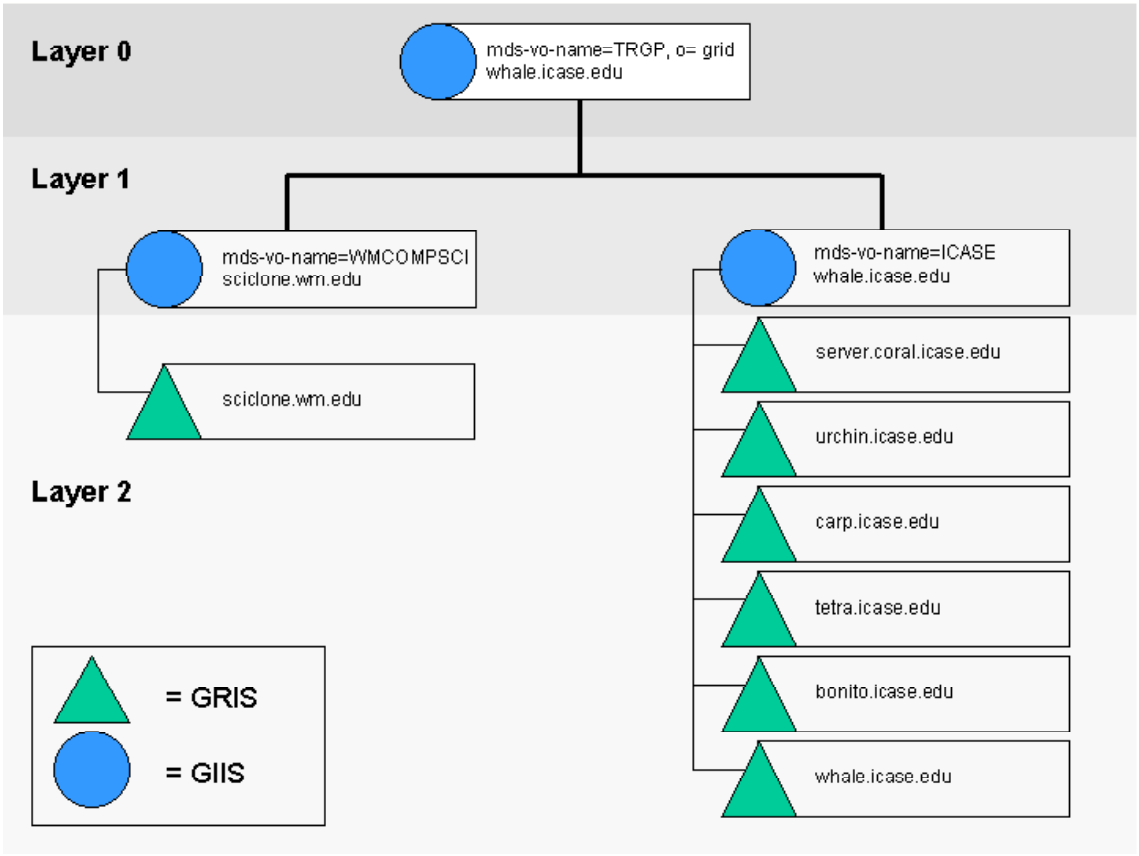

FIGURA A.4: Estructura jerárquica del servicio de información (MDS) de TRGP.

TABLA A.5: Características de los recursos del banco de pruebas UCM-CAB hasta mayo de 2003.

Nombre	**VO**	**Arquitectura**	**Velocidad**	**Núm. procs.**	**OS**	**Mem. total**	**DRMS**
ursa	DACYA	UltraSPARC-IIe	500MHz	1	Solaris	256MB	fork
draco		UltraSPARC-I	117MHz	1	Solaris	128MB	fork
columba		Pentium MMX	233MHz	1	Linux	160MB	fork
cepheus		Pentium Pro	200MHz	1	Linux	64MB	fork
pegasus		Pentium 4	2,4GHz	1	Linux	1GB	fork
solea	QUIM	UltraSPARC-II	300MHz	2	Solaris	256MB	fork
babieca	CAB	Alpha EV67	450MHz	30	Linux	7,5GB	PBS

TABLA A.6: Características de los recursos del banco de pruebas UCM-CAB en la actualidad.

Nombre	VO	Arquitectura	Velocidad	Núm. procs.	OS	Mem. total	DRMS
ursa	DACYA	UltraSPARC-IIe	500MHz	1	Solaris	256MB	fork
pegasus		Pentium 4	2,4GHz	1	Linux	1GB	fork
hydrus		Pentium 4	2,5GHz	1	Linux	512MB	fork
cygnus		Pentium 4	2,5GHz	1	Linux	512MB	fork
cepheus		Pentium III	600MHz	1	Linux	256MB	fork
aquila		Pentium III	700MHz	1	Linux	128MB	fork
babieca	CAB	Alpha EV6	450MHz	30	Linux	7,5GB	PBS

A.3. El Banco de Pruebas IRISGrid

IRISGrid pretende aportar los protocolos, procedimientos y guías de "buenas prácticas" necesarios para construir dentro de España un Grid de investigación coordinando a los diferentes grupos y centros interesados en investigación Grid. Esta iniciativa pretende unir recursos distribuidos geográficamente para que los grupos involucrados tengan un banco de pruebas donde realizar investigación en cualquiera de las áreas Grid.

El objetivo de IRISGrid es facilitar a los grupos interesados la unión de sus recursos al Grid. Queremos resaltar que IRISGrid coordinará este Grid estableciendo procedimientos relacionados principalmente con la autenticación y la monitorización de recursos. La decisión final de qué usuarios podrán usar los recursos (autorización) será, por supuesto, de los propietarios de los mismos siguiendo sus normas locales, ya que la tecnología Grid une dominios de administración sin implicar cambios en las políticas de seguridad o gestión de recursos internas dentro de cada grupo.

A.3.1. Definiciones sobre la Organización de IRISGrid

A continuación se enumeran una serie de definiciones sobre la organización de IRISGrid:

- **Comité ejecutivo**: Comité encargado de tomar decisiones sobre el futuro de IRISGrid. Este comité estará formado por los Representantes de las diferentes Instituciones o Centros del ámbito científico y académico que formen parte del Grid.

FIGURA A.5: Logotipo de IRISGrid.

- **Coordinador**: Miembro del Comité Ejecutivo responsable de mantener el servidor *web* `irisgrid.rediris.es` con los enlaces a la información de los Centros o Instituciones y los diferentes procedimientos y protocolos, y de recibir las solicitudes de ingreso de nuevos Centros e Instituciones.

- **Centro o Institución**: Grupo de Usuarios y Recursos que pertenecen a un mismo Centro o Institución, que desarrollan una actividad común y están interesados en unirse a IRISGrid. Cada Centro o Institución se identifica dentro de IRISGrid mediante un nombre único. Es el modo principal de diferenciar Recursos y Usuarios dentro de IRISGrid. Se recomienda que sea breve. Por ejemplo, DACyA-UCM identifica al Departamento de Arquitectura de Computadores y Automática de la Universidad Complutense de Madrid

- **Representante del Centro o Institución**: Persona responsable de realizar las comunicaciones con los Representantes de los diferentes Centros o Instituciones y con el Coordinador e IRISGrid CA. Adicionalmente, podrá pertenecer al Comité Ejecutivo IRISGrid.

- **Responsable Técnico**: Persona encargado de administrar los Recursos de un Centro o Institución. Será la persona de contacto para la resolución de problemas técnicos. Podrá realizar comunicaciones con los administradores de otros Centros o Instituciones.

- **Recurso de IRISGrid**: Sistema computacional aportado a IRISGrid por un Centro o Institución. Se identifica mediante el nombre de dominio completamente cualificado (*Fully-Qualified Domain Name*, FQDN) del sistema.

- **Usuario de IRISGrid**: Persona perteneciente a un Centro o Institución que usa los Recursos de IRISGrid para fines científicos o académicos. Únicamente se comunicará con el Representante y Responsable Técnico de su Centro o Institución. Se identifica mediante un nombre único (*Distinguished Name*, DN), que incluye su nombre completo y su dirección de *e-mail*.

- **Entidad de certificación de IRISGrid (IRISGrid CA)**: Entidad encargada de firmar los certificados de servicios, Recursos, Usuarios, Representantes y Responsables Técnicos de los diferentes Centros e instituciones.

- **Certificado**: Credenciales X.509 utilizadas para identificar los servicios, Recursos, Usuarios, Representantes, Responsables Técnicos y Coordinador de IRISGrid. Generalmente, deben estar firmados por IRISGrid CA.

- **Canal de Comunicación**: Conjunto de protocolos para realizar la transferencia de información, vía *e-mail*, entre los Representantes, Responsables Técnicos, Coordinador e IRISGrid CA. IRISGrid proporciona los mecanismos para que se puedan realizar comunicaciones de confianza, pero en último término son los miembros que intervienen en una comunicación los que deben implementar las normas.

A.3.2. Componentes Básicos

Globus se ha convertido en el estándar de facto para la computación distribuida y será el soporte sobre el que se desarrollará IRISGrid. La versión que se usará en IRISGrid es la 2.4.

La configuración por defecto de Globus asigna al servicio GRAM (*gatekeeper*) al puerto 2119, todos los Recursos de IRISGrid deben garantizar el acceso desde el exterior a este puerto. El servicio GridFTP se asigna por defecto al puerto 2811, de nuevo todos los Recursos de IRISGrid deben garantizar el acceso a este servicio desde el exterior. Por defecto, el servidor LDAP (`slapd`), tanto para el GRIS como para el GIIS, se asigna al puerto 2135, todos los Recursos de IRISGrid deben garantizar el acceso a este servicio desde el exterior.

El acceso a cada recurso del Grid se controla mediante el archivo `grid-mapfile` del directorio `/etc/grid-security`, que consiste en una serie de asignaciones entre el sujeto (DN) del certificado de un Usuario de IRISGrid que puede usar el Recurso, y un usuario local. Cada uno de los Centros o Instituciones, atendiendo a su propia política de adminis-

tración, deberá decidir qué Usuarios de IRISGrid tienen acceso a cada Recurso, y si éstos se asignan al mismo usuario local o, por el contrario, se crea una cuenta local distinta para cada Usuario de IRISGrid.

Se ha configurado un *host* (albergado por RedIRIS en `giis-irisgrid.rediris.es`) como nodo raíz para el acceso superior GIIS, que a su vez pregunta a los servidores GIIS de cada Centro o Institución con su nombre. Con el fin de simplificar su gestión, todos los grupos, respecto a IRISGrid, están al mismo nivel independientemente de si pertenecen a la misma institución o ubicación geográfica. Internamente los diferentes Centros o Instituciones podrán estar a su vez organizados jerárquicamente.

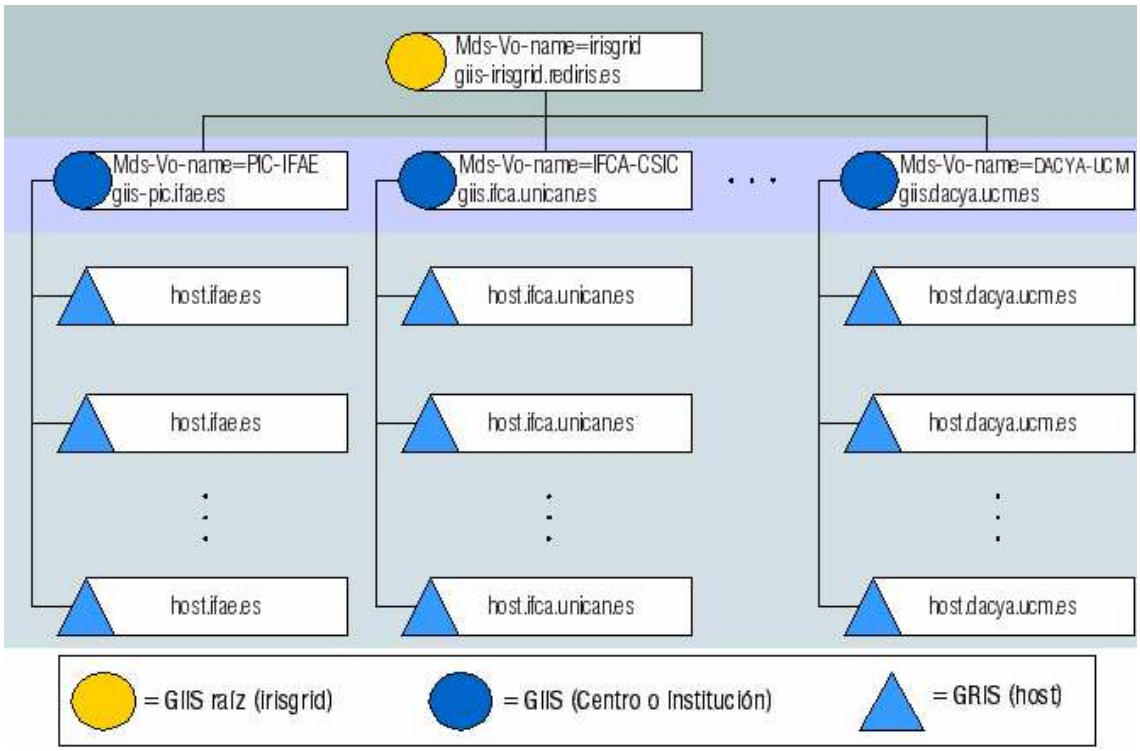

FIGURA A.6: Estructura jerárquica del servicio de información (MDS) de IRISGrid.

A.3.3. Gestión de la Seguridad

La gestión de la seguridad es quizá la componente más importante de un Grid. Debemos establecer procedimientos y políticas de seguridad lo suficientemente robustas como para que en el futuro IRISGrid pueda convertirse en un Grid con calidad de producción.

Protocolos de Comunicación Segura

Los certificados emitidos para los Usuarios de IRISGrid (generados con el comando `grid-cert-request`) se usarán, además de para la explotación segura del Grid, para garantizar las comunicaciones entre Representantes, Responsables Técnicos, Coordinador e IRISGrid CA. Para convertir el certificado en formato PEM (*Privacy Enhanced Mail*) [33] usado por Globus al formato PKCS (*Public Key Cryptography Standards*) [135] usado por los clientes de correo comunes (Netscape y Microsoft OutLook) se puede usar el siguiente comando:

```
% openssl pkcs12 -export -in usercert.pem -inkey userkey.pem -out cert.p12
```

Se han establecido los siguientes dos niveles de seguridad, que los gestores de correo habituales implementan de forma automática:

- **Nivel 1: Integridad y autenticación del emisor del mensaje**: El emisor realizará un resumen del mensaje que encriptará con su clave privada. Este protocolo garantiza que el mensaje no ha sido manipulado y además que el emisor es realmente quien dice ser.
- **Nivel 2: Confidencialidad**: Además de satisfacer el nivel anterior, el mensaje se encriptará con la clave pública del receptor. De este modo, la información viajará por la red encriptada.

Canales de Comunicación

La figura A.7 muestra los Canales de Comunicación usados en IRISGrid:

- **Canal de Gestión**.

  Los Representantes sólo se podrán comunicar con otros Representantes, con el Coordinador y con IRISGrid CA. Este Canal está destinado a:

  - Realizar peticiones de firma de certificados de nuevos Usuarios de IRISGrid (Nivel 1 de seguridad).
  - Realizar peticiones de firma de certificados de nuevos Recursos de IRISGrid (Nivel 1 de seguridad).
  - Solicitar autorización en sistemas (Nivel 1 de seguridad, salvo que sea necesario comunicar alguna contraseña, en cuyo caso se empleará el Nivel 2 de seguridad).

- **Canal Técnico**

  Los administradores se comunicarán entre sí para solventar aspectos técnicos. Cuando un Usuario de un Centro o Institución tenga problemas, se lo comunicará a su Responsable Técnico que se pondrá en contacto con el Responsable Técnico del Centro o Institución al cual pertenece el Recurso remoto. Por defecto, para estas comunicaciones se requiere un Nivel 1 de seguridad. Los problemas de seguridad observados también se reportarán por medio de este Canal usando un Nivel 2 de seguridad.

- **Canal Interno**

  Es el usado por los Usuarios de un Centro o Institución para comunicarse con su Representante y Responsable Técnico. Los protocolos de comunicación y niveles de seguridad de este Canal dependen de las políticas propias de cada Centro o Institución.

Resaltar que los puntos anteriores indican un conducto reglamentario de obligado cumplimiento. Un Usuario, por ejemplo, no podrá contactar con Representantes y Responsables Técnicos de otros Centros o Instituciones. Sin embargo, existe la posibilidad de que una persona desempeñe varios roles (Usuario, Representante o Responsable Técnico).

Requisitos de IRISGrid CA

La entidad de certificación de IRISGrid, IRISGrid CA, es responsable de:

- Crear y mantener una lista de certificados revocados (*Certificate Revocation List*, CRL). La CRL deberá actualizarse inmediatamente después de cada revocación y 7 días antes de su caducidad, que deberá ser de no más de 30 días. Cada Centro o Institución es responsable de actualizar sus copias locales de la CRL. La CRL se publicará en la página `irisgrid.rediris.es`.

- Publicar en `irisgrid.rediris.es` su política de certificación (*Certificate Policy Statement*, CPS), de acuerdo a la plantilla del RFC 2527 [136].

- Publicar sus credenciales en `irisgrid.rediris.es`.

La máquina usada para emitir los certificados debe satisfacer los siguientes requisitos:

- Debe ser una máquina dedicada.

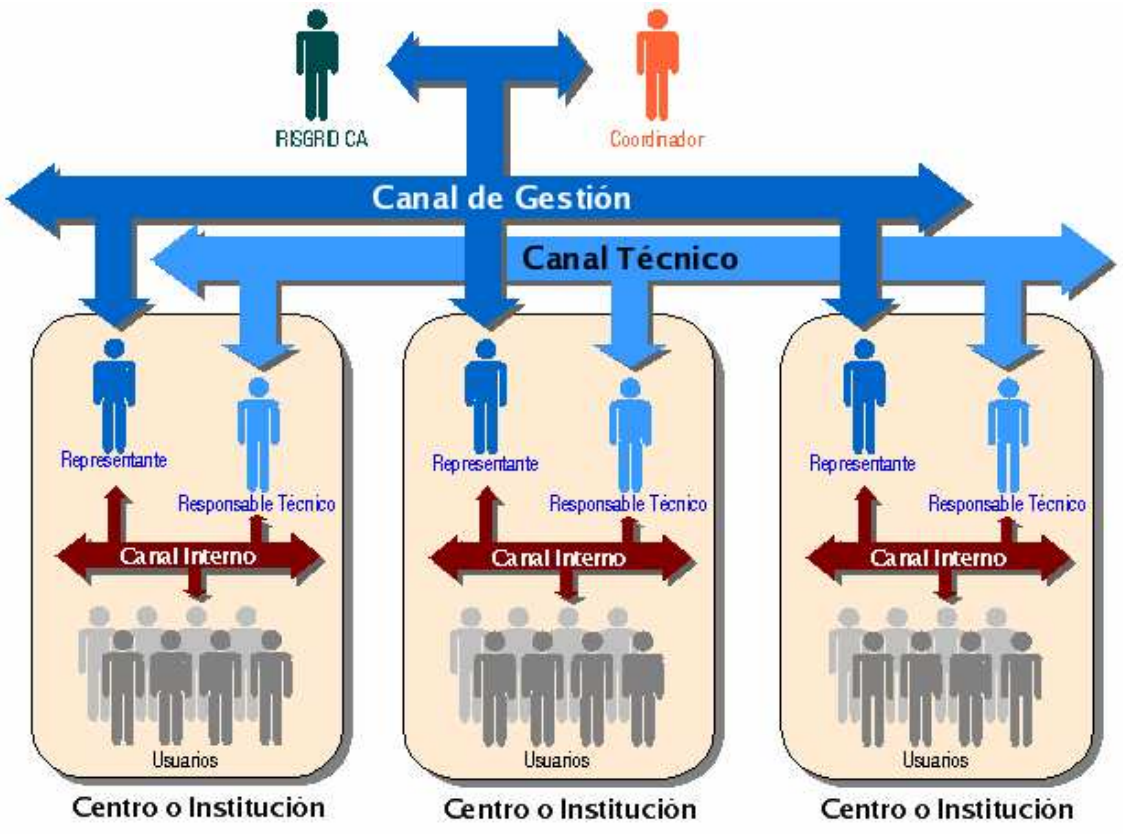

FIGURA A.7: Esquema de los distintos Canales de Comunicación empleados en IRISGrid.

- La ubicación de la máquina ha de ser segura.
- Debe ser administrada por personal cualificado.
- No puede estar conectada a ninguna red pública.
- La clave privada de la CA, así como sus copias, ha de permanecer siempre en un entorno seguro.
- La clave privada ha de ser encriptada con una contraseña no inferior a 10 caracteres y sólo puede conocerla la persona encargada de firmar los certificados.

A.3.4. Procedimientos

Alta en IRISGrid de Centros o Instituciones

Los Centros o Instituciones que deseen participar en la iniciativa IRISGrid deberán seguir los siguientes pasos, independientemente de si el Centro o Institución incluye Recursos,

Usuarios o ambos.

Solicitud de nombre de Institución o Centro:

1. Crear una página *web* que incluya la siguiente información:

   - Nombre propuesto de la Institución o Centro.
   - Representante (nombre, teléfono y *e-mail*).
   - Responsable Técnico (nombre, teléfono y *e-mail*).

2. El Representante deberá solicitar el alta de su Centro o Institución enviado un *e-mail* al Coordinador de IRISGrid con un enlace a la página *web* descrita en el punto anterior.

3. El Coordinador de IRISGrid enviará la solicitud (Nivel 1 de seguridad) a todos los miembros del Comité Ejecutivo para su evaluación durante un periodo máximo de un día hábil y comprobará que el nuevo Centro o Institución cumple los requisitos mínimos. El Coordinador comunicará al Representante del Centro o Institución candidata el resultado de la evaluación del Comité Ejecutivo.

Configuración y solicitud de las credenciales:

1. Instalar los paquetes de IRISGrid CA mediante la secuencia de comandos:

```
% $GPT_LOCATION/sbin/gpt-build \
    globus_simple_ca_41d1861c_setup-0.12.tar.gz <flavour>

% $GPT_LOCATION/sbin/gpt-postinstall

% $GLOBUS_LOCATION/setup/globus_simple_ca_41d1861c_setup/setup-gsi
```

   donde `<flavour>` define las opciones de instalación para el Centro o Institución, por ejemplo, `gcc32dbg`, `vendorcc32pthr`, etc.

2. Una vez instalado, modificar el fichero `globus-user-ssl.conf.41d1861c` del directorio `/etc/grid-security/certificates`:

```
...
[ req_distinguished_name ]
# BEGIN CONFIG
countryName = Country Name (2 letter code)
countryName_default = ES
countryName_min = 2
countryName_max = 2
0.organizationName = Level 0 Organization
0.organizationName_default = IRISGrid
0.organizationalUnitName = Level 0 Organizational Unit
0.organizationalUnitName_default = DACyA-UCM
commonName = Name (e.g., John M. Smith)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 40
# END CONFIG
...
```

3. Crea un nuevo certificado para el Representante y otro para el Responsable Técnico. El DN final debe tener la forma:

   ```
   /C=ES/O=IRISGrid/OU=<Nombre Centro>/CN=<Nombre Completo>/Email=<email>
   ```

   Para solicitar los certificados, usar el comando:

   ```
   % grid-cert-request -ca -int
   ```

   Se creará, entre otros, el archivo `$HOME/.globus/usercert_request.pem`. Ejecutar el siguiente comando para obtener la huella digital (*fingerprint*) de la solicitud (anotar el resultado para comprobarlo posteriormente con IRISGrid CA).

   ```
   % openssl dgst -c -sha1 usercert_request.pem
   ```

   La figura del Responsable Técnico es necesaria, aunque el Centro o Institución sólo incluya Usuarios, para implementar el Canal Técnico.

4. Añadir IRISGrid CA como CA válida en la herramienta de correo electrónico descargando los ficheros adecuados.

5. El Representante enviará la solicitud a IRISGrid CA quien se pondrá en contacto por teléfono con el fin de comprobar la validez de los certificados comparando las huellas digitales (*fingerprints*) de las solicitudes. Una vez comprobados, IRISGrid CA enviará los certificados firmados al Responsable Técnico (Nivel 1 de seguridad).

6. El Responsable Técnico actualizará la página *web* con los enlaces a los certificados firmados.

   - Nombre del Centro o Institución.
   - Representante (nombre, teléfono, *e-mail* y certificado firmado por IRISGrid CA).
   - Responsable Técnico (nombre, teléfono, *e-mail* y certificado firmado por IRISGrid CA).

   Los certificados deben publicarse en la página *web* en formato DER (*Distinguished Encoding Rules*) [137], de forma que puedan ser fácilmente importados por los navegadores más comunes. Para exportarlos, ejecutar el siguiente comando:

   ```
   % openssl x509 -outform der -in usercert.pem -out usercert.der
   ```

7. El Representante y Responsable Técnico importarán los certificados en su gestor de correo para implementar los Canales de Comunicación descritos en la sección A.3.3. Los certificados pueden convertirse en formato PKCS12 [135] (aceptado por los gestores de correo habituales) mediante el comando:

   ```
   % openssl pkcs12 -export -in usercert.pem -inkey userkey.pem \
       -out usercert.p12
   ```

Solicitud de Alta de Nuevos Recursos

El siguiente procedimiento supone la existencia de un Centro o Institución dado de alta de acuerdo al procedimiento anterior.

1. Cuando se añaden nuevos recursos es necesario crear certificados para los Recursos y sus servicios. Estos certificados se crean con el DN recomendado por Globus pero siempre usando el formato:

   ```
   /C=ES/O=IRISGrid/OU=<Nombre Centro>/CN=host/<FQDN maquina>
   ```

   para generar el certificado, ejecutar el comando:

   ```
   % grid-cert-request -ca -host <FQDN>
   ```

2. Actualizar la página *web* para que incluya información del sitio:

   - Nombre del Centro o Institución.
   - Representante (nombre, teléfono, *e-mail* y certificado firmado por IRISGrid CA).
   - Responsable Técnico (nombre, teléfono, *e-mail* y certificado firmado por IRISGrid CA).
   - Credenciales de su CA si es diferente a IRISGrid CA.
   - Sistemas conectados al Grid (nombre, arquitectura, sistema operativo, y componentes Globus instalados).

3. El Representante deberá solicitar la firma de los nuevos certificados enviándolos por *e-mail* a IRISGrid CA (Nivel 1 de seguridad), que firmará los certificados y los devolverá en un *e-mail* al Representante (Nivel 1 de seguridad). El Responsable Técnico instalará los certificados de los nuevos recursos.

Solicitud de Alta de Nuevos Usuarios

El siguiente procedimiento supone la existencia de un Centro o Institución dado de alta de acuerdo al procedimiento anterior.

1. Crea un nuevo certificado para el Usuario. El DN final debe tener la forma:

   ```
   /C=ES/O=IRISGrid/OU=<Nombre Centro>/CN=<Nombre Completo>/Email=<email>
   ```

   Para solicitar los certificados, usar el comando:

```
% grid-cert-request -ca -int
```

2. El Representante deberá solicitar la firma de los nuevos certificados enviando un *e-mail* a IRISGrid CA. Es importante resaltar que el Representante es el encargado de autenticar internamente dentro de su Institución o Centro a los Usuarios. Todos los nuevos Usuarios deberán enviar firmado (primero por fax y luego por correo) el documento de compromiso de seguridad de IRISGrid a la dirección postal de IRISGrid CA. IRISGrid CA firmará los certificados y los enviará en un *e-mail* de vuelta al Representante (Nivel 1 de seguridad).

Solicitud de Autorización para el Uso de Recursos

IRISGrid proporciona autenticación. Sin embargo, la autorización se realiza de forma totalmente independiente dentro de cada Institución o Centro. Para solicitar la autorización a usar un Recurso gestionado por una Institución o Centro, se seguirán los siguientes pasos:

1. En la página *web* de cada Institución o Centro se describe la información necesaria que se deberá enviar a su Representante. Siempre será el Representante de cada Institución o Centro el que solicite autorización a otro Representante (Nivel 1 de seguridad).

2. Cada Institución o Centro decidirá si autoriza el uso de sus Recursos y los procedimientos que deberá utilizar el solicitante, así como las políticas y procedimientos de seguridad internos que deberá respetar. En tal caso podrá (si es necesario) enviar la contraseña del nuevo Usuario al Representante que lo solicitó (Nivel 2 de seguridad).

Solicitud de Baja de un Usuario

El Representante de cada Institución o Centro debe llevar un registro de todas las cuentas que ha solicitado para los Usuarios de su Institución o Centro. Cuando un Usuario no vaya a usar el Grid, el Representante debe informar a los Responsables Técnicos de todas las Instituciones o Centros donde el Usuario tenía cuenta. Además, deberá informar a IRISGrid CA para añadirlo a la lista de certificados revocados (CRL).

A.3.5. Demostración del Grid Nacional IRISGrid

La primera demostración del Grid Nacional IRISGrid se realizó durante los Grupos de Trabajo de RedIRIS (*GT RedIRIS 2003*) en Madrid en mayo de 2003.

TABLA A.7: Características de las máquinas del banco de pruebas usado en la demostración del Grid Nacional IRISGrid.

Nombre	VO	Arquitectura	Rendimiento total	Núm. Procs.	OS	DRMS
pc-ruben	DACyA-UCM	Pentium 4	2,4GFLOPS	1	Linux	fork
solea	QUIM-UCM	US-IIe	1,2GFLOPS	2	Solaris	fork
babieca	LCASAT-CAB	Alpha EV6	27GFLOPS	30	Linux	PBS
bw	CESGA	Pentium III	16GFLOPS	16	Linux	PBS
grid-w2	PIC-IFAE	Pentium 4	8GFLOPS	4	Linux	PBS
ramses	DSIC-UPV	Pentium III	17GFLOPS	20	Linux	PBS
sherlock	DIF-UM	Pentium III	550MFLOPS	1	Linux	fork
Total			75GFLOPS	72		

La tabla A.7 resume las características de las máquinas del banco de pruebas usado en esta demostración. Todas las VOs del banco de pruebas estaban interconectadas mediante la red española de investigación, RedIRIS. La arquitectura del banco de pruebas estaba basada en el *Globus Toolkit* versión 2.X, del que se usaron los siguientes servicios básicos:

- Gestión de recursos: GRAM.
- Gestión de datos: GridFTP y GASS.
- Servicio de información: MDS, usando únicamente el GRIS local de los recursos.
- Infraestructura de seguridad: CA de DataGrid-España y otras CAs existentes en los bancos de pruebas de investigación.

Se utilizó una aplicación para resolver el flujo de un fluido, con número de Reynolds 10^4, sobre una placa delgada mediante un algoritmo multimalla robusto (ver parte izquierda de la figura A.8). Se calculó el coeficiente de rozamiento viscoso para diferentes ángulos de ataque, α, tal y como se ve en la parte derecha de la figura A.8. Para la planificación y ejecución de esta aplicación paramétrica se utilizó la herramienta Grid*W*ay, cuyas características se han descrito a lo largo de esta Tesis.

La figura A.9 muestra la productividad dinámica obtenida durante la demostración, en términos del tiempo medio empleado por cada trabajo conforme la aplicación progresa.

La parte izquierda de la figura A.10 muestra los tiempos medios de transferencia, ejecución y pared para cada recurso del banco de pruebas, junto con su desviación estándar. La

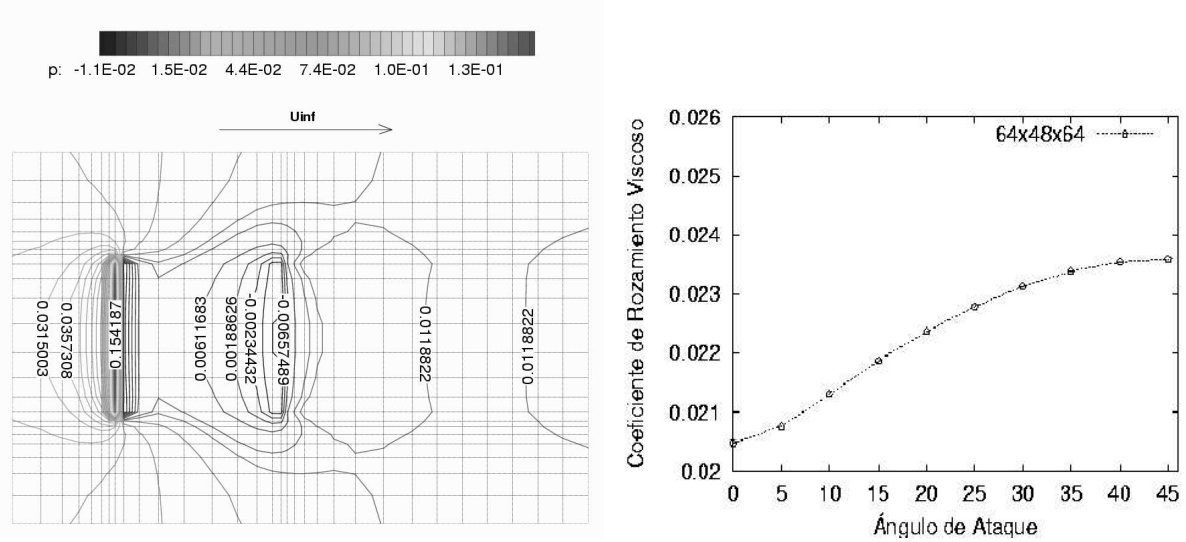

FIGURA A.8: Flujo de un fluido sobre una placa delgada (izquierda) y coeficiente de rozamiento viscoso para diferentes ángulos de ataque (derecha).

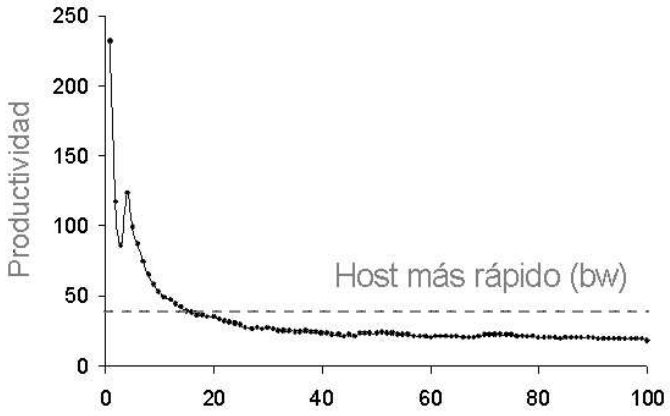

FIGURA A.9: Productividad dinámica durante la demostración.

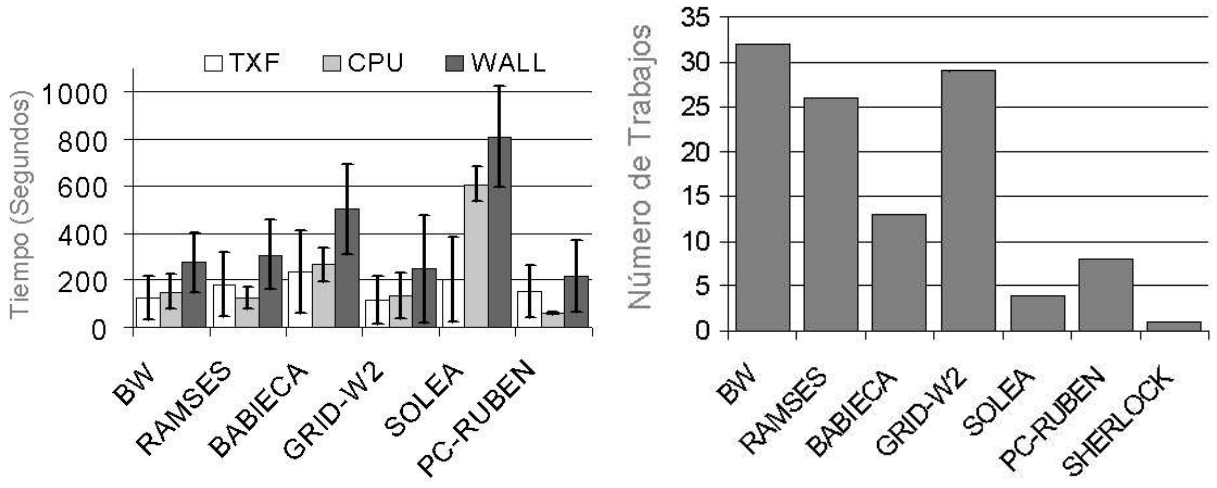

FIGURA A.10: Tiempos medios de transferencia, ejecución y pared durante la demostración (izquierda) y planificación realizada por Grid*W*ay durante la demostración (derecha).

parte derecha de la figura A.10 muestra la planificación realizada por Grid*W*ay durante la demostración como el número de trabajos asignados a cada recurso del banco de pruebas, en función del rendimiento que ofrecen.

B. Manual de Referencia de la Herramienta Grid*W*ay

En este apéndice se describe el modo de uso y programación de la herramienta Grid*W*ay.

B.1. Plantilla del Trabajo

Todos los datos necesarios para la planificación y ejecución de los trabajos se especifican en un fichero de plantilla de trabajo (`job template`). La figura B.1 muestra un ejemplo de plantilla de trabajo para una aplicación de dinámica de fluidos computacional (CFD).

B.1.1. Parámetros de Ejecución

- `EXECUTABLE_FILE`: Fichero ejecutable.
- `EXECUTABLE_PARAMETERS`: Parámetros del ejecutable.
- `STDIN_FILE`: Fichero con la entrada estándar.
- `STDOUT_FILE`: Fichero con la salida estándar.
- `STDERR_FILE`: Fichero con la salida de error estándar.
- `INPUT_FILES`: Ficheros de entrada.
- `OUTPUT_FILES`: Ficheros de salida.
- `RESTART_FILES`: Ficheros de reinicio.
- `PROLOG`: Programa de preparación (prólogo).
- `WRAPPER`: Programa de ejecución (envoltorio).
- `EPILOG`: Programa de finalización (epílogo).

```
#-----------------------------------
# Job Template, CFD simulation
#-----------------------------------
# Executable Parameters
  EXECUTABLE_FILE=NS3D.${GW_ARCH}
  EXECUTABLE_ARGUMENTS=input
# Experiment Files
  INPUT_FILES="input grid32.${GW_ARCH}"
  OUTPUT_FILES="profile.${GW_JOB_ID}"
  RESTART_FILES = checkpoint.ascii
# Standard I/O
  STDIN_FILE=/dev/null
  STDOUT_FILE=ns3d.out.${GW_JOB_ID}
  STDERR_FILE=ns3d.err.{GW_JOB_ID}
# Performance Evaluation Parameters
  PROFILE_DFILE = perf_profile
  PERFORMANCE_EVALUATOR=pde.sh
  MAX_ITERATION_TIME=40
# Resource Selection Parameters
  HOST_REQUIREMENTS_FILE=host_req.ldif
  RANK_FUNCTION_FILE = rank.sh
  MAX_DISCOVERY_TIME=60
```

FIGURA B.1: Ejemplo de plantilla de trabajo (`job template`).

B.1.2. Parámetros de Planificación

- `MAX_RUNNING_TASKS`: Número máximo de tareas en ejecución.

B.1.3. Parámetros de Selección de Recursos

- `RESOURCE_SELECTOR`: Programa de selección de recursos.
- `HOST_REQUIREMENTS_FILE`: Fichero estático con la expresión de requisitos en formato de filtro de búsqueda LDAP [99].
- `HOST_REQUIREMENTS_DFILE`: Fichero dinámico con la expresión de requisitos.
- `RANK_FUNCTION_FILE`: Fichero estático con la expresión de clasificación como un guión que recibe la información de monitorización de los recursos como variables.
- `RANK_FUNCTION_DFILE`: Fichero dinámico con la expresión de clasificación.

- `MAX_DISCOVERY_TIME`: Intervalo de descubrimiento de recursos.

B.1.4. Parámetros de Evaluación del Rendimiento

- `PERFORMANCE_EVALUATOR`: Programa de evaluación del rendimiento.
- `MAX_ITERATION_TIME`: Tiempo máximo por iteración.
- `MAX_SUSPENSION_TIME`: Tiempo máximo de suspensión.

B.1.5. Parámetros de Tolerancia a Fallos

- `NUMBER_OF_RETRIES`: Número de reintentos tras un fallo.
- `ON_FAILURE`: Acción a realizar al agotarse los reintentos.

B.2. Variables de Entorno

A continuación se enumeran las variables de entorno que pueden usarse en la plantilla del trabajo, en los diferentes módulos (*resource selector*, *performance evaluator*, *prologue*, *wrapper* y *epilogue*) y en el propio trabajo.

- `GW_JOB_ID`: Identificador de trabajo.
- `GW_ARRAY_ID`: Identificador de array.
- `GW_TASK_ID`: Identificador de tarea.
- `GW_FILE_SERVER`: Servidor de ficheros local.
- `GW_FILE_PROXY`: Servidor de ficheros del recurso remoto.
- `GW_PREV_FILE_PROXY`: Servidor de ficheros del recurso desde donde se migró.
- `GW_RESTARTED`: Número de veces que el trabajo ha sido reiniciado.
- `GW_DONE`: Indica si el trabajo ha finalizado o si se ha parado.
- `GW_ARCH`: Arquitectura del recurso de ejecución.
- `GW_USER`: Nombre de usuario.
- `GW_DIR`: Directorio de experimento.

B.3. Interfaz de Línea de Comandos

B.3.1. gwd

```
GridWay command reference                                    gwd(1)

NAME
     gwd - GridWay daemon

SYNOPSIS
     gwd

DESCRIPTION
     The GridWay gwd utility launches the GridWay daemon.  All of
     the  other GridWay commands and API functions require having
     the daemon running.  At startup, the program will attempt to
     read ~/gwd.conf where you can configure the daemon according
     to your work load.  If the  file  cannot  be  read,  default
     values will be used.
     You can adjust the value of the following variables, all  of
     them are non-negative integers except for the log and script
     files that must contain the full path to  the  corresponding
     files.

     SCHEDULING_INTERVAL
          Number of seconds between scheduling cycles

     PERFORMANCE_INTERVAL
          Number of seconds between executions of the performance
          degradation evaluator script

     MAX_NUMBER_OF_JOBS
          Size of the job list
```

MAX_NUMBER_OF_ARRAYS
: Size of the array list

MAX_NUMBER_OF_SCHEDULINGS
: Number of individual jobs or arrays scheduled in each scheduling cycle.
: An array is treated as a scheduling unit, due to the fact that all of the jobs in the array share the same job template and therefore a single resource selection is needed, keeping track of the filled processing slots used.

MAX_NUMBER_OF_RESOURCES
: Size of the list of resources returned by the resource selector script

GWD_LOG
: Daemon's log file. If no value is specified, the output will appear in stderr(2) stream.

RS
: Default resource selector used when it is not specified in the job template file

PDE
: Default performance degradation evaluator script used when it is not specified in the job template file

PROLOGUE
: Default prologue script used when it is not specified in the job template file

WRAPPER
: Default wrapper script used when it is not specified in the job template file

```
     EPILOGUE
          Default epilogue script used when it is  not  specified
          in the job template file

EXAMPLES
     gwd

FILES
     ~/gwd.conf
     The file should look like this
     SCHEDULING_INTERVAL=10
     PERFORMANCE_INTERVAL=5
     MAX_NUMBER_OF_JOBS=200
     MAX_NUMBER_OF_ARRAYS=10
     MAX_NUMBER_OF_SCHEDULINGS=5
     MAX_NUMBER_OF_RESOURCES=100
     GWD_LOG=$HOME/gwd.log
     RS=$HOME/GridWay/scripts/rs.sh
     PDE=$HOME/GridWay/scripts/pde.sh
     PROLOGUE=$HOME/GridWay/scripts/prolog.sh
     EPILOGUE=$HOME/GridWay/scripts/epilog.sh
     WRAPPER=$HOME/GridWay/scripts/wrapper.sh

BUGS
     Please report all bugs to gridway@dacya.ucm.es, include core
     dumps.

SEE ALSO
     commands
          gwkill(1),  gwhistory(1),  gwps(1),  gwwait(1),  gwsub-
          mit(1)

     API  gwkill(3),  gwhistory(3),  gwps(3),  gwwait(3),  gwsub-
          mit(3)
```

B.3.2. gwsubmit

```
GridWay command reference                              gwsubmit(1)

NAME
     gwsubmit - submits a job or an array of jobs

SYNOPSIS
     gwsubmit -t jobTemplate [-n numTasks ]

DESCRIPTION
     The GridWay gwsubmit utility submits jobs. The job  template
     should exist and have the proper format.

OPTIONS
     The following options are supported for  the  gwsubmit  com-
     mand:

     -t jobTemplate
          mandatory option: specifies the job's template.

     -n numTasks
          specifies the size of the array job

EXAMPLES
     gwsubmit -t ls_exp/job_template
     gwsubmit -t sleep_exp/job_template -n 5

SEE ALSO
     commands
          gwkill(1), gwhistory(1), gwps(1), gwwait(1), gwd(1)
```

```
     API  gwkill(3),  gwhistory(3),  gwps(3),  gwwait(3),  gwsub-
          mit(3)
```

B.3.3. gwwait

```
GridWay command reference                                gwwait(1)

NAME
     gwwait - waits on jobs

SYNOPSIS
     gwwait jobId | -A arrayId | -l jobIdList | -1 jobIdList

DESCRIPTION
     The GridWay gwwait utility is used to synchronize job execu-
     tion.  The  ids should be valid job identifiers returned by
     gwsumbit. The command will not return  until  the  wait  has
     completed.

OPTIONS
     The following options are supported for the gwwait command:

     -A arrayId
          wait for all of the jobs in the array to finish

     -l jobIdList
          Wait all: doesn't return until all  of  the  jobs  have
          finished.

     -1 jobIdList
          Wait first: waits until one of the  jobs  in  the  list
          finishes
```

```
EXAMPLES
     gwwait 5
          waits on job 5

     gwwait -A 1
          waits on array 1

     gwwait -a 2 12 1 24
          waits on all of the jobs in the list.

     gwwait -1 7 8 12 33 23 1 17
          wait until the first job from the list finishes.

NOTES
     Do not perform more than wait on a job at a  time,  as  they
     can interfere.

SEE ALSO
     commands
          gwkill(1), gwhistory(1), gwps(1), gwd(1), gwsubmit(1)

     API  gwkill(3),  gwhistory(3),  gwps(3),  gwwait(3),  gwsub-
          mit(3)
```

B.3.4. gwps

```
GridWay command reference                                gwps(1)

NAME
     gwps - report job status

SYNOPSIS
     gwps [-d delaySeconds [-#] ]
```

```
DESCRIPTION
     The GridWay gwps utility  prints  information  about  active
     jobs.

     JID  job id returned by gwsubmit, this is the  job's  handle
          for all of the GridWay commands

     AID  array id returned by gwsubmit  if  the  -n  option  was
          used, "--" if the job doesn't belong to an array

     TID  task id, identifies a job within an array, "--" if  the
          job doesn't belong to an array

     DM   dispatch manager state

     SM   submission manager state

     GSM  globus state

     STIME
          submission time (hr:min:sec)

     ETIME
          exit time (hr:min:sec)

     CPUTIME
          execution time (min:sec), can include some overhead

     XFRTIME
          transfer time (min:sec).  Time  taken  to  stage  files
          (prologue script) and retrieve output (epilogue script)

     EXIT job's exit code, captured by the wrapper script
```

```
     TEMPLATE
          job's template file as used in gwsubmit

     HOST host where the job has been submitted to. Use the
          gwkill command with the -m option to request job migra-
          tion, or let the performance degradation evaluator
          script decide when to migrate

OPTIONS
     The following options are supported for the gwps command:

     -d delay
          stay on top with refresh rate delay seconds: displays
          job status, then sleeps delay seconds and re-displays
          job status. This carries on until gwd is closed or the
          gwps process is terminated.

     -#   Clear screen and move to the top left corner before
          printing status.

EXAMPLES
     gwps prints out job status once

     gwps -d 2
          prints out job status every 2 seconds

     gwps -#d 1
          clears the screen before each output, every second.

SEE ALSO
     commands
          gwkill(1), gwhistory(1), gwd(1), gwwait(1), gwsubmit(1)
```

API gwkill(3), gwhistory(3), gwps(3), gwwait(3), gwsubmit(3)

B.3.5. gwhistory

GridWay command reference gwhistory(1)

NAME

gwhistory - prints out the job's history

SYNOPSIS

gwhistory jobId

DESCRIPTION

The GridWay gwhistory utility prints the job's history. The jobId should be a valid job identifier returned by gwsumbit. The command's output is a table, where each row shows the job's history, from oldest to most recent.

HOST host where the job was sent to

RANK value evaluated by the rank script

STIME

submission time (hr:min:sec)

ETIME

exit time (hr:min:sec)

EXETIME

execution time (min:sec)

MIGRATION REASON

```
          description of what forced the migration

EXAMPLES
     gwhistory 14
          get job 14's history

SEE ALSO
     gwkill(1), gwps(1), gwsubmit(1), gwwait(1), gwd(1)
```

B.3.6. gwkill

```
GridWay command reference                                gwkill(1)

NAME
     gwkill - terminate or signal jobs

SYNOPSIS
      gwkill [-a] [-k [-A][-9] | -s  | -r  | -m] jobId|arrayId

DESCRIPTION
     The GridWay gwkill utility sends a signal to the job or
     array specified by the jobId or arrayId operand. This id
     should be a valid identifier returned by gwsumbit.

OPTIONS
     The following options are supported for the gwkill command:

     -a   asynchronous message: the message is delivered and the
          command returns immediately

     -k   kill: abort the job and free it's resources.
          If the -A option is used, the array corresponding to
          the arrayId operand will be freed
```

```
          If the -9 option is used, the  job  or  array  will  be
          freed immediately

     -s   stop: stop job execution

     -r   resume: resume job execution

     -m   migrate: find a new host for job execution

     The default signal, when no others have been specified is -k

EXAMPLES
     gwkill 5
          kills  job  5  synchronously  (command  doesn't  return
          immediately)

     gwkill -k 5
          same as above

     gwkill -s 3
          stops  job  3  synchronously  (command  doesn't  return
          immediately)

     gwkill -ra 2
          resumes  job  2's  execution   synchronously   (command
          returns immediately)

     gwkill -ma 1
          requests  job  1's  migration  synchronously   (command
          returns immediately)

     gwkill -kA 5
          kills array 5 synchronously
```

```
     gwkill -k9 5
          kills job 5 immediately

SEE ALSO
     commands
          gwd(1), gwhistory(1), gwps(1), gwwait(1), gwsubmit(1)

     API  gwkill(3),  gwhistory(3),  gwps(3),  gwwait(3),  gwsub-
          mit(3)
```

B.4. Interfaz de Programación de Aplicaciones

B.4.1. gwsubmit

```
GridWay API reference                                  gwsubmit(3)

NAME
     gwsubmit - submits a job or an array of jobs

SYNOPSIS
     #include "gw_client.h"

     gw_return_code_t gw_client_job_submit( char  *template,  int
     *job_id);

     gw_return_code_t gw_client_job_submit_array( char *template,
     int tasks, int *array_id, int **job_ids);

DESCRIPTION
     gw_client_job_submit
          Submits a single job via the request manager.
          The template parameter should point to  a  string  con-
          taining the path to a job template file.
```

```
          The job_id pointer will hold the  job's  identifier  if
          the call is successful or -1 otherwise.

     gw_client_job_submit_array
          Submits an array job via the request manager.
          The template parameter should point to  a  string  con-
          taining the path to a job template file.
          The tasks parameter must hold the  size  of  the  array
          job.
          The array_id parameter will hold the array's identifier
          if the call is successful or -1 otherwise.
          The job_ids parameter will point to a malloc'd array of
          job identifiers corresponding to each task.  Use free()
          when the array is no longer needed.

RETURN VALUES
     returns a gw_return_code_t code with one  of  the  following
     values

     GW_RC_SUCCESS
          no error

     GW_RC_FAILED
          could not submit

     GW_FAILED_JOB_INIT
          failed job initialization, check job template

     GW_FAILED_NO_MEMORY
          cannot submit any more jobs/arrays

     You can use

     #include "gw_types.h"
```

```
     char *gw_ret_code_string(gw_return_code_t code);

     to decode the return code

SEE ALSO
     commands
          gwd (1), gwkill(1), gwhistory(1),  gwps(1),  gwwait(1),
          gwsubmit(1)

     API  gwkill(3), gwps(3), gwwait(3), gwhistory(3)
```

B.4.2. gwwait

```
GridWay API reference                                    gwwait(3)

NAME
     gwwait - waits on jobs

SYNOPSIS
     #include "gw_client.h"

     gw_return_code_t   gw_client_job_wait(   int   job_id,   int
     *exit_code);

     void gw_client_job_wait_set( int *job_ids, int  *exit_codes,
     gw_return_code_t *return_codes, int num_jobs);

     int   gw_client_job_wait_any_set(    int    *job_ids,    int
     *exit_code, gw_return_code_t *return_code, int num_jobs);

     gw_return_code_t  gw_client_wait_array(  int  array_id,  int
     **job_ids,     int     **task_ids,     int     **exit_codes,
     gw_return_code_t **return_codes, int *num_tasks);
```

DESCRIPTION

The GridWay gwwait family is used to synchronize job execution. The ids should be valid job and array identifiers returned by gwsumbit command or API function. The functions will not return until the wait has completed.

gw_client_job_wait

Waits on a single job, as determined by the job_id argument. exit_code will hold the job's exit code.

gw_client_job_wait_set

Doesn't return until all of the jobs in the job_ids list have finished.

gw_client_job_wait_any_set

Doesn't return until one of the jobs in the list finishes (the first one to do so).

gw_client_wait_array

Waits for the array determined by the array_id argument. job_ids, task_ids, exit_codes and return_codes are pointers to arrays that will automatically be allocated, use malloc when they are no longer needed. num_tasks will hold the size of the lists.

NOTES

Do not perform more than wait on a job at a time, as they can interfere.

RETURN VALUES

returns a gw_return_code_t code with one of the following values

```
     GW_RC_SUCCESS
          no error

     GW_RC_FAILED_BAD_JOB_ID
          job doesn't exist

     You can use

     #include "gw_types.h"
     char *gw_ret_code_string(gw_return_code_t code);

     to decode the return code

SEE ALSO
     commands
          gwd (1), gwkill(1), gwhistory(1),  gwps(1),  gwwait(1),
          gwsubmit(1)

     API  gwkill(3), gwps(3), gwhistory(3), gwsubmit(3)
```

B.4.3. gwps

```
GridWay API reference                                    gwps(3)

NAME
     gwps - get job status

SYNOPSIS
     #include "gw_client.h"

     gw_return_code_t gw_client_job_status(int  job_id,  gw_msg_t
     *status);
```

```
     gw_return_code_t     gw_client_job_status_all(      gw_msg_t
     **status_list, int *num_records);

DESCRIPTION
     gw_client_job_status
          If successful, the status  argument  will  contain  the
          status record of job job_id

     gw_client_job_status_all
          If successful, the status_list argument will point to a
          malloc'd  array  containing a status record for each of
          the jobs that are currently in  the  dispatch  manager.
          The  num_records  argument  will  contain the length of
          this list.

SEE ALSO
     commands
          gwd (1), gwkill(1), gwhistory(1),  gwps(1),  gwwait(1),
          gwsubmit(1)

     API  gwkill(3), gwhistory(3), gwwait(3), gwsubmit(3)
```

B.4.4. gwhistory

```
GridWay API reference                                  gwhistory(3)

NAME
     gwhistory - get job's history records

SYNOPSIS
     #include "gwclient.h"

     gw_return_code_t gw_client_job_history( int job_id, gw_msg_t
```

```
     **history_list, int *num_records);

DESCRIPTION
     The GridWay gwhistory utility requires as input argument the
     job_id  of  the  job, as returned by gwsubmit command or API
     function.

     If the call is successful, history list  will  point  to  an
     allocated  array  containing  the  jobs history records from
     oldest to most recent.  Use free() when you're done with the
     array.
     num_records will hold the length of the array

RETURN VALUES
     returns a gw_return_code_t code with one  of  the  following
     values

     GW_RC_SUCCESS
          no error

     GW_RC_FAILED_BAD_JOB_ID
          job doesn't exist

     You can use

     #include "gw_types.h"
     char *gw_ret_code_string(gw_return_code_t code);

     to decode the return code

SEE ALSO
     commands
          gwd (1), gwkill(1), gwhistory(1),  gwps(1),  gwwait(1),
          gwsubmit(1)
```

```
     API  gwkill(3), gwps(3), gwwait(3), gwsubmit(3)
```

B.4.5. gwkill

```
GridWay API reference                                    gwkill(3)

NAME
     gwkill - terminate or signal jobs

SYNOPSIS
     #include "gw_client.h"

     gw_return_code_t gw_client_job_kill(int job_id, gw_boolean_t
     async, gw_boolean_t hard);

     gw_return_code_t     gw_client_array_kill(int      array_id,
     gw_boolean_t async, gw_boolean_t hard);

     gw_return_code_t gw_client_job_stop(int job_id, gw_boolean_t
     async);

     gw_return_code_t      gw_client_job_resume(int       job_id,
     gw_boolean_t async);

     gw_return_code_t    gw_client_job_reschedule(int     job_id,
     gw_boolean_t async);

DESCRIPTION
     The GridWay gwkill family  require  as  input  argument  the
     job_id  of  the  job, as returned by gwsubmit command or API
     function.
     The async argument determines if the  call  is  asynchronous
```

```
     and  therefore  doesn't wait for a response from the request
     manager/dispatch manager The hard argument determines if the
     job or array will be killed immediately or not.

RETURN VALUES
     returns a gw_return_code_t code with one  of  the  following
     values

     GW_RC_SUCCESS
          no error

     GW_RC_FAILED
          failed

     GW_FAILED_BAD_JOB_ID
          the job doesn't exist

     You can use

     #include "gw_types.h"
     char *gw_ret_code_string(gw_return_code_t code);

     to decode the return code

SEE ALSO
     commands
          gwd(1), gwkill(1),  gwhistory(1),  gwps(1),  gwwait(1),
          gwsubmit(1)

     API  gwhistory(3), gwps(3), gwwait(3), gwsubmit(3)
```

Índice alfabético

Bibliografía

[1] M. P. Forteza. Evolución del Cultivo del Almendro en Castilla-La Mancha Durante el Periodo 1992-2002. Trabajo Fin de Carrera. Universidad de Castilla-La Mancha, 2003.

[2] M. Baker (editor). Cluster Computing White Paper. Informe técnico, IEEE Task Force on Cluster Computing (TFCC), diciembre de 2000. Versión 2.0 disponible en http://www.ieeetfcc.org.

[3] MOSIX. http://www.mosix.cs.huji.ac.il.

[4] Portable Batch System. http://www.openpbs.org.

[5] Sun Grid Engine. http://www.sun.com/gridware.

[6] Load Sharing Facility. http://www.platform.com/products/wm/LSF.

[7] Condor. http://www.cs.wisc.edu/condor.

[8] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[9] A. Chien. Parallel Programming Challenges for Internet-Scale Computing (Entropia). En *Proceedings of the 8th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, página 82. ACM Press, 2001.

[10] A. Chien, B. Calder, S. Elbert, K. Bhatia. Entropia: Architecture and Performance of an Enterprise Desktop Grid System. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.

[11] The SETI@home Project. http://setiathome.ssl.berkeley.edu.

[12] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, Dan Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11):56–61, 2002.

[13] I. Foster, C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, 1999.

[14] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[15] I. Foster. What Is the Grid? A Three Point Checklist. *GRIDtoday*, 1(6), 2002. Disponible en http://www.gridtoday.com/02/0722/100136.html.

[16] W. Gentzsch. Response to Ian Foster's "What Is the Grid". *GRIDtoday*, 1(8), 2002. Disponible en http://www.gridtoday.com/02/0805/100191.html.

[17] The Global Grid Forum. http://www.gridforum.org.

[18] J. M. Schopf, B.Nitzberg. Grids: The Top Ten Questions. *Scientific Programming, special issue on Grid Computing*, 10(2):103–111, agosto de 2002.

[19] Legion. http://www.cs.virginia.edu/∼legion.

[20] Polder. http://www.science.uva.nl/research/scs/PSCS4.html.

[21] MOL. http://www.uni-paderborn.de/pc2/projects/mol.

[22] The Globus Project. http://www.globus.org.

[23] I. Foster, C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[24] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. En *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*, páginas 83–92, 1998.

[25] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. Disponible en http://www.cacr.math.uwaterloo.ca/hac.

[26] ITU-T Recommendation X.509: Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, 1997.

[27] T. Dierks, C. Allen. RFC 2246: The TLS Protocol Version 1.0, enero de 1999. Status: STANDARDS TRACK.

[28] J. Linn. RFC 2743: Generic Security Service API Version 2, Update 1, enero de 2000. Status: STANDARDS TRACK.

[29] J. Wray. RFC 2744: Generic Security Service API Version 2, C-Bindings, enero de 2000. Status: STANDARDS TRACK.

[30] S. Meder, V. Welch, S. Tuecke, D. Engert. GSS-API Extensions. Informe técnico GFD.24, Grid Security Working Group – The Global Grid Forum, 2001.

[31] R. Housley, W. Polk, W. Ford, D. Solo. RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, abril de 2002. Status: STANDARDS TRACK.

[32] ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, 2002.

[33] J. Linn. RFC 1421: Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures, febrero de 1993. Status: STANDARDS TRACK.

[34] S. Tuecke, V. Welch, D. Engert, L. Pearlman, M. Thompson. RFC 3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile, junio de 2004. Status: STANDARDS TRACK.

[35] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. En *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, páginas 62–82, 1998.

[36] J. Bester, I. Foster, C. Kesselman, J. Tedesco, S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. En *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, mayo de 1999.

[37] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A.Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernández, J. Magowan, N. Bieberstein. *Introduction to Grid Computing with Globus*. IBM Redbooks, septiembre de 2003. Disponible en http://www.redbooks.ibm.com.

[38] K. Czajkowski, I. Foster, C. Kesselman. Resource Co-Allocation in Computational Grids. En *Proceedings of the 8th IEEE Symposium on High Performance Distributed Computing (HPDC8)*, páginas 219–228, 1999.

[39] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. En *Proceedings of the 6th IEEE Symposium on High Performance Distributed Computing (HPDC6)*, páginas 365–375, 1997.

[40] M. Wahl, T. Howes, S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3), diciembre de 1997. Status: STANDARDS TRACK.

[41] G. Good. RFC 2849: The LDAP Data Interchange Format (LDIF) - Technical Specification, junio de 2000. Status: STANDARDS TRACK.

[42] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, M. Wilde. Globus Toolkit Support for Distributed Data-Intensive Science. En *Proceedings of Computing in High Energy Physics (CHEP 01)*, 2001.

[43] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V.Nefedova, D. Quesnal, S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, 28(5):749–771, mayo de 2002.

[44] B. Allcock, I. Foster, V.Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, D. Williams. High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. En *Proceedings of the SuperComputing Conference (SC01)*, noviembre de 2001.

[45] W. Allcock (editor). GridFTP: Protocol Extensions to FTP for the Grid. Informe técnico GFD.20, GridFTP Working Group – The Global Grid Forum, 2003. Disponible en http://www.ggf.org/documents.

[46] I. Mandrichenko (editor). GridFTP Protocol Improvements. Informe técnico GFD.21, GridFTP Working Group – The Global Grid Forum, 2003. Disponible en http://www.ggf.org/documents.

[47] J.Postel, J.Reynolds. RFC 959: File Transfer Protocol (FTP), otubre de 1985. Status: STANDARDS TRACK.

[48] I. Foster, C. Kesselman, J.Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Informe técnico, Open Grid Service Infrastructure Working Group – The Global Grid Forum, 2002.

[49] I. Foster, C. Kesselman, J.Nick, S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6), 2002.

[50] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0. Informe técnico, Open Grid Service Infrastructure Working Group – The Global Grid Forum, 2003.

[51] B. Sotomayor. The Globus Toolkit 3 Programmer's Tutorial, 2004. Disponible en http://www.casa-sotomayor.net/gt3-tutorial.

[52] I. Foster, J. Frey, S. Graham et al. Modeling Stateful Resource with Web Services. Disponible en http://www.globus.org/wsrf, marzo de 2004.

[53] K. Czajkowski, D. F. Ferguson, I. Foster et al. The WS-Resource Framework. Disponible en http://www.globus.org/wsrf, marzo de 2004.

[54] M. Baker, R. Buyya, D. Laforenza. Grids and Grid Technologies for Wide-Area Distributed Computing. *Software: Practice and Experience*, 32(15):1437–1466, 2002.

[55] R. Moreno, A. B. Alonso-Conde. Job Scheduling and Resource Management Techniques in Economic Grid Environments. En *Proceedings of the 1st European Across Grids Conference (Across Grids 2003)*, volumen 2970 de *Lecture Notes in Computer Science*, páginas 25–32. Springer-Verlag, 2004.

[56] G. Allen, E. Seidel, J. Shalf. Scientific Computing on the Grid. *Byte*, Spring 2002:24–32, 2002.

[57] Self-Adaptive Numerical Software. http://icl.cs.utk.edu/iclprojects/pages/sans.

[58] J. M. Schopf. Ten Actions when Superscheduling. Informe técnico GFD.4, Scheduling Working Group – The Global Grid Forum, 2001. Disponible en http://www.ggf.org/documents.

[59] J. M. Schopf. A General Architecture for Scheduling on the Grid. Informe técnico ANL/MCS-P1000-10002, Argonne National Laboratory, 2002. Enviado a Journal of Parallel and Distributed Computing, special issue on Grid Computing. Disponible en http://www-unix.mcs.anl.gov/∼schopf.

[60] J. M. Schopf. Ten Actions when Grid Scheduling: The User as Grid Scheduler. En J.Nabrzyski, J. M. Schopf, J. Weglarz, editores, *Grid Resource Management: State of the Art and Future Trends*, capítulo 2. Kluwer Academic, 2003.

[61] R. Raman, M. Livny, M. Solomon. Matchmaking: An Extensible Framework for Distributed Resource Management. *Cluster Computing*, 2(2):129–138, 1999.

[62] E. Huedo, R. S. Montero, I. M. Llorente. Experiences on Grid Resource Selection Considering Resource Proximity. En *Proceedings of the 1st European Across Grids Conference (Across Grids 2003)*, volumen 2970 de *Lecture Notes in Computer Science*, páginas 1–8. Springer-Verlag, 2004.

[63] C. Liu, L. Yang, I. Foster, D. Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. En *Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC11)*, 2002.

[64] R. Brobst, W. Chan, F. Ferstl, J. Gardiner, J. P. Robarts, A. Haas, B.Ñitzberg, H. Rajic, J. Tollefsrud. Distributed Resource Management Application API Specification 1.0. Informe técnico GFD.22, DRMAA Working Group – The Global Grid Forum, 2004. Disponible en http://www.ggf.org/documents y en http://www.drmaa.org.

[65] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. M. Figueira, J. Hayes, G. Obertelli, J. M. Schopf, G. Shao, S. Smallen, N. T. Spring, A. Su, D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(5):369–382, 2003.

[66] R. Buyya, D. Abramson, J. Giddy. Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid. En *Proceedings of the 4th IEEE International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia)*, 2000.

[67] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. En *Proceedings of the 9th Heterogeneous Computing Workshop (HCW2000)*, mayo de 2000.

[68] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf. The Cactus Worm: Experiments with Dynamic Resource Selection and Allocation in a Grid Environment. *International Journal of High-Performance Computing Applications*, 15(4), 2001.

[69] R. S. Montero, E. Huedo, I. M. Llorente. Experiences about Job Migration on a Dynamic Grid Environment. En *Proceedings of the 5th International Conference on Parallel Computing (ParCo 2003)*, volumen 13 de *Advances in Parallel Computing*. Elsevier Science, otubre de 2004.

[70] G. Lanfermann, G. Allen, T. Radke, E. Seidel. Nomadic Migration: A New Tool for Dynamic Grid Computing. En *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, agosto de 2001.

[71] S. Vadhiyar, J. Dongarra. A Performance Oriented Migration Framework for the Grid. En *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, 2003.

[72] A. DeVivo, M. Yarrow, K. M. McCann. A Comparison of Parameter Study Creation and Job Submission Tools. Informe técnico NAS-01-002, NASA Advanced Supercomputing (NAS), NASA Ames Research Center, Moffett Field, CA, 2001. Disponible en http://www.nas.nasa.gov/Research/Reports/Techreports/2001.

[73] M. Yarrow, K. M. McCann, R. Biswas, R. F. Van der Wijngaart. ILab: An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid. En *Proceedings of the 1st International Workshop on Grid Computing (GRID 2000)*, 2000.

[74] E. Huedo, R. S. Montero, I. M. Llorente. Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. En *Proceedings of the 12nd International Conference on Parallel, Distributed and Network based Processing (PDP 2004)*, páginas 28–33. IEEE Computer Society Press, febrero de 2004.

[75] E. Huedo, R. S. Montero, I. M. Llorente. Adaptive Scheduling and Execution on Computational Grids. *Journal of Supercomputing*, 2004. (en prensa).

[76] E. Huedo, R. S. Montero, I. M. Llorente. A Framework for Adaptive Execution on Grids. *Software: Practice and Experience*, 34(7):631–651, junio de 2004.

[77] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke. Condor/G: A Computation Management Agent for Multi-Institutional Grids. En *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, agosto de 2001.

[78] The European DataGrid Project. http://www.eu-datagrid.org.

[79] F. Giacomini, Francesco Prelz. Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description. Disponible en http://www.infn.it/workload-grid/documents.html, 2001.

[80] Enabling Grids for E-science in Europe. http://www.eu-egee.org.

[81] R. Buyya, J. Giddy, D. Abramson. A Case for Economy Grid Architecture for Service-Oriented Grid Computing. En *Proceedings of the 10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, abril de 2001.

[82] R. Buyya, D.Abramson, J. Giddy. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. *Future Generation Computer Systems*, 2002.

[83] R. Buyya, K. Branson, J. Giddy, D. Abramson. The Virtual Laboratory: A Toolset for Utilising the World-Wide Grid to Design Drugs. En *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, mayo de 2002.

[84] The AppLeS Project. http://apples.ucsd.edu.

[85] H. Casanova, G. Obertelli, F. Berman, R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. En *Proceedings of the SuperComputing Conference (SC00)*, 2000.

[86] G. Shao, F. Berman, R. Wolski. Master/Slave Computing on the Grid. En *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*, mayo de 2000.

[87] The GrADS Project. http://hipersoft.cs.rice.edu/grads.

[88] F. Berman, A. Chien, K. Cooper et al. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15(4):327–34, 2001.

[89] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torezon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt, D. A. Reed, J. Dongarra, S. Vadhiyar, R. Wolski. Toward a Framework for Preparing and Executing Adaptive Grid Applications. En *Proceedings of the NSF Next Generation Systems Program Workshop, International Parallel and Distributed Processing Symposium*, abril de 2002.

[90] F. Vraalsen, R. A. Aydt, C. L. Mendes, D. A. Reed. Performance Contracts: Predicting and Monitoring Grid Application Behavior. En *Proceedings of the 2nd International Workshop on Grid Computing (GRID 2001)*, noviembre de 2001.

[91] The ScaLAPACK Project. http://www.netlib.org/scalapack.

[92] A. Petitet, S. Balckford, J. Dongarra et al. Numerical Libraries and the Grid: The GrADS Experiments with ScaLAPACK. *International Journal of High Performance Computing Applications*, 15(4):359–374, 2001.

[93] The Cactus Code Server. http://www.cactuscode.org.

[94] The GridLab Project. http://www.gridlab.org.

[95] E. Seidel, G. Allen, A. Merzky, J.Nabrzyski. GridLab: A Grid Application Toolkit and Testbed. *Future Generation Computer Systems*, 18(8):1143–1153, 2002.

[96] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel. The Cactus Code: A Problem Solving Environment for the Grid. En *Proceedings of the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9)*, mayo de 2000.

[97] The Triana Workflow. http://www.triana.co.uk.

[98] E. Huedo, R. S. Montero, I. M. Llorente. An Experimental Framework for Executing Applications in Dynamic Grid Environments. Informe técnico 2002-43, Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hamptom VA, noviembre de 2002.

[99] T. Howes. RFC 2254: The String Representation of LDAP Search Filters, diciembre de 1997. Status: STANDARDS TRACK.

[100] Hierarchical Data Format Version 5. http://hdf.ncsa.uiuc.edu/HDF5.

[101] S. Vazhkudai, S. Tuecke, I. Foster. Replica Selection in the Globus Data Grid. En *Proceedings of International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001)*. IEEE Computer Society Press, 2001.

[102] R. S. Montero, I. M. Llorente, M. D. Salas. Robust Multigrid Algorithms for the Navier-Stokes Equations. *Journal of Computational Physics*, 173:412–432, 2001.

[103] M. Prieto, R. Santiago, I. M. Llorente, F. Tirado. A Multigrid Solver for the Incompressible Navier-Stokes Equations on a Beowulf-class System. En *Proceedings of the 30th. International Conference on Parallel Processing (ICPP01)*, 2001.

[104] The NorduGrid Project. http://www.nordugrid.org.

[105] M. Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 1995.

[106] R. S. Montero, E. Huedo, I. M. Llorente. Grid Resource Selection for Opportunistic Job Migration. En *Proceedings of the 9th International Conference on Parallel and Distributed Computing (Euro-Par 2003)*, volumen 2790 de *Lecture Notes in Computer Science*, páginas 366–373. Springer-Verlag, agosto de 2003.

[107] A. Fuentes, E. Huedo, R. S. Montero, I. M. Llorente. A Grid Scheduling Algorithm Considering Dynamic Interconnecting Network Quality. En *Proceedings of the 3rd Cracow'03 Grid Workshop*, páginas 151–158. Academic Computer Centre CYFRONET AGH, otubre de 2003.

[108] Iperf: The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf.

[109] C. H. Hsu, U. Kremer. IPERF: A Framework for Automatic Construction of Performance Prediction Models. En *Workshop on Profile and Feedback-Directed Compilation (PFDC)*, otubre de 1998.

[110] R. Wolski, N. Spring, J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, 15(5-6):757–768, 1999.

[111] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, 1998.

[112] S. Vazhkudai, J. Schopf, I. Foster. Predicting the Performance of Wide-Area Data Transfers. En *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.

[113] E. Huedo, R. S. Montero, I. M. Llorente. Adaptive Grid Scheduling of a High-Throughput Bioinformatics Application. En *Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM 2003)*, volumen

3019 de *Lecture Notes in Computer Science*, páginas 840–847. Springer-Verlag, abril de 2004.

[114] E. Huedo, R. S. Montero, I. M. Llorente. The GridWay Framework for Adaptive Scheduling and Execution on Grids. En *Proceedings of Workshop on Adaptive Grid Middleware (AGridM 2003), in conjunction with the 12th Conference on Parallel Architectures and Compilation Techniques (PACT 2003)*, Journal of Parallel and Distributed Computing Practices. Nova Science, 2004. (en prensa).

[115] E. Huedo, U. Bastolla, R. S. Montero, I. M. Llorente. Computational Proteomics on the Grid. *Journal of New Generation Computing, special issue on Grid Systems for Life Sciences*, 22(2):191–192, febrero de 2004.

[116] U. Bastolla, E.W. Knapp, M. Vendruscolo. How to Guarantee Optimal Stability for Most Protein Native Structures in the Protein Data Bank. *Proteins: Structure, Function and Genetics*, 44:79–96, 2001.

[117] 5th round of Critical Assessment of techniques for protein Structure Prediction. http://PredictionCenter.llnl.gov/casp5.

[118] R. van Ham, J. Kamerbeek, C. Palacios, C. Rausell, F. Abascal, U. Bastolla, J.M. Fernandez, L. Jimenez, M. Postigo, F.J. Silva, J. Tamames, E. Viguera, A. Latorre, A. Valencia, F. Moran, A. Moya. Reductive Genome Evolution in Buchnera Aphidicola. *Proceedings of the National Academy of Sciences, USA*, 100:581–586, 2003.

[119] U. Bastolla, A. Moya, E. Viguera, R. van Ham. Genomic Determinants of Protein Folding Thermodynamics. (en preparación), 2004.

[120] J. Herrera, E. Huedo, R. S. Montero, I. M. Llorente. Developing Grid-Aware Applications with DRMAA on Globus-Based Grids. En *Proceedings of 10th International Conference on Parallel and Distributed Computing (Euro-Par 2004)*, volumen 3149 de *Lecture Notes in Computer Science*, páginas 429–435, 2004.

[121] J. Herrera, R. S. Montero, E. Huedo, I. M. Llorente. DRMAA Implementation within the GridWay Framework. En *Proceedings of the 12th Global Grid Forum (GGF12)*, 2004.

[122] Grid Benchmarking Research Group – The Global Grid Forum. http://www.nas.nasa.gov/GGF/Benchmarks, 2004.

[123] A. Snavely, G. Chun, H. Casanova, R. F. Van der Wijngaart, M. A. Frumkin. Benchmarks for Grid Computing: A Review of Ongoing Efforts and Future Directions. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):27–32, 2003.

[124] R. F. Van der Wijngaart, M. A. Frumkin. NAS Grid Benchmarks Version 1.0. Informe técnico NAS-02-005, NASA Advanced Supercomputing (NAS) Division, NASA Ames Research Center, Moffett Field, CA, 2002.

[125] M. A. Frumkin, R. F. Van der Wijngaart. NAS Grid Benchmarks: A Tool for Grid Space Exploration. *Journal of Cluster Computing*, 5(3):247–255, 2002.

[126] R. Wolski, J. Brevik, G. Obertelli, N. Spring, A. Su. Writing Programs that Run EveryWare on the Computational Grid. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1066–1080, 2001.

[127] D. H. Bailey, E. Barszcz, J. T. Barton. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, 1991.

[128] E. Deelman, J. Blythe, Y. Gil et al. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[129] Distributed Resource Management Application API Working Group – The Global Grid Forum. http://www.drmaa.org, 2004.

[130] N. Karonis, B. Toonen, I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63:551–563, 2003.

[131] E. Huedo, A. Lepinette, R. S. Montero, I. M. Llorente, L. Vázquez. Simulations of Mars Impact Cratering on a Grid Environment. En *Proceedings of the 1st International Workshop on Grid Computing and Its Application to Data Analysis (GADA'04)*, Lecture Notes in Computer Science. Springer-Verlag, 2004. (en prensa).

[132] Tidewater Research Grid Partnership. http://www.tidewaterrgp.org.

[133] SciClone Cluster Project. http://www.compsci.wm.edu/SciClone.

[134] Coral Cluster Project. http://www.icase.edu/CoralProject.html.

[135] RSA Laboratories. PKCS 12 v1.0: Personal Information Exchange Syntax, junio de 1999.

[136] S. Chokhan, W. Ford. RFC 2527: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework, marzo de 1999. Status: INFORMATIONAL.

[137] ITU-T Recommendation X.690: Information Technology - ASN.1 Enconding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 2002.