UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR DE ALBACETE

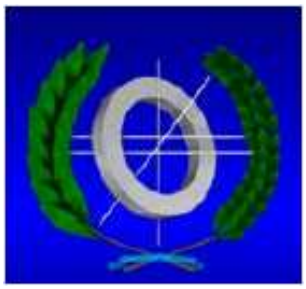
INGENIERÍA EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Implantación de un sistema Grid basado en GridWay en el Laboratorio RAAP del I3A

Autor: Iván Fernández Hernández

Diciembre, 2007



UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR DE ALBACETE

Departamento de Sistemas Informáticos

PROYECTO FIN DE CARRERA

Implantación de un sistema Grid basado en GridWay en el Laboratorio RAAP del I3A

Autor: Iván Fernández Hernández

Directores: Mª Carmen Carrión Espinosa y Mª Blanca Caminero Herráez

Diciembre, 2007

Quisiera dedicar este trabajo a mis padres y a mi hermana, por el apoyo que me han prestado siempre en todo lo que he hecho. Quiero agradecer a todos los compañeros del I3A la inestimable ayuda prestada durante la realización de mi Proyecto, y especialmente a Jesús Delicado Martínez y a Juan Antonio Villar Martín, sin los cuales, este documento no habría sido posible. También quiero agradecer la ayuda prestada por mi tutora, Mª Blanca Caminero Herráez, por las interminables horas dedicadas a corregir errores gramaticales para que este documento resultara medio decente, y a Mª Carmen Carrión Espinosa por contar conmigo para este Proyecto Fin de Carrera.

Resumen:
El objetivo de este proyecto es dotar al laboratorio de Redes de Altas Prestaciones (RAAP) de una infraestructura Grid basada en el metaplanificador de trabajos *GridWay*, y la capa base de servicios de *Globus Toolkit*. Se realizará el despliegue en los recursos disponibles por el laboratorio RAAP, de una capa base de servicios Grid. Esta capa de servicios será utilizada por el metaplanificador GridWay, que se encargará de gestionar los recursos adecuadamente en función de las necesidades de los usuarios. Una vez desplegada esta infraestructura, se establecerán conexiones con varios grupos de investigación con el fin de crear una federación Grid integrando recursos de las Universidades de: Murcia, Valencia y Politécnica de Valencia. Se evaluará la plataforma Grid conjunta mediante benchmarks específicos.
Se desarrollará una aplicación gráfica que permita diseñar flujos de trabajo (Work-Flows) y posteriormente generar scripts para el metaplanificador, con el fin de lanzar flujos de trabajo complejos, en el Grid desplegado.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción a la Computación Grid

Actualmente, a principios de este siglo XXI, el nivel de desarrollo tecnológico que ha alcanzado la civilización humana ha sido en gran medida, gracias a los grandes descubrimientos científicos y avances tecnológicos en todos los campos obtenidos en el siglo pasado. Uno de los campos de investigación más novedosos y que más frutos ha dado a lo largo de la segunda mitad del siglo pasado ha sido la Informática. Ésta ciencia sentó sus bases en el siglo XIX cuando George Boole inventó las bases lógicas de la computación. Fue más tarde, durante la Segunda Guerra Mundial, cuando fue impulsada por matemáticos como un intento desesperado de desencriptar las comunicaciones de los alemanes cifradas bajo el sistema *Enigma* y frustrar así sus planes de conquista. A lo largo de la segunda mitad del siglo XX, esta ciencia todavía en pañales, ha contribuido de forma única a la resolución de millones de problemas en diferentes ámbitos y disciplinas. Constituyendo hoy en día un motor de procesamiento y fuente de recursos del que no se puede prescindir. La informática siempre ha ayudado a la ciencia a progresar por campos no experimentales, mediante simulaciones a pequeña o a gran escala, manejando grandes volúmenes de información y realizando cálculos que incluso 25 años antes parecerían inabordables ni en varias vidas de hombre. Según iba creciendo el conocimiento, más cantidad de datos o más cantidad de procesamiento se requería para las simulaciones o para las herramientas software disponibles. En algunos casos, se ha tenido que reinventar el concepto de computación para poder abordar problemas que antes no tenían solución. Los supercomputadores nacieron de la necesidad de resolución de problemas complejos con requerimientos de ingentes cantidades de cálculo en coma flotante, en incluso de memoria de almacenamiento. A principios de la década de los 80 surgieron los clusters de ordenadores para poder satisfacer estas grandes necesidades de memoria de almacenamiento y capacidad de cálculo. Estas *granjas* de ordenadores, seguían conservando los recursos de forma dedicada, es decir, eran recursos exclusivamente para ejecutar simulaciones o realizar tareas de cómputo de altas prestaciones sobre datos de entrada. Estos *clusters* de ordenadores son bastante caros, y no todas las organizaciones podían conseguir un *cluster* afín a los requerimientos de cálculo que la organización requería porque era imposible económicamente, ya no por costos de hardware, sino por el consumo eléctrico que tener esos recursos suponía. A mediados de la década de los 90, dos investigadores, Ian Foster y Carl Kesselman,

crearon un nuevo concepto de computación llamado *Computación Grid*, y definieron éste como: *Una infraestructura software y hardware que proporciona un entorno barato, consistente y fiable de computación de altas prestaciones*. Uno de los precursores de la *Computación Grid* es la red eléctrica. La red eléctrica está proporciona electricidad a lo largo del territorio. Esta red puede conectar nuevos generadores y el usuario final no tiene por qué enterarse. El usuario únicamente ve un entorno fiable del que puede suministrarse de energía eléctrica. Por lo tanto, ¿por qué no aplicar este concepto a la computación? Un usuario podría conectar su ordenador y éste, automáticamente formar parte de una infraestructura Grid que lo utilice para diferentes fines, utilizando únicamente los recursos computacionales (memoria, CPU) que el usuario no use en cada momento. Al igual, desde la otra parte, un usuario podría ver el Grid formado por recursos muy dispares como un entorno fiable y confiable donde ejecutar sin fallos sus programas, o almacenar u obtener información. Desafortunadamente, en la práctica, los parecidos entre la *Computación Grid* y la red eléctrica no llegan muy lejos. No es simplemente *enchufar* y listo. Hay que desplegar capas de servicios para que todo este tinglado funcione y los usuarios puedan utilizar los recursos, y éstos ser utilizados por los usuarios. En la Figura 1.1 se muestra a Ian Foster y Carl Kesselman, padres de la *Computación Grid*. Según ellos, hay una lista mínima de propiedades que un sistema Grid debe poseer:

- Se deben coordinar recursos sin un control centralizado, aunque estos pertenezcan a varias organizaciones distintas, y estén situados a lo largo de distintas fronteras administrativas. Al mismo tiempo es necesaria una mínima seguridad, confiabilidad.
- El sistema Grid debe usar protocolos e interfaces abiertas basadas en estándares para proveer al Grid de servicios de autenticación, autorización y de mecanismos de descubrimiento y acceso a recursos.
- El sistema debe proporcionar al usuario la imagen de un gran *computador virtual* unificado, que es tolerante a fallos. Todo esto se consigue a través de capas de software que abstraen los recursos en una interfaz única para todos.

Figura 1.1: Carl Kesselman e Ian Foster, padres de la computación Grid.

El Grid fue concebido originalmente como un modo de proporcionar de forma barata a los usuarios investigadores, o miembros de organizaciones, grandes cantidades de recursos computacionales para ejecutar aplicaciones de altas prestaciones como son

los simuladores. El incremento en la velocidad de las redes de área local y las redes de área amplia ha permitido a los científicos hacer más grandes sus simulaciones y experimentos incluyendo parámetros en éstas que antes ni se habían planteado debido a la explosión computacional a la que se verían sometidas estas aplicaciones. Los resultados de salida de una aplicación de este tipo pueden ser transmitidos de un continente a otro en menos tiempo de lo que se creía posible hace una década, y ello ha favorecido el nacimiento de diversos programas de colaboración entre grupos de investigación, llamados genéricamente *eScience*. Este tipo de aplicaciones tienen como objetivos proporcionar a los científicos instrumentos con los que poder almacenar grandes cantidades de información de forma distribuida, e incluso potencia de cómputo distribuida. Un escenario de *eScience* muy conocido en la actualidad es el que proporciona la infraestructura para el almacenamiento masivo distribuido y el cómputo de altas prestaciones al proyecto *Large Hadron Collider (LHC)* del *CERN* [35], que es internacional debido a los altísimos costes del proyecto, quizás el mayor proyecto de física de partículas de Europa y del mundo en la actualidad. En la Figura 1.2 se muestra un escenario típico de *eScience* en el que un científico tiene acceso a masivas cantidades de datos distribuidos, puede colaborar con otros científicos mediante visualización remota, acceder a servicios de cálculo de altas prestaciones, etcétera. Todo esto era imposible a esta escala sin disponer de una infraestructura Grid.

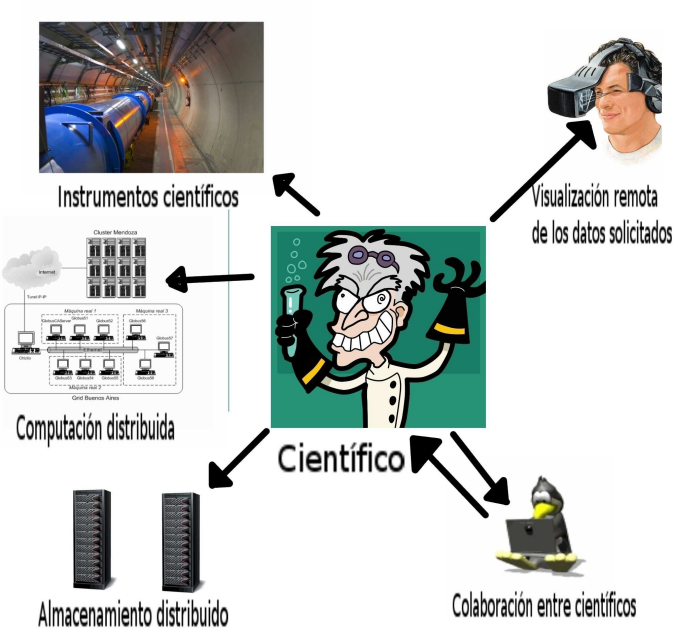


Figura 1.2: Escenario de aplicaciones *eScience*.

1.2. Infraestructura Grid

El desarrollo de la infraestructura Grid tanto software como hardware está de moda entre la comunidad de investigadores y desarrolladores, tanto en el sector académi-

co y de investigación como en el comercial. Muchos de los problemas con los que se encuentran los desarrolladores de Grid son:

- Proporcionar gestión distribuida de los recursos manteniendo el control total sobre los recursos locales.
- Mejorar la disponibilidad de los datos distribuidos.
- Proporcionar a los científicos un entorno sencillo con el que interactuar con los recursos Grid.

La forma de tener acceso a los recursos Grid, que actualmente están disponibles de forma dinámica, es disponer en el Grid de varios Servicios de Descubrimiento de Recursos. Los usuarios finales enviarían sus peticiones de trabajo con los requisitos de éstas a un agente Grid llamado *Grid Resource Broker*. Éste consultaría a los distintos proveedores de información por los recursos disponibles y asignaría las tareas a los recursos Grid más afines a éstas. Todo esto debe realizarse en un marco seguro, en el que las transferencias de datos entre los usuarios, los recursos Grid y el *broker* deberían realizarse consultando la identidad del que envía los trabajos y el recurso al que se le envían.

1.2.1. Componentes del Grid

Algunos de los componentes que pueden encontrarse en una infraestructura Grid son:

- **Componentes de almacenamiento distribuido** con servicio de replicación.
- **Componente de Seguridad**. Proporciona un medio por el que realizar Autorización y Autenticación de los recursos y usuarios.
- **Componente de acceso a recursos**, de procesamiento o de datos.
- **Componente de descubrimiento de recursos**.
- **Componente de planificación de trabajos**. Metaplanificadores para planificar trabajos de usuarios a recursos del Grid.
- **Componente de monitorización de trabajos**. Se debe mantener al usuario informado del estado de un trabajo enviado al Grid. Por lo menos se debe indicar si se está o no ejecutando y dónde.
- **Componente de transmisión de datos**. Un componente que realice labores de *File staging* o labores de envío de ficheros necesarios para la ejecución de un trabajo desde el usuario u otros servicios de almacenamiento distribuido hasta el nodo de cómputo, y de vuelta al usuario o a otro componente de almacenamiento remoto.

- **Componente de auditoría**. Se debe tener constancia de los trabajos que se han ejecutado en cada recurso y por quién, además se deben realizar estadísticas de uso para los diferentes recursos que pueden ser utilizadas como entrada para diversas políticas de planificación de recursos basadas en el uso de éstos.

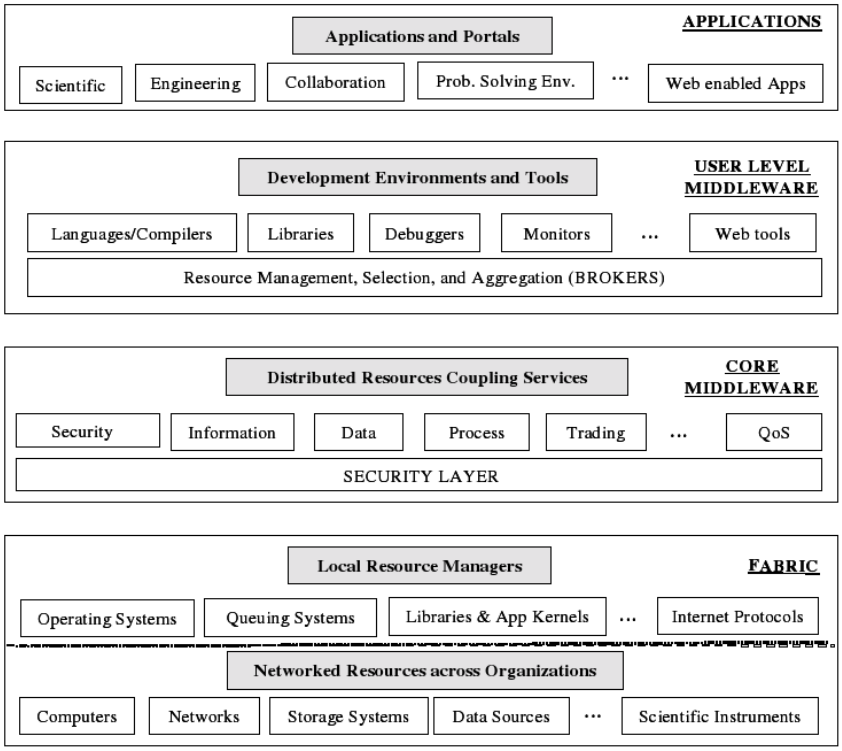


Figura 1.3: Estructura en capas de la arquitectura Grid.

Estos componentes se enmarcan en una estructura en capas de lo que es el software y el hardware de los recursos del Grid:

1. **Capa base o *Grid Fabric Layer***. Esta capa ocupa la capa más baja de la jerarquía. La representan los recursos computacionales de distintas arquitecturas como pueden ser los clusters, supercomputadores, servidores, estaciones de trabajo, ordenadores de sobremesa, etcétera. Incluso pueden ser sensores de instrumentos científicos que provean información al Grid.

2. **Capa de servicios o *Core Grid Middleware Layer***. Esta capa ofrece servicios de alto nivel a los recursos de la capa base. Proporciona gestión remota de procesos, localización de recursos, acceso a almacenamiento y seguridad. Incluso puede que tenga soporte a una muy limitada *Calidad de Servicio (QoS)*, como reserva de recursos. Esta capa hace que la capa base aparezca de forma unificada a los usuarios, aunque todavía esta capa es muy baja en cuanto a nivel de abstracción.

3. **Capa de usuario o *User-level Grid Middleware***. Esta capa utiliza la capa de servicios anterior para proporcionar servicios de mayor nivel orientados a los

usuarios finales. Estos servicios incluyen APIs para desarrollo de aplicaciones basados en estándares, como es el *DRMAA* de OGF (se hablará de él en siguientes capítulos), y metaplanificadores para planificar trabajos entre todos los recursos del Grid global.

4. **Aplicaciones y Portales Grid**. Estas aplicaciones y portales abstraen la capa anterior estableciendo un entorno amigable con el que comunicarse con ella por parte de los usuarios.

En la Figura 1.3 se muestran las capas de una arquitectura Grid básica.

Hay que tener en cuenta que aunque se realiza una abstracción de los recursos del Grid como si fueran un único recurso, en realidad no lo son. Algunos recursos pueden estar lejos entre sí, conectados a través de líneas WAN comerciales con bajo ancho de banda. Por lo tanto, aunque se intente, no se pueden ejecutar todo tipo de aplicaciones en este tipo de entornos si no se quiere obtener incluso menos rendimiento del que se obtendría sin paralelizar una aplicación. Las aplicaciones más señaladas para ejecutarse en este tipo de entornos son las aplicaciones que pueden dividirse en un número dado de tareas independientes entre sí, con lo cual no se transmitirán datos entre ellas a través de enlaces con mucha latencia. Lo más recomendable para este tipo de entornos son tareas con mucha carga computacional sobre los datos de entrada, con un tiempo de cómputo muy superior al de la transferencia de los datos. Cuánto más tiempo de cómputo por tarea independiente, y mayor número de tareas paralelas, mayor será la productividad del Grid comparada con la productividad sin paralelizar en un único recurso.

Capítulo 2

Motivación y objetivos

El objetivo de este proyecto es dotar al laboratorio de Redes de Altas Prestaciones (RAAP) del Instituto de Investigación en Informática de Albacete (I3A) de las ventajas que la *Computación Grid* puede ofrecerle, como unificar los diferentes recursos de los que éste dispone bajo un único recurso al que el usuario puede tratar como un sistema unificado, confiable y tolerante a fallos. Las herramientas elegidas para dotar a los recursos computacionales del laboratorio de una interfaz abierta y unificada a los demás recursos es el *middleware* de código abierto *Globus ToolKit*. Este *middleware* o capa base de servicios Grid, permite compartir recursos computacionales incluso a través de fronteras administrativas a través de las redes de área amplia. Los servicios que ofrece permiten a los usuarios acceder de forma remota a los recursos del mismo modo que si estuvieran accediendo de forma local, permitiendo el control local sobre quién o cómo puede usar esos recursos y cuándo. Utilizando esta capa básica de servicios que se explicará en siguientes capítulos, se instalará otra capa superior de servicios orientada al usuario que utilice esta capa base de servicios de forma transparente. Además deberá gestionar eficientemente todos los recursos a los que tenga acceso e incluso descubrir nuevos recursos, dando una visión unificada al usuario. Se utilizará el metaplanificador de trabajos *GridWay* para unificar todos los recursos del laboratorio RAAP bajo esta interfaz, con la que los usuarios podrán interactuar a través de interfaces de programación estándar como el *DRMAA* del *Open Grid Forum*.

Una vez conseguida la unificación de recursos locales del laboratorio RAAP, se creará un prototipo de interconexión entre diversas máquinas pertenecientes a 3 grupos de investigación de la Universidad Politécnica de Valencia (UPV), la Universidad de Valencia (UV), la Universidad de Murcia (UM), y el grid desplegado en el laboratorio RAAP. Dichos grupos de investigación participan en un Proyecto conjunto financiado por el Ministerio de Educación y Ciencia, con fondos FEDER, dentro de la iniciativa Consolider-Ingenio 2010. Uno de los objetivos de dicho proyecto consiste */... poner en común las capacidades de al menos 5 clusters de altas prestaciones.../*. Esta *puesta en común* de recursos se llevará a cabo mediante tecnología Grid. En este Proyecto, se hará la primera aproximación a dicha interconexión, que servirá como experiencia piloto a la hora de implantar procedimientos y recomendaciones de uso para la consecución del mencionado objetivo estratégico.

Para conseguir esta unificación de recursos de forma segura, se instalará una

capa superior al metaplanificador GridWay que *virtualizará* el Grid de cada Universidad en un único recurso accesible desde exterior. Esta nueva capa superior es provista por *GridGateWay*, desarrollada por el mismo grupo de investigación que el metaplanificador *GridWay*.

A lo largo de los siguientes capítulos se irá documentando el despliegue de toda está infraestructura Grid en los recursos disponibles del laboratorio RAAP, desde las capas inferiores de servicios a la capa más alta de servicios. Desde integrar un único recurso local a la organización, hasta integrar los recursos de los 4 grupos de investigación. Con todo esto se conseguirá un entorno mixto de recursos de varias Universidades a las que un usuario autorizado del Grid, de cualquier Universidad pueda acceder de forma transparente.

Se validará el Grid desplegado y se realizarán pruebas de rendimiento a los diferentes recursos añadidos, evaluando las capacidades de *GridWay* como metaplanificador y de *GridGateWay* como puerta de acceso a un Grid.

Basándose en los principios de transparencia y unificación del Grid, se desarrollará una aplicación gráfica para que los usuarios puedan diseñar sus flujos de trabajo para el Grid, y que ésta genere los correspondientes ficheros de órdenes que el metaplanificador pueda interpretar para planificar y enviar sus trabajos. Incluso, la aplicación será capaz de interactuar con el metaplanificador pudiendo enviar trabajos al Grid conectado desde la máquina del usuario de forma remota.

2.1. Metodología del Trabajo

Con el fin de cumplir los objetivos marcados, el proyecto se desarrollará siguiendo los siguientes pasos:

1. Se llevará a cabo un proceso de investigación sobre la Computación Grid, con el objetivo de comprender mejor como funciona este tipo de tecnología, y que ventajas puede proporcionar a la computación paralela de altas prestaciones.

2. En los recursos disponibles en el laboratorio RAAP, se instalará la capa base de servicios Grid escogida, es decir, *Globus Toolkit 4.0*.

3. Se investigará en el campo de la metaplanificación de tareas en entornos Grid, con el objetivo de familiarizarse con las políticas de planificación utilizadas en este tipo de planificadores, y el tipo de middlewares de los que éstos se sirven para comunicarse con los recursos a bajo nivel.

4. Se instalará el metaplanificador *GridWay* en un recursos escogido del laboratorio, y se configurará adecuadamente para utilizar los recursos disponibles. Se configurarán las políticas de planificación oportunas del metaplanificador con el fin de familiarizarse con ellas y mejorar la utilización de los recursos disponibles.

5. Se extenderá el Grid creado en el laboratorio RAAP conectándo recursos proporcionados por los grupos de investigación de la UPV, UV y UM. Para ello se

empleará la puerta de enlace al Grid, *GridGateWay*, instalándose en los recursos de los 4 grupos de investigación implicados en la conexión.

6. Una vez la infraestructura Grid haya sido desplegada, se procederá probando los distintos recursos agregados mediante el empleo de benchmarks específicos.

Capítulo 3

Estado del Arte

3.1. Introducción a la Computación Grid

Este nuevo Siglo XXI ha empezado con muchos retos para los científicos. Éstos retos cada vez requieren de soluciones más complejas, que requieren grandes cantidades de cálculos computacionales para su resolución o simulación. La computación Grid ofrece a éstos, un marco sobre el que realizarse utilizando los recursos conjuntos cientos de grupos de investigación dispersos por el planeta, cuya cooperación sería imposible sin este tipo de tecnología.

3.1.1. Proyectos Grid de Computación voluntaria

Desde la aparición a mediados de los 90, de un pequeño cliente llamado SETI@HOME [54], que procesaba señales procedentes del espacio en busca de patrones inteligentes, en los ordenadores personales de usuarios, la computación Grid ha avanzado mucho. Actualmente existen muchos y variados proyectos de computación Grid voluntarios. La mayoría de estas infraestructuras Grid están basadas en aplicaciones clientes, que se instalan en los ordenadores de los usuarios que actúan como recursos del Grid. Digamos que los recursos tienen instalada una capa base de servicios de la infraestructura Grid. La capa middleware de cliente, que utiliza las capas base de los clientes, son los servidores de los Proyectos, que envían los trabajos a los recursos. Estos servidores tienen intalados metaplanificadores que tienen acceso a los cientos de miles de recursos, dando la visión a éstos de un supercomputador gigante con millones de procesadores. Estos entornos Grid son muy propensos a fallos, debido a que la organización que controla el proyecto no tiene control sobre ninguno de los recursos de procesamiento, como no ocurre en otras infraestructuras Grid de investigación. Para evitar fallos, debido a la manipulación malintencionada de usuarios en los resultados, o a recursos hardware defectuosos, las unidades de trabajo se mandan replicadas a varios nodos. Para que una unidad de trabajo se de como terminada y validada, debe de cumplirse un determinado *quórum*, es decir, deben coincidir los resultados de varias tareas para que se de por validada una unidad de trabajo. Existen proyectos de computación

Grid voluntarios para todas las disciplinas de la ciencia:

- **La climatología**, es de las disciplinas que más atención recibe en la actualidad, debido al cambio climático. Se nutre de la computación distribuida para realizar sus simulaciones, que son costosas en tiempo debido a los algoritmos complejos y a la gran cantidad de datos de entrada, y variables que procesar. Las simulaciones cada vez requieren de más *resolución*, por la que los recursos que necesitan para ejecutarse crecen con la resolución. Algunos proyectos de computación Grid actuales han surgido de esta disciplina. Uno de ellos es el proyecto *Climate Prediction* [58]. Este proyecto es el experimento más grande que se ha desarrollado para predecir el clima del siglo XXI. Se nutre de ordenadores personales de todo el mundo para ejecutar pequeños trabajos llamados *Work Units*, sobre los cuales, los procesadores de los usuarios realizan procesamiento y devuelven resultados. Se incentiva a los usuarios del proyecto facilitándoles *créditos* por unidades de trabajo procesadas. Otro proyecto en este campo es el *BBC Climate Change Experiment* [6]. Este proyecto está financiado por la BBC y la Universidad de Oxford, y pretende simular distintos modelos de climatología desde principios del siglo XX hasta la actualidad, para ver si el modelo se parece al clima actual, y si es así, se prosigue con el modelo hasta el año 2080. Se pretende predecir con exactitud el clima que habrá en la Tierra para el año 2080, contando con la influencia humana claro. Actualmente el proyecto ha terminado, y se están procesando los resultados obtenidos para sacar conclusiones.

- **La medicina**, otra de las disciplinas que actualmente también se nutre de las infraestructuras Grid, para la simulación del plegado de proteínas, con el objetivo de desarrollar nuevas drogas y curas para las dolencias del hombre. Uno de los proyectos más conocidos en esta disciplina es el proyecto *Folding@Home* [17] de la Universidad de Stanford, California. Este proyecto simula el plegado de proteínas, y ayuda a comprender cuál es su función. Actualmente consiste en una aplicación que descarga unidades de trabajo, y envía resultados. Existen versiones para diversas arquitecturas, incluso para la arquitectura del procesador Cell de IBM, Toshiba y Sony, basado en PowerPC. También existe un cliente que puede ejecutarse en tarjetas gráficas ATI, en la que se aprovecha la potencia de cálculo paralelo de los vertex shaders de estas tarjetas gráficas. Actualmente este proyecto logra una potencia de cálculo común de 1.3 PetaFlops, el mayor proyecto de computación distribuida del planeta. El mayor supercomputador dedicado actualmente logra los 600 TeraFlops de procesamiento y cuesta millones de dólares.

- **Astronomía**, actualmente existen varios proyectos en este campo, como es el *cosmology@HOME* [9] y el *Einstein@HOME* [15]. El primero no se centra en ningún problema en concreto, sino que simplemente los astrofísicos proponen problemas que necesitan resolver y piden la ayuda del proyecto *cosmology* para obtener recursos de cómputo para sus simulaciones. El proyecto *Einstein@HOME*, trata sobre el análisis de los datos obtenidos por detectores de ondas gravitacionales, un controvertido proyecto que pretende descubrir ondas gravitacionales procedentes de la evaporación de agujeros negros debido a la radiación de Hawking, cosa que está aún por demostrar.

Existen varios proyectos englobados dentro de lo que se denomina el *Berkeley Open Infrastructure on Network Computing* o *BOINC* [29], como son *SETI@HOME* [54], *EINSTEIN@HOME*, *Cosmology@HOME*, etcétera. *BOINC* es una infraestructura Grid que provee a los proyectos de investigación de una capa de servicios que pueden utilizar en los recursos, y un modelo de programación, para que los investigadores creen sus programas que generen pequeñas unidades de trabajo que se repartan a los recursos de los usuarios, utilizando la capa de servicios base que es un cliente BOINC.

3.1.2. Proyectos de colaboración Grid entre comunidades científicas

Varios campos de la ciencia que surgieron a finales del siglo pasado, como la teoría de las Supercuerdas en su vertiente de la Teoría M, no pueden ser explorados por vía de la experimentación, por lo menos por ahora. Se requieren cantidades astronómicas de energía para poder explorar la matería a un nivel cercano a la longitud de Planck (del orden de $10^{-30}m$). Las *SuperCuerdas* de la teoría andan próximas a esta longitud y por lo tanto, actualmente, son inalcanzables por la tecnología humana actual, por ello, los científicos echan mano de las herramientas de simulación. Actualmente, los científicos han predicho mediante simulaciones, la existencia de una partícula llamada *bosón de Higgs*, que según dicen provee de masa a las demás partículas del Modelo Estándar. Su existencia, entre otras cosas, será probada experimentalmente en el Gran Colisionador de Hadrones (LHC) [**?**]. Este experimento, el más ambicioso en física de partículas del mundo, requerirá ingentes cantidades de almacenamiento para la gran cantidad de información que este generará al año, que se prevee de 10 Petabytes. El proyecto se proveerá de una infraestructura Grid internacional que proveerá de almacenamiento y computación distribuida a los datos generados por el proyecto. También facilitará a los científicos el acceso. Este acelerador de partículas entrará en funcionamiento en el 2008. En la Figura 3.3, puede verse la infraestructura Grid que proporcionará soporte a este proyecto internacional

En los Estados Unidos existen varios Grids de producción como *TeraGrid* [55], que provee de 40 teraflops mediante la colaboración de 8 centros en los Estados Unidos. Los centros están conectados por líneas de datos de 10 a 30 Gbps. Otro proyecto es el *Grid3* [2], que une a 25 centros de Estados Unidos y Corea del Sur, para investigar de forma colaborativa en física de alta energía, astronomía y biología. Estos proyectos Grid utilizan como diversas capas de servicios:

- ***Globus Toolkit*** [46], que proporciona una capa base de servicios Grid,para abstraer los gestores locales de recursos específicos, como *LSF*, *Condor*, *PBS*, etcétera.
- ***Condor*** [27], que permite infraestructuras Grid mediante *Condor-G*.
- ***Unicore*** [60]. Proporciona una capa de servicios que permite un acceso uniforme a los gestores locales de los recursos Grid.
- ***gLite*** [39]. Proporciona una capa de servicios pre WS, no orientada a servicios

Web. Es un proyecto europeo de código abierto utilizado para el proyecto *EGEE* [47].

En Europa existe lo que se conoce como el proyecto*EGEE* [47] (*Enabling Grids for E-sciencE)*. Desarrollado por el Reino Unido. Este proyecto propone desarrollar una infraestructura Grid sólida, capaz de proporcionar a los científicos una plataforma fiable para desplegar sus aplicaciones. Otro proyecto enteramente Europeo es el *GridLab* [22], que proporciona un paquete para desarrollar aplicaciones llamado Cactus. También el proyecto *CrossGrid* [10] en el que participan varios centros educativos de España, en la Figura 3.2, puede verse el mapa de distribución de colaboradores a lo largo de Europa.

En Asia existen proyectos como el *NAREGI* [41] de Japón, *APAC Grid* [3] de Australia, el *National Grid* en China, el *K* Grid* [34] en Corea del Sur. Las capas de middleware que usan estos proyectos son varias:

- **El *Ninf*** [44] de Japón, que implementa llamadas RPC sobre servicios *Globus*.
- **El *Grid Datafarm (GFarm)*** [18], que implementa un sistema de ficheros compartido y distribuido en red. Propone una solución alternativa a NFS, y pretender funcionar a mayor escala y de forma más rápida.
- ***Nimrod*** [43] de Australia, que proporciona herramientas de modelado de aplicaciones distribuidas.
- ***Alchemi*** [1]. Es un marco de trabajo que permite agregar recursos computacionales de forma sencilla a un Grid creado con este software. Es utilizado por metaplanificadores como *GridBus*. Como curiosidad está implementado en Windows, por lo tanto es posible ejecutar programas compilados para este sistema operativo. Ofrece una API de programación en .NET. Se desarrolló en la Universidad de Melbourne (Australia).

En España existe la red *IrisGrid* [48], que es una red Grid académica y de investigación. Actualmente cuenta con 500 procesadores obtenidos de la unión de diferentes centros educativos a lo largo del territorio español. En la Figura 3.1, puede verse un mapa de la infraestructura *IrisGrid*. Existen aún proyectos más pequeños como RenderGrid [51], y *GaliGrid*, de la *Xunta de Galicia*.

3.2. Planificación en entornos Grid: Metaplanificadores

La planificación dentro del entorno de computación Grid consiste en encontrar recursos dentro del Grid capaces de satisfacer las necesidades de los trabajos que en este entorno se lanzan. Se toman en consideración diversas métricas como son:

- Carga computacional de los recursos (trabajos ejecutándose en ellos y recursos de sistema consumidos)

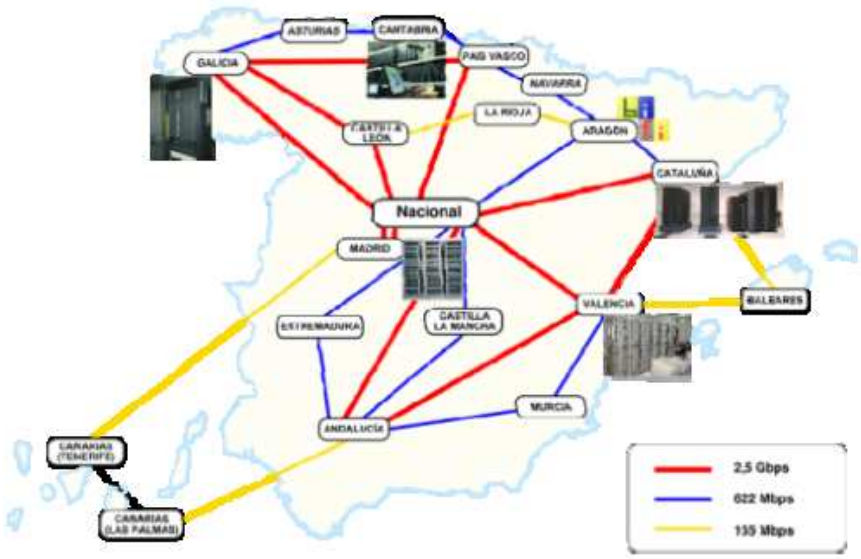


Figura 3.1: Mapa nacional de *IrisGrid*.

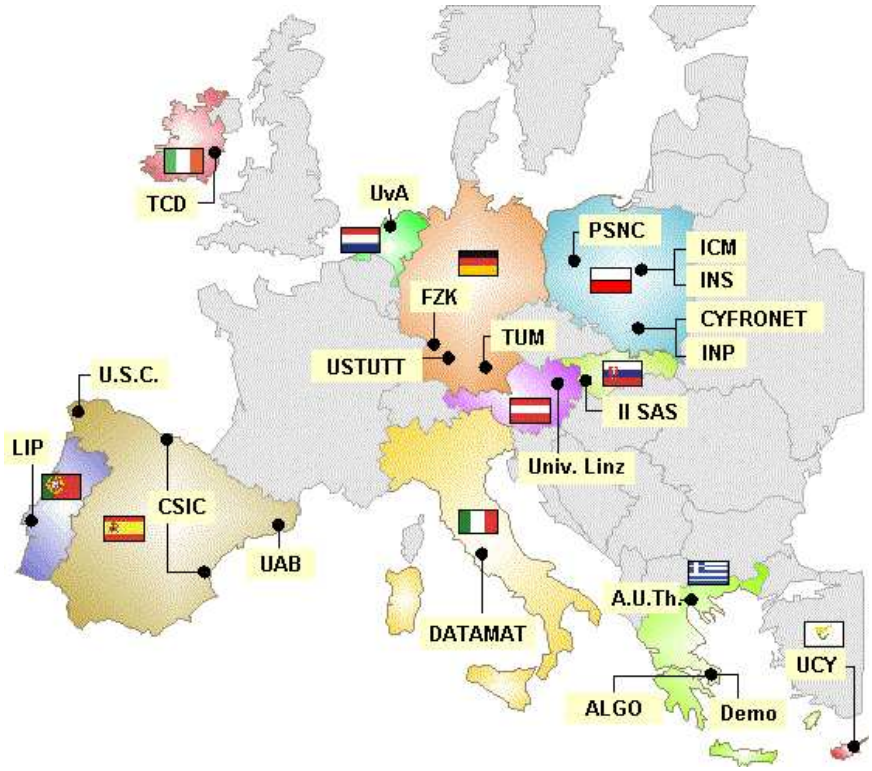


Figura 3.2: Mapa europeo de CrossGrid

- Prioridad de los trabajos.
- Preferencias de las tareas lanzadas en cuanto a los recursos en los que se ejecutarán.

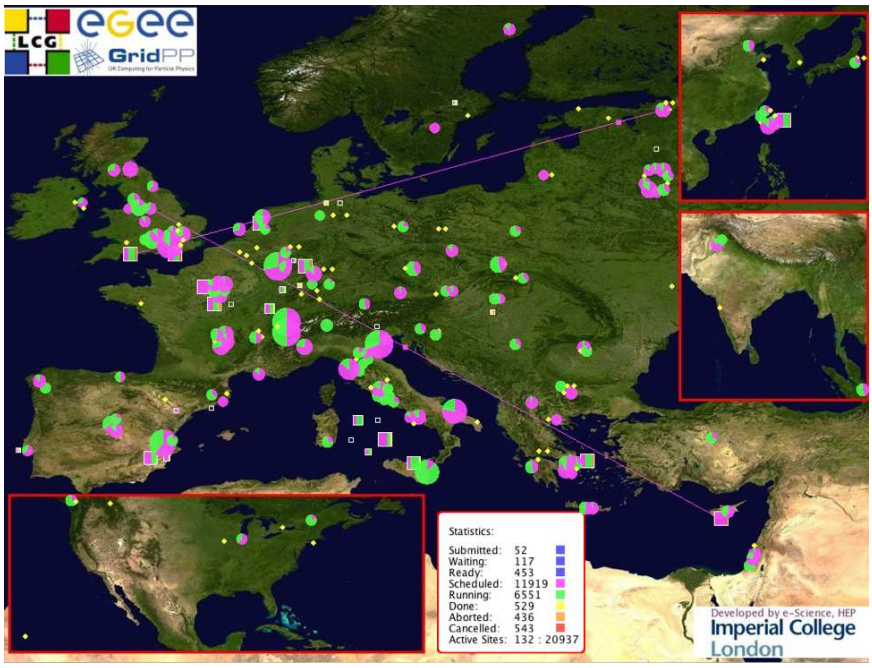


Figura 3.3: Mapa del Grid LHC.

- Requisitos obligatorios de las tareas lanzadas en cuanto a los recursos en los que se ejecutarán.
- Tasa de fallos de un recurso computacional determinado.
- ...

El problema de planificación ha sido ampliamente estudiado en el pasado y como resultado se han obtenido algoritmos muy eficientes que han sido probado en diferentes plataformas de cómputo. A pesar de la experiencia ganada en planificación Grid, todavía hay características que difieren de forma dramática de las plataformas clásicas de computación homogéneas y locales no basadas en Grid (Clusteres, multiprocesadores...). Estas características, que difieren de las plataformas clásicas son:

- Distintos dominios administrativos presentes en *Partner Grids*.
- Control limitado sobre algunos recursos.
- Heterogeneidad y entornos dinámicos que cambian con el tiempo. (Nodos de cálculo que se caen, fallos en la red).
- ...

Existen tres escenarios en los que un metaplanificador debe funcionar:

- **Grid de empresa o *Enterprise Grid***. Los recursos son propiedad de una única organización. Existe un único metaplanificador que interactúa con los recursos.

- **Grid amigos o *Partner Grids***. Se comparten recursos entre varias organizaciones. Puede compartirse todo tipo de recursos. Metaplanificadores pueden enviar trabajos a otros metaplanificadores o a gestores locales de recursos.

- **Grid de Altas Prestaciones**. Se comparten con otras organizaciones recursos HPC (High Performance Computers). Es necesaria una organización jerárquica.

Existen actualmente varios metaplanificadores Grid que merece la pena mencionar como son:

- **CSF (Community Scheduler Framework)** [61]. Es un marco de desarrollo de metaplanificadores, es código abierto y parte del proyecto *Globus*. Soporta la implementación de metaplanficadores basados en servicios *Globus* (GRAM, MDS, RFT,etc...). Permite la comunicación entre otros planificadores a nivel local, ya que permite reserva de recursos (para calidad de servicio), trae varias mecanismos simples de planificación, y es extensible vía plugins. En la Figura 3.4 puede verse la arquitectura *CSF*.

- **GridWay** [38]. Es un proyecto código abierto, que utiliza servicios Globus para las versiones con y sin servicios Web. Como característica principal es que soporta el estándar DRMAA de GGF [57], permite dependencias entre tareas (WorkFlows) y soporta migración de trabajos de forma adaptativa.

- **GSB (Grid Service Broker) GridBus** [19][56]. Permite utilizar varias capas base de servicios Grid, como *Globus* pre-WS y WS, *Unicore*, *Alchemi*. Proporciona servicios orientados a servicios de cómputo de pago. Es desarrollado por la Universidad de Melbourne (Australia). En la Figura 3.5 se muestra la arquitectura del agente de servicios Grid *GSB*.

- **GRMS GridLab Resource Management System** [21]. Funciona utilizando los servicios Globus pre-WS y WS. Se planea soportar dependencias entre tareas. Se define un lenguaje de definición de trabajos llamado (GJD). Permite migración de trabajos. En la Figura 3.6 se muestra la arquitectura de este gestor de recursos Grid.

- **Moab/Maui** [40]. Es un metaplanificador usado en clusters de altas prestaciones. Soporta Globus como gestor de recursos de recursos Grid. También soporta múltiples gestores de cluster.

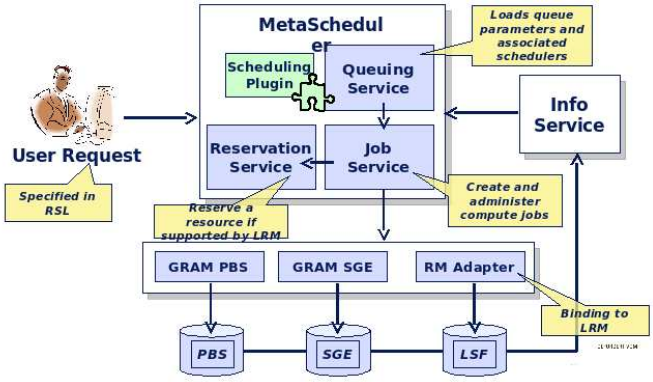


Figura 3.4: Arquitectura de *CSF*.

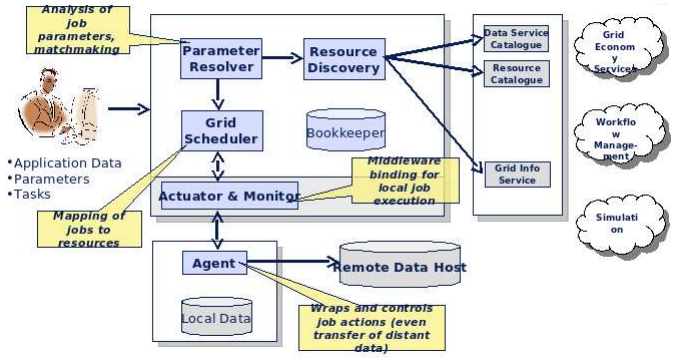


Figura 3.5: Arquitectura de *GSB*.

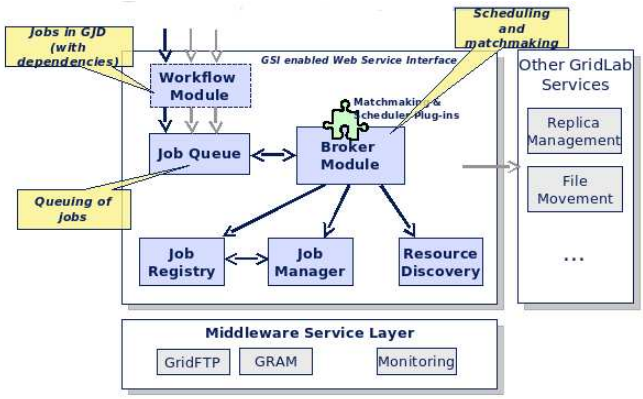


Figura 3.6: Arquitectura de *GRMS*.

Capítulo 4

Globus ToolKit como capa base de servicios Grid

4.1. Introducción

Globus Toolkit [46] es un proyecto de código abierto disponible para sistemas operativos Linux, resultado de la *Globus Alliance* [46], una comunidad de organizaciones que persiguen crear tecnologías para el Grid que permitan compartir recursos con potencia de cálculo, bases de datos, información instrumental y otras herramientas en red de forma segura, a lo largo de distintas organizaciones, instituciones e incluso fronteras administrativas sin sacrificar la gestión autónoma de los recursos por parte de la organización que los posee. Se ha escogido *Globus Toolkit* ya que el metaplanificador *GridWay* del que se hablará en los siguientes capítulos, necesita de la capa de servicios que *Globus Toolkit* ofrece a las capas superiores. Concretamente, necesita bibliotecas de la API de programación C de *Globus*. La versión más actual es la 4.0.5. Desde la versión 4.0 de *Globus* se vienen usando servicios web para implementar los servicios. Versiones anteriores a la 4.0 no utilizaban servicios web y los servicios que ofrecen suelen llamarse *pre-WS*. Los servicios se enumeran a continuación:

- **Gestor de asignación de recursos *(Globus Resource Allocation Manager (GRAM))***. Éste componente traduce peticiones expresadas en lenguaje de especificación de recursos *RSL*, a comandos propios del gestor local de recursos instalado en el recurso. Por ejemplo, Condor, PBS, LSF. El *RSL* especifica en un lenguaje cuya sintaxis consiste en pares atributo-valor, la descripción de los recursos requeridos por el trabajo (CPU, memoria, etcétera). Permite la ejecución remota mediante el envío de un fichero RSL al servicio GRAM del nodo de procesamiento. Este componente está implementado de forma distinta para cada gestor local de recursos. Por ejemplo, la implementación de Fork (ejecución normal en un computador corriente con Linux) es distinta de la implementación de Condor (véase capítulo 5).
- **Componente de Infraestructura Grid Segura *(Grid Secure Infraestructure (GSI))***. Proporciona servicios de autenticación y autorización a los recursos y usuarios que usan el Grid.

- **Servicios de Información *(Grid Information Services (GIIS))***. Proporciona servicios de publicación de información del recurso, monitorización y descubrimiento.
- **Servicios de Acceso a almacenamiento secundario *Global Access to Secondary Storage (GASS)***. Proporciona servicios por lo que escribir o leer bloques de datos de discos de almacenamiento remotos.
- **Servicio de transferencia de ficheros *Grid File Transfer Protocol (GridFTP)***. Proporciona un servicio FTP eficiente, seguro y confiable para realizar transferencia de ficheros entre recursos en entornos Grid. GridFTP utiliza el componente seguro de infraestructura Grid (GSI) para autenticar a los clientes y recursos durante las transferencias de datos.

Para todos los componentes anteriores, se proporciona una API para programar en C, Java, y Python, aunque *Globus* está desarrollado en lenguaje C. Además es posible interactuar con los servicios Globus mediante un lenguaje al estilo *Command Line Lenguaje* (CLI). En la Figura 4.1 puede verse la estructura en capas de los servicios de *Globus Toolkit*.

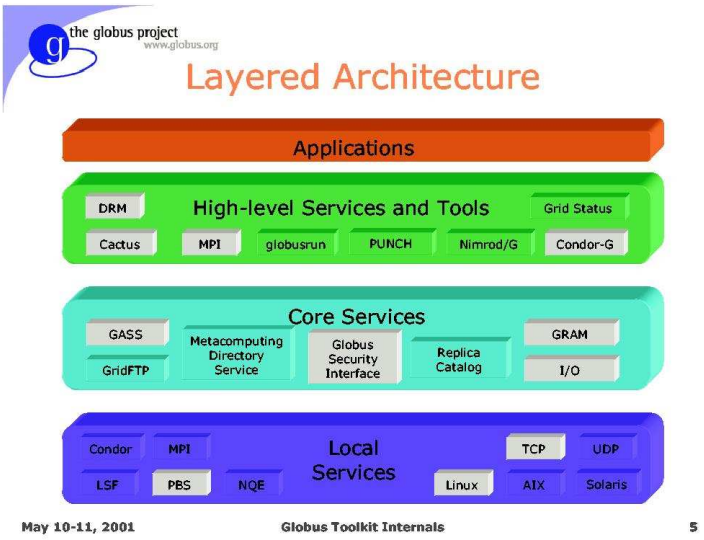


Figura 4.1: Arquitectura de *Globus ToolKit* [36].

4.2. Arquitectura Globus

La arquitectura *Globus* está dividida en varias capas. La capa más baja representa a los recursos locales del Grid. Estos recursos, como se ha visto anteriormente, pueden ser accedidos directamente mediante la creación directa de un proceso (Fork), o estar gestionados por un gestor de recursos local (LRMS) como Condor, LSF, PBS,

etcétera. Por encima de esta capa se encuentra la capa de servicios de Globus , en la que se encuentran los componentes que abstraen los recursos locales para proporcionar una visión única de éstos de cara a las capas superiores en las que se encontrarán las APIs de programación y la interfaz CLI para interactuar con los servicios Globus . Por encima de esta capa se encuentran las aplicaciones de terceros que utilizan las APIs proporcionadas por la capa inferior. En las siguientes secciones se centrará la atención en los componentes de la capa *Globus* utilizados por el metaplanificador *GridWay* que se instalará en el Laboratorio RAAP como está establecido en los objetivos del proyecto. Estos servicios son los que se explican en las siguientes secciones.

4.2.1. Capa de seguridad de Globus *GSI Security Layer*

Esta capa proporciona métodos de autenticación y comunicación segura a los usuarios y recursos. Se basa en la arquitectura *SSL*, *PKI*[67] y *certificados X.509*. Esta capa de Servicios Grid Seguros *Grid Secure Infraestructure (GSI)* utiliza criptografía de clave pública también llamada criptografía asimétrica, en la que el mensaje se codifica con la clave pública del receptor y se descodifica con la clave privada del receptor del mensaje. La clave pública se transmite a todos los elementos con los que se desea establecer una comunicación, y la clave privada únicamente tiene el receptor. El mensaje una vez codificado con la clave pública sólo puede ser desencriptado con la clave privada. Para obtener más información acerca del módulo de seguridad de *Globus Toolkit*, consultar [69].

4.2.2. Componente GRAM

El componente Gestor de Asignación de Recursos o *Globus Resource Allocation Manager* procesa las peticiones a recursos para la ejecución remota de aplicaciones. El componente asigna los recursos seleccionados para la ejecución de trabajos y los ejecuta. También proporciona al componente *MDS* información sobre el estado de los recursos locales para que este las publique. Éste componente proporciona una API para enviar y cancelar trabajos, al igual que consultar el estado de un trabajo determinado. Las peticiones de asignación de trabajos a un recursos se especifican en lenguaje *RSL* o Lenguaje de Especificación de Recursos. Por lo tanto el componente GRAM es responsable de:

- Analizar y procesar el fichero con lenguaje *RSL*. El fichero especifica los recursos necesarios por la tarea a ejecutar.
- permitir la monitorización remota y la gestión del trabajo creado.

Para obtener más información acerca del componente GRAM de ejecución de trabajos, consultar [69].

4.2.3. Componente MDS

Provee de soporte para publicar y solicitar información de recursos remotos o locales. La información se devuelve en XML y se representan varias propiedades del sistema, como son la velocidad de la CPU, memoria libre, porcentaje de utilización de la CPU, etcétera. Este componente tiene una estructura de tres capas:

1. **La capa 1**. Están los proveedores de información, que obtienen la información de los recursos y la procesan de tal manera que pueda ser enviada en un fichero XML donde se han definido todas las clases posibles, para cada parámetro a monitorizar.

2. **La capa 2**. Está el Servicio Grid de Información *(Grid Resource Information Service (GRIS))*. El demonio del servicio se ejecuta en cada recurso del Grid.

3. **Índice de Servicios de Información Grid o *(Grid Information Index Service (GIIS))***. Este servicio indexa la información proporcionada por otros servicios *GRIS* o *GIIS*. Suele haber un índice de servicios o varios índices de servicios establecidos de forma jerárquica dentro de una organización. Los recursos de la organización estarán registrados en uno de los *GIIS* presentes para su organización, de tal forma, que los *GRIS* presentes en cada recurso envíen su información al *GIIS* configurado.

4.2.4. Componente GridFTP

El componente *GridFTP*[68] proporciona seguridad y confiabilidad a las transferencias de datos a través de las redes de WAN públicas. Está basado en el protocolo FTP y tiene un conjunto de características y extensiones que se detallan a continuación:

- Seguridad GSI sobre varios canales de datos.
- Múltiples canales de datos para transferencias paralelas.
- Envíos parciales de datos, con capacidad de resumir.

Para obtener más información acerca del módulo de transferencia de ficheros GridFTP, consultar [69].

4.3. Pasos para desplegar un Grid con Globus

Los pasos recomendados para desplegar un Grid con middleware *Globus Tookkit* sobre recursos disponibles, es recomendable consultar [69].

Capítulo 5

Integrando un cluster en el Grid

5.1. Introducción

Un gestor local de recursos (LRMS) se encarga de administrar los recursos locales optimizando su uso. En las arquitecturas cluster se encargan de administrar una agrupación local de recursos computacionales autónomos y de prestaciones similares. Los componentes se administran atendiendo a criterios comunes. Los nodos de computación están integrados en una red de área local *LAN* o de área de sistema *SAN*, y el sistema operativo se instala completamente en todos los nodos. Uno de los sistemas operativos más usados es Linux. El gestor de recursos generalmente asume el control exclusivo del sistema administrando los trabajos que llegan a este optimizando el uso de los recursos monitorizando una serie de variables como pueden ser la productividad, carga, etc...

La capa de servicios que ofrece Globus permite desplegarse sobre cualquier *LRMS* soportado por su versión actual. La versión 4.0.5 permite utilizar *PBS*, *Condor* y *LFS*. como gestores locales de recursos (LRMS).

5.2. Condor como gestor de recursos

Figura 5.1: Logo Condor.

Condor es un sistema de software libre originalmente desarrollado por la Universidad de Wisconsin-Madison, que permite computación paralela distribuida en tareas intensivas de grano grueso y grano fino ya que soporta *MPI* y *PVM*. Puede usarse para gestionar la carga de un cluster de estaciones de trabajo o para repartir trabajo

a ordenadores libres cuando estos están en estado ocioso. Esto es lo que se llama *cycle scavenging*. Condor puede ser ejecutado en Linux, Unix, Mac OS X y FreeBSD, e incluso en sistemas operativos Windows. Con condor como gestor de recursos se pueden integrar recursos dedicados como pueden ser *racks* de procesadores u ordenadores personales normales no dedicados (desktop computers) cuando estos están en estado ocioso o *ocioso*. Todo esto bajo un mismo entorno, transparente para el usuario final de Condor que será el que lance trabajos sobre este.

5.3. Condor y Grid

5.3.1. Condor-G

Condor permite también la utilización de recursos distribuidos con diferentes gestores locales de recursos. Esto añade una capa de virtualización que permite añadir al conjunto de recursos que gestiona Condor un Grid de terceros que a su vez puede tener diferentes recursos gobernados por diferentes gestores locales de recursos. Condor-G permite utilizar recursos *Globus* mediante la capa de servicios que Globus ofrece.

La pregunta es: *¿Cuál es la diferencia entre Globus y Condor-G?, y ¿Cuándo instalar Globus o Condor-G?*. La respuesta es sencilla. Condor-G usa la instalación de *Globus Toolkit* para ejecutar un trabajo en una máquina remota, ajena al cluster. Por lo tanto *Condor-G* ofrece una *ventana al Grid* para los usuarios que deseen acceder a recursos a través del Grid e instantáneamente ver como sus trabajos se están ejecutando. Globus es un paquete de servicios que permite integrar cualquier recurso computacional a un Grid corporativo o entre varias corporaciones, ya que ofrece servicios de descubrimiento y monitorización de recursos, servicios de ejecución y transferencia segura y confiable de ficheros por medio de *GSIFTP* y *RFT, Reliable File Transfer*. Por lo tanto si tenemos un cluster con Condor y añadimos un Cluster con otro gestor de local de recursos, por ejemplo *LSF*, entonces instalando el paquete de servicios de Globus podemos integrar ambos clusters como si fueran un único cluster para el usuario. Sin Globus ToolKit esto no sería posible.

5.3.2. Globus GRAM para Condor

Globus permite utilizar Condor como gestor local publicando información de este vía *MDS*, ejecutando trabajos y cancelándoles si es necesario. Para ello, Globus tiene implementado un gestor de ejecución que utiliza comandos Condor para interactuar este LRMS. La obtención de información del cluster se limita a obtener el número de nodos actualmente en el cluster, número de trabajos activos y números de trabajos en estado ocioso (ocioso). Esta información es tratada por *Globus* y publicada mediante su servicio *MDS* para que pueda ser consultada por planificadores en capas superiores u otros gestores de recursos que soporten *Globus* como capa base de servicios Grid.

5.4. Configurando Globus ToolKit para utilizar un recurso Condor

La manera más sencilla de instalar Globus en un Cluster es mediante la instalación a partir de su código fuente. Existen diferentes paquetes para diversas distribuciones tipo Debian o FedoraCore, o para arquitecturas Itanium 64 bits, AMD x86_64 y x86, pero lo más sencillo es instalarlo desde código fuente, ya que de esta forma no habrá conflicto alguno con nuestro sistema si todas las dependencias de éste están resueltas.

5.4.1. Compilar *Globus* con soporte GRAM para Condor

Partimos de una instalación de Condor configurada y funcionando correctamente.

1. En el nodo principal del cluster, es decir, el nodo raíz NFS o nodo con interfaz de red pública o accesible a las conexiones del exterior, hay que crear un usuario llamado *globus* que será el usuario sobre el que se ejecutarán los servicios de *Globus* .

2. Verificar una correcta instalación de Condor ejecutando *condor_status* sobre el usuario *globus* creado para la instalación de *Globus*. La salida debe mostrar los nodos del sistema y los trabajos que actualmente se estén ejecutando si es que hubiera alguno. La salida de *condor_status* para un cluster hipotético es:

```
Name               OpSys      Arch   State      Activity   LoadAv Mem   ActvtyTime

myr1.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.000  1519  0+01:30:04
myr11.i3a.ucl      LINUX      INTEL  Owner      ocioso      1.000   756  3+20:38:22
myr13.i3a.ucl      LINUX      INTEL  Unclaimed  ocioso      0.000   756  0+09:36:16
myr14.i3a.ucl      LINUX      INTEL  Unclaimed  ocioso      0.020   756  0+00:30:52
myr16.i3a.ucl      LINUX      INTEL  Owner      ocioso      0.510   756[?????]
myr2.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.000   756[?????]
myr3.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.000   756[?????]
myr5.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.000   756[?????]
myr6.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.000   756  0+08:41:05
myr9.i3a.uclm      LINUX      INTEL  Unclaimed  ocioso      0.070   756  0+00:06:50

                     Machines Owner Claimed Unclaimed Matched Preempting

          INTEL/LINUX      10     2       0         8       0          0

                Total      10     2       0         8       0          0
```

3. Descargamos la última versión Globus ToolKit de la página oficial de Globus ToolKit [46]. Preferiblemente la versión de código fuente.

4. Descomprimimos el fichero tar.gz con el código fuente.

5. Ejecutamos el script *configure* con opciones *–prefix=/$GLOBUS_LOCATION* y *–enable-wsgram-condor* para indicar que Globus se instale en la localización $GLOBUS_LOCATION y se compile e instale el soporte para Condor.

   Es posible indicar cualquier otro LRM soportado. Para consultar las opciones disponibles ejecutar*configure –help*.

6. Ejecutar make y si no ha habido errores make install.

5.4.2. Configurar seguridad y servicios de Globus

1. Configurar la seguridad del Grid. Antes de configurar la seguridad de Globus, es decir, crear y desplegar certificados, es recomendable tener las máquinas que vayan a pertenecer a la *VO* sincronizadas en su hora y fecha mediante **NetWork Time Protocol (NTP)**. Problemas derivados de la variación de hora entre instalaciones de *Globus* en la misma *VO* o entre distintas *VO* puede desencadenar en errores en los certificados difíciles de resolver si no se conoce de antemano que el error está causado por variación de la hora entre, por ejemplo, la entidad emisora y una estación que ha solicitado un certificado para su host. Un breve tutorial para instalar *NTP* en una máquina puede encontrarse en la bibliografía[8].

   Partiendo de que el tiempo en las máquinas está sincronizado con *NTP*, hay dos posibilidades a la hora de configurar la seguridad:

   a) Disponer de una entidad raíz creada en otra máquina con otra instalación de *Globus* . En cuyo caso bastaría con seguir los pasos proporcionados por la guía rápida de *Globus* [25] para segundas máquinas.

b) No disponer de una entidad raíz y por lo tanto necesitar crear una entidad raíz desde cero [25].

c) Una vez configurado lo anterior, hay que crear un usuario, que será el usuario de *Globus* . Hay que tener en cuenta que el usuario *globus* únicamente es creado para ejecutar los servicios de Globus, y el usuario que se cree ahora será un cliente de esos servicios. Para que el usuario pueda utilizar en el terminal los comandos *Globus* sin necesidad de ir a la ruta de *Globus* es necesario incluir en el fichero *.bashrc* situado en el HOME de usuario la línea:

source $GLOBUS_LOCATION/etc/globus-user-env.sh

Antes de seguir hay que tener en cuenta una serie de consideraciones:

- La máquina sobre la que estamos desplegando *Globus* no tiene una entidad raíz certificadora y por lo tanto existen usuarios ya creados en otra máquina, que es la que tiene la entidad certificadora (en la mayoría de los casos). Únicamente hay que transferir el contenido de la carpeta *.globus* contenida en el HOME del usuario creado al nuevo usuario creado en la máquina local.
- La entidad certificadora existe en la máquina local y el usuario creado es un nuevo usuario de Globus. En este caso hay que realizar una serie de pasos para configurar la seguridad para el nuevo usuario.
  1) Como máquina sobre la que se está instalando *Globus* posee la entidad raíz certificadora y por lo tanto todavía no se ha creado ningún certificado para ningún usuario de *Globus* en la organización virtual *VO*. Hay que ejecutar el comando: *grid-cert-request* para usuario.
  Se creará un certificado sin firmar y una fichero de clave privada *.key* en el directorio oculto *.globus* situado en el HOME de usuario.
  El usuario *globus* con la entidad certificadora raíz tiene que recibir el fichero *usercert_request.pem* y firmarlo. Todo esto se realiza con:
  grid-ca-sign -in usercert_request.pem -out firmado.pem
  El fichero firmado.pem deberá ser transferido a la cuenta de usuario al directorio oculto *.globus* y sobreescribir *usercert.pem*.
  Hay que cerciorarse que el usuario es propietario de *usercert.pem* y no el usuario *globus*.

d) A continuación hay que copiar o generar de nuevo, dependiendo de la situación anterior, el fichero *grid-mapfile* que contiene una lista de los usuarios que tienen certificados en Globus. Para ello hay que crear o sobreescribir en la carpeta */etc/grid-security* el fichero *grid-mapfile*. Para más información acerca de como rellenar el fichero consultar guía *Globus* [25].
Para comprobar que todo ha ido correctamente ejecutar comando *grid-mapfile-check-consistency* cuya salida debe ser:

```
globusdev:/etc/grid-security# grid-mapfile-check-consistenc
Checking /etc/grid-security/grid-mapfile grid mapfile
Verifying grid mapfile existence...OK
Checking for duplicate entries...OK
Checking for valid user names...OK% use packages: array
```

e) Por último crear un proxy válido en la cuenta de usuario. Todas las llamadas a servicios de *Globus* remotos como transferencia segura FTP, GRAM o monitorización se realizarán a través de este proxy. Para probar que el proxy se crea y verificar que se ha creado correctamente basta con:

grid-proxy-init -verify

Hay que proporcionar la clave dada durante la creación del certificado para la creación del proxy. La salida del comando debe ser:

```
usuario@globusdev:/etc/grid-security\$ grid-proxy-init -verify
Your identity: /O=Grid/OU=GlobusTest/ \
OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=usuario
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Proxy Verify OK
Your proxy is valid until: Wed Nov 14 02:21:46 2007
```

Si por cualquier motivo aparece un error en la creación del proxy revisar los permisos de los certificados de */etc/grid-security* y del usuario en: */home/¡globus-user¿/.globus/usercert.pem y userkey.pem*

2. Una vez hecho esto hay varios caminos que se pueden tomar en función de la configuración presente del Grid. Hay dos posibilidades:

   Existe una base de datos *RFT* instalada en otra máquina en la que se registrarán los recursos locales, por lo tanto no necesitamos instalar una base de datos local, pero se debe indicar la URL y nombre de la base de datos remota. A continuación se detallan los pasos necesarios para indicar la situación de la base de datos de *RFT*:

   a) En el fichero GLOBUS_LOCATION/etc/globus_wsrf_rft/jndiconfig.xml hay editar el parámetro *connectionString* e indicarle la máquina remota y nombre de la base de datos. Los parámetros *username* y *password* deben sobreescribirse con el nombre de usuario y contraseña empleado en la creación de la base de datos.

   No existe una base de datos *RFT* instalada en otra máquina y por lo tanto el cluster actuará como base de datos *RFT*. En este caso es necesario instalar y configurar una base de datos local RFT que va en PostgreSQL. Los pasos para instalar la base de datos pueden encontrarse de forma detallada en el manual rápido de *Globus* en su página web[25] en la sección instalación de *RFT*. Cabe considerar que en ocasiones la guía rápida de *Globus* no es del todo detallada para la configuración de la base de datos PostgreSQL, ya que existen distintas versiones de éste funcionando actualmente, y disponibles en los repositorios de paquetes de cada distribución, y cada una se configura de forma distinta. A continuación se detallará de forma detallada para las versiones 7.4 y 8.1 la configuración del servidor de base de datos PostgreSQL para que acepte conexiones TCP y configurar los usuarios y máquinas que tendrán permiso de acceso a ésta.

   a) **Linux basado en Debian e PostgreSQL 7.4:** en el directorio */etc/-postgresql/7.4/main/* editar fichero *postgresql.conf* y descomentar línea: *tcpip_socket = true*. Con esto se habilitan las conexiones tcpip a la base de

datos. Ahora hay que dar permiso al usuario *globus* para que se conecte desde la máquina local. Para ello hay que editar el fichero *pg_hba.conf* y añadir la línea:

host rftDatabase "globusIP ADDRESS"SUBNET_MASK trust

b) **Linux basado en Debian e PostgreSQL 8.1:** en el directorio /etc/PostgreSQL/8.1/main editar fichero *PostgreSQL.conf* y descomentar línea:

listen_addresses = "localhost".

Para las conexiones de usuarios es igual que en el caso anterior: editar el fichero *pg_hba.conf*.

3. Configurar servicio de monitorización y descubrimiento *MDS* de *Globus* . Este servicio publica información local de los recursos sobre los que está instalado. Es recomendable pero no necesario realizar la instalación de *Ganglia*[26] como herramienta de monitorización del cluster. Para consultar información sobre como realizar la instalación y despliegue de Ganglia consultar capítulo *"Ganglia como herramienta de monitorización para el Grid"*. Es recomendable su instalación aunque no obligatoria ya que *Globus* compilado para un cluster obtiene su información a través del comando Condor *condor_status*, por lo que es necesario que este comando funcione correctamente (véase capítulo 7 que versa sobre la Instalación y configuración de Ganglia como herramienta de monitorización del Grid). Para que *Globus* publique esta información en su interfaz MDS es necesario editar el fichero
$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/gluerp.xml y dejar el fichero de este modo:

   En la versión de *Globus ToolKit 4.0.5* hay que realizar un paso de configuración adicional. Hay que ejecutar el comando *Globus mds-gluerp-configure* para que se genere el recurso *GLUE* apropiado para el gestor de recursos local. El recurso *GLUE* obtiene información de estado del recurso local y se la proporciona al servicio *MDS*. Hay que realizar tres pasos:

   a) Cambiar al directorio $GLOBUS_LOCATION/etc/gram-service-<LOCAL_LRMS> (Fork, Condor, LSF, PBS).

   b) Ejecutar *mds-gluerp-configure ¡LRMS¿ganglia*. Se generará un fichero gluerp-config.xml.

   c) Editar el fichero gluerp-config.xml para asegurarse que los puertos de Ganglia y la localización de este están correctamente configuradas.

   d) Arrancar los servicios *GRAM* y *MDS* para poder ejecutar y monitorizar trabajos en Globus. Existe una explicación detallada en la guía de instalación de Globus [25].

   e) Configurar servicio de transferencia de ficheros seguro *GSIFTP*.[25]. Para verificar que el servicio se ha iniciado correctamente ejecutar la orden *netstat -an — grep 2811*. Debe aparecer un socket escuchando en ese puerto, que será el servidor gsiftp.

   f) Una vez configurado todo hay que probar la instalación de Globus. Para ello primero se probará el servicio de transferencia de ficheros *GSIFTP*. Abrir

```
<config xmlns="http://mds.globus.org/2004/10/gluerp-config">
<!--
    <defaultProvider>none</defaultProvider>



To enable the use of ganglia to provide cluster information, replace the above+\\
with the following:+\

-->

<defaultProvider>java org.globus.mds.usefulrp.glue.GangliaElementProducer</defaultProvider>

</config>
<config xmlns="http://mds.globus.org/2004/10/gluerp-config">

  <defaultProvider>none</defaultProvider>


<!--

To enable the use of ganglia to provide cluster information, replace the above
with the following:

-->

<defaultProvider>java org.globus.mds.usefulrp.glue.GangliaElementProducer</defaultProvider>


</config>
<config xmlns="http://mds.globus.org/2004/10/gluerp-config">

 <defaultProvider>none</defaultProvider>

<!--

To enable the use of ganglia to provide cluster information, replace the above
with the following:

-->

<defaultProvider>java org.globus.mds.usefulrp.glue.GangliaElementProducer</defaultProvider>

</config>
```

un terminal para usuario, y ejecutar *grid-proxy-info* para cerciorarse de que existe un proxy válido. La salida debe ser similar a ésta:

```
subject   :
/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/ \
OU=dyndns.info/CN=usuario/CN=1959447498
issuer    : /O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/
OU=dyndns.info/CN=usuario                                          \
identity  : /O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/O
U=dyndns.info/CN=usuario                                           \
type      : Proxy draft (pre-RFC) compliant impersonation proxy
strength  : 512 bits
path      : /tmp/x509up\_u1001
timeleft  : 11:51:50
```

Si el proxy no es válido, es decir, su tiempo *timeleft* es 0:00:00 hay que crear un proxy nuevo con *grid-proxy-init*. Si por cualquier motivo ha surgido algún error hay que volver a ejecutar grid-proxy-init con la opción *-debug*.

Si todo ha ido bien realizar una transferencia de prueba de un fichero local a un directorio local mediante:

globus-url-copy gsiftp://¡hostaddress¿/etc/group file:///tmp/prueba

Si la copia se ha realizado correctamente no debe aparecer ningún mensaje de error. Si por cualquier motivo hay un error con el nombre de la máquina revisar el fichero */etc/hosts/* y añadir el nombre de la máquina local para la dirección de red de bucle local (127.0.0.1) y para la dirección ip de la interfaz de red ya sea pública o privada. Si el problema persiste consultar el fichero $GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd

y cerciorarse de que existe la entrada:

```
<globalConfiguration>
<parameter name="logicalHost" value=IP ADDRESS />
```

g) Verificar que la interfaz MDS publica información referente a los recursos del cluster. Ejecutando:

wsrf-query -s https://globusdev.dyndns.info:8443/wsrf/services/DefaultIndexService

El comando debe devolver:

```
...
   <ns1:Host ns1:Name="myr3.i3a.uclm.es" ns1:UniqueID="myr3.i3a.uclm.es">
        <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0" ns1:CacheL1I="0" ns1:CacheL2="0"
ns1:ClockSpeed="1800"
 ns1:InstructionSet="x86"/>
        <ns1:MainMemory ns1:RAMAvailable="40" ns1:RAMSize="756" ns1:VirtualAvailable="2541"
ns1:VirtualSize="3275"/>
        <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="2.6.10-1.770\_FC2"/>
        <ns1:Architecture ns1:SMPSize="1"/>
        <ns1:FileSystem ns1:AvailableSpace="41039" ns1:Name="entire-system"
ns1:ReadOnly="false" ns1:Root="/"
 ns1:Size="58465"/>
        <ns1:NetworkAdapter ns1:IPAddress="192.168.19.3" ns1:InboundIP="true" ns1:MTU="0"
ns1:Name="myr3.i3a.uclm.es"
 ns1:OutboundIP="true"/>
        <ns1:ProcessorLoad ns1:Last15Min="0" ns1:Last1Min="0" ns1:Last5Min="0"/>
       </ns1:Host>
       <ns1:Host ns1:Name="myr12.i3a.uclm.es" ns1:UniqueID="myr12.i3a.uclm.es">
        <ns1:Processor ns1:CacheL1="0" ns1:CacheL1D="0" ns1:CacheL1I="0" ns1:CacheL2="0"
ns1:ClockSpeed="1800"
 ns1:InstructionSet="x86"/>
        <ns1:MainMemory ns1:RAMAvailable="477" ns1:RAMSize="756" ns1:VirtualAvailable="2988"
ns1:VirtualSize="3275"/>
        <ns1:OperatingSystem ns1:Name="Linux" ns1:Release="2.6.10-1.770\_FC2"/>
        <ns1:Architecture ns1:SMPSize="1"/>
        <ns1:FileSystem ns1:AvailableSpace="43427" ns1:Name="entire-system"
ns1:ReadOnly="false" ns1:Root="/"
 ns1:Size="59885"/>

        <ns1:NetworkAdapter ns1:IPAddress="192.168.19.12" ns1:InboundIP="true"
ns1:MTU="0"
 ns1:Name="myr12.i3a.uclm.es" ns1:OutboundIP="true"/>
        <ns1:ProcessorLoad ns1:Last15Min="4" ns1:Last1Min="47" ns1:Last5Min="11"/>
       </ns1:Host>
...
```

h) Verificar que el servicio *GRAM* de ejecución de trabajos funciona correctamente. Para ello ejecutamos la siguiente orden en el terminal del usuario de *Globus* :

globusrun-ws -s -submit -S -F MAQUINA LOCAL -c /bin/uname -a

La orden debe devolver:

```
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:8a7cc956-91f0-11dc-8c14-0018f3150400
Termination time: 11/14/2007 13:58 GMT
Current job state: Active
Current job state: CleanUp-Hold
Linux globusdev.dyndns.info 2.6.18-4-686
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
```

Llegados a este punto podemos integrar el Cluster en el Grid de la organización *VO*, e incluso en un Grid mayor integrado por los sub-Grids de diferentes organizaciones.

Capítulo 6

Cooperación entre organizaciones virtuales (VO)

Como *Organización Virtual* (*Virtual Organization (VO)*) se conoce a una entidad sin ánimo de lucro, educativa o de otro modo entidad productiva, que no tiene ninguna localización geográfica central ni fija, y existe únicamente a través de las redes de telecomunicaciones, es decir, una *Organización Virtual* está formada por una serie de organizaciones independientes que comparten medios y recursos para conseguir un fin, que incluso no está limitado por una alianza económica entre esas entidades. La interacción entre miembros de la organización virtual se realiza principalmente a través de las redes de telecomunicaciones. Por lo tanto una *Organización Virtual* es un ejemplo de redes colaborativas.

Actualmente existen varias instituciones que están emergiendo como comunidades de usuarios con incluso economía propia dentro de *mundos virtuales*. Uno de estos mundos es *Second Life*[53] o *World of Warcraft*[66]. Hasta tal punto llegan estas instituciones virtuales que incluso poseen su propia economía y clases sociales que se forman debido a la interacción entre usuarios distribuidos a lo largo del mundo a través de la red de redes. Como curiosidad los *ciudadanos* de *Second Life*, como así se denominan los usuarios de este mundo virtual, tienen derechos virtuales de propiedad privada sobre terrenos virtuales y derechos para comerciar con estas propiedades.

Figura 6.1: Logo Second Life.

En *Computación Grid* una *Organización Virtual* es un grupo de individuos o instituciones que comparten sus *recursos computacionales* para un fin común. Estos

recursos que pueden ser de cómputo o almacenamiento se comparten en un Grid. Aquí es donde entra en juego *Globus* como plataforma de servicios para la creación de Grid corporativos y entre *VO*.

La iniciativa *IRISGrid* [48], evolución de RedIris, lanzada en el 2002, surge con el objetivo de coordinar a nivel académico y científico a los grupos de investigación interesados en esta tecnología, tanto en su desarrollo, implantación y aplicaciones. Además de esta coordinación, IRISGrid tiene como objetivo crear la infraestructura GRID nacional que permita el uso de esta tecnología tanto a nivel de aplicabilidad en diferentes ámbitos, como a nivel de desarrollo e innovación en este campo. Actualmente, en la iniciativa IRISGrid participan diferentes grupos del ámbito científico y académico, y está reconocida como la iniciativa nacional para la implantación, desarrollo e investigación en entornos Grids.

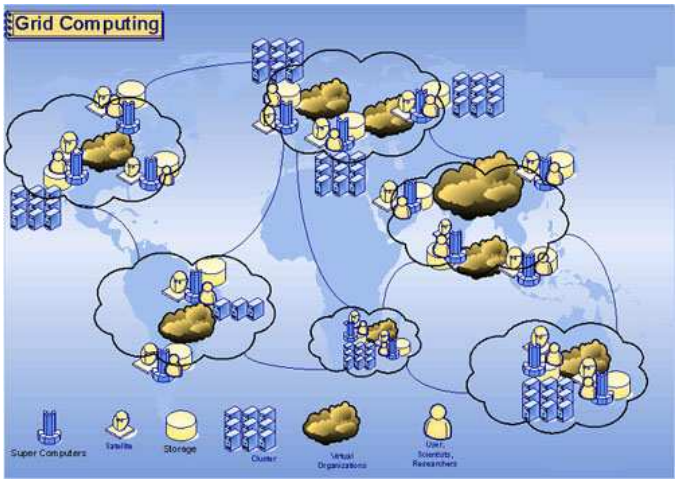


Figura 6.2: Organizaciones virtuales, VOs.

6.1. Establecer lazos de unión entre VOs

Para la creación de un Grid entre varios *sub-Grids* pertenecientes a organizaciones distintas hay que establecer primero una confianza entre ellas, es decir, no es posible realizar una unión entre dos organizaciones sin permiso expreso del administrador o administradores de cada una de ellas. El objetivo de dicha unión es siempre el de unir recursos computacionales que beneficien de forma directa a ambas y al fin que persiguen.

La confianza entre organizaciones puede ser temporal o permanente. Suele ser temporal puesto que las organizaciones virtuales son eso, virtuales. Esta confianza se da a través de certificados X509 que proporcionan las organizaciones virtuales. Estos certificados son proporcionados por una entidad emisora raíz, que cada organización debe poseer, y las competencias en la emisión de certificados válidos para usuarios de esa organización son competencia de cada organización virtual. A la hora de realizar la unión de varias VO hay que realizar un intercambio de certificados y de usuarios

para que de forma transparente los usuarios de ambas organizaciones vean ambas como una VO más grande formada por todos los recursos. Hay que puntualizar que puede haber usuarios de una organización que tengan vetado el acceso a los recursos de la otra organización aunque exista una confianza entre ambas. Todo esto puede ser establecido por el paquete de servicios *Globus* ToolKit.

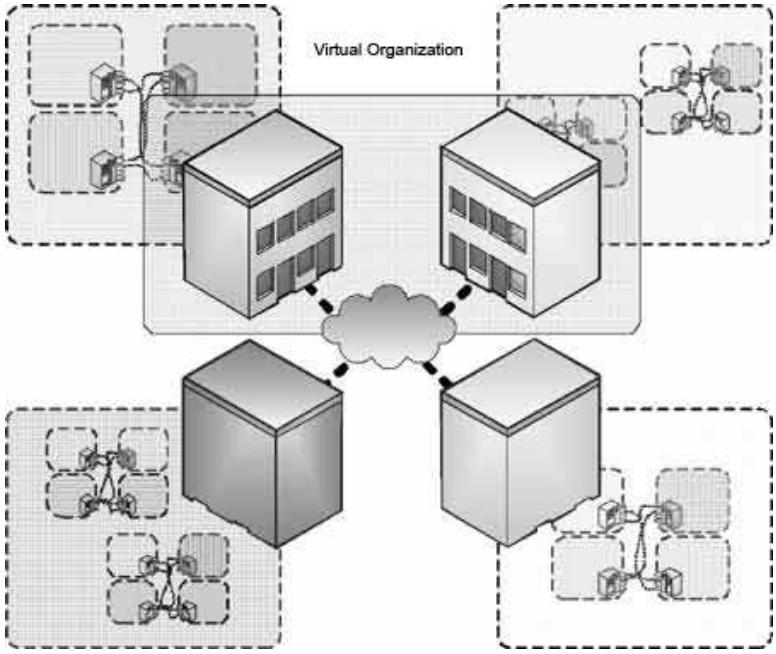


Figura 6.3: Unión entre varias VO a través de la red

6.2. Configurando *Globus* para la unión de dos VO

6.2.1. Unirse a una Organización Virtual

Para unirse a una organización virtual como usuario es necesario disponer del certificado de la *Autoridad certificadora (CA)*. Una vez hecho esto será posible generar certificados para la máquina y los usuarios. Los certificados para la máquina sólo serán necesarios si la máquina va a ser utilizada como recurso en el Grid. Si únicamente necesitamos usuarios que usarán recursos existentes sólo necesitamos los certificados de usuario. Una vez generados estos certificados deben ser firmados por la *CA* para que estos cobren validez temporal (al cabo de un tiempo que se establece durante su firma por la *CA*, estos certificados dejan de ser válidos).

Partiendo de una instalación de *Globus ToolKit* válida, los pasos para la unión de un usuario a una VO son:

1. Instalación del certificado de la entidad emisora de la VO en la máquina local. Para ello hay que obtener los ficheros .0 y .signing_policy proporcionados desde la

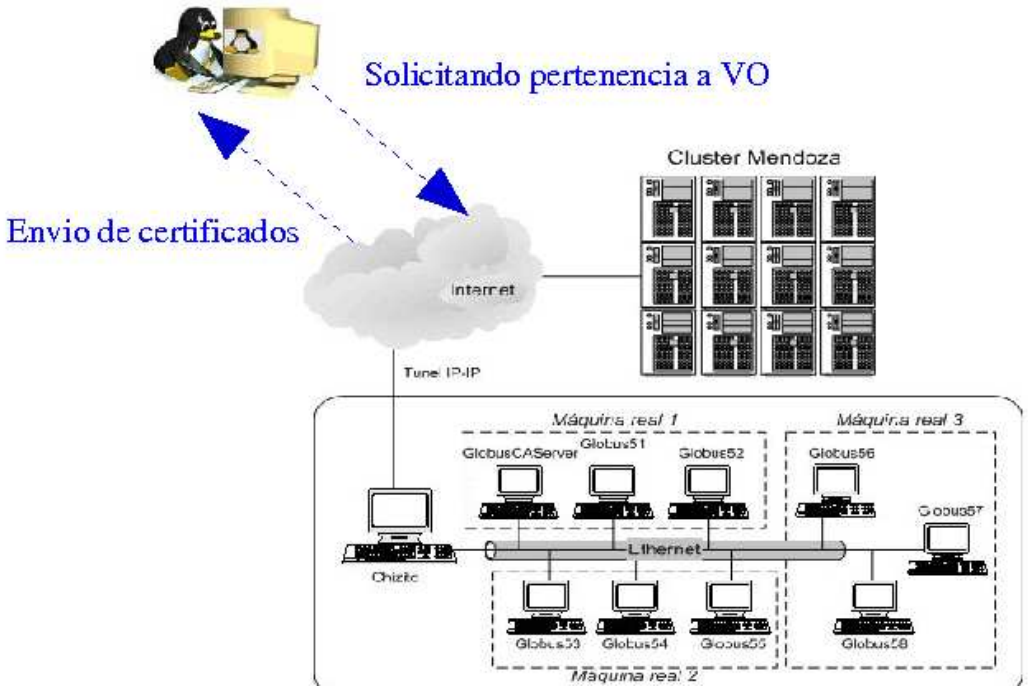


Figura 6.4: Proceso de unión a VO.

VO e instalarlos en la máquina local donde queremos crear y/o utilizar el usuario. Hay que copiarlos a la carpeta */etc/grid-security/* También es posible realizar la instalación desde el paquete *setup* proporcionado por la entidad emisora. Este paquete estará en formato gz y debe instalarse con el comando *gpt-build*.

2. Si la máquina actual va a formar parte del Grid como servicio es necesario generar un certificado de host con:

grid-cert-request -host 'hostname'

Este comando generará en la carpeta */etc/grid-services/* un fichero de clave privada y otro de solicitud de certificado, que tendrá que ser enviado a la entidad emisora para que sea firmado. Una vez este sea firmado por la entidad, deberá ser enviado a la máquina local y allí renombrarse por *hostcert.pem* y como propietario el usuario *root*.

Ahora es turno del usuario, para crear un certificado para un usuario local hay dos formas dependiendo de si el usuario existe ya o no en la VO:

- Si el usuario existe, solicitar envío de certificados ya firmados, copia de otros instalados en otras máquinas de la VO. Una vez obtenidos la clave privada *userkey.pem* y el certificado firmado *usercert.pem*, copiar éstos a la carpeta *$USER_HOME/.globus*.

- Si no existe el usuario hay que entrar como el usuario local y ejecutar la orden:

  grid-cert-request

  Se generará un clave privada para el usuario local y una solicitud de certificado (usercert_request.pem) en el directorio *$USER_HOME/.globus* que deberá ser enviado a la entidad emisora para que sea firmado y por lo tanto cobrar validez temporal. Una vez hecho ésto, el certificado *usercert.pem* deberá ser copiado con propietario, el usuario local, en la la carpeta: *$USER_HOME/.globus*.

Una vez los certificados creados e instalados hay que probarlos ejecutando el comando *grid-proxy-init*, que devuelve:

```
usuario@globusdev:/etc/grid-security/certificates$ grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=usuario
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Thu Nov 15 16:19:15 2007
```

Si el proxy se crea correctamente los certificados son válidos y podremos usar los servicios que ofrecen otras máquinas de la VO. Por ejemplo podemos ejecutar la orden:

```
usuario@globusdev:/etc/grid-security/certificates$ globusrun-ws -s \
-submit -S -F aulasun15.dsi.uclm.es -c /bin/uname -a
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:25ec45c8-932d-11dc-b495-0018f3150400
Termination time: 11/16/2007 03:45 GMT
Current job state: Active
Current job state: CleanUp-Hold
Linux aulasun15.dsi.uclm.es 2.6.16-2-amd64-k8-smp
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
```

Si queremos que la máquina local actúe como servicio para el Grid de la VO es necesario:

- Haber creado antes o crear ahora un certificado para la máquina.
- Generar el fichero *grid-mapfile* que contendrá la lista de usuarios autorizados a usar los servicios *Globus* locales (ejecutar trabajos, obtener información MDS, transferencia de ficheros GSIFTP). Para generar el fichero, basta simplemente con copiar la salida de *grid-proxy-info* al fichero */etc/grid-security/grid-mapfile*. El fichero grid-mapfile quedaría:

  ```
  "/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/␣\
  OU=dyndns.info/CN=usuario" usuario"
  ```

  Si se quiere que otros usuarios de la VO tengan acceso a los servicios locales, sólo hay que añadir más líneas al fichero *grid-mapfile*. Por ejemplo si se quiere que otro usuario de *Globus* llamado *fulano*, que no es usuario de sistema en nuestra máquina, pueda acceder al recurso local, hay que editar el fichero *grid-mapfile* que quedaría así:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=usuario" usuario

"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=fulano" usuario
```

- Editar el fichero */etc/sudoers* como root para permitir al usuario globus (contiene la instalación de *Globus* y los servicios de *Globus* se ejecutan en su cuenta) que pueda ejecutar trabajos como un usuario diferente. La configuración de este fichero puede consultarse en la guía rápida de Globus[25].

Estableciendo jerarquía de índice de servicios

Cada entidad una *jerarquía de índice de servicios*, es decir una jerarquía de máquinas que contienen índices de servicios, que en realidad son un servicio web que proporciona información sobre las máquinas disponibles y los recursos presentes. Al unirse a una VO es necesario registrar el nuevo recurso en el índice de servicios, para que pueda ser descubierto y usado por otros recursos de esa misma VO.

```
<config>

<!-- <upstream> elements specify remote index services that the local index
     will be registered to.

     Set an upstream entry for each VO index that you wish to participate in.
-->

     <upstream>https://globusdev.dyndns.info:8443/wsrf/services/DefaultIndexService</upstream>



<!-- <downstream> elements specify remote index service which will be
      registered into the local index. You do not need to configure
      <downstream> services unless you are building your own VO. -->

<!--     <downstream>https://member-host:8443/wsrf/services/DefaultIndexService</downstream> -->

</config>
```

Hay que añadir un campo *upstream* por cada VO en la que se registrará a este recurso.

```
<config>

<!-- <upstream> elements specify remote index services that the local index
     will be registered to.

     Set an upstream entry for each VO index that you wish to participate in.
-->

     <upstream>
https://globusdev.dyndns.info:8443/wsrf/services/DefaultIndexService</upstream>
     <upstream>
https://globus1.uclm.es:8443/wsrf/services/DefaultIndexService<
/upstream>

<!-- <downstream> elements specify remote index service which will be
      registered into the local index. You do not need to configure
      <downstream> services unless you are building your own VO. -->

<!--     <downstream>
https://member-host:8443/wsrf/services/DefaultIndexService
</downstream> -->

</config>
```

6.2.2. Establecer confianza entre dos VO

Una vez hay establecida una VO compuesta por recursos computacionales distribuidos, usuarios, etc... es hora de saltar a otro nivel para colaborar con otras VO afines. Como se dijo antes, hay que establecer una confianza entre las dos VO que se basa en el intercambio de certificados y usuarios.

Cuando se establece la unión entre dos VO va a ser necesario que al menos uno o más recursos de una VO contacten con uno o más recursos de la otra VO. No siempre es necesario establecer una confianza *N:N* entre todas las máquinas de la primera VO y la segunda VO.

Partiendo de que *Globus* está instalado y configurado en las máquinas sobre las que se va a establecer la confianza *inter-CA*. Pueden definirse una serie de pasos que se explican a continuación:

1. Intercambio de certificados de la autoridad emisora raíz de cada VO. Hay que intercambiarse los ficheros *.0* y *.signing_policy* contenidos en el directorio */etc/grid-security/certificates/* de cada entidad emisora

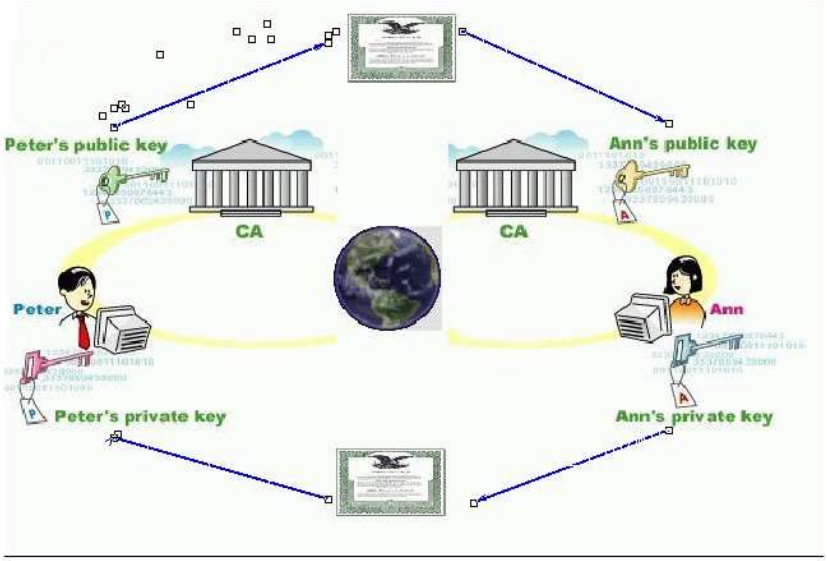


Figura 6.5: Intercambio de certificados de las entidades emisoras.

y añadirlos al directorio */etc/grid-security/certificates*. Si se ejecuta el comando *grid-default-ca–list* se obtiene la lista de las autoridades de certificación en las que se confía, o lo que es lo mismo, las distintas *VO* en las que se confía.

Puede observarse en la salida que está realizada sobre la máquina instalada en el Laboratorio de *Redes de Altas Prestaciones*[49] dentro del *Instituto de Investigación en Informática de Albacete* UCLM que se confia en dos VO aparte de la entidad emisora local. También se muestra qué entidad raíz está seleccionada por defecto a la hora de expedir certificados locales.

```
usuario@globusdev:/home/rayban/.gw_gwadmin_3$ grid-default-ca -list
The available CA configurations installed on this host are:

Directory: /etc/grid-security/certificates

1) d85f0ec4 -  /O=Grid/OU=GlobusTest/OU=simpleCA-maijaus.dyndns.info/CN=Globus S imple CA
2) e16a2917 -  /O=Grid/OU=GlobusTest/OU=simpleCA-globus1.uclm.es/CN=Globus Simpl e CA
3) fa9133d3 -  /O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/CN=Globus  Simple CA


The default CA is: /O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/CN=Globus Simple CA
        Location: /etc/grid-security/certificates/fa9133d3.0
```

2. Intercambiar usuarios entre las dos VO. Como puede observarse en la figura 6.5 en la que hay dos usuarios llamados: *Ana* y *Pedro*, pertenecientes a dos VO distintas. Para que *Pedro* pueda acceder a los recursos de la VO de *Ana* y viceversa, es necesario crear en los recursos que se comparten entre las dos VO, nuevas entradas en el fichero de autorización de usuarios de *Globus*, fichero *grid-mapfile*. Partiendo del contenido del fichero *grid-mapfile* en un recurso de la VO de *Pedro*, añadimos una nueva entrada para que *Ana* pueda acceder a este recurso, mapeándose en un usuario nuevo recién creado para la ocasión, o como usuario *Pedro*.

   Hay dos formas de proceder:

   - Para hacer las cosas bien sería necesario crear un usuario genérico de *Globus* en la VO de *Pedro* para que el usuario *Ana* u otros usuarios de la VO de *Ana* puedan utilizar recursos de la VO de *Pedro*. el fichero *grid-mapfile* situado en el recurso de la VO de *Pedro* quedaría:

```
...
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=Pedro" pedro

"/O=Grid/OU=GlobusTest/OU=simpleCA-foreignVO.domain.com \
/OU=domain.com/CN=Ana" anasvo

"/O=Grid/OU=GlobusTest/OU=simpleCA-foreignVO.domain.com \
/OU=domain.com/CN=Pepa" anasvo
...
```

     Donde *anasvo* es un usuario nuevo creado en el sistema con permisos de ejecución para *globus*, sobre el que se mapearán todos los usuarios de la VO de *Ana*. Por cada usuario hay que añadirle una línea al fichero.

     Para añadir el usuario *anasvo* al sistema, y que éste pueda ejecutar trabajos como *globus* hay que ejecutar:

```
$addgroup foreign-globus-user
$adduser anasvo foreign-globus-user
$sudo vi /etc/sudoers
$sudo cat /etc/sudoers

Runas_Alias     FOREIGN_USER = %foreign-globus-user

# User privilege specification
root    ALL=(ALL) ALL

globus  ALL=(pedro,FOREIGN_USER)  NOPASSWD:
/usr/local/globus-4.0.4/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/globus-4.0.4/libexec/globus-job-manager-script.pl *

globus  ALL=(pedro,FOREIGN_USER)  NOPASSWD:
/usr/local/globus-4.0.4/libexec/globus-gridmap-and-execute -g
/etc/grid-security/grid-mapfile
/usr/local/globus-4.0.4/libexec/globus-gram-local-proxy-tool *
```

Se ha creado un grupo especialmente para albergar a usuarios foráneos a la VO local. Se crea el nuevo usuario sobre el que se ejecutarán todos los usuarios pertenecientes a la VO de *Ana* y se añade este nuevo usuario al grupo *foreign-globus-user*. Este grupo coge un alias y se le autoriza a ejecutar trabajos en el recurso de la VO de *Pedro*.

Cuando el usuario *Ana* ejecute un trabajo en un recurso de la VO de *Pedro*, ésta podrá ejecutar trabajos como usuario *anasvo*.

- Es posible mapear los usuarios de la VO de *Ana* a usuarios ya existentes en la VO de *Pedro*, aunque es mejor del modo anterior creando un usuario nuevo como mínimo para cada VO, ya que así se controla más a los grupos de usuarios de cada organización al mapearse sobre usuarios locales distintos. El fichero *grid-mapfile* quedaría así:

```
...
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=Pedro" pedro

"/O=Grid/OU=GlobusTest/OU=simpleCA-foreignVO.domain.com \
/OU=domain.com/CN=Ana" pedro
...
```

Cuando llegue un trabajo perteneciente a *Ana* a un recurso del VO de *Pedro*, esta ejecutará trabajos sobre el usuario local de *Pedro*.

Todo esto puede aplicarse a *Ana* para que *Pedro* u otros usuarios de la VO de *Pedro* puedan acceder a recursos de la VO de *Ana*.

No es necesario cambiar la ruta de la base de datos que almacenará todas las transferencias *RFT* seguras, empleadas para ejecutar trabajos que necesitan *file staging* (véase bibliografía para más información[25]).

6.3. Probando envío mutuo de trabajos entre dos VO

Una vez establecida la confianza entre las dos VO para dos o más máquinas de ambas es necesario probar los servicios de *ejecución*, *monitorización y descubrimiento*; y transferencia segura de ficheros.

1. Partimos de *Ana*, que se dispone a utilizar servicios de un recurso de la VO de *Pedro* al que tiene acceso. Lo primero que necesita es un proxy válido. Ejecutando *grid-proxy-init* deberá obtener un proxy temporal válido.

2. Antes de proceder a acceder el recurso es importante que las máquinas de las dos organizaciones estén configuradas con NTP. (véase bibliografía [8]).

3. Obtener una lista de servicios del recurso del VO de *Pedro* es tan sencillo como ejecutar la orden:

   wsrf-query -s https://machineVOPedro:8443/wsrf/services/DefaultIndexService

El comando nos debe devolver un contenido en formato xml y no debe producirse ningún error. Un error es fácilmente distinguible ya que éste se devuelve en texto sin formato y no en XML.

Errores típicos:

- **Errores relacionados con la validez de los certificados**. Debidos a que las dos máquinas que han entrado en contacto tienen distintas versiones de tiempo al no estar sincronizadas con NTP[8], pero este no es el caso ya que ambas han sido configuradas para utilizar este protocolo.
- **Errores relacionados con nombre de recursos**: hay ocasiones en las que se producen errores asociando el nombre de host especificado en los certificados a la dirección de red de la máquina, con el consiguiente error. Esto puede comprobarse realizando transferencias de ficheros desde y hacia el host local desde otros recursos de la VO. Si el error se produce hay que editar el fichero */etc/hosts* y añadir una línea:

  <recurso> <IP ADDRESS>

- **Errores relacionados con configuraciones de cortafuegos**. Es necesario abrir ciertos puertos al exterior que de otro modo estarían cerrados hacia fuera de la organización donde se encuentran los recursos, o cerrados en cada uno de los recursos. *Globus Toolkit* como capa de servicios para el *Grid* utiliza los puertos:
  - **8443** Para GRAM y MDS.
  - **5432** Para la base de datos RFT (utilizada para el file-staging). Aunque no suele compartirse una misma base de datos debido a las latencias debido al paso por redes WAN entre dos VO distintas.
  - **2811** Para el servidor GSIFTP utilizado en las transferencias de ficheros y por lo tanto trabajos.

  Como mínimo deben estar abiertos al exterior los puertos 8443 y 2811 que son los básicos para que se pueda obtener información de recursos y envío de trabajos entre dos o más VO.

  Opcionalmente pueden abrirse más puertos para obtener información del demonio de monitorización *Ganglia* o del demonio *Gmetad* que obtiene información de éste y la agrupa. Todo esto con motivo de mostrar bajo una única interfaz web los recursos disponibles en cada organización por separado o en conjunto (véase capítulo 7).

  Los puertos que utiliza Ganglia y Gmetad por defecto son:
  - **8649** para Ganglia.
  - **8651** para Gmetad.

  En el capítulo 7 que versa sobre la instalación y configuración de Ganglia, puede obtenerse más información al respecto sobre que puertos utilizar dependiendo de la forma de agrupación de los datos que se desee.

4. Realizar una transferencia de ficheros segura mediante los comandos:

globus-url-copy gsiftp://maquinaPedro/etc/group file:///tmp/transferana

globus-url-copy gsiftp://maquinaAna/tmp/transferana gsiftp://maquinaPedro/tmp/transferana

Los comandos no deben devolver nada con las opciones que se les han pasado, lo anterior significa que se ha copiado el fichero *group* dentro de la máquina local de *Ana* en el directorio */tmp/* como *transferana*.

Si por casualidad se obtuviera algún error se debe volver a pasos anteriores anteriores en función de la salida del error.

5. Enviar un trabajo sencillo a la máquina de *Pedro* desde la máquina de *Ana* y viceversa.

globusrun-ws -s -submit -S -F maquinaPedro -c /bin/true

Con todo esto ya tendríamos plena confianza entre los dos recursos de las dos VO para los usuarios *Ana* y *Pedro*. Se ha visto como se establece la confianza, como se trasladan usuarios de un VO a otro de las diferentes formas que hay, y como utilizar los servicios remotos de otra VO.

6.4. Estableciendo límites en el uso de recursos inter-VO

Aún habiendo plena confianza entre las distintas *organizaciones virtuales*, es necesario establecer un control local a nivel de VO de los trabajos y recursos a los que se accede. Todo esto se consigue en cierto modo con los certificados, pero no es posible únicamente con *Globus*, limitar el uso de estos recursos en función de la carga de éstos y/o por cuáles de los usuarios estarán usados en un cierto momento, ya pertenezcan a una u otra VO. Como se comentaba en anteriores apartados, para la plena utilización de recursos por parte de ambas organizaciones era necesario añadir al fichero de autorización de *Globus*, una línea por usuario de otra VO con la que se estableció confianza (se han visto varias formas de hacerlo en el apartado 6.2.2). Estos son algunos de los motivos por los que se necesita una capa superior a *Globus* que proporcione metaplanificación para trabajos locales y remotos además de una interfaz que *virtualize* los recursos disponibles ya sea dentro de la propia VO como hacia fuera de la VO.

En los capítulos 8 y 9 se abordará este problema mediante la instalación del *metaplanificador Gridway* y una interfaz *Globus* a este metaplanificador llamada *GridgateWay* ambos complementarios, desarrollados por la Universidad Complutense de Madrid[63].

Capítulo 7

Ganglia como herramienta de monitorización para el Grid

7.1. Introducción

La aplicación de la tecnología Grid dentro del entorno de una VO implica la necesidad de controlar de algún modo el uso de los recursos que se están usando dentro de esta VO. Se hace necesario un modo de poder monitorizar el estado de estos recursos, es decir, si están o no disponibles, cuanta memoria libre tienen, CPUs libres, conectividad, etc... Todo ello deja obsoletas las tradicionales herramientas de monitorización centralizadas, que se instalan en un recurso (host) y se ejecutan únicamente de forma local, proporcionando información local. Para ello surge Ganglia, que se diseñó como un sistema distribuido de monitorización pensando para sistemas de computación de altas prestaciones como clusters, y posteriormente Grids. Está basado en un diseño jerárquico enfocado a crear jerarquías de clusters. El éxito de Ganglia como sistema de monitorización es debido a que está basado en tecnologías libres ampliamente utilizadas como es *XML* para la representación de datos, *XDR* para compactación y transporte de datos y *RRDtool* para el almacenamiento de datos y visualización. Ganglia está disponible para una amplia gama de sistemas operativos y arquitecturas hardware. Como curiosidad Ganglia es frecuentemente usado para unir clusters de campus universitarios bajo una interfaz de monitorización común. Ha sido probado que es capaz de soportar clusters con hasta 2000 nodos.

Como proyecto *open-source* en un principio desarrollado por la *Universidad de California*[62] en el llamado *Berkeley Millenium Project* que fue financiado principalmente por la *Infraestructura nacional en computación avanzada* y la *Fundación científica nacional*. Actualmente es usado en cientos de clusters y estaciones de trabajo a lo largo del mundo.

En los Grids desplegados con middleware de Globus ToolKit ha cobrado una gran importancia puesto que este proporciona información sobre los recursos locales que puede ser publicada en el servicio Globus MDS. De otro modo parámetros como la velocidad de CPU, memoria disponible, espacio en disco disponible, sistema operativo, etc... No podían ser incluidos de forma dinámica en la información publicada por MDS.

En el apartado 7.4 se explica como hacer que el middleware Globus obtenga información de Ganglia.

7.2. Estructura

Ganglia funciona como dos demonios independientes llamados *Ganglia Monitoring Daemon (gmond)* y *Ganglia Meta Daemon*.

7.2.1. Ganglia Monitoring Daemon (gmond)

El **demonio de monitorización Ganglia** (gmond) es un demonio multi hilo que se ejecuta en cada uno de los nodos del cluster que se desea monitorizar. Su instalación es muy sencilla. No tiene porque realizarse en un sistema de archivos NFS común ni configurar ninguna base de datos, *gmond* tiene su propia base de datos distribuida. Este demonio tiene 4 responsabilidades:

- Registrar los cambios de estado de un host.
- Publicar los cambios vía *multidifusión*.
- Escuchar el estado de otros nodos ganglia vía el canal multidifusión anterior.
- Responder a peticiones XML de estado del cluster.

El demonio Gmond tiene hilos que escuchan por el canal muticast y escriben la información obtenida al canal de forma muy rápida en una tabla hash en memoria. Toda las medidas tomadas dentro de un nodo del cluster son procesadas y almacenadas. Al contrario de lo que la gente puede pensar, el almacenamiento de esta información no requiere mucho espacio. Por ejemplo, en un cluster de 1024 nodos, si se está monitorizando 25 parámetros distintos en cada máquina el demonio gmond únicamente utilizará 16 + 136*1024 + 364*25 = 148380 bytes = 144 kB de memoria. La tabla hash en memoria es accedida de forma segura frente a hilos, por o tanto múltiples instancias pueden escribir o leer simultáneamente en esta tabla hash de información local del nodo.

Gmond puede transmitir en dos modos diferentes:

- multidifusión con representación de datos XDR. El demonio sólo realiza un multidifusión de la métrica del nodo local cuando un cambio en el valor de la métrica ha superado cierto umbral. El umbral asegura que el demonio sólo realice multidifusión cuando realmente lo necesita. Si no hay cambio de estado no se transmite nada. Esto reduce de forma drástica el tráfico por el canal.
- Enviando XML a través de una conexión TCP. Se envía la información en XML enviando la descripción completa del estado del cluster si se ha realizado una

petición explícita a través un puerto específico. Todo esto se detallará en los siguientes apartados.

Todos los demonios en el cluster almacenan la información del cluster en su totalidad.

Cada vez que el demonio recibe un paquete multidifusión de un nuevo nodo, éste expira el umbral de tiempo de sus métricas y reenvía todas sus métricas por el canal.

El demonio Gmond también es tolerante ante errores. Es inevitable que los nodos de un cluster se caigan ocasionalmente. Cuando esto ocurre y no se recibe información de un nodo que antes se tenía almacenado en la tabla hash después de que expire el umbral de retransmisión de información este pasa a estado *down* o caído. Incluso si alguna vez el sello de hora *timestamp* de un latido del demonio *heartbeat* cambia entonces esto significa que se ha producido un reinicio del demonio en otro nodo. Esto provoca que se lance una alerta para expirar el umbral de tiempo en todas las métricas con objetivo de actualizar el nuevo demonio.

Toda la información local del demonio se procesa de forma multidifusión a través de la interfaz de bucle local *loopback*.

Cuando se realiza una petición al demonio, éste escribirá toda la información del cluster completo en XML incluyendo su propia información por supuesto. Sin embargo, gmond solo compartirá datos que contenga en su lista de *host confiables*, *trusted hosts* especificada en su configuración.

Pueden enviarse métricas personalizables por el canal de las docenas que puede monitorizar.

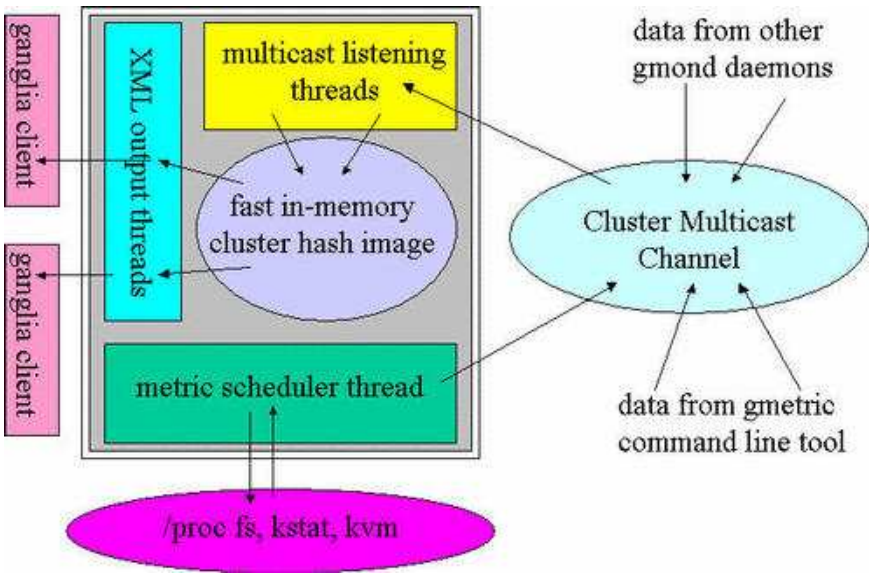


Figura 7.1: Demonio Gmond.

En la Figura 7.1 el rectángulo verde representa al demonio gmond con sus componentes dentro: el planificador de las métricas, el hilo de escucha al canal, la tabla hash en memoria y los hilos de respuesta XML.

- El hilo del planificador chequea el estado del host que está ejecutando gmond. Se decide si ha habido cambios relevantes consultando los umbrales de cambio. Si la diferencia entre el valor anterior y el actual supera el umbral se transmite el cambio. También, si el lapso de tiempo desde que la anterior retransmisión es mayor al umbral de tiempo se realizará la retransmisión sin importar su valor. Cada umbral de tiempo y de valor es específico de la métrica.

- El hilo de escucha del canal realiza la escucha en el canal multidifusión esperando mensajes desde él (loopback) o desde cualquier otro nodo. Todos los datos son almacenados en la tabla hash en memoria.

- El hilo de respuesta XML es el encargado de procesar las peticiones de conexión entrantes. Cuando es realizada una petición de XML, se consulta si el host que ha realizado la petición de conexión está dentro de la lista de host conocidos (trusted list), especificada en el fichero */etc/gmond.conf*. Si el host no se encuentra en esta lista, el hilo cierra la conexión; en caso contrario, se envía una completa descripción del estado del cluster en formato XML. Cuando se habla del cluster se habla de todos los nodos al alcance a través del canal multidifusión. Se puede obtener información de un nodo del cluster simplemente mediante un telnet al puerto *8649*:

  telnet localhost 8649

7.2.2. Ganglia Meta Daemon (gmetad)

El demonio de metadatos de Ganglia se encarga de almacenar la información histórica y exportar información XML que la interfaz web de Ganglia pueda usar para publicar capturas del estado de todos los hosts monitorizados.

En la Figura 7.2 se muestra una representación gráfica de como el demonio de metadados (gmetad) trabaja junto con el demonio de monitorización (gmond) para permitir una monitorización completa del cluster a través de canales *unicast* (redes WAN). Mientras que gmond utiliza canales *multidifusión* en modo *igual a igual (p2p)*, gmetad genera una descripción completa de los recursos del cluster u otros demonios gmetad, a través de un enlace unicast. Es decir reúne toda la información del cluster o clusters monitorizados en un conjunto llamado Grid y la envía como respuesta a peticiones que se realizan por redes unicast mediante conexiones TCP.

La configuración de Gmetad se realiza editando el fichero */etc/gmetad.conf*. Por efecto este fichero contiene las instrucciones de todos los parámetros que es posible configurar. De hecho no hay ninguna guía oficial de cómo configurar el demonio salvo las instrucciones contenidas en los ficheros de configuración.

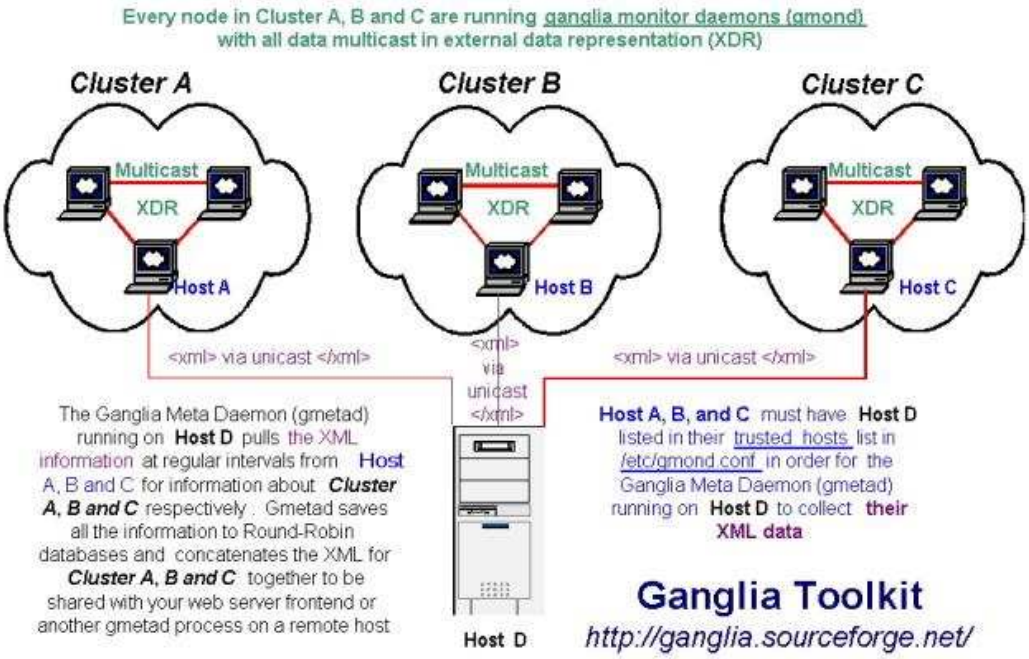


Figura 7.2: Demonio Gmetad.

7.2.3. Ganglia Web Frontend

La interfaz web de Ganglia proporciona una visión global y centralizada del grid o cluster monitorizado a través de páginas web dinámicas (php). Lo más importante, se muestra información de forma gráfica para que los administradores de sistema y usuarios de ordenadores puedan sacar conclusiones rápidas del uso de los recursos de su sistema.

Algunos de los recursos que se visualizan por defecto son:

- Utilización de CPU en el tiempo.
- Uso de Memoria.
- Uso de disco.
- Estadísticas de red en la última hora, día, semana o año.
- Número de procesos en ejecución.
- ...

El interfaz web se ejecuta bajo Apache[28] y necesita de la existencia de un demonio gmetad en la máquina local que obtenga información de los distintos recursos Ganglia.

La interfaz web se encarga de abrir un socket en el puerto local 8651 (por defecto) y esperar a recibir información de los recursos Ganglia.

Si en el fichero de configuración del demonio gmetad se han especificado por ejemplo cuatro recursos (pueden ser ganglia o gmetad), en la interfaz web se mostrará esos cuatro recursos con el nombre que se les haya dado en sus configuraciones locales y los agrupará bajo el nombre local. Por ejemplo: si existe un cluster y estaciones de trabajo con el demonio gmond instalado y tenemos un demonio gmetad y la interfaz web ejecutándose en otro ordenador, entonces se mostrará información de cada estación de trabajo por separado(ya que se tratan de recursos independientes) y el cluster por otro lado. Con esto se consigue una monitorización global con posibilidad de ampliar la resolución en un recurso determinado.

7.3. Instalación y configuración

7.3.1. Consideraciones previas

La maquina y la red deben estar habilitadas de manera multidifusión para poder ejecutar Ganglia, si las maquinas están conectadas a un mismo switch entonces no hay que realizar pasos adicionales. El Gmond esta listo para ser usado sin ninguna configuración previa. Si las máquinas del cluster están separadas por un encaminador, entonces se necesitará configurar la opción mcast_TTL en el /etc/gmond.conf. Este parámetro se utiliza para configurar el campo TTL de los paquetes IP que se transmiten por el canal multidifusión. Todos los encaminadores decrementan en 1 el campo TTL de un datagrama IP cada vez que encaminan un datagrama. Si las maquinas del cluster están separadas por un encaminador, habrá que configurar la opción mcast_TTL en el /etc/gmond.conf. Esto es para elevar el 1 que esta por omisión. También hay que asegurarse de que los encaminadores estén configurados para permitir el tráfico multidifusión.

7.3.2. Descarga e instalación

La herramienta de monitorización Ganglia puede descargarse de la página web del proyecto [26]. Es posible descargar la herramienta en código fuente y paquetes para distribuciones Debian y Red-Hat en formatos rpm y deb.

Si se dispone de una distribución que tiene la herramienta *apt* es posible instalarla de manera muy sencilla siguiendo la Guía Web [30]. Si no se dispone de la herramienta *apt-get*[4] es recomendable descargarla desde código fuente.

Si la instalación se está realizando en el nodo principal de un cluster o en una estación de trabajo que se desee monitorizar de forma independiente hay que descargar la herramienta rrdtool[52] de generación de gráficas necesaria para el *frontend web* de Ganglia.

A continuación se detallan los pasos necesarios para compilar Ganglia versión 3.0.5 con sus demonios *gmond* y *gmetad*; y generar su ficheros de configuración:

1. Se extrae el contenido del fichero comprimido tar.gz con:

   $tar -xvf ganglia-3.0.5.tar.gz

2. **Este paso es opcional, únicamente se realiza si se está sobre una estación de trabajo independiente o en el nodo principal de un cluster**. Se ejecuta el script de configuración indicando que se compile el demonio gmetad. También es posible indicar el directorio de instalación donde se instalarán los ficheros generados. Por defecto se crean en /usr/local/bin, aunque es posible especificarse mediante el parámetro *–prefix= path*

   $./configure –with-gmetad –prefix= path

3. Si no ha habido ningún error debido a dependencias inexistentes entonces proceder al compilado e instalación.
   $make
   ...
   $make install

   Los ficheros se generarán en donde se haya especificado en el parámetro *–prefix*.

7.3.3. Configuración de los demonios

1. Generar los ficheros de configuración. Para ello hay que ejecutar desde la carpeta en donde se ha instalado *ganglia* (se indica mediante el parámetro *–prefix* al script de configuración):

   $gmond –default_config gmond.conf
   $gmetad

   Para el caso de gmetad no hay que especificar ningún parámetro, se creará el fichero de configuración gmetad.conf de forma automática en el mismo directorio donde está el binario *gmetad* que se ha ejecutado.

2. Copiar los ficheros *gmond.conf* y *gmetad.conf* a la carpeta */etc/*.

3. Copiar los ficheros binarios *gmond* y *gmetad* a la carpeta */etc/init.d/*. Hay que darles permisos de ejecución. Además crear enlaces simbólicos:

   - */etc/rc3.d/gmond -¿/etc/init.d/gmond*
   - */etc/rc3.d/gmetad -¿/etc/init.d/gmetad*

   Esto garantiza que los demonios se ejecuten durante el arranque del sistema. Todo esto debe ser realizado desde root.

Una vez en este instalado y configurado Ganglia pueden arrancarse los demonios mediante:

$/etc/init.d/gmond start
$/etc/init.d/gmetad start

7.3.4. Instalación y configuración del Web Frontend de Ganglia

Hay que descargar desde la página web de Ganglia [26] el fichero comprimido con los fuentes de Ganglia, también es posible bajarse un paquete .rpm para distribuciones Linux que lo soporten. En este documento se seguirán las instrucciones para hacer funcionar el frontend de Ganglia desde los fuentes. Partiendo de la descarga del fichero de la página web hay que seguir los siguientes pasos:

1. Extraer el contenido del fichero comprimido.
2. Instalar el servidor web Apache si es que previamente no estaba instalado.
3. Instalar la herramienta de visualización *RRDtool*.
4. configurar Apache para php.

```
cd /etc/apache
vi http
# 20070420 - CLC - Inicio
# Para ganglia
  <IfDefine HAVE\_PHP4>
     LoadModule php4\_module     extramodules/libphp4.so
     AddModule mod\_php4.c
  </IfDefine>
  AddType  application/x-httpd-php  .php .php4 .php3 .phtml
  AddType  application/x-httpd-php-source  .phps
# 20070420 - CLC - Fin
```

5. Reiniciar el servidor apache.
6. Copiar la carpeta *web* de la carpeta extraída 1 a */var/www/ganglia*.
7. Cambiar el propietario de la carpeta al usuario *www-data*.
   chown -R www-data /var/www/ganglia
8. Actualizar el archivo de versión a la versión actualmente instalada de gmond.
   cd /var/www/ganglia cp version.php.in version.php gmond –version
9. Actualizar el archivo según resultado del comando anterior.

```
vi version.php
<?php
# This file is autogenerated

$majorversion = 3;
$minorversion = 0;
$microversion = 5;

$ganglia\_version = "3.0.5";
$ganglia\_release\_name     = "";

?>
```

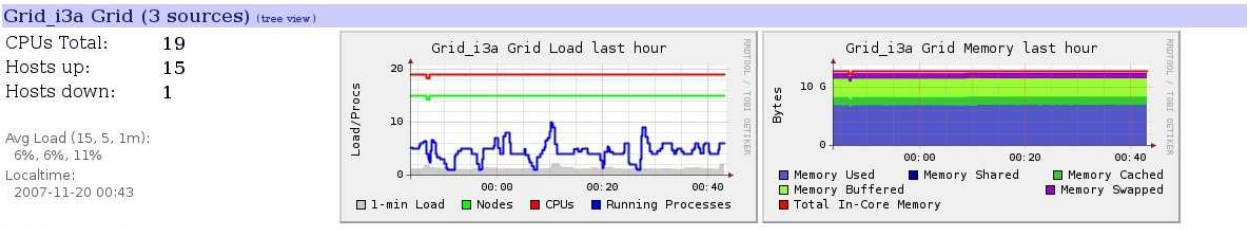


Figura 7.3: Gráfica de carga computacional del Grid I3A.

7.4. Modificar el servicio Globus MDS para obtener información de Ganglia

Basta con modificar *$GLOBUS_LOCATION/etc/globus_wsrf_mds_usefulrp/gluerp.xml*. Hay que dejar el fichero como se indica en la Figura 7.4.

```
<config xmlns="http://mds.globus.org/2004/10/gluerp-config">
<!--
    <defaultProvider>none</defaultProvider>

To enable the use of ganglia to provide cluster information,
replace the above
with the following:
-->

<defaultProvider>java
org.globus.mds.usefulrp.glue.GangliaElementProducer
</defaultProvider>

</config>
```

Figura 7.4: Código de configuración de gluerp.xml.

En la versión de Globus ToolKit 4.0.5 hay que realizar un paso de configuración adicional. Hay que ejecutar el comando Globus *mds-gluerp-configure* para que se genere el recurso GLUE apropiado para el gestor de recursos local. El recurso GLUE obtiene

información de estado del recurso local y se al proporciona a Globus. Hay que realizar tres pasos:

1. Cambiar al directorio $GLOBUS_LOCATION/etc/gram-service-<LOCAL_LRMS> (Fork, Condor, LSF, PBS).
2. Ejecutar *mds-gluerp-configure <lrms> ganglia*. Se generará un fichero gluerp-config.xml.
3. Editar el fichero gluerp-config.xml para asegurarse que los puertos de Ganglia y la localización de este están correctamente configuradas.

7.5. Ejemplo real de configuración de Ganglia y Gmetad

Para el ejemplo real de configuración se va a tomar como referencia la instalación de Ganglia como herramienta de monitorización del Grid del Laboratorio de Redes de Altas prestaciones del Instituto de Investigación en Informática de Albacete (I3A) [49].

Este grid está formado por:

a) Un cluster Myrinet compuesto por 12 máquinas monoprocesador. El nodo principal es el que tiene dos interfaces de red, una Myrinet para comunicarse con los otros nodos y el otro para comunicarse con el *mundo exterior*.

b) 4 estaciones de trabajo Sun con procesador de doble núcleo AMD Opteron 64 bits.

c) Estación principal con servidor web que será la que ejecute el *web frontend* de Ganglia.

7.5.1. Configuración del cluster

En cada nodo del cluster se lleva a cabo la instalación de Ganglia siguiendo los pasos detallados en apartados anteriores.

A partir de aquí hay que aplicar una configuración específica. Todos los nodos del cluster deben agruparse dentro de un *nombre de cluster* que se especifica editando el fichero **gmond.conf** de cada nodo del cluster.

```
cluster {
  name = "LINCE"
}
```

Puede observarse que el atributo *nombre de cluster* toma como valor "LINCE", el cluster Myrinet a partir de ahora se llamará "LINCE".

En el nodo principal: *myr1*, gmond se configura igual pero además hay que configurar gmetad7.2.2, ya que este nodo, que es el único que tiene acceso desde el

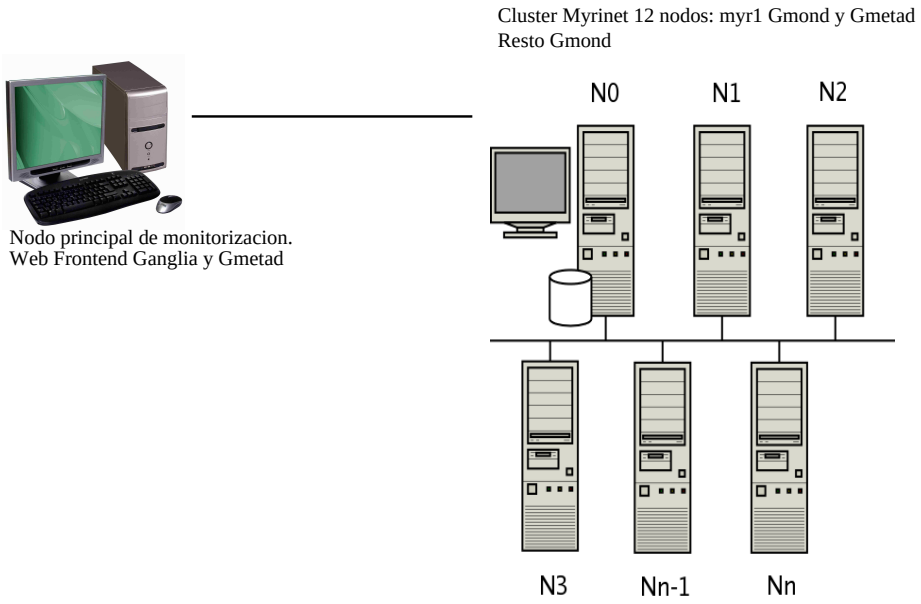


Figura 7.5: configuración Grid con Cluster.

exterior, se encargará de recopilar la información de los demás nodos y publicarla mediante XML a través de su interfaz pública. En la página siguiente se expone parte del contenido del fichero *gmetad.conf*.

```
#-------------------------------------------------------------------------------
# What to monitor. The most important section of this file.
#
# The data\_source tag specifies either a cluster or a grid to
# monitor. If we detect the source is a cluster, we will maintain a complete
# set of RRD databases for it, which can be used to create historical
# graphs of the metrics. If the source is a grid (it comes from another gmetad),
# we will only maintain summary RRDs for it.
#
# Format:
# data\_source "my cluster" [polling interval] address1:port addreses2:port ...
#
# The keyword 'data\_source' must immediately be followed by a unique
# string which identifies the source, then an optional polling interval in
# seconds. The source will be polled at this interval on average.
# If the polling interval is omitted, 15sec is asssumed.
#
# A list of machines which service the data source follows, in the
# format IP:port, or name:port. If a port is not specified then 8649
# (the default gmond port) is assumed.
# default: There is no default value
#
# data\_source "my cluster" 10 localhost  my.machine.edu:8649  1.2.3.5:8655
# data\_source "my grid" 50 1.3.4.7:8655 grid.org:8651 grid-backup.org:8651
# data\_source "another source" 1.3.4.7:8655  1.3.4.8

#data\_source "my cluster" localhost

data\_source "lince\_condor" localhost  myr2 myr3 myr4 myr5 myr6 myr7
myr8 myr9 myr10 myr11 myr12 myr13 myr14

#
# Round-Robin Archives
# You can specify custom Round-Robin archives here (defaults are listed below)
#
# RRAs "RRA:AVERAGE:0.5:1:240" "RRA:AVERAGE:0.5:24:240" "RRA:AVERAGE:0.5:168:240" \
"RRA:AVERAGE:0.5:672:240" \
#       "RRA:AVERAGE:0.5:5760:370"
#
```

Siguiendo las instrucciones del fichero de configuración hay que crear un *data_source* que es un string que identifica un origen de datos. Pueden agregarse varios recursos al origen de datos, para ello hay que especificar la ruta del recurso y el puerto al que lanzar peticiones. Por defecto se hacen peticiones al puerto **8649** de Ganglia, pero también pueden enviarse consultas a otros demonios gmetad, que escucha por defecto en el puerto **8651**.

Advertencia: Si el host sobre el que se está realizando la instalación de Ganglia dispone de dos interfaces puede que haya problemas estableciendo el canal *multidifusión*. Puede que este canal se enrute a otra interfaz que no sea la que está conectada a los demás nodos del cluster. Esto puede consultarse en los logs de sistema */var/log/services*. Si disponemos de dos interfaces, una pública y otra privada y el canal multidifusión es necesario que sea hacia la interfaz privada es necesario añadir a la tabla de enrutamiento una regla que indique que el canal muticasting se enruta a través de la interfaz de red privada o interna.

route add 239.2.11.71 192.168.0.1 Donde *239.2.11.71* es el canal multidifusión y *192.168.0.1* es la dirección de la interfaz de red privada.

7.5.2. Configuración de estaciones de trabajo SUN

La configuración de las estaciones de trabajo puede realizarse siguiendo los pasos anteriores 7.3, con la peculiaridad de que la estación *aulasun16.dsi.uclm.es* ejecutará el demonio gmetad y publicará información de las 4 estaciones de trabajo SUN. En *gmond.conf* hay que especificar para todas las máquinas el nombre del Grid del que se forma parte, en este caso *AulaSUN*. El contenido del fichero de configuración *gmetad.conf* de dicha máquina:

data_source “AulaSUN”0 localhost aulasun14.dsi.uclm.es aulasun15.dsi.uclm.es aulasun15.dsi.uclm.es aulasun17.dsi.uclm.es

Se genera una fuente de datos *AulaSUN* que contiene los recursos de las 4 máquinas SUN.

7.5.3. Configuración de la estación que soporta el Web Frontend de Ganglia

Siguiendo los pasos descritos anteriormente en 7.3 únicamente hay que aplicar una configuración específica a los demonios gmond y gmetad para que obtengan información de los recursos del Grid del I3A consultando los puertos correctos que se corresponden con el puerto del demonio gmetad, ya que los recursos van a estar agrupados en distintos grupos:

- LINCE_CLUSTER.
- AulaSUN.
- Grid.

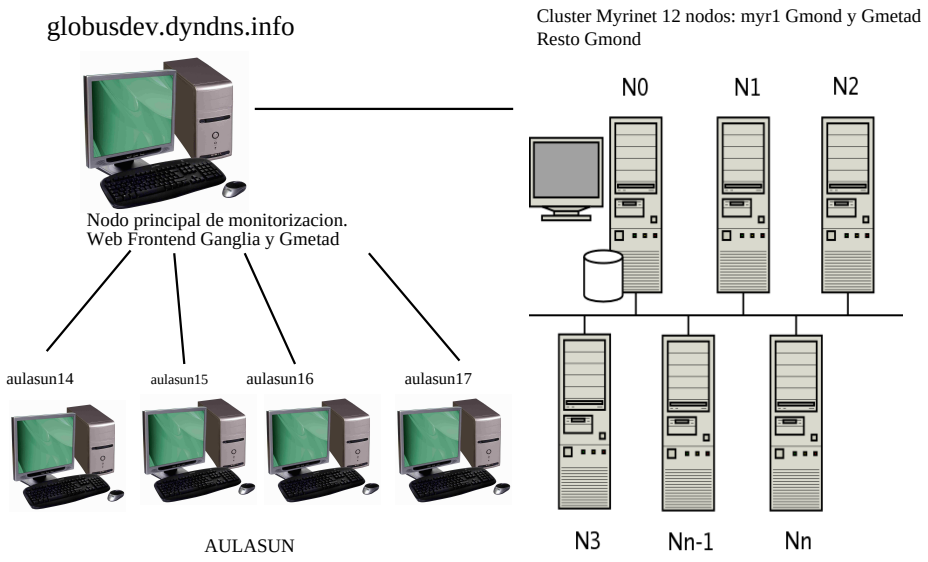


Figura 7.6: configuración Grid con máquinas SUN.

La máquina debe tener ejecutándose los demonios gmetad y gmond (ella misma también se publica en la web). El contenido del fichero de configuración *gmetad.conf* es:

```
data_source "GRID1" 10 localhost
data_source "GRID2" 10 aulasun15.dsi.uclm.es:8651
data_source "GRID3"  10 Myrinet.i3a.info:8651
```

Puede observarse que se definen tres fuentes de datos distintas. Una obtiene datos de *localhost*, otra del demonio gmetad del AulaSUN y otra para el cluster Myrinet.

Figura 7.7: Captura del frontend web para el Grid I3A.

El nombre que se les asigna a cada fuente de datos se sobreescribe por el nombre proporcionado por el demonio gmetad al que se consulta para esa fuente de datos.

Puede consultarse el estado del Grid del laboratorio RAAP en la página: *http://globusdev.dyndns.info/ganglia*

7.5.4. Agregando recursos Grid de otras VO

Si se está desarrollando una federación de Grids entre varias VO, en los que los recursos foráneos se integran con los recursos locales y viceversa, sería recomendable poder monitorizar de forma centralizada toda la información sobre los recursos disponibles en los distintos Grids. Que mejor que realizarlo con Ganglia, y poder visualizar toda la información de todos los Grids de forma conjunta. Para instalar el frontend Web sería necesario otra máquina con un demonio gmetad ejecutándose y recopilando información de los Grids que componen la federación de Grids. Esta máquina deberá tener acceso a los puertos de los demonios gmetad remotos de los que se pretende obtener información. Los distintos demonios gmetad remotos recopilarán información sobre sus respectivos recursos de su VO. Puede verse un diagrama de esta configuración en la Figura 7.8 como ejemplo para cuatro organizaciones distintas. En este caso, universidades.

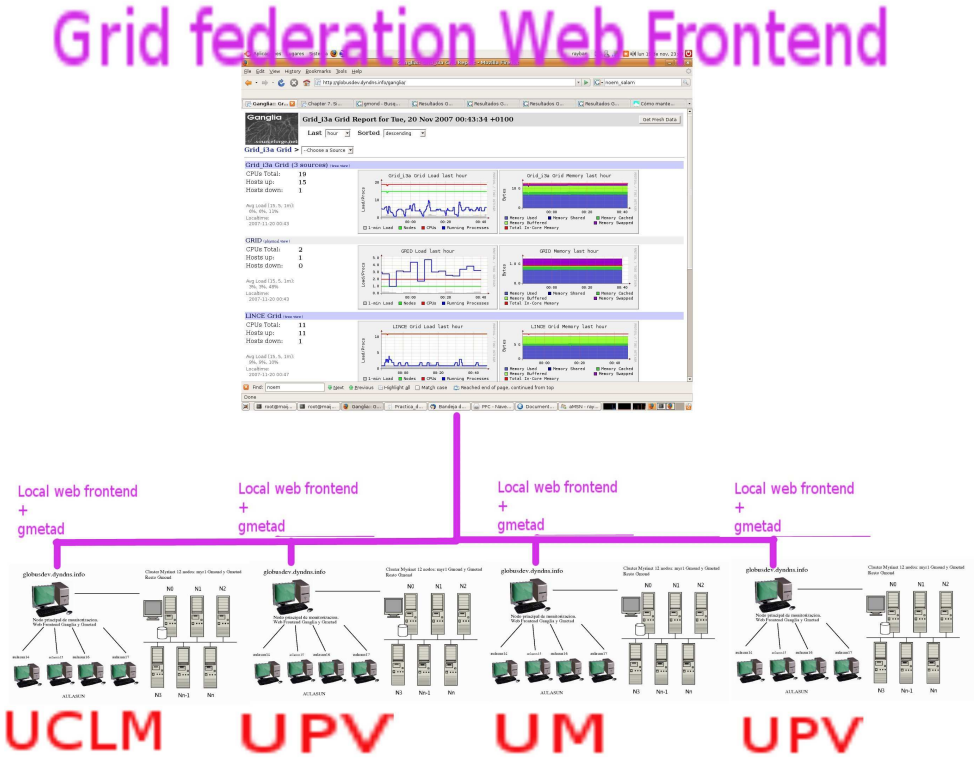


Figura 7.8: Monitorización de los recursos de cuatro cuatro organizaciones que forman una federación Grid.

La configuración del web frontend principal se realizará editando el demonio gmetad, que será el que recopilará información de los distintos Grids.

```
gridname "Grid Federado"
...
datasource "GRIDUCLM" globusdev.dyndns.info:8651
datasource "GRIDUM" gridum.um.es:8651
datasource "GRIDUV" griduv.uv.es:8651
datasource "GRIDUPV" gridupv.upv.es:8651
...
```

Habitualmente la información de cada organización se obtiene del demonio gmetad de cada una. El demonio por defecto escucha peticiones en el puerto **8651**. Es necesario que este puerto sea accesible desde el exterior de la organización.

Capítulo 8

Metaplanificador GridWay

8.1. Introducción

GridWay es un middleware Grid de capa de usuario, de código abierto, con licencia Apache 2.0 en principio desarrollado por el *Departamento de Arquitectura de Computadores y Automática* de *la Facultad de Informática* de la *Universidad Complutense de Madrid*. Cuando nos referimos a *GridWay* como middleware hablamos de *grid middleware* de alto nivel, es decir, que usa los servicios proporcionados por otro *Grid middleware*, en este caso usa los servicios de Globus como núcleo del Grid. *GridWay* es un gestor de carga que realiza gestión de trabajos y agente de recursos del grid, compuesto por distintas plataformas de cómputo gestionadas por servicios Globus. *GridWay* soporta:

- Ejecución desatendida, confiable y eficiente de trabajos simples, matrices de trabajos y flujos de trabajo complejos con interdependencias.
- Planificación y envío de trabajos de forma totalmente transparente para el usuario final.
- Tolerancia ante errores y técnicas de recuperación.
- Planificación dinámica de tareas que incluso actualmente se están ejecutando. Una tarea que se está ejecutando en un recurso inferior a otro que ha quedado libre puede trasladarse al recurso que ha quedado libre si la tarea así lo especifica.

8.2. Arquitectura

La arquitectura de *GridWay* consiste en 5 componentes:

- **Interfaz de usuario** que proporciona al usuario final comandos tipo *DRM*, típicos de los gestores de recursos locales: *submit, kill, migrate, monitor y synchronize*. Además se incluye una interfaz de programación de aplicaciones DRMAA

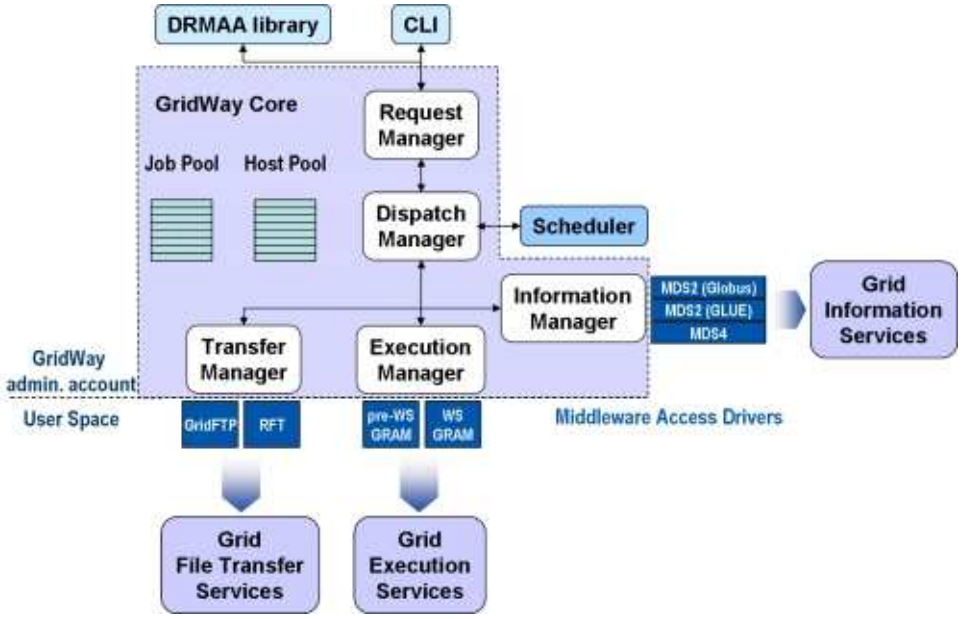


Figura 8.1: Arquitectura del metaplanificador GridWay.

(Distributed Resource Management Application API) afín con el estándar GGF (Global Grid Forum) [57]. Esta API está disponible para aplicaciones C y Java.

- **El núcleo de GridWay** se ejecuta como un demonio que consta de:

  - El **Gestor de emisión de trabajos** (Dispatch Manager) realiza todas las acciones de emisión de trabajos basándose en las necesidades de cómputo de la tarea para escoger el recurso.
  - El **Gestor de información y descubrimiento** obtiene la información del medio del Grid a través de los *MADS* (Medium Access Driver). Este controlador de acceso al medio es el responsable del descubrimiento de hosts y monitorización de recursos y trabajos en esos recursos.
  - **El Gestor de Ejecución** a través del *MAD* es responsable del *file staging* o traslado de ficheros antes y después de la ejecución, creación de directorios de trabajo remotos e inicialización del entorno de ejecución del trabajo. También lleva a cabo las tareas de limpieza del nodo de cómputo una vez la ejecución del trabajo finaliza.

- El **driver de Gestión de Información** y descubrimiento (Information Manager MAD) es utilizado por el núcleo del metaplanificador para acceder a los servicios de monitorización y descubrimiento proporcionados por los recursos Globus.

- **El driver de Gestión de Ejecución** (Execution Manager MAD) lo utiliza el núcleo del metaplanificador para acceder al servicio GRAM de Globus, que es el servicio de gestión de ejecución de los recursos Globus del Grid.

8.3. Infraestructuras con GridWay

GridWay, como metaplanificador de trabajos dentro de un Grid, ofrece la posibilidad de implementar varias estructuras dependiendo del nivel al que esta tecnología de metaplanificación funcione.

8.3.1. Infraestructuras formadas por VO amigas (partner infrastructure)

Las Infraestructuras formadas por varias VO amigas dentro del contexto de diferentes proyectos de investigación, se despliegan con objetivo final de proporcionar una manera confiable y segura de compartir recursos a gran escala entre ellas. Como se dijo en el apartado 6.4, es necesario administrar los recursos existentes en la infraestructura formada por estas VO amigas. Esto hace difícil una gestión centralizada inter-VO de los recursos pertenecientes a varias VO. Existen niveles a los que el metaplanificador puede planificar:

- **Planificación a nivel de usuario**: la mayoría de los metaplanificadores actuales son herramientas a nivel de usuario que proporcionan planificación y gestión de trabajos, pero por cada usuario. Se creará una instancia de estas herramientas por cada usuario del sistema, por lo que cada metaplanificador competirá con el resto de metaplanificadores por el control de los recursos del Grid.

- **Planificación a nivel de VO**: los problemas derivados de la planificación a nivel de usuario son los mismos que si no tenemos ningún metaplanificador, ya que al final, los usuarios acceden sin control a los recursos de la infraestructura con la consiguiente falta de seguridad y control de trabajos ya que hay que abrir los puertos del firewall para que se pueda acceder desde cualquier lugar a los servicios Globus y además dar permisos de ejecución para muchos usuarios en lugar de *mapear* los usuarios en pocos usuarios autorizados para favorecer el control de la seguridad. Aquí entra en juego la planificación centralizada a nivel de VO que da soporte para múltiples usuarios intra-organización por cada estancia del metaplanificador. Esto significa que hay una instancia por organización, con el consiguiente control para los usuarios intra-organización e inter-organización. Llegados a este punto, quién compite ahora por los recursos, son los metaplanificadores de cada organización, a nivel de organización, no a nivel de usuario. Este modo de instalación y configuración del planificador *GridWay*, llamado *instalación multiusuario*, es administrado por un usuario administrador de *GridWay* por VO, y los usuarios finales que se comunican con *GridWay* desde sus hosts no necesitan tener instalado ni los servicios de Globus ni el metaplanificador GridWay.

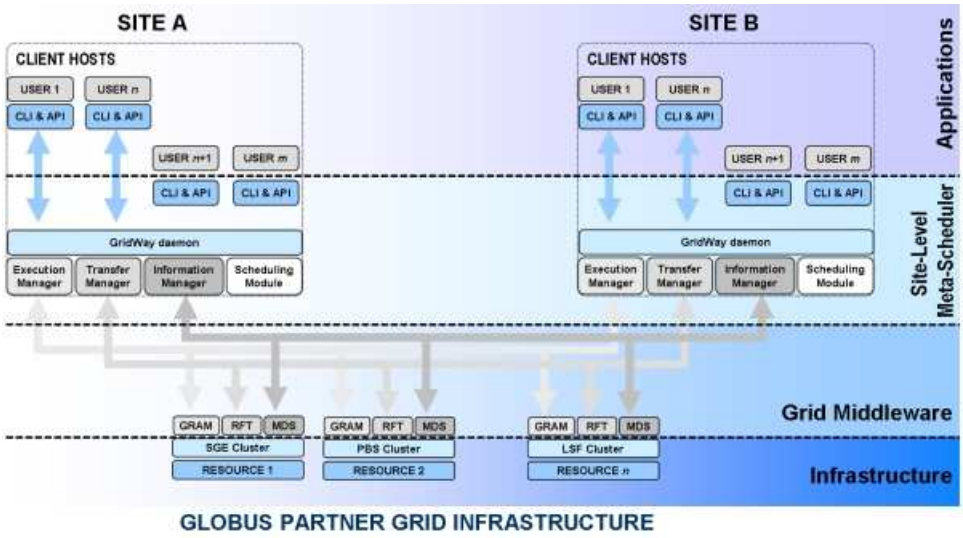


Figura 8.2: Infraestructura *GridWay* a nivel de VO.

8.3.2. Infraestructuras de Empresa

Los Grids en empresas (*Enterprise Grids*), mejoran la gestión de sus recursos permitiendo un mejor aprovechamiento de éstos con la consiguiente ganancia con respecto a la inversión realizada en tecnologías de la información (hardware y software). Los recursos disponibles dentro de una compañía se explotan mejor y la sobrecarga de administración se minimiza mediante la tecnología Grid. Todos estos recursos pertenecen a una misma compañía, y por lo tanto al mismo dominio administrativo. Por ello, esta infraestructura que puede ser o no virtual es centralizada, ya que utiliza políticas centralizadas para la planificación de trabajo y administración de acceso. El administrador debería disponer de políticas de acceso, uso, monitorización e histórico de forma centralizada. En este tipo de infraestructura se aplicaría la configuración *multi-usuario* de GridWay, en la que una única instancia de *GridWay* da soporte a varios usuarios, y estos, como en el apartado anterior no necesitan disponer de una instalación de Globus ni de *GridWay* en sus sistemas.

8.4. Funcionamiento del metaplanificador GridWay

La interacción del usuario final con *GridWay* se puede llevar a cabo mediante su interfaz de línea de comandos CLI o mediante la API DRMAA que proporciona GridWay, en los dos casos la comunicación se lleva a cabo con el demonio *gwd* de GridWay. Por defecto el demonio de *GridWay* espera peticiones de usuarios en el puerto 6725 .

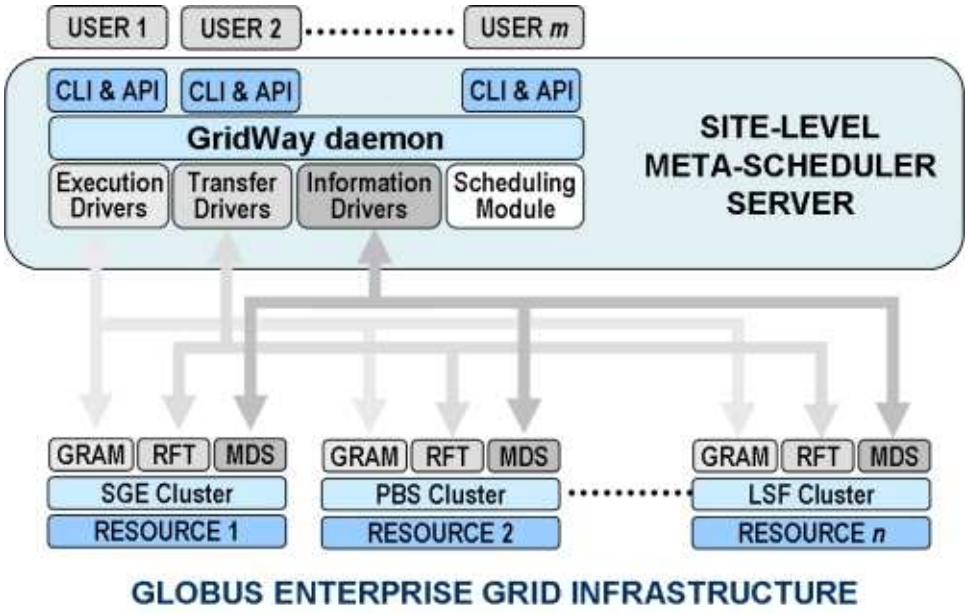


Figura 8.3: Infraestructura *GridWay* en empresas.

8.4.1. Estados de un trabajo

Los trabajos, una vez que llegan a *GridWay*, adquieren un estado en función de lo que se le indique mediante el lenguaje CLI o la API de programación.

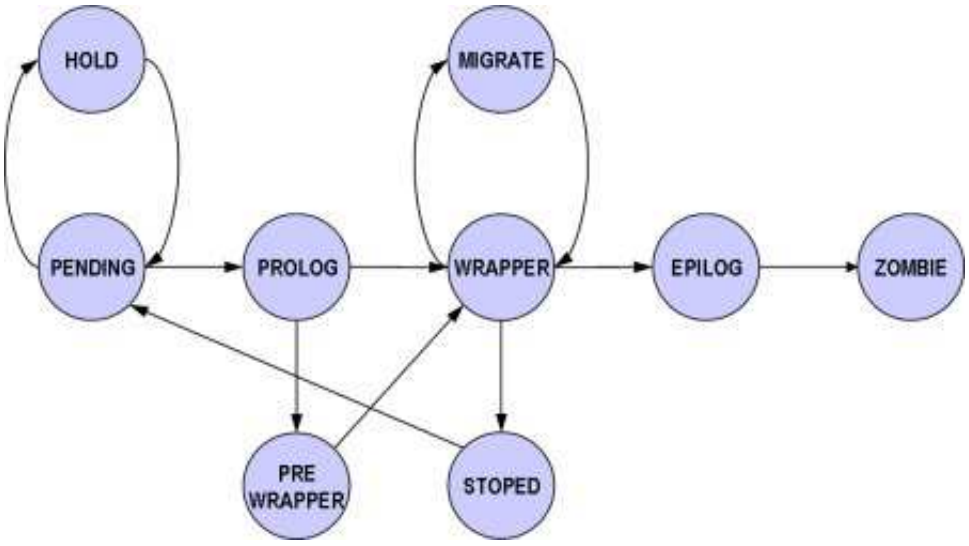


Figura 8.4: Máquina de estados simplificada de un trabajo en GridWay.

Los estados de la Figura 8.4 se detallan a continuación:

- **PENDING (pend)** (Pendiente): el trabajo está esperando un recurso en el que

ejecutarse. Se llega a este estado al entrar en el sistema o cuando el trabajo se ha reiniciado debido a un fallo, ha habido una parada o una automigración debida a recursos insuficientes una vez asignado a un recurso que no puede soportar el trabajo.

- **HOLD (Mantenido)**: el propietario del trabajo o el administrador de *GridWay* puede mantener un trabajo. Este trabajo no será planificado hasta que no reciba una señal de *release*.
- **PROLOG (prol)**: el trabajo se está preparando en el nodo de cómputo creando los directorios de trabajo temporales y transfiriendo los ficheros de entrada, ejecutables y ficheros de restauración en caso de migración.
- **WRAPPER (wrap)**: el trabajo está ejecutando el programa cargador o Wrapper. Este programa se encarga de preparar el sistema para la ejecución del ejecutable principal y ejecutarlo. Este programa cargador también ejecuta un programa de monitorización si se especifica. El programa monitor mira el tiempo de CPU que es utilizado por el programa ejecutable.
- **EPILOG (epil)**: el trabajo está finalizando. En esta fase los ficheros de salida y de restauración son transferidos a la máquina de origen desde el nodo de cómputo. Los ficheros y directorios se limpian del directorio remoto si todo va bien.
- **MIGRATION (migr)**: el trabajo se está migrando de un recurso a otro, cancelando la ejecución del wrapper y limpiando los directorios en el viejo recurso (como en la fase Epilog). Además se realizan las tareas de preparación del trabajo en el nuevo recurso (como en la fase Prolog).
- **Stopped (stop)**: el trabajo está parado. Si tiene ficheros de restauración que han sido definidos en la plantilla del trabajo, éstos son transferidos de vuelta al cliente, y serán usados cuando el trabajo se re-arranque de nuevo, si se re-arranca.
- **FAILED (fail)**: el trabajo ha fallado por cualquier motivo. Conviene consultar el log de error.
- **DONE (done)**: el trabajo se ha realizado sin problema alguno y ha finalizado correctamente.

A su vez, el estado Wrapper tiene **subestados** que se explican a continuación:

- **PENDING (pend)**: el trabajo ha sido enviado con éxito al gestor local de recursos DRMS del recurso Grid, y está esperando a que el sistema DRM local lo ejecute. En un recurso con un LRMS tipo FORK, el trabajo simplemente está esperando a que se ejecute. En un LRMS tipo Condor por ejemplo, el trabajo está esperando a que el gestor de trabajos de Condor lo envíe al nodo de cómputo del cluster más conveniente para Condor y para el trabajo.
- **SUSPENDED (susp)**: el trabajo ha sido suspendido por el gestor local de recursos.
- **ACTIVE (actv)**: el trabajo ha entrado en ejecución en el nodo de cómputo.

- FAILED (fail): el trabajo ha fallado al ejecutarse en el nodo de cómputo.

- **DONE (done)**: el trabajo se ha terminado correctamente en el nodo de cómputo. A partir de aquí entraría en fase epilog y habría que traer de vuelta a la máquina principal de *GridWay* los resultados, o dónde se especifique en la plantilla de trabajo.

8.4.2. Como *GridWay* ejecuta trabajos

Cuando *GridWay* recibe un trabajo, lo primero que hace es leer su plantilla de trabajo. De esta obtiene información acerca de todos sus requerimientos como ficheros de entrada y salida; entradas y salidas estándar; fichero ejecutable, argumentos, etcétera. Con esta información *GridWay* genera un fichero *job.env* en la carpeta *$GW_LOCATION/var/jobid/*. Este fichero contiene variables de entorno que se definen para la ejecución de un programa *wrapper* o cargador que se enviará al nodo de cómputo donde se ejecutará el trabajo y servirá para "cargar" todos los ficheros de entrada y ejecutar el trabajo en sí. El contenido de un fichero *job.env* hipotético puede verse en la Figura 8.4.2. Además del fichero *job.env* se genera un fichero en lenguaje *Resource Specification Language (RSL)*, que será el que se encargué de ejecutar el programa cargador en el nodo destino. El contenido de este fichero puede verse en la Figura 8.4.2 y en él se define el fichero ejecutable, en este caso el fichero *wrapper* que será el programa cargador. Una vez están todos los ficheros generados y *GridWay* selecciona el trabajo para ejecución se producen las siguientes actividades:

1. El trabajo se selecciona para la ejecución a través del servicio *GridFTP* de *Globus*. Se genera la estructura de carpetas en el nodo de cómputo, donde serán copiados el wrapper, el fichero de entorno, los ficheros de entrada y el ejecutable del trabajo. El estado del trabajo sería *PROLOG*.

2. Si todo ha ido bien, *GridWay* realiza la ejecución de la orden *globus-run-ws* con el fichero *job.rsl* de descripción del trabajo, que ejecuta el programa cargador en el destino. La tarea pasaría del estado *PROLOG* al estado *WRAPPER PEND*.

3. El programa cargador realizará tareas de comprobación en el nodo remoto. Comprobará que existe el directorio de trabajo y los ficheros de entrada. Si es así, el programa daría permisos de ejecución al fichero ejecutable y lo lanzaría en *background*. Se pasaría a estado *WRAPPER ACTIVE*.

4. Si por cualquier motivo se cancela el trabajo desde el nodo cliente, a través del servicio GRAM de Globus se enviará una señal al trabajo para que muera, en este caso, el programa cargador. Al morir el programa cargador también morirá la tarea ejecutada en background.

5. Si el trabajo finaliza correctamente el programa cargador termina. En este momento el trabajo pasará al estado *WRAPPER DONE*.

6. Al terminar la ejecución del programa, Globus se encarga de transferir de vuelta los ficheros *stdout.wrapper* y *stderr.wrapper*, ya que se ha especificado en el fichero RSL.

7. *GridWay* se encarga de transferir de vuelta desde el nodo de cómputo los ficheros de salida al nodo de GridWay. Esta fase se conoce como *EPIL*.

8. Cuando todos los ficheros están en nodo de GridWay, en el directorio del usuario se pasaría al estado *DONE* y el trabajo finalizaría.

En la Figura 8.7 puede verse el diagrama de flujo de la ejecución del programa *wrapper* en un nodo recurso.

```
export GW_CPULOAD_THRESHOLD="0"
export GW_OS_NAME="Linux"
export GW_OS_VERSION="2.6.10-1.770_FC2"
export GW_CPU_MODEL="x86"
export GW_CPU_MHZ="2400"
export GW_MEM_MB="537"
export GW_DISK_MB="310560"
export GW_PARAM="4"
export GW_MAX_PARAM="8"
export GW_ARCH="x86"
export GW_EXECUTABLE="file:///home/usuario/copy/ED"
export GW_ARGUMENTS="A -i 4 "
export GW_STDIN="/dev/null"
export GW_STDOUT="stdout.133"
export GW_STDERR="stderr.133"
export GW_STAGING_URL="Myrinet.i3a.info"
export GW_JOB_HOME="/home/usuario"
export GW_INPUT_FILES="file:///home/usuario/copy/sp.A sp.A,\
file:///home/usuario/copy/libgfortran.so.1 libgfortran.so.1,\
file:///home/usuario/copy/libgcc_s.so.1 libgcc_s.so.1"
```

Figura 8.5: Contenido de un hipotético fichero *job.env*.

```
<job>
<executable>.gw_umuser_138/.wrapper</executable>
<environment>
<name>GW_HOSTNAME</name> <value>aulasun15.dsi.uclm.es</value>
</environment>
<environment> <name>GW_USER</name> <value>umuser</value><
/environment>
<environment> <name>GW_JOB_ID</name> <value>138</value><
/environment>
<environment> <name>GW_TASK_ID</name> <value>0</value>
</environment>
<environment> <name>GW_ARRAY_ID</name> <value>-1</value><
/environment>
<environment> <name>GW_TOTAL_TASKS</name> <value>0</value>
</environment>
<environment> <name>GW_RESTARTED</name> <value>0</value>
</environment>
 <stdout>.gw_umuser_138/stdout.wrapper</stdout>
<stderr>.gw_umuser_138/stderr.wrapper</stderr>
 <queue>default</queue>
</job>
```

Figura 8.6: Contenido de un hipotético fichero *job.rsl*.

8.4.3. Planificación en GridWay

Un planificador Grid o metaplanificador (véase capítulo 3.2) es el responsable de asignar los trabajos que se envian al Grid a los recursos Grid disponibles. Las decisiones de *cuándo* y *dónde* se ejecutará un trabajo son decisiones tomadas periódicamente por el planificador. La frecuencia de planificación se puede ajustar en función de las necesidades de los usuarios y los recursos disponibles.

El planificador, para tomar decisiones, obtiene información de las siguientes fuentes:

- Trabajos enviados al sistema, los cuales incluyen trabajos pendientes y trabajos ejecutándose. Los trabajos que fallaron o que no pueden ser ejecutados puesto que los requisitos que requieren no pueden ser satisfechos no entran en el grupo.
- Resultados de emparejamiento trabajo-recurso. El MAD de gestión de información obtiene del Grid información de los recursos disponibles para realizar un seguimiento de éstos. Esta información es usada por el núcleo de *GridWay* para crear una lista de recursos recomendados para la tarea en función de los requisitos de ésta y el estado de éstos.
- Comportamiento de los recursos. Se penalizan los recursos que fallan demasiado, es decir, hacen que fallen un gran número de trabajos que se le envían.
- Histórico de uso de los recursos. El tiempo de espera en en su LRMS local, tiempo de transferencia, etc...

Si ningún recurso ha sido asignado a una tarea en estado PENDING, en una decisión de planificación, ésta quedará en estado PENDING hasta la siguiente decisión de planificación, en la que competirá de nuevo por los recursos disponibles con las demás tareas del sistema.

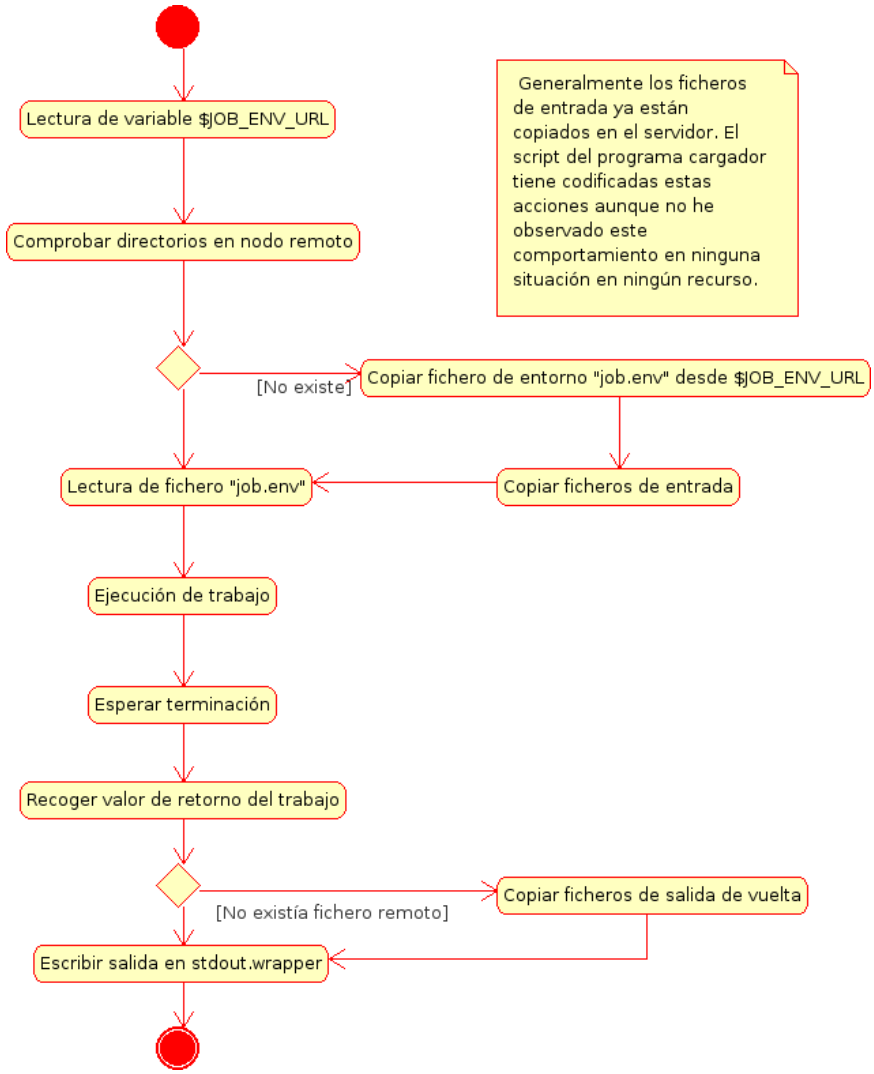


Figura 8.7: Diagrama de flujo del programa *wrapper*.

Una vez que llega un trabajo al sistema, éste puede llegar en estado *HOLD* o *PENDING*. El trabajo pasaría a un *banco* de trabajos y NO una cola, en la que los trabajos esperarían a que les tocara para ser ejecutados. Una vez aquí entrarían en juego las políticas de planificación. Las políticas de planificación se dividen en dos grupos:

- Políticas de priorización de trabajos.
- Políticas de priorización de recursos.

Políticas de priorización de trabajos

En *GridWay* hay implementadas cuatro políticas de planificación de trabajos: *fixed, share, deadline* y *waiting-time*. Los trabajos que llegan al sistema se planifican mezclando las políticas, dando más peso a unas o a otras en función de la configuración

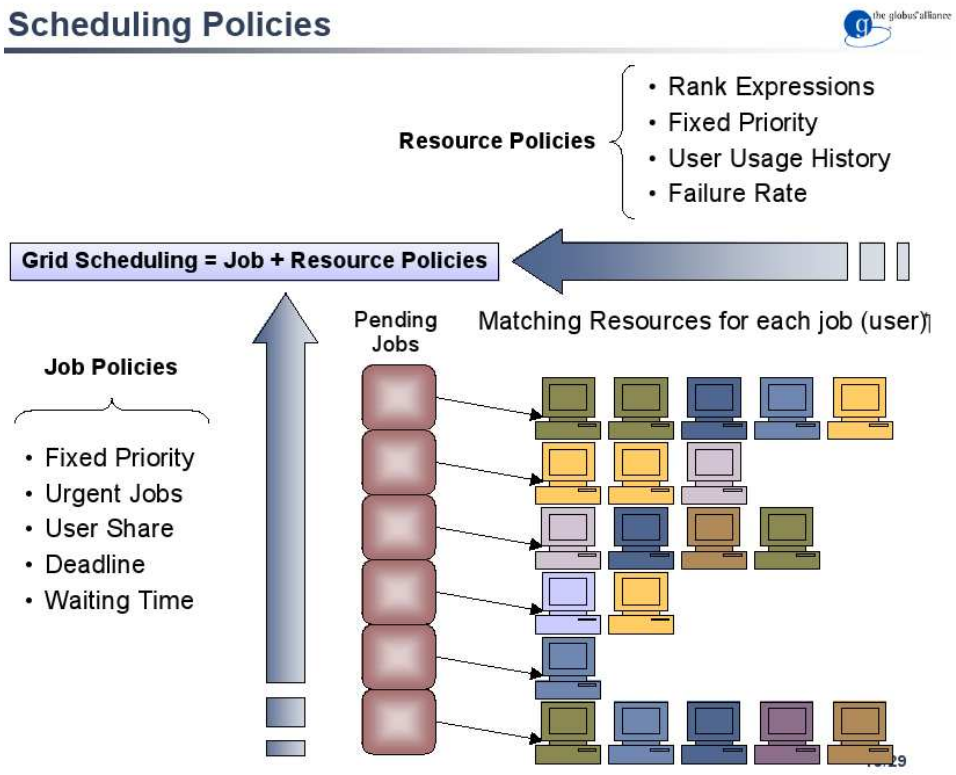


Figura 8.8: Planificación en GridWay.

del planificador (véase apartado 8.7.5). Los trabajos se ordenan en función de los valores obtenidos al evaluar esas políticas, y en función de ese orden podrán ser o no ejecutadas (véase Figura 8.8). Todo esto es a nivel de *dominio de planificación* o instancia en ejecución de GridWay.

- **Política FIXED**:
  Esta política asigna una prioridad fija a cada tarea. Esta prioridad varia entre la prioridad más baja *0* y la prioridad más alta *19*. Los trabajos con una prioridad más alta serán ejecutados antes de los que tienen menor prioridad. La prioridad por defecto de los trabajos es asignada por el administrador del Grid usando los criterios siguientes:

  Usuario: a todos los trabajos del usuario se les asignará una prioridad fija establecida en el fichero de configuración del planificador (véase apartado 8.7.5).

  Grupo: a todos los trabajos pertenecientes a usuarios de un Grupo dado se les asignará una prioridad fija.

  La prioridad del usuario prevalecerá sobre la prioridad del grupo al que el usuario pertenece. Los usuarios pueden cambiar su prioridad respecto a la prioridad por defecto (especificada en el fichero de configuración apartado 8.7.5) mediante *gwsubmit -p prioridad ...*, no obstante la prioridad definida no puede ser mayor que la establecida por el administrador del Grid en el fichero de configuración. Esta posibilidad no existe desde la API DRMAA.

- **Política Urgent Job:**
  El administrador del Grid tiene como privilegio ajustar la prioridad de cualquier trabajo enviando a 20, que es superior a la prioridad máxima que maneja la política *FIXED PRIORITY* que es 19. Cuando una tarea consigue prioridad 20, ésta se transforma en un trabajo urgente. Los trabajos urgentes son despachados tan pronto como sea posible, descartando todas las demás políticas de planificación de recursos.

- **Fair-Share (SH)** Esta política permite establecer un ratio de despacho de tareas entre todos los usuarios de un dominio de planificación. Por ejemplo hay dos usuarios y la política fair-share establece que los trabajos de uno y otro se despachen en un ratio 2:5, los trabajos del primer usuario se despacharán 2 mientras que del otro usuario se despacharán 5. La ventana de observación para hacer los ratios es ajustable en el fichero de configuración. Esta política sirve por ejemplo para ajustar el ratio de trabajos que se despachan en una cierta infraestructura Grid por una u otra organización, si varias organizaciones forman esta infraestructura.

- **Waiting-Time o Tiempo de Espera:**
  El objetivo de esta política es prevenir la inanición (starvation) de tareas de baja prioridad. Los trabajos que permanecen en estado PENDING durante mucho tiempo sin ser enviados a recursos del Grid son priorizados con el tiempo de forma lineal con el tiempo de espera.

- **Deadline Time (DL):** Tiempo máximo de espera. *GridWay* incluye soporte para especificar el tiempo de deadline en las plantillas del trabajo. El planificador incrementará la prioridad de la tarea a medida que se aproxima el tiempo máximo de espera para esa tarea. Esta política no garantiza que la tarea vaya a despacharse a un recurso antes de que el tiempo máximo de espera aumente, no obstante ayuda a que ésta pueda despacharse aumentando su prioridad. El parámetro ajustable en esta política es su peso y el tiempo en el que esta tarea alcanzará la mitad de la prioridad asignada por esta política.

- **Cálculo de la prioridad de una tarea:** Se calcula de la siguiente manera:

$$Prioridad = \sum_i w_i p_i$$

donde

$$i = FP, SH, WT, DL$$

Dónde w_i son los pesos de cada política y p_i es la prioridad obtenida por cada política.

Políticas de priorización de recursos

Las políticas de priorización de recursos permiten a los administradores del Grid influenciar a los usuarios a que usen unos recursos más que otros, es decir, guiar

sus tareas a unos recursos antes que a otros recursos. En los planificadores de recursos locales (LRMS), el uso del recurso es administrado por una cola de recursos. Las tareas llegan y el planificador saca un recurso libre de la cola y se lo asigna a la tarea. Esto es lo que sucede en el LRMS *Condor*. En GridWay, el planificador construye una meta-cola (cola de colas de recursos de cada LRMS de los recursos Grid), para cada tarea de cada usuario, basado en sus requisitos obligatorios. Por defecto esta cola se ordena en función del orden de descubrimiento de los recursos. Este orden puede variar influenciado por las siguientes políticas de planificación de recursos que implementa GridWay.

- **Fixed Resource Priority Policy (RP):**
  Esta política es muy similar a la Fixed para las tareas. Consiste en priorizar los recursos en función del gestor de información que encontró esos recursos o índice de servicios en el que están registrados esos recursos. También es posible asignar prioridades fijas a los recursos individuales. Por ejemplo, se pueden especificar recursos de la red local gestionado por un driver llamado intranet y luego recursos foráneos gestionados por otro driver. La prioridad del recurso foráneo puede ser menor ya que siempre las tareas tardarán más en ejecutarse en un nodo remoto antes que en un nodo cercano al usuario. En igualdad de condiciones ganan los recursos locales, pero si éstos están colapsados por tareas se utilizan los recursos remotos que están libres. Cuando un recurso obtiene una prioridad de 0, éste pasa a ser prohibido, y por lo tanto no será usado por ninguna tarea emitida al metaplanificador del dominio de planificación local. De este modo, un administrador del Grid puede inhabilitar un recurso simplemente con añadir un 0 a su línea de configuración en el fichero de configuración del planificador (véase apartado 8.7.5).

- **Rank Policy (RA)**
  El objetivo de esta política es priorizar aquellos recursos que son más recomendables para el trabajo, desde su propio punto de vista. Un trabajo puede escoger recursos con CPUs rápidas antes que recursos con CPUs más lentas. Esta política se configura a través del atributo *RANK* en la plantilla de trabajo. (Véase 8.7.5).

- **Usage Policy (UG)**
  Esta política se basa en la utilización de los recursos del Grid. Se consultan estadísticas de las ejecuciones de las tareas. Las variables de rendimiento, tiempo medio de estancia en colas o las veces que se realiza una transferencia son consideradas a la hora de planificar un trabajo. Esta política se deriva de la suma del histórico y el estado actual. Es decir, se consultan datos del fichero histórico de tareas de los últimos días y también del último trabajo que se ha ejecutado en un recurso.

$$T = (1 - w)(T^h\_exe + T^h_{xfer} + T^h_{queue}) + w(T^c_{exe} + T^c_{xfer} + T^c_{queue})$$

  Dónde T^c son las estadísticas de ejecución el último trabajo y T^h las estadísticas de ejecución del histórico para el recurso.

  Esta política toma como parámetros configurables el número de días a tener en cuenta del histórico y el peso w que se le asigna a la última ejecución.

- **Failure Rate Policy (FR)**
  Cuando un recurso falla en la ejecución de una tarea, *GridWay* implementa una estrategia de *back-off* o ventana de contención a nivel del recurso y por usuario. Cada vez que falla un recurso para un usuario, éste queda prohibido para ese usuario por T segundos. La fórmula para calcular el tiempo es:

$$T = T_{\infty}(1 - e^{-\frac{\Delta t}{C}})$$

  Donde T_{∞} es el tiempo máximo que una tarea puede ser baneada o prohibida. Δt es el tiempo desde el último fallo. C es una constante de tiempo.

- **Cálculo de la prioridad de un recurso:**

  La lista de todos los recursos candidatos para una tarea (meta-cola) se ordena por el valor obtenido al computar todas las políticas de planificación de recursos con sus respectivos pesos en el total. La ecuación para obtener la prioridad de un recurso es:

$$P_h = \sum_i w_i p_i$$

  donde

$$i = RP, UG, FR$$

  Dónde w_i es el peso de cada política en el total y p_i es la contribución de prioridad de cada política.

8.5. Interactuar con GridWay

GridWay ofrece al usuario, de forma transparente una manera segura y confiable de usar los recursos Globus en un Grid, ya sea de la infraestructura que sea, sean gestionados por los gestores locales de recursos que soporte Globus (PBS, SGE, LSF, Condor). Para esto ofrece dos alternativas con las que trabajar con él: mediante la API DRMAA o mediante una interfaz de línea de Comandos (Command Line Interface CLI).

GridWay permite manejar los trabajos enviados como si se trataran de procesos de Unix. Cada trabajo tiene un identificador de trabajo llamado *Job identifier* o JID. Si el trabajo pertenece a un array de trabajos, entonces, el identificador del array se llama *Array id*. Para identificar un trabajo dentro de un array de trabajo se utiliza el valor *Task ID* o identificador de tarea TID. Mediante las variables anteriores, es posible identificar de forma inequívoca un trabajo en GridWay.

8.5.1. Interfaz de línea de Comandos (CLI)

La interfaz de línea de comandos es similar a cualquier CLI de gestores de recursos locales tales como Condor, PBS o LSF. Actualmente *GridWay*, en su versión 5.2.3, tiene 9 comandos que se describen en a continuación:

Comandos CLI

- **gwsubmit**: Este comando se encarga de enviar los trabajos a GridWay, es decir, encolar el trabajo en las colas de trabajo de GridWay.
- **gwps**: Muestra información de todos los trabajos que actualmente se han lanzado en GridWay, no significa que se hayan ejecutado todavía.
- **gwhistory**: Muestra un histórico de un trabajo.
- **gwhost**: Muestra información sobre los recursos del Grid que se han descubierto por el servicio de descubrimiento. En la Figura 8.9 puede verse la salida del comando *gwhost*.

```
HID PRIO   OS                ARCH    MHZ %CPU  MEM (F/T)      DISK (F/T)
N (U/F/T) LRMS HOSTNAME
0   1      Linux2.6.16-2-a x86_6 1794   192    58/1004    61570/72968
0/2/2 Fork                  aulasun15.dsi.uclm.es
1   1      Linux2.6.16-2-a x86_6 1794   199    58/1004    61570/72968
0/2/2 Fork                  aulasun16.dsi.uclm.es
2   1      Linux2.6.16-2-a x86_6 1794   198    58/1004    61570/72968
0/2/2 Fork                  aulasun17.dsi.uclm.es
3   1      Linux2.6.10-1.7 x86    2400   100   109/1519 385641/1069327
0/7/8 Condor                Myrinet.i3a.info
```

Figura 8.9: Salida del comando gwhost para el Grid I3A.

- **gwkill**: Controla la ejecución de los trabajos en el Grid. Es parar , resumir y terminar trabajos. También es posible replanificar de nuevo el trabajo.
- **gwwait**: Sincroniza con un trabajo. Esta llamada toma un identificador de tarea como parámetro. Esta llamada es bloqueante, y hace que el proceso que la invoca se bloquee a la tarea, o tareas terminen.
- **gwuser**: Monitoriza usuarios en GridWay. Se muestran los usuarios que actualmente están usando GridWay.
- **gwacct**: Muestra información de las cuentas de usuarios de los usuarios de *GridWay* y el uso de los recursos por parte de ellos.

Para más información acerca de la invocación y acciones avanzadas de los comandos consultar [38] Un cliente que necesite enviar trabajos a *GridWay* para su ejecución en el Grid simplemente necesita una cuenta de usuario en la máquina que ejecuta el demonio *GridWay*. El usuario no necesita nada más que un cliente de SSH para poder utilizar comandos *CLI* de GridWay. (Véase 8.2).

Plantilla de trabajo:

Las plantillas de trabajo con extensión *.jt* contienen toda la información del trabajo que se va a ejecutar salvo las dependencias a otros trabajos y el número de tareas del array de tareas. A continuación se detallan los campos que soporta la plantilla de trabajo.

- **NAME**: nombre del trabajo. Nombre del fichero de trabajo por defecto.
- **EXECUTABLE**: fichero ejecutable. Pueden añadirse variables de sustitución.
- **ARGUMENTS**: argumentos del fichero ejecutable.
- **ENVIRONMENT**: definición de variables de entorno que necesita el ejecutable para ejecutarse.
- **TYPE**: tipo de trabajo, si es single (simple), múltiple o MPI.
- **NP**: número de procesos en trabajos MPI.
- **INPUT_FILES**: ficheros de entrada que se trasladarán al nodo de cómputo.
- **OUTPUT_FILES**: ficheros de salida que se trasladarán desde el nodo de cómputo hasta el directorio de trabajo de GridWay.
- **STDIN_FILE**: especifica un fichero de entrada para la entrada estándar del ejecutable. Este fichero puede trasladarse o no desde la carpeta de trabajo de *GridWay* al nodo de cómputo o no, según se especifique.
- **STDOUT_FILE**: especifica un fichero de salida para la salida estándar del ejecutable.
- **STDERR_FILE**: especifica un fichero de salida para la salida de error del ejecutable. Todos los errores causados por un mal funcionamiento del programa ejecutable, y sean impresos por la salida de error serán impresos en este fichero.
- **RESTART_FILES**: estos ficheros son gestionados por el programador y son independientes de la arquitectura. Se crean capturas o instantáneas de la ejecución del ejecutable en otro lugar. El nombre del fichero se especifica aquí.
- **CHECKPOINT_INTERVAL**: especifica cada cuanto tiempo se creará una copia de los RESTART_FILES en el servidor de checkpoint que se especifica a continuación.
- **CHECKPOINT_URL**: se especifica como una dirección del protocolo GRIDFTP. Aquí se almacenarán los ficheros de restauración.
- **REQUIREMENTS**: es una expresión booleana que es evaluada por cada host disponible. Si la evaluación devuelve verdadero para un host, este es considerado para emitir el trabajo en él. El host con el mayor RANK será el escogido para ejecutar el trabajo. La expresión tiene que pertenecer al lenguaje reconocido por la gramática que se define en la Figura 8.10.
  Ejemplo de sintaxis: *REQUIREMENTS = ARCH ="i686"& CPU_MHZ > 1000*;

```
stmt::= expr?;?
expr::= VARIABLE ?=? INTEGER
      | VARIABLE ?>? INTEGER
      | VARIABLE ?<? INTEGER
      | VARIABLE ?=? STRING
      | expr ?&? expr
      | expr ?|? expr
      | ?!? expr
      | ? (? expr ?)?
```

Figura 8.10: Gramática para el lenguaje que define los *REQUIREMENTS*.

- **RANK**: es una expresión numérica que se evalúa por cada host descubierto y que su REQUERIMENTS sea verdadero. El host con el mayor RANK será el escogido para ejecutar el trabajo. La expresión tiene que pertenecer al lenguaje reconocido por la gramática que se define en la Figura 8.11.
- **RESCHEDULING_INTERVAL**: cada cuánto tiempo *GridWay* busca mejores recursos para ejecutarse en ellos. Si se especifica a 0 entonces estás acciones no se llevarán a cabo nunca.
- **RESCHEDULING_THRESHOLD**: umbral de replanificación. Si se ha encontrado un recurso mejor para el trabajo y el tiempo de CPU actualmente ejecutado en el nodo de cómputo es menor que este umbral (indicado en segundos), entonces el trabajo se migra al recurso más apropiado para las necesidades del trabajo.
- **DEADLINE**: tiempo de deadline. Si la tarea no se ha ejecutado antes de [[D:]H:]M] el trabajo falla.
- **SUSPENSION_TIMEOUT**: si el trabajo está suspendido en el host durante un intervalo superior a SUSPENSION_TIMEOUT, éste se migra a otro recurso.
- **CPULOAD_THRESHOLD**: si la CPU asignada para el trabajo es inferior a este umbral, el trabajo se migra a otro recurso.
- **MONITOR**: especifica la ruta de un monitor de trabajos alternativo.
- **RESCHEDULE_ON_FAILURE**: *yes* o *no*, especifica si se desea una planificación del trabajo en caso de fallo.
- **NUMBER_OF_RETRIES**: especifica el número máximo de reintentos de una tarea fallida.
- **WRAPPER**: especifica un wrapper alternativo.
- **PRE_WRAPPER**: especifica la ruta de un ejecutable que tiene que ejecutarse antes de que se ejecute el wrapper.
- **PRE_WRAPPER_ARGUMENTS**: especifica los argumentos para el ejecutable prewrapper.

Ejemplo de plantilla de trabajo:

```
stmt::= expr?;?
expr::= VARIABLE
      | INTEGER
      | expr ?+? expr
      | expr ?-? expr
      | expr ?*? expr
      | expr ?/? expr
      | ?-? expr
      | ? (? expr ?)?
```

Figura 8.11: Gramática para el lenguaje que define el *RANK*.

```
EXECUTABLE  = pi
ARGUMENTS   = \${TASK_ID} \${TOTAL_TASKS} 100000000
STDOUT_FILE = stdout_file.\${TASK_ID}
STDERR_FILE = stderr_file.\${TASK_ID}
```

8.5.2. API DRMAA

Uno de los aspectos más importantes de la Computación Grid es su habilidad para ejecutar tareas distribuidas que se comunican entre sí. La interfaz de programación DRMAA o *Distributed Resource Management Application API* [14] constituye una interfaz homogénea a los diferentes gestores de recursos locales (DRMS) para gestionar el envío, control de ejecución y recuperación de datos de las tareas remotas. Por lo tanto, DRMAA ayuda a científicos e ingenieros a expresar sus problemas de cómputo de forma transparente a una única interfaz DRMS. Esto provoca que las aplicaciones sean portables, ya que el lenguaje DRMS es un estándar que cada recurso tiene implementado por debajo de forma distinta en función del gestor de recursos que utilice. Hay varios proyectos que implementan la especificación DRMAA en diferentes DRMS locales como SGE, Torque o Condor. *GridWay* también tiene su implementación de la especificación del estándar DRMAA para utilizar Globus como DRMS local. Como dato importante, la API DRMAA de *GridWay* permite compilar y ejecutar sin problemas el test desarrollado por el proyecto DRMAA[13]. La API DRMAA permite interactuar con los DRMS inferiores como SGE o Condor, pero con *GridWay* únicamente se considerará el nivel del Grid, es decir, se aplicará sobre los servicios de Globus, no sobre niveles inferiores como los DRMS locales.

Como puede apreciarse en la Figura 8.12 los científicos suelen fragmentar los problemas computacionales más grandes en tareas más pequeñas que permiten tratar el problema de una manera más simple y estructurada. Estas tareas más sencillas son susceptibles de ser paralelizadas de forma total o parcial, es decir, que no dependen unas de otras. Por lo tanto pueden ejecutarse en paralelo o por el contrario es necesario una comunicación de unas y otras para al final poder proseguir cada una por su lado con los datos intercambiados. Existen varios esquemas de programación en función del nivel de paralelismo presente en ellas que se explican a continuación:

Figura 8.12: Modelo de desarrollo Grid con DRMAA.

- **Embarrassingly distributed (ED):** son aplicaciones con paralelismo de grado grueso, es decir, que pueden ser divididas en un número de tareas "gruesas"que se ejecutan cada una de forma no sincrónica e independiente entre ellas y que cuando terminan producen datos que formarán parte de la solución final y que deberán juntarse con los de las demás tareas.

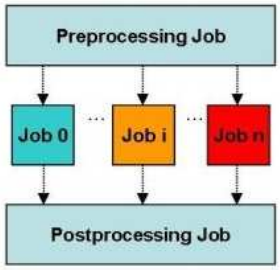


Figura 8.13: Aplicación Embarrassingly distributed (ED)

- **Master Worker:** un conjunto de tareas realiza trabajo que luego otras tareas necesitan para ejecutarse continuando el trabajo. Existen muchos familias de problemas que se basan en este esquema como son el *Helical Chain (HC), Visualization Pile (VP) y Mixed Bag (MB).*

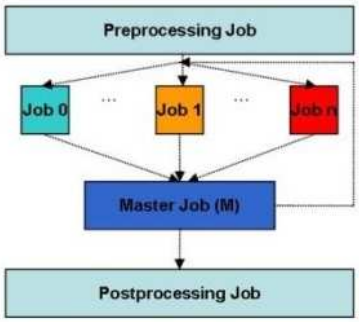


Figura 8.14: Aplicación Maestro Trabajador, Master-Worker

Primeros pasos con programación DRMAA

Antes de empezar a programar una aplicación que tener en cuenta varios pasos a realizar antes de programar cualquier aplicación con DRMAA.

1. Crear una sesión DRMAA (DRMAA Session). Esta sesión es usada para gestionar los trabajos enviados por nuestra aplicación Grid. Antes de terminar el programa hay que terminar las sesiones que han sido anteriormente iniciadas con el fin de liberar estructuras de datos internas a la API.

2. Capturar la excepción DRMAA_Exception si se programa sobre JAVA o consultar en cada llamada el código de error que devuelve (todas las llamadas devuelven un código de error que puede ser DRMAA_ERRNO_SUCCESS, que significa que todo ha ido bien u otro que indica el caso contrario). Esto puede utilizarse programando la API de C, que no soporta excepciones ya que está escrita en lenguaje C.

Para más información consultar el manual de referencia DRMAA Java [12]. Si se está desarrollando en lenguaje C, el correspondiente manual de referencia se encuentra en [11].

8.5.3. Recomendaciones

La utilización de uno u otro modo de interacción con GridWay: si CLI o DRMAA está determinada por. La API DRMAA Java o C da soporte a aplicaciones que generan trabajos de forma dinámica, generalmente interfaces gráficas que dejan a los usuarios diseñar flujos de trabajo, e incluso desplegar y controlar sus flujos de trabajo por el Grid. Un ejemplo de esto es Grid SuperScalar [24][71][72] que es un entorno de programación desarrollado por el equipo de Computación Grid del *Centro nacional de SuperComputación* (BSC), situado en la ciudad de Barcelona [5]. Orientado a extraer paralelismo de aplicaciones secuenciales y dividirlo en tareas sensibles de ser ejecutadas en un Grid (Embarrassingly distributed y Master Worker, ver figuras 8.13 y 8.14).

Este entorno de trabajo genera código para el estándar DRMAA y por lo tanto este código puede ser compilado con la biblioteca DRMAA de *GridWay* y utilizar este para desplegarse y ejecutarse en un Grid. La falta de DRMAA es que no se puede expresar dependencias entre tareas de forma explícita, es decir, no hay manera de indicarle a DRMAA que una tarea espere a otra de forma programativa, ya que simplemente el estándar DRMAA no lo soporta. Pero el lenguaje CLI de *GridWay* si que lo soporta. Es posible indicar que tareas van a esperar a que tareas o arrays de trabajos en el momento de enviar los trabajos al pool de *GridWay* (véase apartado 8.5.1), a partir de entonces *GridWay* se encarga de mantener tareas que dependen de otras en estado HOLD (ver 8.4.1) hasta que las segundas terminan. Mediante la DRMAA esto se viene haciendo mediante el método *synchronize* de la API DRMAA indicando la lista de tareas a la que se debe esperar o a través del método *getJobProgramStatus*, que devuelve el estado de un trabajo. La utilización de la primera implica la presencia de hilos de ejecución en la aplicación, si se quieren varias tareas en paralelo. La segunda implica un bucle iterativo que vaya consultando las diferentes tareas en paralelo para ver cada una si sus dependencias se han completado. Esta consulta se habrá de realizar cada cierto tiempo ya que la llamada *getJobProgramStatus* no es bloqueante como la de *synchronize* (véase Figura 8.5.3).

```
....
status = session.getJobProgramStatus (array_bt[n_mg]);
if  (status == Session.DRMAA_PS_DONE)
{
System.out.println ("Activando el trabajo mg iteración " + (n_mg*3+1));

session.control (array_mg[n_mg], Session.DRMAA_CONTROL_RELEASE);

printJobStatus (array_mg[n_mg], session);
n_mg++;
}
.....
```

Figura 8.15: Ejemplo de código de consulta DRMAA Java no bloqueante

8.6. Instalación de GridWay

GridWay puede ser descargado en código fuente de la página oficial del proyecto [38]. La versión actual es la *5.2.3*.

8.6.1. Requisitos

- Instalación de Globus ToolKit funcional, con usuarios de globus creados con certificados apropiados. Deben existir usuarios capaces de crear un proxy válido y puedan ejecutar servicios de Globus. Estos usuarios deben pertenecer al grupo *globus*.
- Instalación de Globus Development Kit para gcc32dbg. Se instalan las bibliotecas

necesarias para la compilación de *GridWay* mediante el GNU Compiler. Para ello ejecutar desde el usuario *globus*:

```
$gpt-build --nosrc gcc32dbg
```

- C compiler: Versiones probadas gcc 3.4.2, 3.4.4, 4.0.3, 4.0.3 and 4.1.2
- J2SE versiones a partir de la 1.4.2_10+
- GNU Make
- Sudo. Únicamente requerido en la versión multi-usuario
- Biblioteca Berkeley Database version 4.4.20 (necesaria para compilar el módulo de histórico)

8.6.2. Instalación multi-usuario

Es la instalación más usual y la que se recomienda para desplegar un Grid en una Organización. Si se desea más información acerca de la instalación mono-usuario consultar la Guía de instalación de *GridWay* [38].

1. Hay que escoger un usuario de *Globus* para convertirlo en administrador de *GridWay*.
2. Se crea una carpeta en /usr/local llamada *gw* que pertenecerá al usuario administrador de *GridWay* y tendrá permisos 755 (todo para usuario y lectura y ejecución para grupo y otros).
3. Se ejecuta el comando: *source $GLOBUS_LOCATION/etc/globus-dev-env.sh*

   Esto prepara el entorno de desarrollo de Globus.
4. Se ejecutará el script de configuración con los parámetros recomendados:

   ./configure –prefix=/usr/local/gw –with-doc –with-tests –with-flavor=gcc32dbg –with-db=<berkeley db location>. Consultar instalación específica de la base de datos BerkeleyDB para cada distribución.
5. Ejecutar *make*.

   Si se ha producido un error al compilar algunos ficheros relacionados con jsdl hay que eliminar esta opción de la configuración para que no se compile el soporte para JSDL con *–disable-jsdl*
6. Ejecutar la instalación:

   make install

   Si todo ha ido bien se habrán generado en la carpeta */usr/local/gw* toda la instalación de GridWay.

8.7. Configuración de GridWay

Para la configuración de *GridWay* se va a seguir el ejemplo de configuración para un entorno multiusuario. Es la configuración más frecuente en un Grid de producción y no de pruebas.

8.7.1. Creación de usuarios

Una vez instalado *GridWay* hay que configurar los usuarios que tendrán acceso al demonio y podrán usar los controladores de acceso al medio para acceder a los recursos del Grid. Hay que editar el fichero */etc/sudoers* como *root* para que los usuarios de *GridWay*, que serán creados dentro del grupo del usuario *usuario*, administrador de gridway. Hay que tener en cuenta que el demonio se ejecuta bajo el usuario administrador de gridway, pero los MADs son ejecutados bajo los usuarios de *GridWay* que lanzan y monitorizan trabajos. Todo esto se hace en la máquina donde se ejecuta el demonio del metaplanificador, no en los demás recursos del grid.

Hay que añadir las líneas al fichero *sudoers* especificadas en la Figura 8.16:

```
usuario ALL= (GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_em_mad_ws *
usuario ALL= (GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_tm_mad_ftp *
usuario ALL= (GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_em_mad_prews *
usuario ALL= (GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_im_mad_mds4_thr *
usuario ALL= (GW_USERS)    NOPASSWD: /usr/local/gw/bin/gw_im_mad_mds4 *
Defaults>GW_USERS env_keep="GW_LOCATION_GLOBUS_LOCATION"
```

Figura 8.16: Fichero /etc/sudoers.

8.7.2. Creación de certificados para esos usuarios

Los usuarios que se creen especificamente para *GridWay* tienen que crearse perteneciendo al grupo *globus* y al grupo del usuario *usuario*, administrador de *GridWay gwgroup*. Para utilizar los servicios de globus hay que crear certificados para la GSI (Grid Secure Infraestructure). Desde sus cuentas de usuario:

```
$grid-cert-request
```

Se solicitará una contraseña que posteriormente se utilizará para arrancar el proxy del usuario. Se creará una clave privada y un fichero de solicitud de certificado *user-*

cert_request.pem en el directorio $HOME/.globus. Este fichero debe ser firmado por la entidad certificadora para que cobre validez. La firma se realiza desde el usuario *globus* por el administrador de globus:

```
$grid-ca-sign -in usercert_request.pem -out usercert.pem
```

Este certificado debe de ser copiado en el $HOME/.globus/ del usuario recién creado. El usuario puede probar su proxy ejecutando *grid-proxy-init*.

Configuración de fichero de autorización grid-mapfile

En la máquina en la que se ejecuta el demonio hay que añadir una entrada al fichero de autorización *grid-mapfile*. Con ello se autoriza al nuevo usuario para utilizar los servicios Globus en esa máquina.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/ \
OU=dyndns.info/CN=usuario" usuario
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=nuevousuario" nuevousuario
```

IMPORTANTE: hay que añadir una entrada en los ficheros grid-mapfile de todos los recursos accesibles en el Grid de la organización que vaya a usar este usuario, es decir, los se vayan a descubrir con GridWay. No hace falta crear una nueva cuenta de usuario, sino utilizar una ya creada para un usuario globus genérico. Por ejemplo se utilizará el usuario *usuario*, presente en todos los recursos y con los certificados ya instalados. La entrada en el fichero *grid-mapfile* quedaría:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/ \
OU=dyndns.info/CN=usuario" usuario
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info \
/OU=dyndns.info/CN=nuevousuario" usuario
```

De este modo el usuario *nuevousuario* se mapeará bajo el usuario *usuario* en todos los recursos del Grid. Esto resulta bastante cómodo ya que sino habría que crear una cuenta nueva de usuario y copiar certificados en todos los recursos del grid. Esto puede hacerse de este modo ya que los recursos del Grid no lanzan trabajos, sino que los reciben. Es importante tener certificados si vas a enviar trabajos, pero para recibir trabajos únicamente hace falta una cuenta de usuario en la que se mapeen los demás usuarios. OJO, estos usuarios deben tener certificados en la máquina donde está instalado el demonio, que es donde se lanzaran trabajos a través de su proxy de usuario válido.

8.7.3. Configurando los MADs

Los MADs o controladores de acceso al medio se encargan de utilizar los servicios de Globus presentes en los recursos del Grid a través de los servicios Globus presentes en la máquina local. Hay que especificar qué MADs de *GridWay* se cargaran

ya que existen diferentes tipos dependiendo del tipo de servicios globus a los que se quiere acceder.

Hay tres tipos:

- Gestores de ejecución para acceder a servicios pre-WS GRAM y WS GRAM.
- Gestores de información para acceder a servicios MDS2 y MDS4 (gt 4).
- Gestores de transferencia para acceder a servicios GridFTP.

Gestores de Ejecución

Los controladores de acceso al medio para los servicios de ejecución son responsables de la gestión de la ejecución de las tareas a bajo nivel, a nivel de los servicios de Globus. La distribución *GridWay* de la versión 5.2.3 incluye los siguientes MADs de ejecución:

- Pre-WS GRAM (Globus Toolkit 2.4 en adelante).
- WS GRAM (Globus Toolkit 4.0)

El uso de estos MADs requiere el uso de un proxy válido. Configuración: En el fichero *$GW_LOCATION/etc/gwd.conf* hay que editar:

```
EM_MAD = <mad_name>:<path_to_mad>:<args>:<rsl|rsl_nsh|rsl2>
```

Donde:

- mad_name: es la etiqueta para referenciar al driver de ejecución.
- path_to_mad: es el nombre del fichero ejecutable del driver. Debe estar emplazado en: *$GW_LOCATION/bin/*.
- args: son los parámetros pasados al driver cuando este se ejecute.

Ejemplo:

```
EM_MAD = ws:gw_em_mad_ws::rsl2 # ws GRAM
```

Gestores de Información y Descubrimiento

Son los responsables del descubrimiento de recursos y monitorización de trabajos. En *GridWay* se incluyen los siguientes gestores de información:

- Información Estática de Recursos.
- MDS2 con esquema MDS soportado por Globus ToolKit 2.4.

- MDS2 con esquema GLUE.
- MDS4 soportado por Globus ToolKit 4.0 (Actual).

Hay tres modos de operación del Driver:

- **Static Discovery and Monitoring (SS mode):** en este modo, los host son definidos estáticamente leyéndolos de un fichero lista. También se leen los atributos de los hosts. **Ejemplo:**

```
IM_MAD = static:gw_im_mad_static:-l examples/im/host.static \
::gridftp:ws
```

Para más información consultar Guía de Instalación y configuración de *GridWay* [38]

- **Static Discovery and Dynamic Monitoring (SD mode):** los hosts son descubiertos leyendo como en el caso anterior un fichero. Sin embargo, la información de cada host es obtenida consultando su servicio Globus de información (GRIS en MDS2 o DefaultIndexService en MDS4).

Ejemplo:

```
IM_MAD = glue:gw_im_mad_mds2_glue:-l examples/im/host.list:: \
gridftp:prews
```

- **Dynamic Discovery and Monitoring (DD mode):** en este modo, los recursos son descubiertos y monitorizados directamente accediendo al servicio de información Grid. Este modo es recomendable cuando los recursos del Grid son muy cambiantes, es decir, pueden aparecer y desparecer recursos en cuestión de horas o días. **Ejemplo:**

```
IM_MAD = mds4:gw_im_mad_mds4:-s hydrus.dacya.ucm.es::gridftp:ws
```

Cosas a tener en cuenta:

- Pueden usarse simultáneamente hasta diez gestores de información.
- Pueden mezclarse varios tipos de gestores: MDS2, MDS4.
- Pueden mezclarse varios modos de descubrimiento en el mismo gestor de información: SS, SD y DD.

Gestores de Transferencia GridFTP

El driver de Transferencia GridFTP es el responsable del traslado de ficheros necesarios para la ejecución de las tareas, creación de directorios remotos y limpiar después de obtener los resultados. En la distribución de *GridWay* 5.2.3 se incluyen:

- Servidor GridFTP: versión 1.1.2 o superior.
- Dummy Transfer Driver: para ser usado con clusters sin NFS.

El uso de este driver requiere un proxy válido, al igual que los drivers de ejecución.

Ejemplo:

```
TM_MAD = <mad_name>:<path_to_mad>:[arguments]>
```

8.7.4. Configurando *GridWay* Core

El demonio *GridWay* (*gwd*) tiene su configuración definida en *./etc/gwd.conf*. A continuación se detallan los campos de configuración:

- **GWD_PORT**: puerto TCP/IP donde el demonio escuchará a las peticiones de usuario. Por defecto es el 6725. Si este puerto está usado, *gwd* irá buscando los siguientes puertos hasta que encuentre uno que esté libre. El puerto actualmente usado por el demonio se encuentra en *$GW_LOCATION/var/gwd.port*.
- **MAX_NUMBER_OF_CLIENTS:** número máximo de clientes simultáneos.
- **NUMBER_OF_JOBS**: el número máximo de trabajos que será gestionado por *GridWay* en un momento dado.
- **NUMBER_OF_ARRAYS:** el número máximo de arrays de tareas que serán gestionados por *GridWay* en un momento dado.
- **NUMBER_OF_HOSTS:** el número máximo de recursos que *GridWay* gestionará en un momento dado.
- **NUMBER_OF_USERS:** el número máximo de clientes que se soportarán a la vez en un determinado momento.
- **SCHEDULING_INTERVAL:** periodo en segundos entre dos decisiones de planificación.
- **DISCOVERY_INTERVAL:** periodo en segundos entre dos búsquedas de nuevos recursos.
- **MONITORING_INTERVAL:** periodo en segundos entre dos consultas en el estado de los recursos actualmente descubiertos.
- **POLL_INTERVAL:** periodo en segundos entre dos consultas de estado de un trabajo en un recurso.
- **MAX_ACTIVE_IM_QUERIES:** máximas peticiones de información a recursos que se realizarán en un momento dado.

- **DM_SCHED:** nombre de fichero del ejecutable de un planificador que se usará para planificar las tareas. Si los usuarios mejoran el planificador actual pueden cargar el nuevo planificador editando este campo.

8.7.5. Configurando el planificador

El planificador tiene su configuración un fichero llamado *sched.conf* situado en el directorio de *GridWay*. Se explicarán las variables editables de cada política de planificación englobadas dentro de las políticas de prioridad de recursos y las de prioridad de trabajos.

Prioridad de trabajos

En la sección del fichero dedicada a la prioridad de trabajos se encuentran los campos configurables de las políticas de priorización de tareas (véase apartado 8.4.3).

- **DISPATCH_CHUNK:** el número máximo de tareas que se despacharán en cada decisión de planificación.
- **MAX_RUNNING_USER:** el número máximo de trabajos simultáneos de un usuario dado.
- **Sección política FIXED:**

  FP_WEIGHT: peso de la política. Se permiten números reales.

  FP_USER[<**username**>]**:** prioridad que se le asigna a los trabajos lanzados por el usuario.

  FP_GROUP[<**groupname**>]**:** prioridad que se le asigna a los trabajos lanzados por usuarios del grupo.
- **Sección política SHARE:**

  SH_WEIGHT: peso de la política. Se permiten números reales.

  SH_USER[<**username**>]**:** número de trabajos del usuario para realizar el ratio.

  SH_WINDOW_DEPTH: número de intervalos almacenados de despache de trabajos para cada usuario con motivo de calcular el ratio de envío de trabajos al Grid entre varios usuarios.

  SH_WINDOW_SIZE: el tamaño de cada intervalo en días. Se permiten números reales.
- Sección política DEADLINE:

  DL_WEIGHT: peso de la política. Se permiten números reales.

  DL_HALF: número de días antes de la fecha de deadline cuando el trabajo alcance la mitad de su prioridad máxima.

Prioridad de recursos

En la sección del fichero dedicada a la prioridad de recursos Grid se encuentran los campos configurables de las políticas de priorización de recursos (véase apartado 8.4.3).

- **MAX_RUNNING_RESOURCE:** máximo número de tareas que se enviarán a un recurso determinado, por muchos procesadores que este tenga.
- **Sección política FIXED:**

  RP_WEIGHT: peso de la política. Se permiten números reales.

  RP_HOST[<host>]: prioridad de los slots (capacidad para aceptar tareas) de un recurso.

  RP_IM[<etiqueta im>]: prioridad de todos los recursos descubiertos por este gestor de información especificado por su etiqueta.
- Sección política Usage:

  UG_WEIGHT: peso de la política. Se permiten números reales.

  UG_HISTORY_WINDOW: número de días de histórico para realizar el cómputo de la política.

  UG_HISTORY_RATIO: peso para estimar el tiempo estimado de cómputo en un recurso (véase apartado 8.4.3).
- **Sección política RANK:**

  RA_WEIGHT: peso de la política. Se permiten números reales.
- Sección política Failure Rate:

  FR_MAX_BANNED: tiempo máximo que un recurso Grid es prohibido o bloqueado.

  FR_BANNED_C: constante exponencial utilizada para calcular el tiempo de baneo del recurso (véase apartado 8.4.3).

8.8. Arrancando el demonio

Se va a partir de una configuración multi-usuario de demonio, puesto que se va a trabajar con un Grid con mútiples usuarios y acceso centralizado a los recursos, es decir, una única instancia de *GridWay* ejecutándose en el servidor que dará a los usuarios acceso al Grid. Los usuarios se conectan de forma remota al servidor para lanzar trabajos en el Grid. Para arrancar el demonio es necesario realizarlo desde el usuario administrador de *GridWay* y no desde **superusuario**. Este usuario es necesario que disponga de un proxy de Globus válido para poder realizar pruebas, aunque no es necesario porque luego los usuarios utilizarán sus propios proxies para utilizar los servicios Globus del Grid de la propia VO, o de otras otras VO. Para arrancarlo:

```
$gwd –m
```

8.9. Pruebas básicas

Para las pruebas básicas se necesita que el usuario administrador de *GridWay* tenga creado un proxy válido si este usuario es de globus o en su defecto un usuario de *GridWay* que tenga un proxy válido.

```
$grid-proxy-init -valid 60:00 #Crea un proxy con 60 horas de validez.
```

8.10. Conexión de un usuario a GridWay

Para poder enviar trabajos al Grid a través del *GridWay* configurado con los usuarios que se han creado dentro del grupo de usuarios admitidos para GridWay. En la máquina del usuario no es necesario tener instalado ni *GridWay* ni Globus. Únicamente es necesario disponer de un cliente SSH y conectividad al puerto SSH de la máquina con GridWay. Otra opción sería disponer de un sistema de ficheros NFS en nuestra máquina y que las bibliotecas DRMAA y cuentas de usuario estén çompartidas.[en]tre varias máquinas dentro del entorno NFS. Las cuentas de usuario y carpetas HOME deberían estar compartidas entre los hosts de los usuarios y el servidor GridWay. Al igual que el directorio de *Globus*, que debería tener derechos de lectura para grupo.

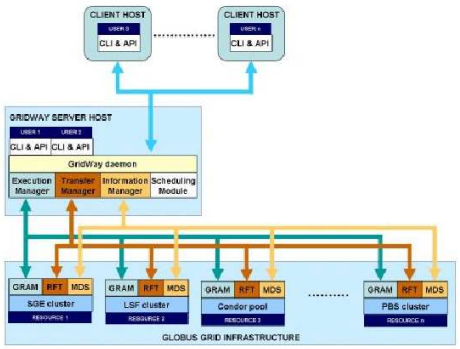


Figura 8.17: Clientes conectándose a *GridWay* por **SSH**

Una vez el usuario se ha conectado a la máquina con *GridWay* ya puede interactuar con él mediante el lenguaje CLI. Además es posible ejecutar aplicaciones escritas con la API DRMAA compiladas de forma local en la máquina de *GridWay* que será la que tiene las bibliotecas DRMAA.

8.10.1. Ejecución de un trabajo básico

Con la instalación de *GridWay* se instalan ejemplos de programas script que utilizan la interfaz CLI de DRMAA y también sus respectivas API C y Java. Para el ejemplo básico se va a usar el lenguaje CLI, y de los ejemplos existentes para ese lenguaje un ejemplo en particular que es el *WORKFLOW*. Este ejemplo consiste en cuatro tareas con dependencias entre ellas. Aquí se aprecia la potencia del lenguaje CLI de *GridWay* ya que mediante varias ordenes sencillas de script es posible generar un workflow complejo. Esto mediante las APIs C y Java puede realizarse de forma programativa mediante la espera activa de tareas con consulta de estado de los trabajos, no de manera desatendida. Se hablará de esto más tarde en el capítulo 10.

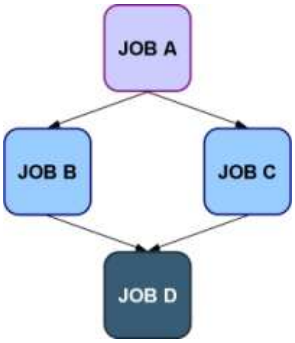


Figura 8.18: Workflow de ejemplo.

El ejemplo trae un script CLI que se detalla en la Figura 8.19.

```
#!/bin/sh

A_ID=`gwsubmit -v -t A.jt | cut -f2 -d':' | cut -f2 -d' '`
B_ID=`gwsubmit -v -t B.jt -d "$A_ID" | cut -f2 -d':' | cut -f2 -d' '`
C_ID=`gwsubmit -v -t C.jt -d "$A_ID" | cut -f2 -d':' | cut -f2 -d' '`
D_ID=`gwsubmit -v -t D.jt -d "$B_ID $C_ID"| cut -f2 -d':' | cut -f2 -d' '`

gwwait $D_ID

echo "Random number `cat out.A`"
echo "Workflow computation `cat out.workflow`"
```

Figura 8.19: Script Workflow

El script genera las cuatro tareas a partir de sus plantillas de trabajo, y luego les asigna las dependencias entre ellas. Espera a su ejecución y muestra el resultado por pantalla. La salida del trabajo se muestra en la Figura 8.20.

```
USER            JID DM   EM    START    END      EXEC     XFER     EXIT NAME HOST
usuario         0   pend ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   A.jt  --
usuario         1   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   B.jt  --
usuario         2   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   C.jt  --
usuario         3   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   D.jt
.....
USER            JID DM   EM    START    END      EXEC     XFER     EXIT NAME HOST
usuario         0   prol ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   A.jt
aulasun15.dsi.uclm.es/Fo
usuario         1   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   B.jt  --
usuario         2   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   C.jt  --
usuario         3   hold ----- 21:07:17 --:--:-- 0:00:00  0:00:00  --   D.jt  --
......
USER            JID DM   EM    START    END      EXEC     XFER     EXIT NAME HOST
usuario         0   wrap actv 21:08:26 --:--:-- 0:00:03  0:00:01  --   A.jt
aulasun15.dsi.uclm.es/Fo
usuario         1   hold ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   B.jt  --
usuario         2   hold ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   C.jt  --
usuario         3   hold ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   D.jt  --
.......
USER            JID DM   EM    START    END      EXEC     XFER     EXIT NAME HOST
usuario         0   done ----- 21:08:26 21:08:52 0:00:03  0:00:03  0    A.jt
aulasun15.dsi.uclm.es/Fo
usuario         1   pend ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   B.jt   --
usuario         2   pend ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   C.jt   --
usuario         3   hold ----- 21:08:26 --:--:-- 0:00:00  0:00:00  --   D.jt   --
.......
Random number 8364
Workflow computation 16730
```

Figura 8.20: Salida de la ejecución WorkFlow + pasos de ejecución.

8.11. Implantando *GridWay* en Laboratorio RAAP (I3A)

Hasta este momento se ha visto la instalación y configuración de GridWay, en un entorno multi-usuario. Se han explicado las formas de interacción para con éste, así como cual usar dependiendo de lo que se quiera hacer y a que nivel. Llegados a este punto, en el que se conocen todas las opciones que *GridWay* ofrece, es hora de implantarlo en el Grid en producción desplegado en el Laboratorio RAAP [49] con servicios Globus (véase capítulo 2). El Grid consta de un cluster Myrinet con Condor provisto de 12 procesadores, que se ha añadido en el capítulo 5; 4 estaciones de trabajo SUN con procesadores Opteron x86_64 con un total de 2 núcleos; y una máquina con dos núcleos Pentium IV. En total un Grid con 18 CPU en el que se quiere mejorar el aprovechamiento de los recursos y controlar el acceso a estos por parte de los usuarios del Grid. Para eso se ha escogido *GridWay* como metaplanificador.

8.11.1. Elección de host

Hay que escoger una máquina en la que se ejecutará el demonio GridWay, la máquina debería ser accesible desde Internet al puerto SSH para que los usuarios

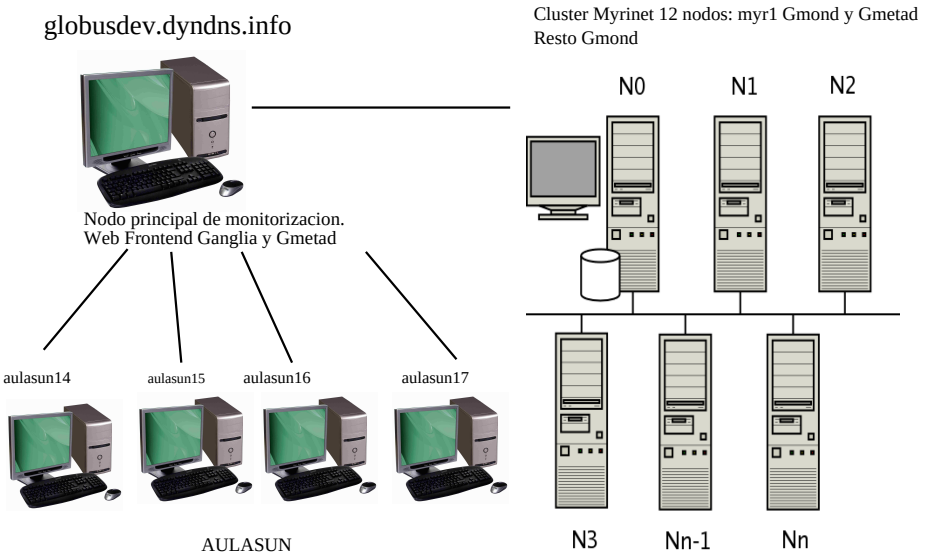


Figura 8.21: Recursos del RAAP integrados al Grid.

puedan conectarse y mandar trabajos al Grid. Para lograr esto en el laboratorio RAAP sería posible realizar esto de dos maneras:

- Permitir tráfico al puerto SSH de la máquina *GWHOST* añadiendo reglas al cortafuegos del encaminador o encaminadores que impidan este tráfico.
- Añadir esta máquina al DMZ de la red (espacio de direcciones a las que el encaminador permitirá acceso total desde fuera).

La segunda opción es la más elegante puesto que no habrá que añadir reglas a los cortafuegos de los encaminadores. Simplemente hay que tocar cableado para que topológicamente el host *GWHOST* pertenezca de esta red. Visto lo visto, de los recursos disponibles, el más propicio a ser el portador del *GridWay* será *globusdev.dyndns.info*. Las máquinas del AulaSUN tienen sus puertos cerrados al exterior y además están situadas en un centro (edificio) distinto al del Cluster y la máquina *globusdev.dyndns.info*, por lo tanto no son la opción más indicada. El cluster de cálculo Myrinet tiene ya el cableado hecho y no resulta atractivo ni recomendable cambiarlo a DMZ. Principalmente por motivos de seguridad. Por lo tanto *globusdev.dyndns.info* es el candidato elegido por mayoría.

8.11.2. Instalación de GridWay

En la máquina globusdev.dyndns.info que puede verse en la Figura 8.21 están instalados la base de datos RFT utilizada para las transferencias de ficheros (file staging) y el índice de servicios dentro del cual se registran los recursos del Grid local. Esta máquina dispone de los siguientes usuarios:

- globus: usuario sobre el que se ejecutarán los servicios.

- usuario: usuario de globus que podrá utilizar los servicios de Globus.

Es posible escoger un usuario de globus ya existente como *usuario* o por el contrario crear uno nuevo para albergar GridWay. En la práctica se escogió *usuario* como usuario de GridWay.

En el usuario *usuario* se realizará la instalación de *GridWay* siguiendo el apartado 8.6.2.

Creación de usuarios de GridWay

Se ha escogido utilizar el usuario *usuario* para albergar el demonio GridWay, pero los usuarios del Grid no ejecutarán los trabajos con ese usuario, necesitarán el suyo propio. Para realizar pruebas de instalación se creará un usuario de pruebas llamado *rayban* perteneciente al grupo *usuario* de usuarios de *GridWay*.

```
#adduser rayban usuario
...
#adduser rayban globus
....
```

Los usuarios deben pertenecer al grupo globus o como se haya especificado el grupo de usuarios que vayan a usar globus. Todo esto se con efectos del fichero */etc/sudoers*, que permite a usuarios distintos a globus ejecutar los servicios de globus (véase apartado 6.2.2 del capítulo 6).

Se crearán los certificados para el usuario siguiendo el apartado 8.7.2. Las entradas de los ficheros */etc/grid-mapfile* en los distintos recursos del Grid quedarán:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/ \
OU=dyndns.info/CN=usuario" usuario
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/ \
OU=dyndns.info/CN=rayban" usuario
```

El usuario *usuario* partiendo de una instalación de Globus funcional en los recursos del Grid, debe estar presente en todos los recursos con sus certificados, para que los usuarios que se creen para *GridWay* puedan mapearse en este usuario como se explica en el apartado 8.7.2.

8.11.3. Configuración de GridWay Core

En este paso hay que configurar los controladores de acceso al medio del *GridWay Core*.

MAD de Ejecución

Como se explica en el apartado 8.7.3 este controlador se encarga de acceder a los servicios de ejecución de Globus. Como se está trabajando dentro de un Grid con Globus 4.X que funciona mediante servicios web, es necesario seleccionar el driver preparado para este tipo de accesos. En el fichero *./etc/sched.conf* la configuración del MAD GRAM quedaría así:

```
EM_MAD = ws:gw_em_mad_ws::rsl2
```

MAD de Información

Hay varios aspectos a considerar antes de configurar este controlador de acceso. Si se desea obtener información de los recursos de forma dinámica como se explicará a continuación, es necesario que dispongan de una herramienta de monitorización ganglia instalada que proporcione información a su interfaz de monitorización y descubrimiento (MDS) para que publique la información recopilada por esta herramienta (véase 7

- ¿Con cuánta frecuencia cambiará el Grid?
- ¿Deseo que todas los recursos registrados en el Índice de servicios se utilicen por los usuarios de GridWay?.

En un Grid que cuyos recursos cambian mucho, es decir, aparecen nuevos recursos frecuentemente, a la vez que desaparecen otros, sería necesaria una configuración dinámica del driver. Éste obtendría la información directamente del Índice de Servicios de la VO local en la que se registrarían los nuevos recursos una vez se añadieran. También se eliminarían una vez que estos desaparecieran. En el Grid del RAAP esta no es la situación más usual, ya que los recursos suelen añadirse una vez y permanecen asignados a éste de forma indefinida. En este caso sería necesaria una configuración estática del driver GridWay, escogiendo la configuración: *Static Discovery and Dynamic Monitoring* (véase apartado 8.7.3). El driver toma como entrada una lista de recursos que será editada por el administrador del Grid para definir los recursos que formarán parte del repertorio de recursos que el metaplanificador empleará en sus decisiones de planificado de trabajos. La entrada del driver IM en el fichero *./etc/gwd.conf* queda:

```
IM_MAD = grid3a:gw_im_mad_mds4_thr:-l lista.host:gridftp:ws
```

Cabe destacar que el nombre del conjunto de recursos que se encuentran en la lista proporcionada al driver de información se nombra bajo la etiqueta *grid3a*. Esta etiqueta se utiliza por el planificador para clasificar los recursos descubiertos por cada driver de información y aplicar las consiguientes políticas de planificación sobre estos. El fichero de recursos debe estar situado en el directorio de instalación de *GridWay*. Contendría las siguiente entradas:

```
aulasun14.dsi.uclm.es
aulasun15.dsi.uclm.es
aulasun16.dsi.uclm.es
aulasun17.dsi.uclm.es
Myrinet.i3a.info
```

Cabe destacar que no existe una entrada para la máquina *globusdev.dyndns.info*. Hay una explicación razonable y es que la máquina que albergará *GridWay* no debe ejecutar trabajos del Grid, puesto que corre el riesgo de consumir los recursos del sistema y hacer que *GridWay* se ejecute mal. Además esta máquina ejecuta más cosas además de *GridWay* como es la base de datos RFT y el índice de servicios. Si por cualquier motivo se escogiera introducir la máquina *globusdev.dyndns.info* habría que limitar de algún modo la utilización de esta. Un modo posible de hacerlo sería en la configuración del planificador, sección de priorización de recursos (véase apartado 8.4.3), añadiendo la prioridad más baja al recurso globusdev.

```
...
RP_HOST[globusdev.dyndns.info] = 1
RP_HOST[DEFAULT] = 10
...
```

Si en un momento dado se decide eliminar el recurso globusdev, es decir, que no se use por GridWay, basta con poner a 0 su prioridad. Automáticamente el recurso quedaría fuera de juego por GridWay.

MAD de Transferencia

Como se explica en el apartado 8.7.3 este controlador se encarga de realizar las transferencias de ficheros y preparación de directorios para los trabajos en los recursos remotos. En el Grid del RAAP los servicios que se utilizan son los de Globus 4, por lo tanto son servicios web. Tampoco hay un sistema de ficheros compartido entre los distintos recursos por lo tanto no se usa el *Dummy driver*. La configuración del driver de transferencia quedaría:

```
TM_MAD = gridftp:gw_tm_mad_ftp:
```

8.11.4. Configuración de políticas de planificación

Consultando la sección 8.4.3 y 8.4.3 se pueden analizar las diferentes políticas de planificación para trabajos y recursos que aplica *GridWay* 5.2.3. Teniendo en cuenta la configuración actual del Grid del RAAP: compuesto por 16 CPUs (hemos eliminado globusdev) y el tipo de trabajos que se lanzarán en él: *Embarrassingly distributed*, la mayor parte simulaciones que no utilizan paralelismo pero sí una gran cantidad de recursos de memoria, disco y CPU; será necesario la creación de usuarios con la misma prioridad. Los usuarios preferirán ejecutar trabajos en máquinas con mucha memoria y mucha capacidad de CPU (para que no lleve días), sobre la disponibilidad de un

cluster con soporte para MPI, con más procesadores pero más lentos, y con menos memoria principal y menos disco. Por lo tanto el cluster como recurso Grid deberá tener menos prioridad que máquinas más rápidas y con más memoria principal como son las máquinas del AulaSUN añadidas al Grid:

```
aulasun14.dsi.uclm.es
aulasun15.dsi.uclm.es
aulasun16.dsi.uclm.es
aulasun17.dsi.uclm.es
```

La configuración del planificador para los trabajos de los usuarios será para *FIXED PRIORITY*:

```
...
FP_WEIGHT          = 1
FP_USER[gwuser] = 10
FP_USER[DEFAULT] = 15
...
```

La política *SHARED* se dejará de momento por defecto. La política *WAITING TIME* se habilitará poniendo su peso en 1 (véase apartado 8.4.3). La política de *DEADLINE* debe dejarse habilitada para que no se produzca inanición de ningún trabajo al ser lanzado y no ejecutado antes de que expire.

En políticas de recursos cabe destacar la política *FIXED* ya que esta permitirá asignar prioridad a los recursos del Grid. Puede reducirse la prioridad de ciertos recursos respecto a otros. Por ejemplo el cluster Myrinet puede tener menos prioridad que las máquinas del AulaSUN ya que estas son más rápidas, aunque si estas están recibiendo muchos trabajos pueden enviarse al cluster de cálculo.

```
...
     RP_IM[grid3a] = 10
        #Se reduce la prioridad de este recurso
        # frente a los demás del grid3a
     RP_HOST[Myrinet.i3a.info] = 05
...
```

Por defecto los recursos del *grid3a* tendrán prioridad 10 salvo por el cluster que tendrá prioridad 5. Como de momento se están realizando pruebas con el Grid la política *USAGE* puede deshabilitarse debido a que las estadísticas no van a ser fiables. Lo mismo ocurre con la política *Failure Rate* que penaliza temporalmente a los recursos que han dado fallos de ejecución. (véase 8.4.3). La política *RANK* puede dejarse habilitada ya que es propio de los trabajos preferir ejecutarse en unos recursos sobre en otros.

```
...
UG_WEIGHT       = 0
...
FR_MAX_BANNED = 0
...
```

8.11.5. Probando *GridWay* en RAAP

Llegados a este punto hay una instalación de *GridWay* operacional implantada en el laboratorio RAAP utilizando los servicios Globus de los recursos disponibles. Los usuarios pueden conectarse y mandar trabajos. Si son necesarios nuevos usuarios simplemente hay que realizar el paso descrito en el apartado 8.7.1 en el que se describe como añadir un nuevo usuario al grupo de usuarios del administrador de GridWay. Las pruebas pueden proceder según el apartado 8.9. En la Figura 8.11.5 pueden observarse los distintos recursos del Grid *grid3a* y su prioridad.

```
HID PRIO OS              ARCH   MHZ %CPU MEM (F/T)      DISK (F/T)          N (U/F/T) LRMS    HOSTNAME
0   5    Linux2.6.10-1.7 x86    2400 100  649/1519     385830/1069327      0/10/10 Condor Myrinet.i3a.info
1   10   Linux2.6.16-2-a x86_6  1794 198   75/1004       61599/72968       0/2/2 Fork   aulasun15.dsi.uclm.es
3   10   Linux2.6.16-2-a x86_6  1794 200   75/1004       61599/72968       0/2/2 Fork   aulasun16.dsi.uclm.es
4   10   Linux2.6.16-2-a x86_6  1794 200   75/1004       61599/72968       0/2/2 Fork   aulasun17.dsi.uclm.es
```

Figura 8.22: Salida del comando gwhost en el *GridWay* del laboratorio RAAP.

8.12. Integrando un recurso Grid foráneo en el Grid del RAAP con GridWay

Hasta aquí únicamente se contaba con los recursos de las máquinas del laboratorio RAAP y del AulaSUN. Se dará un paso adelante integrando al Grid del laboratorio, a *GridWay* un recurso foráneo de otra VO. En este caso la VO de la Universidad de Murcia [65]. Los primeros pasos para hacer esto posible, son permitir la comunicación entre los servicios Globus de nuestra VO y la VO de la Universidad de Murcia, es decir, intercambiar certificados y usuarios entre las VO. Sería recomendable consultar el capítulo 6 en el que se detalla todo esto. El intercambio de usuarios y certificados no se lleva a cabo en todos los recursos de ambos Grids, sino en las máquinas que interaccionan entre ambos. En el Grid de la Universidad habrá que instalar los certificados de la entidad CA local, y además se deberá modificar su fichero *grid-mapfile* por cada usuario de *GridWay* del Grid del I3A. Los usuarios deben mapearse en un usuario creado específicamente para los usuarios del Grid del I3A. En este caso, un usuario llamado *grid3a*. Este usuario es un usuario globus y debe ser creado con todos los certificados pero de la Universidad de Murcia. Una vez establecida la confianza mutua hay que configurar *GridWay* para que pueda utilizar este nuevo recurso y además asignarle la prioridad que se merece.

8.12.1. Configurando *GridWay* para utilizar los recursos de la nueva VO

En la configuración de *GridWay* fichero *./etc/gwd.conf* sería recomendable separar los recursos de una y otra en distintos drivers, es decir, que un driver gestione

los recursos locales y el otro driver se encargue de los recursos que proporciona la UM. En el apartado 8.7.3 donde se detallan los drivers que proporciona GridWay, se explica que éste puede soportar hasta 10 drivers distintos. La configuración de los drivers de información quedaría:

```
...
IM_MAD = grid3a:gw_im_mad_mds4_thr:-l lista.host:gridftp:ws
IM_MAD = gridUM:gw_im_mad_mds4_thr:.-s inf-205-154.um.es:gridftp:ws
...
```

El nuevo recurso del que obtener información es: *inf-205-154.um.es* que contendrá un índice de servicios sobre el que se registrarán los recursos locales a aquella VO. Al añadir otro driver de información es recomendable pero no necesario añadir información de planificación a las políticas de planificación de recursos (véase 8.4.3). Recomendable puesto que no obligatorio, ya que si no se añade nada se le asignaría al nuevo recurso la prioridad por defecto. Debido a los costes en el envío de trabajos al nuevo recurso causados por la latencia y el bajo ancho de banda presente en las redes WAN públicas (que se utilizarán entre universidades y otros centros) se hace recomendable reducir la prioridad a los recursos foráneos al Grid de la VO para evitar enviar trabajos a éstos teniendo recursos libres en el Grid local. Los tiempos de transferencia en local y en remoto pueden diferir en órdenes de magnitud por lo tanto hay que reducir la prioridad del nuevo recurso hasta la más baja de entre todos los recursos presentes. Si por cualquier motivo se necesitan más recursos que los que hay disponibles en el Grid local a la VO se utilizarán los recursos del Grid foráneo, en este caso de la Universidad de Murcia. La política *FIXED* para los recursos quedaría:

```
...
RP_IM[gridUM]   = 02
RP_IM[grid3a] = 10
RP_HOST[Myrinet.i3a.info] = 05
...
```

El recurso *gridUM*, donde gridUM es la etiqueta que describe al driver que obtiene información de *inf-205-154.um.es*, tiene la prioridad más baja del Grid. Si el Grid se satura, se enviarán trabajos a los recursos detectados por el driver *gridUM*.

8.13. Consideraciones

Todo lo anterior puede aplicarse a cualquier VO añadida posteriormente a nuestro Grid con GridWay. Del mismo modo que pueden integrarse recursos foráneos a nuestro Grid, también pueden incluirse nuestros recursos locales en otros Grids. Si se quiere que nuestro *GridWay* pueda enviar trabajos a un recurso descubierto consultando el índice de servicios de otra VO es necesario que este recurso sea accesible a los puertos **8443** y **2811** que son los servicios de Globus de ejecución y gsiftp.

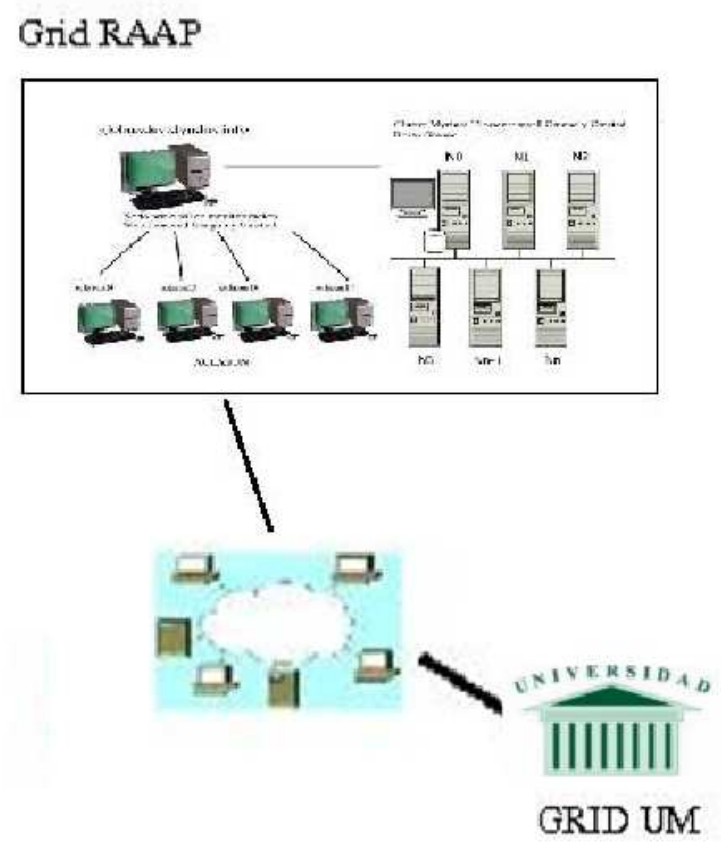


Figura 8.23: Grid formado por Grid RAAP y Grid UM.

Capítulo 9

Virtualizando el Grid con GridGateWay

9.1. Introducción

La computación Grid desarrollada en ámbitos científicos a principios de la última década del siglo pasado ha experimentado un gran difusión debido a la expansión de las redes de área amplia y la capacidad de los recursos computacionales. Esta tecnología innovadora en los 90 para utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetas a un control centralizado, ha empezado a entrar en el mercado comercial siguiendo la idea de la llamada *Utility Computing* [73]. Los servicios convencionales de Internet tienen la capacidad de organizar sus recursos para ser utilizados por sus usuarios, por ejemplo, almacenamiento para hosting, o capacidad de ancho de banda para permitir un gran volumen de tráfico de subida para servidores bancarios, etcétera El término *Utility Computing* se refiere a la virtualización de esos recursos para que los usuarios no vean espacios separados de almacenamiento y servidores distintos, sino un único recurso cuya capacidad de almacenamiento y cómputo es el conjunto de los recursos que anteriormente tenía el usuario alquilados. Esto abre las puertas a organizaciones que únicamente se dedican a ofrecer servicios de cómputo alquilando cierta capacidad de almacenamiento y capacidad de computación de forma suficientemente flexible como para ajustarse a las exigencias de los usuarios. Los usuarios que necesiten estos servicios contratarán éstos solicitando cierto cupo de potencia de cálculo y almacenamiento, y se le proporcionará al usuario un recurso que pueda utilizar apto para sus necesidades, sin necesidad de que éste gaste montones de dinero adquiriendo hardware nuevo. Incluso, el usuario podría aumentar sus exigencias de un momento a otro (siempre que no supere los recursos disponibles del proveedor) y obtener la potencia exigida. Todo ésto bajo un estricto control de uso y auditoría para que luego el usuario pague por los recursos usados.

En el paradigma de la *Utility Computing* hay que separar de forma total el usuario de los recursos. El cliente requiere un modo uniforme, seguro y confiable para acceder al servicio *Utility Computing*, y el proveedor requiere una infraestructura escalable, flexible y adaptativa, que perfectamente puede ser una infraestructura Grid,

para proporcionar servicios al cliente.

Desde el año 2000 muchas compañías en el sector de la computación han entrado en el mercado, pero todavía hay muchas pequeñas organizaciones que han usado o usan utility Computing. Principalmente esta se usa para el renderizado de películas de animación o creación de efectos especiales para el cine. Empresas como la del multimillonario Steve Jobs y no me refiero a Apple sino a Pixar utilizaron estos servicios para el renderizado de películas como Monstruos S.A.

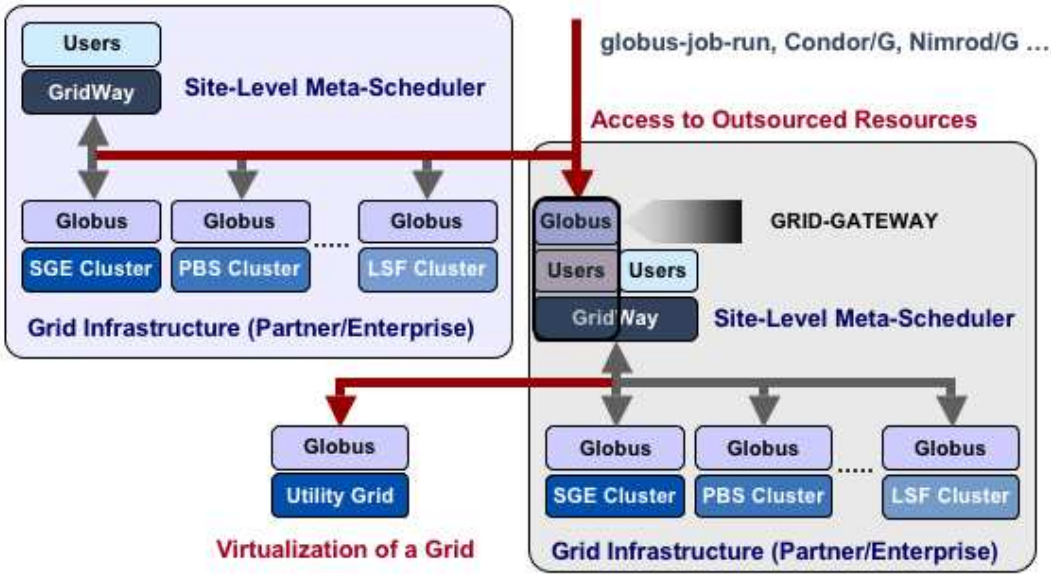


Figura 9.1: Federación de Grids basada en *Globus* y el metaplanificador *GridWay* .

9.2. Federación de infraestructuras Grid

La federación de infraestructuras es la explotación coordinada y simultánea de recursos pertenecientes a diferentes infraestructuras. Las aproximaciones actuales a esta problemática se pueden agrupar en las siguientes aproximaciones:

- Arquitectura de explotación simultánea de infraestructuras: en este tipo de arquitecturas, el sistema de meta-planificación debe estar instalado en los diferentes clientes y adaptado para que pueda interactuar simultáneamente con los middlewares de ambas infraestructuras.

- Arquitectura jerárquica de infraestructuras: esta arquitectura aisla ambas infraestructuras y las comunica por medio de puertas de enlace (Gateways) que transfieren los trabajos entre ambas. *GridGateWay* se pertenece a este tipo.

Existen varias soluciones de software que proporcionan una solución para la primera aproximación, una de ellas es *MOAB Grid Suite* de Cluster Resources [7], que proporciona gestión de carga de trabajo e integra planificación, gestión, monitorización y reparto de cargas de trabajo entre clusters independientes. El problema, es que estas soluciones no están basadas en estándares y requieren que el mismo gestor de carga esté instalado en todos los recursos. El grupo de investigación OGF *Grid Scheduling Architecture* [45] [23] está trabajando en una arquitectura estándar para la interacción de diferentes metaplanificadores, desarrollados por grupos de investigación distintos y de forma independente. También existen iniciativas para conseguir interoperabilidad entre los distintos middlewares como *Globus* con gLite (basado en pre-ws) y UNICORE. El grupo *grid4utility* desarrolló un banco de pruebas mixto basado en *Globus* para ejecutar una aplicación de Bioinformática a través del metaplanificador GridGateWay. El banco de pruebas se construyó a partir de recursos asignados a IRISGrid [48] y la EGEE [47], que es la mayor infraestructura Grid mundial con nivel de producción. Se demostró que era posible conseguir el uso coordinado y simultáneo de los recursos de ambas infraestructuras (IRISGrid y EGEE). El middleware *GridWay* permite la utilización de recursos con interfaces pre-servicio Web como gLite [39].

El uso de arquitecturas jerárquicas para la federación de infraestructuras Grid entre varias VO es la alternativa más escalable y flexible de las soluciones planteadas. Además, se trata del único modo de interconectar infraestructuras con middlewares incompatibles (Globus 4.X y gLite). El uso de estas arquitecturas jerárquicas requiere el desarrollo de nuevos módulos que agreguen la información de ambas infraestructuras. El nivel de agregación de la información es un parámetro que se debe analizar con el fin de que el metaplanificador tenga la información necesaria para la gestión eficiente de los trabajos. Existe un problema al añadir más capas a la infraestructura, toda esta novedosa arquitectura jerárquica implica cambios en los algoritmos de planificación. Hay que tener en consideración las sobrecargas en el envío y gestión de trabajos, así como los retardos entre cambios de estado. Un trabajo que se ejecute en un Grid *virtualizado* tardará más sobre los mismos recursos que un Grid accediéndose a través de su capa más baja del middleware. Este es un factor fundamental que hay que mejorar (disminuir latencia) para aumentar el rendimiento.

Como se comentaba en el capítulo 6, en el apartado 6.4, es necesario establecer un cierto control en los recursos sobre los usuarios que van a acceder a ellos y de que manera. Además el problema de tener que abrir los puertos de los servicios de *Globus* en los encaminadores para que se acceda a todos los recursos desde el exterior, intercambiar certificados, etcétera. La arquitectura presentada como federación de Grids tiene un gran número de ventajas sobre la tradicional forma de acceder *todos a todos*:

- Se mejora la seguridad, ya que ahora únicamente *GridGateWay* será accesible desde Internet. Las cuentas de usuario permitidas serán manejadas únicamente en un único sitio, sin necesidad de modificar todos los recursos accesibles.

- Las diferentes políticas de mapeo de usuarios pueden ser definidas a cada nivel, y el conjunto de autoridades certificadoras reconocidas puede ser diferente en los diferentes niveles. Un Grid formado por varios sub-Grids diferentes pertenecientes a distintas VO tienen distintas entidades certificadoras, pero no todos los sub-Grids tienen que tener compartidos con todos los certificados de sus CA. En cada

nivel se realiza un mapeo de certificados de un usuario de una CA remota a un usuario de la CA actual.

- En los diferentes niveles pueden aplicarse distintas políticas de planificación, por ejemplo en un Grid de una VO foránea puede que el cluster de cálculo Myrinet tenga menos prioridad que un recurso normal, mientras que en el Grid local no.

- No es necesario realizar una distribución masiva de la información de un recurso a todos los usuarios por los que este será usado, por parte de las herramientas de monitorización tipo Ganglia (véase capítulo 7). Únicamente se mostrará la información global de todos los recursos de un nivel.

Aquí entra en juego GridGateWay, un producto del proyecto de investigación *Grid4Utility, Federation of Grid Infrastructures and Utility Computing* del departamento de Arquitectura de Sistemas distribuidos de la Universidad Complutense de Madrid [20], que proporciona un acceso uniforme, flexible y adaptativo a los recursos computacionales de un Grid desplegado en una arquitectura basada en *Globus ToolKit*. En el capítulo 6 apartado 6.4 se habla de controlar el acceso a los recursos por parte de miembros de varias VO así como limitar el uso de los recursos que pueden verse y accederse en cada momento. *GridGateWay* encapsula un Grid dentro de otro grid, simulando un único recurso con un número de CPUs igual al que tienen todos los recursos del Grid conjuntamente. De este modo, la gestión de la seguridad, ejecución, información y ficheros es independiente en ambos Grids. La puerta de enlace actúa de representante (proxy) de los usuarios de la primera infraestructura en la segunda, mientras que los usuarios de la primera infraestructura obtienen información agregada y resumida de los recursos disponibles en la segunda infraestructura. Como se comentaba en el capítulo 6 no hace falta portar todos los usuarios de la primera organización en la segunda organización ni viceversa como se haría hasta ahora. En la Figura 9.1 puede verse a modo de ejemplo ilustrativo, la arquitectura básica de dos organizaciones. Una de ellas utiliza los recursos de la otra a través de *GridGateWay* que proporciona una virtualización de todos los recursos presentes en ella. Poco a poco según avance el capítulo se complicará esta arquitectura básica para permitir la virtualización de varias VO conjuntas en lo que se llama federación de Grids.

9.3. Arquitectura de GridGateWay

GridGateWay es un parche para *Globus* que modifica el servicio GRAM, RFT y MDS de un recurso Grid con *Globus* ToolKit (véase Figura 9.2. El parche añade un adaptador al servicio GRAM para que los trabajos que se envíen al recurso local sean ejecutados en el LRMS (Local Resource Management System) de *GridWay* en lugar de Fork. Los trabajos que se envíen por el servicio GRAM al recurso local, se delegarán a un usuario local y se enviarán al demonio *GridWay* instalado en el recurso local. El Servicio de Información *MDS* también se modifica por el parche de tal modo que ahora también se muestra la información de los recursos detectados por los drivers de acceso al medio de *GridWay* (véase capítulo 8). El metaplanificador *GridWay* se encarga del resto, enviando los trabajos que le llegan por la interfaz del servicio GRAM a los recursos de los que dispone, por lo tanto, desde el punto de vista de la capa de Grid (Grid

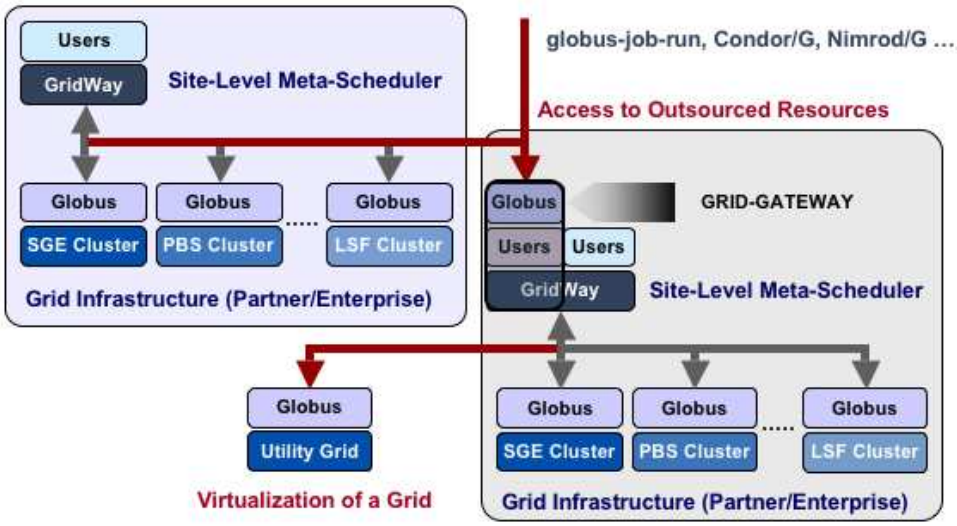


Figura 9.2: Arquitectura de GridGateWay.

Resource Management System) (véase Figura 9.3) es como si *Globus ToolKit*, además de su capa habitual de servicios, implementara un metaplanificador con las políticas más punteras de planificación de trabajos en Grids. En un futuro se espera añadir a *GridWay* toda la funcionalidad de *GridGateWay* más nuevas políticas de planificación a nivel de organización o conjunto de organizaciones. Además características típicas de los sistemas *Utility Computing* como es la facturación.

9.4. Instalación de GridGateWay

Los requisitos de *GridGateWay* son:

- *Globus* ToolKit 4.X instalado.
- Metaplanificador *GridWay* instalado en configuración multiusuario.
- GPT *Globus* packaging tools.

1. Hay que configurar el entorno de desarrollo *Globus* mediante la orden:
   ```
   $gpt-build --nosrc gcc32dbg
   ```
2. Descargar el parche para *Globus* y extraer dentro del usuario globus.
3. Aplicar el parche con:

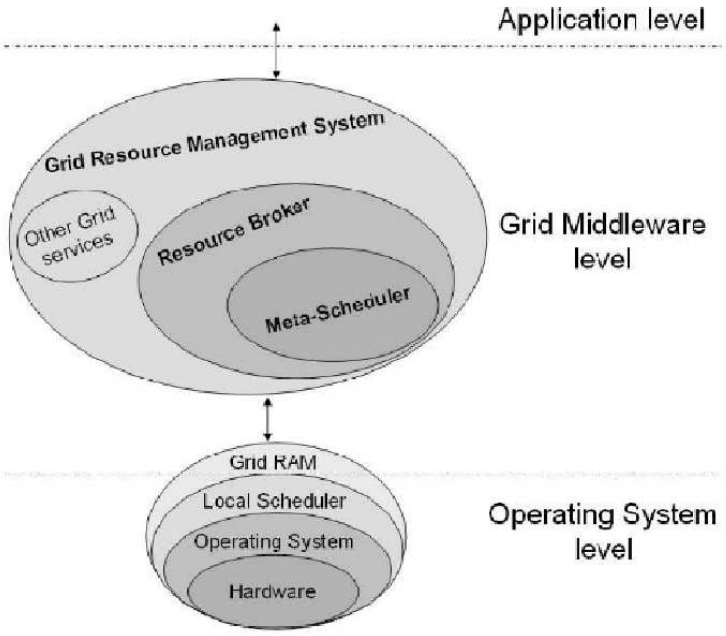


Figura 9.3: Anatomía de los gestores de recursos del Grid.

```
$ gpt-build -force gcc32dbg \
    globus_gram_job_manager_setup_gw.tar.gz \
    globus_scheduler_event_generator_gw.tar.gz \
    globus_scheduler_provider_setup_gw.tar.gz \
    globus_wsrf_gram_service_java_setup.tar.gz
```

4. Por último ejecutar un script para aplicar la configuración:

   $gpt-postinstall -force

   Después de hacer esto, hay que revisar el fichero de configuración *$GLOBUS_LOCATION/etc/globus_wsrf_rft/jndi-config.xml*, ya que se modifican los parámetros para el acceso a la base de datos RFT.

9.5. Configuración de usuarios de GridGateWay

La forma de crear usuarios debe seguir la filosofía del apartado 6.2.2 del capítulo 6. Se añadirán los certificados de las CA remotas a *Globus* para que confíe en ellas. Se creará un usuario por cada VO foránea que utilizará la interfaz de *GridGateWay* con todos los certificados y en ellos se mapearán todos los usuarios de las VO, cada una a su usuario. Estos usuarios recién creados deben ser usuarios de *GridWay* , ya que a través de estos usuarios se podrá acceder al metaplanificador *GridWay* y lanzar trabajos en nuestro Grid. La configuración de *GridWay* se ve en profundidad en el capítulo 8, apartados 8.11 y 8.12.

9.6. Configuración de GridGateWay

Después de la instalación, hay que iniciar el demonio de metaplanificación *GridWay* en modo multiusuario. El servicio MDS de *Globus* ahora publicará la información obtenida de la ejecución del comando *GridWay gwhost* (véase capítulo 8 apartado 8.5.1), por lo tanto el demonio debe de estar ejecutándose para cuando los servicios de *Globus* se arranquen. La información que muestra ahora puede ser obtenida ejecutando el comando
wsrf-query -s https://machineVOPedro:8443/wsrf/services/DefaultIndexService.

```
....
 <ns1:
ResourceID
xmlns:ns1="http://www.globus.org/namespaces/2004/10/gram/job">
GW</ns1:ResourceID>
<ns1:Info ns1:GRAMVersion="4.0.3"
 ns1:LRMSType="GW" ns1:LRMSVersion="1.0" ns1:TotalCPUs="17"/>
....
```

9.6.1. Configuración del servicio de descubrimiento MDS

Aparece una nueva interfaz que contiene todos los recursos del Grid detectados por *GridWay* . La información que se publica para la interfaz GW puede ser configurada modificando un script en
$GLOBUS_LOCATION/libexec/globus-scheduler-provider-gw. El script básicamente ejecuta el comando *gwhost* de *GridWay* , obtiene todas las CPUs de los recursos y las suma. Luego hace lo mismo con las CPUs libres.

9.6.2. Configuración del servicio de ejecución GRAM

El nuevo servicio GRAM que aparece en el servicio de descubrimiento MDS utiliza con LRMS un gestor llamado GW. Este gestor es el de GridGateWay. Puede configurarse incluso que en el script de configuración del MDS de *GridGateWay* que no aparezca el tipo GW. En este caso los usuarios remotos que utilicen este servicio no sabrán si se trata de un gestor Fork, Condor, PBS, etcétera, salvo por un matiz que hay que señalar, y es que por defecto GW no soporta *file-staging.* La información de file-staging tanto de entrada como de salida se trunca cuando se recibe el fichero de descripción de trabajo en la interfaz GRAM. La información de *file-staging* indica los ficheros a transferir para la ejecución del trabajo. El siguiente fragmento es de un fichero RSL (lenguaje de descripción de recursos), es eliminada directamente por GridGateWay.

```
...
<fileStageIn>
<transfer>
<sourceUrl>gsiftp://cognito.mcs.anl.gov:2811/bin/echo</sourceUrl>
<destinationUrl>file:///${GLOBUS_USER_HOME}/my_echo</destinationUrl>
</transfer>
</fileStageIn>
...
```

Cuando los trabajos se envían desde un metaplanificador *GridWay* a un recurso GridGateWay, es decir, no directamente desde clientes Globus, se detecta el tipo de recurso como *GW* (si el usuario no ha cambiado la configuración del proveedor MDS de GridGateWay). De este modo esa información de file-staging se añade en la sección de extensiones *extensions* del fichero RSL. De esta manera:

```
...
<extensions>
<gw>
INPUT_FILES=/bin/echo my_echo
</gw>
</extensions>
...
```

OJO esto no sería posible desde un cliente *Globus*, mediante *globusrun-ws* sobre un fichero RSL, puesto que *GridGateWay* elimina esa sección. Sería posible si se instalara un parche para *Globus* 4.x que es posible descargar de la página de *GridWay* [38]. Este parche permitiría que se incluyera información de file-staging al fichero RSL, el parche renombraría los campos del file-staging para poder ser lanzado en el *GridWay* encapsulado en GridGateWay. Desde *GridWay* no haría falta ningún parche ya que se utilizan extensiones.

El gestor de ejecución *Globus*, del que provee GridGateWay, obtiene un fichero RSL con los datos sobre el trabajo, y genera un directorio temporal para él. Este directorio temporal contiene la plantilla de trabajo, y ficheros de entrada copiados mediante el *two hop file staging*, del que se hablará a continuación. El script Perl que se encarga de generar las carpetas temporales también se encarga luego de limpiar todo, cuando el trabajo ha sido enviado de vuelta, es decir, ha pasado al estado *done* tanto en el Grid local como en el usuario remoto que ha lanzado el trabajo a través de GridGateWay.

File staging

La transferencia de los ficheros se realiza mediante dos saltos (*two hop file staging*). Al principio los ficheros son transferidos por el host cliente al servidor GridGateWay. Allí se genera un directorio de trabajo temporal donde quedan almacenados junto con las recién generadas plantillas de trabajo y demás parafernalia que necesita *GridWay* para enviar un trabajo. Posteriormente *GridWay* , cuando envía el trabajo a un recurso, se transfieren los ficheros desde este directorio temporal al objetivo, seleccionado para ejecutar el trabajo. Y posteriormente, cuando el trabajo termina, *GridWay* hace el paso inverso y copia los ficheros de vuelta al directorio temporal. *GridGateWay* se encarga de transferir los ficheros de nuevo de vuelta al host remoto que ha enviado el trabajo a través de éste. Existe otro modo llamado *one hop file staging* en el que

los ficheros se transmiten directamente desde el host remoto al recurso seleccionado, pero este modo se utiliza a través de clientes Globus. En el proyecto únicamente se han estimado los clientes *GridWay* s.

9.7. Implementando una federación de Grids entre varias Universidades mediante GridGateWay

Uno de los objetivos del proyecto es crear una federación de Grids entre tres universidades cercanas a la nuestra, y nuestra universidad, la Universidad de Castilla-La Mancha [64]. Concretamente, se trata de unir un Grid experimental desplegado con recursos computacionales del laboratorio RAAP del Instituto de Investigación en Informática de Albacete y Grids creados en tres universidades:

- Universidad de Murcia.
- Universidad Politécnica de Valencia.
- Universidad de Valencia.

Las Fases por las que se pasará para desplegar el Grid:

1. Implantación de un Grid con servicios *Globus* en los recursos disponibles en el RAAP.
2. Instalación y configuración de un metaplanificador en ese Grid para poder centralizar el uso de los recursos y usuarios que los usan, y planificar las tareas que se envian al Grid mediante políticas de planificación Grid punteras.
3. Instalación y configuración de *GridGateWay* sobre *GridWay* para poder ofrecer una interfaz GRAM hacia fuera de nuestra universidad.

El cuarto paso se realizará ahora, instalando el middleware necesario en cada universidad y preparando usuarios y certificados para permitir la comunicación entre Universidades a través de GridGateWay. En cada Universidad se instalarán los mismos componentes que se han instalado en el Grid experimental del RAAP:

- Globus.
- *GridWay* .
- GridGateWay.

Para preparar los equipos que servirán de puerta de enlace entre universidades es necesario seguir los pasos explicados en el capítulo 6 apartado 6.2.2

9.7.1. Universidad de Castilla-La Mancha

En la máquina que contiene el *GridGateWay* de la UCLM, se crearán usuarios genéricos para los usuarios de las 3 universidades. Por lo tanto, se crearán certificados locales para estos usuarios firmados por la CA de nuestra Universidad. Estos se llamarán:

- *umuser*. Para la Universidad de Murcia
- *uvuser*. Para la Universidad de Valencia.
- *upvuser*. Para la Universidad Politécnica de Valencia.

Estos usuarios tendrán que ser añadidos al fichero *grid-mapfile* de **todos los recursos del Grid local** según se indica en el apartado 8.7.2 del capítulo 7. El contenido del grid-mapfile para la máquina que ejecuta el demonio *GridWay* es el que se muestra en la Figura 9.4.

Los usuarios de cada universidad se mapean en los usuarios creados especificamente para englobar a los usuarios de cada universidad. Por ejemplo, los usuarios de la Universidad de Murcia, en este caso *gwadmin*, se mapearán en el usuario *umuser*.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=usuario" usuario
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=umuser" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=upvuser" upvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=uvuser" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=gwadmin" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=gwadmin" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-banca100.gap.upv.es/OU=upv.es/CN=gwadmin" upvuser
```

Figura 9.4: Fichero *grid-mapfile* para el host con *GridGateWay* de la UCLM

Al planificador *GridWay* se le crearán tres drivers para obtener información de estos recursos además del driver para los recursos locales que no se incluye:

```
IM_MAD = gridum:gw_im_mad_mds4_thr:-l \
listamurcia:gridftp:ws
IM_MAD = griduv:gw_im_mad_mds4_thr:-l \
listavalencia:gridftp:ws
IM_MAD = gridupv:gw_im_mad_mds4_thr:-l \
listaupv:gridftp:ws
```

Donde las listas son ficheros donde se define de forma estática la localización de los recursos remotos (véase apartado 8.7.3 del capítulo 8). En la configuración del planificador se le añadirá la misma prioridad a los recursos de las tres universidades puesto que estas están a la misma distancia (véase 8.4.3).

```
...
RP_IM[gridum]   = 02
RP_IM[griduv] =  02
RP_IM[gridupv] = 02
RP_IM[gridi3a] = 10
RP_HOST[Myrinet.i3a.info] = 05
...
```

A los trabajos de los usuarios se les añadirá distinta prioridad, siendo la de los usuarios locales mayor que las de los usuarios foráneos (véase 8.4.3).

```
...
FP_USER[uvuser]   = 02
FP_USER[upvuser] =  02
FP_USER[umuser] = 02
FP_USER[usuario] = 15
...
```

9.7.2. Universidad de Murcia

En la máquinas de la Universidad de Murcia que contiene el *GridGateWay* y será accesible desde el exterior, se crean los siguientes usuarios, donde se mapearán los usuarios remotos, de las otras universidades:

- *uclmuser*. Para la Universidad de Castilla-La Mancha
- *uvuser*. Para la Universidad de Valencia.
- *upvuser*. Para la Universidad politécnica de Valencia.

Hay que añadirlos al fichero de autorización *grid-mapfile* Figura 9.5.
Los drivers de información de *GridWay* quedan así:

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=gwadmin" gwadmin
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=uclmuser" uclmuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=uvuser" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=upvuser" upvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=gwadmin" upvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=gwadmin" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=usuario" uclmuser
```

Figura 9.5: Fichero *grid-mapfile* para el host con *GridGateWay* de la Universidad de Murcia

```
IM_MAD = griduclm:gw_im_mad_mds4_thr:-l \
listauclm:gridftp:ws
IM_MAD = griduv:gw_im_mad_mds4_thr:-l \
listavalencia:gridftp:ws
IM_MAD = gridupv:gw_im_mad_mds4_thr:-l \
listaupvs:gridftp:ws
```

```
...
RP_IM[gridum]   = 10
RP_IM[griduv] =  02
RP_IM[gridupv] = 02
RP_IM[griduclm] = 02
...
```

A los trabajos de los usuarios se les añadirá distinta prioridad, siendo la de los usuarios locales mayor que las de los usuarios foráneos (véase 8.4.3).

```
...
FP_USER[uvuser]   = 02
FP_USER[upvuser] =  02
FP_USER[usuario_local] = 15
FP_USER[uclmuser] = 02
...
```

9.7.3. Universidad Politécnica de Valencia

En las máquina que contiene el GridGateWay, de la Universidad Politécnica de Valencia se crearán los siguientes usuarios:

- *umuser*. Para la Universidad de Murcia
- *uvuser*. Para la Universidad de Valencia.
- *uclmuser*. Para la Universidad de Castilla-La Mancha.

Estos usuarios tendrán que ser añadidos al grid-mapfile de todos los recursos del Grid local según se indica en el apartado 8.7.2 del capítulo 7. El contenido del grid-mapfile para la máquina que ejecuta el demonio *GridWay* es el que se muestra en la Figura 9.6.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=gwadmin" gwadmin
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=uclmuser" uclmuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=uvuser" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=umuser" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=gwadmin" uvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=gwadmin" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=usuario" uclmuser
```

Figura 9.6: Fichero *grid-mapfile* para el host con *GridGateWay* de la UPV

Los drivers de *GridWay* quedan:

```
IM_MAD = gridum:gw_im_mad_mds4_thr:-l \
listamurcia:gridftp:ws
IM_MAD = griduv:gw_im_mad_mds4_thr:-l \
listavalencia:gridftp:ws
IM_MAD = griduclm:gw_im_mad_mds4_thr:-l \
listauclm:gridftp:ws
```

```
...
RP_IM[gridum]   = 02
RP_IM[griduv] =  02
RP_IM[griduclm] = 02
RP_IM[gridupv] = 10
...
```

A los trabajos de los usuarios se les añadirá distinta prioridad, siendo la de los usuarios locales mayor que las de los usuarios foráneos (véase 8.4.3)

```
...
FP_USER[uvuser]   = 02
FP_USER[usuario_local] =  15
FP_USER[umuser] = 02
FP_USER[uclmuser] = 02
...
```

9.7.4. Universidad de Valencia

Los usuarios a crear en la máquina con *GridGateWay* de Valencia son:

- *umuser*. Para la Universidad de Murcia
- *upvuser*. Para la Universidad Politécnica de Valencia.
- *uclmuser*. Para la Universidad de Castilla-La Mancha.

El fichero *grid-mapfile* resultante puede verse en la Figura 9.7.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=gwadmin" gwadmin
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=uclmuser" uclmuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=upvuser" upvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-griduv.uv.es/OU=uv.es/CN=umuser" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-inf-205-154.um.es/OU=um.es/CN=gwadmin" umuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-bancal00.gap.upv.es/OU=upv.es/CN=gwadmin" upvuser
"/O=Grid/OU=GlobusTest/OU=simpleCA-globusdev.dyndns.info/OU=dyndns.info/CN=usuario" uclmuser
```

Figura 9.7: Fichero *grid-mapfile* para el host con *GridGateWay* de Valencia

Los drivers de *GridWay* quedarían así:

```
IM_MAD = gridum:gw_im_mad_mds4_thr:-l \
listamurcia:gridftp:ws
IM_MAD = gridupv:gw_im_mad_mds4_thr:-l \
listaupv:gridftp:ws
IM_MAD = griduclm:gw_im_mad_mds4_thr:-l \
listauclm:gridftp:ws
```

La prioridad de los recursos descubiertos por cada driver puede verse a continuación:

```
...
RP_IM[gridum]   = 02
RP_IM[griduv] =  10
RP_IM[griduclm] = 02
RP_IM[gridupv] = 02
...
```

Del mismo modo la prioridad de los usuarios se muestra a continuación:

```
...
FP_USER[usuario_local]   = 15
FP_USER[upvuser] =  02
FP_USER[umuser] = 02
FP_USER[uclmuser] = 02
...
```

Los usuarios creados deben crear un proxy válido para poder utilizar la infraestructura segura de *Globus* (GSI), que se usará al utilizar el demonio *GridWay* al lanzar trabajos en el Grid local a través del driver de ejecución de *GridWay* .

9.7.5. Problemas encontrados

Mediante esta configuración en la que los distintos *GridWay* s proporcionan información a los demás meta-planificadores presenta un problema importante y es que se producirán ciclos de realimentación en la obtención de los recursos por parte de *GridWay* de otros GridGateWays. Esto ocurre de esta manera, para no complicar las cosas vamos a trabajar en el ejemplo únicamente con dos universidades, la UCLM y la UM:

1. Al principio la UCLM publica información en su interfaz *GridGateWay* sobre sus recursos locales. Es decir 16 CPUs.
2. La Universidad de Murcia publica información en sus recursos locales en su interfaz GridGateWay. 1 CPU.
3. La UCLM descubre a la UM y añade su CPU a la interfaz *GridGateWay* (GridGateWay obtiene información MDS de *GridWay* mediante gwhost), por lo tanto ahora publica un recurso con 16+1 = 17 CPUs.
4. La UM descubre a la UCLM y añade los recursos publicados por esta, es decir 17 CPUs a su interfaz. Por lo tanto ahora esta tiene 17 + 1 = 18 CPUs.
5. La UCLM obtiene información actualizada de la UM y ve que ahora tiene 18 CPUs, y las añade a sus recursos locales por lo que ahora tiene 16 + 18 = 34 CPUs.
6. La UM obtiene información actualizada de la UCLM y ve que ahora tiene 34 CPUS, la añade a las suyas con lo que ahora tiene 1 + 34 = 35 CPUs.
7. Y así sucesivamente...

9.7.6. Posibles soluciones

Una posible solución **sin tocar los scripts de configuración** de *GridGateWay* es añadir a cada Grid de cada Universidad un metaplanificador nuevo, instalado en un host distinto, sin acceso desde el exterior y para usuarios locales del Grid. El otro metaplanificador seguirá tal que está pero sólo ejecutará trabajos que vengan de fuera. El metaplanificador para trabajos locales tendrá una visión global de todos los Grids, ya que los incluirá en su driver IM para poder descubrirlos. El metaplanificador que ejecuta trabajos para usuarios foráneos deberá eliminar de su lista de descubrimiento

los Grids foráneos. De este modo desde un Grid foráneo que obtenga información de *GridGateWay* únicamente verá los recursos locales y no todos los recursos de todos los Grid que se irán realimentando con cada actualización de los recursos. Esta es una solución poco elegante pero sencilla. Aunque presenta un problema debido a que ahora los usuarios foráneos de un Grid competirán con los usuarios locales a éste ya que estamos hablando de dos metaplanificadores distintos sobre una misma *partner infrastructure* (véase capítulo 8 apartado 8.3.1) compitiendo por los mismos recursos. Esta configuración se parece mucho a la configuración monousuario de *GridWay* en la que todos los usuarios tienen su metaplanificador y acceden a los recursos individualmente compitiendo con los demás. No es la solución más aceptada que usuarios pertenecientes a Grids foráneos, compitan con los usuarios locales.

La mejor solución para arreglar el problema de los ciclos es modificar el script que proporciona información de *GridWay* vía MDS para que no publique los recursos descubiertos del tipo *GW*, es decir, de tipo GridGateWay. De este modo se evitarán los ciclos de realimentación. La configuración anterior para usuarios y políticas de planificación sigue siendo válida para todas las universidades. Hay que modificar el script *$GLOBUS_LOCATION/libexec/globus-scheduler-provider-gw*. Pero para ello hay que hacerlo de forma **indirecta** modificando el script *$GLOBUS_LOCATION/setup/globus/globus-scheduler-provider-gw.in* y luego ejecutando:

```
gpt-postinstall -force
```

Hay que evitar la adicción de recursos *GW* foráneos, a la salida del GridGateWay. El script proveedor de información obtiene información del comando *gwhost* de *GridWay* y de ahí obtiene el número de recursos ocupados y lo disponibles. A partir de esto, genera una salida para la interfaz MDS con un recurso con LRMS=GW y con CPUs igual a todos los recursos del Grid. Parte del script se muestra en la Figura 9.7.6

En la salida del comando *gwhost* hay que ver los recursos del tipo *GW* y no contabilizarlos a la hora de mostrar las CPUs. El cambio puede verse en la Figura 9.7.6

En la Figura 9.10 puede observarse la federación de infraestructuras Grids formada desde el punto de vista de la UCLM, ya que la UM la UPV y la UV usan recursos de la UCLM. Cada universidad estaría en la cima de esta federación.

En la Figura 9.7.6 puede observarse la vista que tiene la Universidad Murcia del Grid federado. La salida es consecuencia del comando *gwhost* realizado en el servidor *GridWay* de esa universidad. Del mismo modo, en la Figura 9.7.6 se puede ver los recursos accesibles por la Universidad de Valencia. El metaplanificador del Grid desplegado en el laboratorio RAAP de la UCLM en Albacete puede verse en la Figura 9.7.6.

```
system ("host `hostname` | grep -v mail | cut -d' ' -f1 > /tmp/hostname.tmp");
open (FP, "/tmp/hostname.tmp");
$hostname = <FP>;
chomp $hostname;
close FP;
system ("rm /tmp/hostname.tmp");

#-----------------------------------------------------
# from the gwhost command, get total and free nodes also calculate total cpus
#
open (FP, "$gwhost_cmd |");

my $free=0; $used=0; $total=0;

while ( <FP> )
{
my ($hid, $nice, $os, $arch, $mhz, $cpu, $mem, $disk, $nodes, $lrms) = split (' ',$_);
chomp $lrms;
my ($u, $f, $t) = split ('/', $nodes);
chomp $t;
$used += $u;
$free += $f;
$total += $t;
}
close FP;
...
```

Figura 9.8: Script de configuración globus-scheduler-provider-gw

Se muestra una parte del script proveedor de informacion para GridGateWay. Puede observarse que se obtiene información del comando *gwhost* para luego analizarla y de ahí sacar el número de CPUs disponibles y ocupadas además incorporar esta información a la salida del servicio MDS.

```
#-----------------------------------------------------
# from the gwhost command, get total and free nodes also calculate total cpus
#
open (FP, "$gwhost_cmd |");

my $free=0; $used=0; $total=0;

while ( <FP> )
{
my ($hid, $nice, $os, $arch, $mhz, $cpu, $mem, $disk, $nodes, $lrms) = split (' ',$_);
homp $lrms;
if ($lrms ne "GW")
{
   my ($u, $f, $t) = split ('/', $nodes);
   chomp $t;

   $used += $u;
   $free += $f;
   $total += $t;

}
close FP;
```

Figura 9.9: Fragmento de código modificado de GridGateWay

El fragmento modificado de código en el script *globus-scheduler-provider-gw* evita que los recursos *GW* se añadan al recuento de recursos que se publicarán en la interfaz MDS.

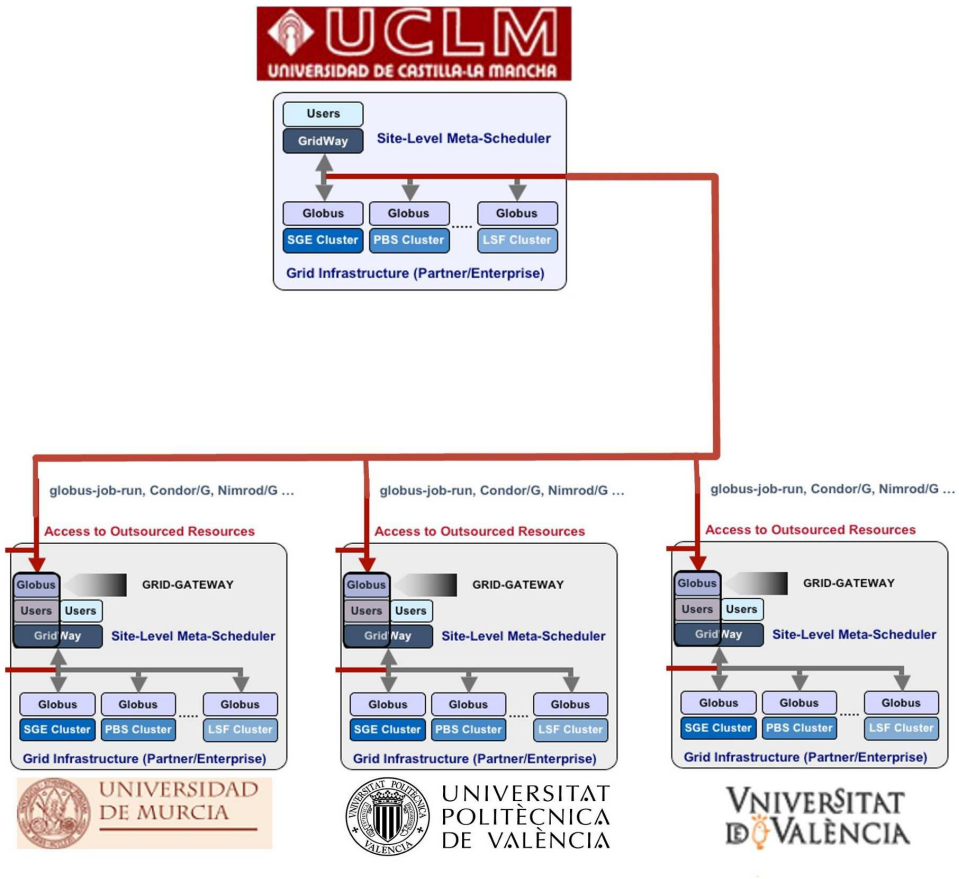


Figura 9.10: Federación de infraestructuras Grid formada por las 4 Universidades

```
HID PRIO   OS                  ARCH   MHZ  %CPU  MEM (F/T)      DISK (F/T)
N (U/F/T) LRMS                      HOSTNAME
0   5      Linux2.6.18-4-6     x86    3400  191     15/948  215782/250360      0/18/19 GW
globusdev.dyndns.info
1   5      Linux2.6.16.13-     x86    2448  100      6/504   59801/79734
0/1/1 GW cipriano.uv.es
2   1      Linux2.6.22-14-     x86    1992   91      6/503   50896/59416
0/1/1 GW inf-205-154.um.es
```

Figura 9.11: Salida del comando *gwhost* en la Universidad de Murcia.

```
HID PRIO  OS                    ARCH   MHZ %CPU  MEM (F/T)      DISK (F/T)
N (U/F/T) LRMS     HOSTNAME
0   10    Linux2.6.16-2-a x86_6 1794  197  491/1004   65461/72968        0/2/2 Fork
aulasun14.dsi.uclm.es
1   10    Linux2.6.16-2-a x86_6 1794  199  491/1004   65461/72968        0/2/2 Fork
aulasun15.dsi.uclm.es
2   10    Linux2.6.16-2-a x86_6 1794  197  491/1004   65461/72968        0/2/2 Fork
aulasun16.dsi.uclm.es
3   10    Linux2.6.16-2-a x86_6 1794  199  491/1004   65461/72968        0/2/2 Fork
aulasun17.dsi.uclm.es
4   10                            0    0      0/0          0/0        0/0/0
gridi3a.dyndns.info
5   5     Linux2.6.10-1.7 x86   2400  100   14/1519 309769/1069327
0/7/8 Condor Myrinet.i3a.info
6   2     Linux2.6.22-14- x86   1992  100     6/503   50896/59416
0/1/1 Fork     inf-205-154.um.es
7   2     Linux2.6.16.13- x86   2448  100     6/504   59824/79734
0/1/1 Fork     cipriano.uv.es
```

Figura 9.12: Salida del comando *gwhost* en el laboratorio RAAP del I3A.

```
HID PRIO  OS                    ARCH   MHZ %CPU  MEM (F/T)      DISK (F/T)
N (U/F/T) LRMS                      HOSTNAME
0   5     Linux2.6.18-4-6 x86   3400  187    15/948 215782/250360      0/18/19 GW
globusdev.dyndns.info
1   15    Linux2.6.16.13- x86   2448  100     6/504   59801/79734
0/1/1 GW cipriano.uv.es
2   1     Linux2.6.22-14- x86   1992   91     6/503   50896/59416
0/1/1 GW inf-205-154.um.es
```

Figura 9.13: Salida del comando *gwhost* en la Universidad de Valencia.

Capítulo 10

Experimentos

Los experimentos a realizar van encaminados a validar el funcionamiento de la plataforma Grid desplegada en los recursos disponibles del laboratorio RAAP, así como de los recursos añadidos al Grid pertenecientes a las 3 Universidades que se definieron en los objetivos del Proyecto. Una vez validados todos los recursos que componen la plataforma Grid, se procederá a evaluar el comportamiento del metaplanificador GridWay sobre el Grid desplegado. Se evaluará la productividad de los recursos bajo la plataforma Grid, ejecutando cargas de trabajo real en los distintos recursos que la componen. La arquitectura *GridGateWay* empleada para unir los Grids de las distintas Universidades, será analizada mediante el análisis de la productividad del sistema al enviar trabajos entre ellas.

10.1. Descripción de las pruebas

La realización de las pruebas se llevará a cabo en el Grid experimental descrito en el capítulo 9 apartado 9.7 con algunas modificaciones realizadas para realizar las pruebas. Se realizarán dos tipos de pruebas:

1. **Productividad.** Se evaluará la productividad del sistema utilizando todos los recursos del Grid a través de GridGateWay y/o directamente.
2. **Latencia.** Se evaluará el tiempo de ejecución de tareas con muchos requerimientos de ancho de banda de red desde las distintas Universidades en el Grid.

10.1.1. Recursos disponibles

Cada Universidad ha donado recursos computacionales que han sido agregados a a la federación Grid. Los recursos computacionales de cada organización se describen a continuación:

Universidad	Recurso	CPUs
UCLM	4 x SUN Workstations AMD Opteron Linux 2.6.16-2-amd64-k8-smp	8
UCLM	1 x Cluster Myrinet con gestor Condor. Linux 2.6.10 x86 FC2 Pentium(R) 4 2.40GHz	10
UM	1 x PC Pentium(R) 4 2.00 GHz Linux 2.6.22-14-generic Ubuntu 7.10	1
UV	1 x PC Pentium(R) 4 CPU 2.4 GHz Linux 2.6.16.13-4-default SuSE 10.1	1
UPV	1 x Cluster Myrinet SuSE 9.3 Pentium 3	0

Nota: El cluster Myrinet de la Universidad Politécnica de Valencia, no pudo ser añadido al Grid debido a un problema software en los requisitos de instalación de la capa middleware, necesaria para integrar el recurso en el entorno Grid. Debido a que ya existe un cluster Myrinet, con gestor de recursos Condor, se considera suficiente con los recursos proporcionados por las demás Universidades, para realizar las pruebas propuestas.

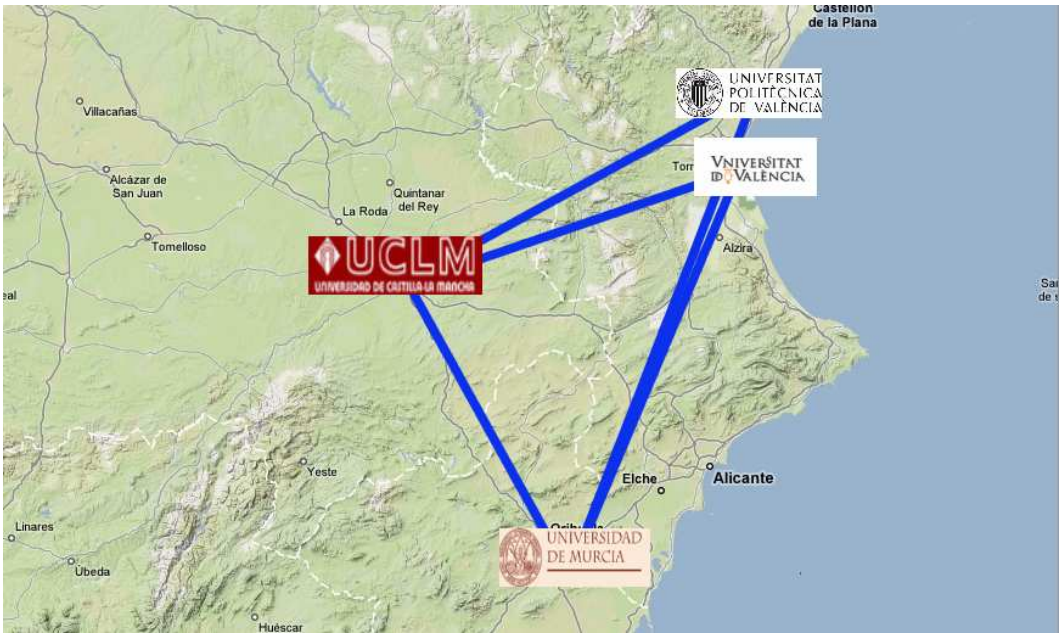


Figura 10.1: Mapa del Grid entre las 4 Universidades.

Los recursos de la Universidad de Castilla-La Mancha están ocultos tras un interfaz GridGateWay. Por lo tanto, para enviar trabajos a sus recursos, las otras Universidades tienen que pasar a través de esta interfaz GRAM. Los recursos de las demás Universidades son accedidos de forma directa, ya que únicamente han proporcionado un único recurso al Grid y es innecesario instalar una puerta de enlace puesto que lo único que ésta hace, es abstraer varios recursos inaccesibles de otra forma desde el exterior por políticas de cortafuegos en un único recurso accesible desde el exterior. De ésta forma, en la Figura 10.1 puede observarse el grid formado para las pruebas.

HID	PRIO	OS	ARCH	MHZ	%CPU	MEM(F/T)	DISK(F/T)	N(U/F/T)	LRMS	HOSTNAME
0	10	Linux2.6.16-2-a	x86_6	1794	200	142/1004	64411/72968	0/2/2	Fork	aulasun16.dsi.uclm.es
1	10	Linux2.6.16-2-a	x86_6	1794	200	142/1004	64411/72968	0/2/2	Fork	aulasun14.dsi.uclm.es
2	5	Linux2.6.10-1.7	x86	2400	97	196/1519	310566/1069327	0/8/9	Condor	myrinet.i3a.info
3	2	Linux2.6.22-14-	x86	1992	99	7/503	50171/59416	0/1/1	Fork	inf-205-154.um.es
4	10	Linux2.6.16-2-a	x86_6	1794	199	142/1004	64411/72968	0/2/2	Fork	aulasun17.dsi.uclm.es
5	10	Linux2.6.16-2-a	x86_6	1794	199	142/1004	64411/72968	0/2/2	Fork	aulasun15.dsi.uclm.es
6	2	Linux2.6.16.13-	x86	2448	100	6/504	56821/79734	0/1/1	Fork	cipriano.uv.es
7	10			0	0	0/0	0/0	0/0/0		gridi3a.dyndns.info

Figura 10.2: Recursos visibles por GridWay en el RAAP.

HID	PRIO	OS	ARCH	MHZ	%CPU	MEM(F/T)	DISK(F/T)	N(U/F/T)	LRMS	HOSTNAME
0	5	Linux2.6.18-4-6	x86	3400	130	316/948	213369/250360	0/18/19	GW	globusdev.dyndns.info
1	0	Linux2.6.22-14-	x86	1992	99	7/503	50171/59416	0/1/1	Fork	inf-205-154.um.es
2	0	Linux2.6.16.13-	x86	2448	100	5/504	56821/79734	0/1/1	Fork	cipriano.uv.es

Figura 10.3: Recursos visibles por GridWay en la Universidad de Valencia.

HID	PRIO	OS	ARCH	MHZ	%CPU	MEM(F/T)	DISK(F/T)	N(U/F/T)	LRMS	HOSTNAME
0	5	Linux2.6.18-4-6	x86	3400	130	316/948	213369/250360	0/18/19	GW	globusdev.dyndns.info
1	0	Linux2.6.16.13-	x86	2448	100	5/504	56821/79734	0/1/1	Fork	cipriano.uv.es
2	0	Linux2.6.22-14-	x86	1992	94	5/503	50171/59416	0/1/1	Fork	inf-205-154.um.es

Figura 10.4: Recursos visibles por GridWay en la Universidad de Murcia.

En la Figura 10.2 pueden verse los recursos disponibles desde el metaplanificador del Laboratorio RAAP. En la Figura 10.3 y 10.4 se muestran los recursos accesibles por la Universidad de Valencia y por la Universidad de Murcia.

10.1.2. Carga de trabajo

Como carga de trabajo se utilizará una versión modificada de los benchmarks paralelos *NAS Parallel Benchmarks (NPB)*, concretamente los *NAS Grid Benchmarks (NGB)*. El banco de pruebas original *(NPB)* fue concebido en 1991 con objetivo de evaluar el hardware y el software de sistemas paralelos, que en aquella época eran muy diversos debido a la inexistencia de estándares. Se utilizaron componentes de cálculo intensivo de dinámica de fluídos de la *NASA*. La primera versión del banco de prueba consistía en una especificación de éstos, que dejaba los aspectos de implementación al programador para que realizara el banco de pruebas para su arquitectura paralela. Los programas de referencia usados estaban escritos en Fortran, pero había que realizar cambios para adaptarlo a la arquitectura paralela a evaluar. Estas prueban cobraron gran importancia entre investigadores y fabricantes. El banco de pruebas *NPB* está diseñado para arquitecturas paralelas. Por lo tanto, hardware dedicado interconectado por redes de altas prestaciones como multicomputadores o multiprocesadores, pero no para entornos Grid donde el hardware pasa a ser no dedicado como ordenadores personales o estaciones de trabajo, y las redes de altas prestaciones pasan a ser redes de área amplia *(WAN)*, donde la calidad de servicio y latencia son bastante peores. Por

ello, se diseñaron los bancos de pruebas de Grid *(NGB)*, que intentan proporcionar una herramienta para evaluar los entornos Grid.

Descripción del banco de pruebas *NGB*

Las pruebas *NGB* [59][70] consisten en un conjunto de problemas de cálculo de dinámica de fluidos basados en el banco de pruebas paralelo original. Son las siguientes:

- **Embarrassingly Distributed(ED)**. Consisten en aplicaciones llamadas paramétricas, ya que se lanzan muchas de forma paralela con diferentes parámetros y devuelven sus resultados para cada ejecución. No hay comunicación entre las tareas lanzadas y por lo tanto no se genera tráfico de red que ralentice la ejecución de la prueba, salvo por la inicialización de la prueba y por la recogida de resultados. Puede verse su diagrama de tareas en la Figura 10.5.
- **Helical Chain(HC)**. Consisten en cadenas de pruebas repetitivas que se ejecutan una tras otra. Más que para evaluar la productividad de un Grid, están diseñadas para calcular la latencia que añade el Grid a la ejecución secuencial de tareas. Estos tests se ejecutan mejor como procesos dentro de una máquina y no repartidos por recursos del Grid, a no ser que los recursos remotos sean más potentes en cuanto potencia de cálculo en coma flotante (FPUs muy rápidas). En cada iteración es necesario traer los resultados de vuelta y posteriormente enviarlos junto con el trabajo a otro recurso para que compute la siguiente solución, y así sucesivamente hasta el número especificado de iteraciones. Puede verse un diagrama de esta prueba en la Figura 10.6.
- **Visualization Pipeline (VP)**. La prueba consiste en tres problemas diferentes: BT, MG y FT. Cada uno necesita datos obtenidos por los demás para comenzar su ejecución. Esta prueba es un intermedio entre la ED y la HC, puesto que puede extraerse algo de paralelismo de las tareas (cosa que no ocurre con la *HC*) pero no tanto como en la *ED*. En la Figura 10.7 puede verse el diagrama de la prueba.
- **Mixed Bag (MG)**. La prueba es muy similar a la prueba *VP*, ya que implica un flujo de tareas que dependen unas de otras. La diferencia radica en las cantidades asimétricas datos enviadas de una tarea a otra.

De los conjuntos de pruebas descritos se utilizarán modificaciones de los ED, HC y VP.

10.2. Pruebas de productividad

Como productividad se define el cociente entre los trabajos que realiza un sistema y el tiempo que tarda en realizarlos (trabajos/hora). Con estas pruebas se pretende evaluar la capacidad de GridWay para optimizar la utilización de los recursos del Grid. Se utilizará un conjunto de pruebas *ED* ejecutadas al mismo tiempo para que consuman todos los recursos disponibles. Luego en función de los trabajos que

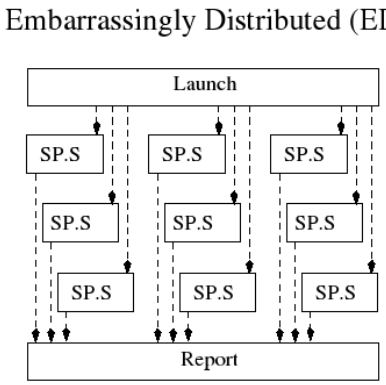


Figura 10.5: Prueba ED.

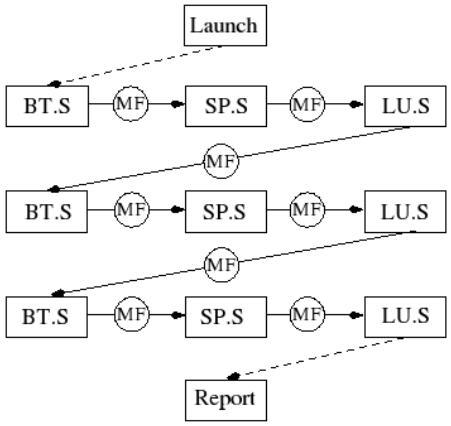


Figura 10.6: Prueba HC.

se ejecuten por unidad de tiempo se obtendrá la productividad del sistema en varias situaciones:

1. Variando los tiempos de monitorización de recursos y de planificación.
2. Utilizando acceso directo a los recursos o a través de GridGateWay lanzando trabajos desde la Universidad de Murcia o de Valencia al Grid de la UCLM.

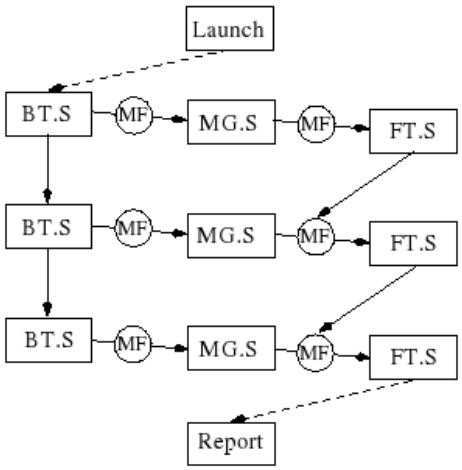


Figura 10.7: Prueba VP.

Para las siguientes pruebas se han utilizado 10 pruebas *ED*, cada prueba *ED* está formada por 9 tareas paramétricas independientes, lo que hace un total de 90 tareas. Se envian estas 90 tareas a GridWay y se monitorizan los trabajos que van acabando cada minuto con un script de monitorización consultando los logs de salida. El script es:

```
#!/bin/sh
while [ true ]
do
echo `ls -l | grep stdout. | wc -l`
sleep 60
```

10.2.1. Variando parámetros del metaplanificador

Lo que se pretende conseguir es evaluar el nivel de utilización por parte del metaplanificador GridWay de los recursos del Grid. Para ello se han modificado los tiempos de monitorización de los recursos y de planificación por parte del demonio GridWay. En la Figura B.13 puede observarse las dos curvas que representan la productividad del sistema con ciertos valores de intervalo de planificación y monitorización de recursos. Al principio la productividad es nula puesto que las tareas se han lanzado, y no ha terminado ninguna. Según va pasando el tiempo la productividad va creciendo hasta llegar a una series de picos que son los picos de productividad del sistema. Esos picos indican que la utilización de recursos es la máxima posible para la configuración actual de planificación y monitorización. Una vez pasado un pico, la productividad en ambas curvas cae en picado debido a que los recursos se saturan y la herramienta de monitorización Ganglia, de la que Globus se surte para publicar la información sobre el estado del recursos, tiene una cierta latencia que hace que los recursos parezcan ocupados cuando en realidad no lo están. Además de Ganglia, Globus también introduce un retardo en la publicación de su estado. Una vez el estado de saturación

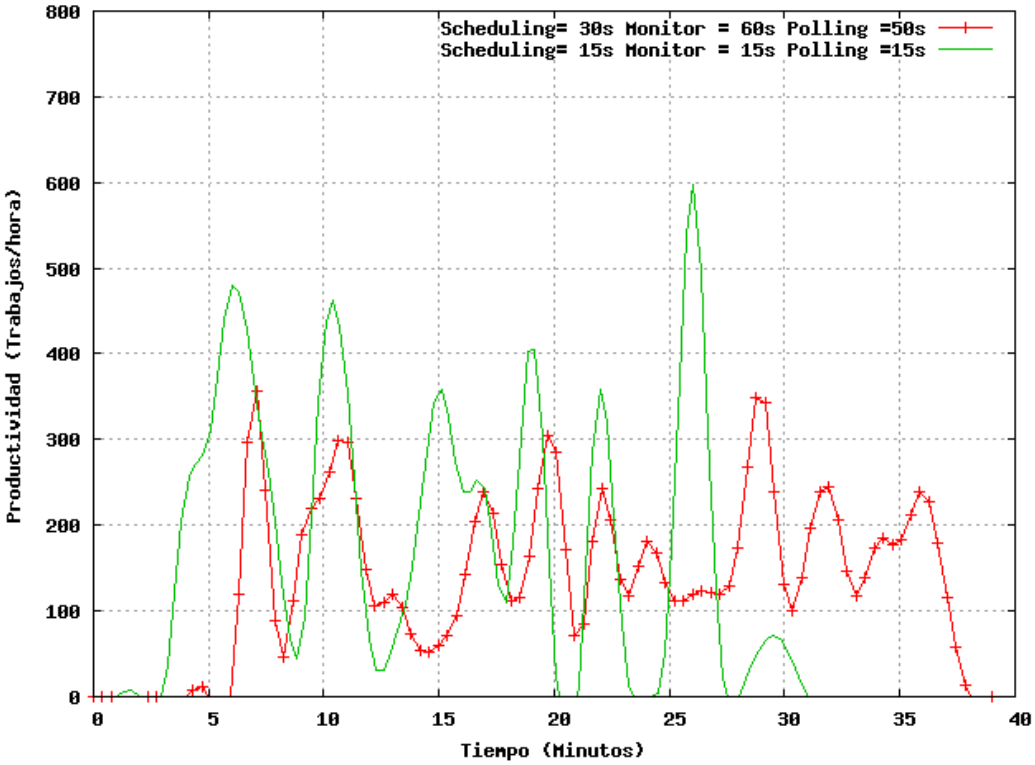


Figura 10.8: Gráfica de la productividad del Grid modificando los intervalos de planificación.

pasa, GridWay empieza a utilizar de nuevo recursos enviando trabajos a ellos. Puede observarse que la curva de productividad obtenida por la configuración de GridWay con mayores intervalos de planificación y monitorización es más lenta en responder después de una saturación que la configuración con tiempos más pequeños. La productividad media conseguida por la configuración de mayor tiempo de planificación es de **135.49 trabajos/hora** mientras que en la configuración más rápida, la productividad media es de **172.5 trabajos/hora**. Si se reduce aún más el intervalo de planificación y monitorización de recursos, se incrementa el tráfico de la red y el consumo de memoria y de procesador que hacen Globus y GridWay del servidor sobre el que se ejecutan. En la Figura 10.9 puede observarse que la productividad incluso se ha incrementado llegando a una media de **199.28 trabajos/hora**. Se ha superado en un 15 % la productividad conseguida por la configuración más rápida obtenida con 15 segundos de intervalo de planificación y 15 segundos de monitorización.

10.2.2. Acceso directo o a través de GridGateWay

Se pretende evaluar cuanta sobrecarga añade GridGateWay a la latencia y por lo tanto a la productividad de los recursos disponibles. GridGateWay actúa de puerta de enlace entre un Grid y otro. Los trabajos enviados deberán ser copiados primero en la puerta de enlace y posteriormente lanzarse al metaplanificador local del Grid.

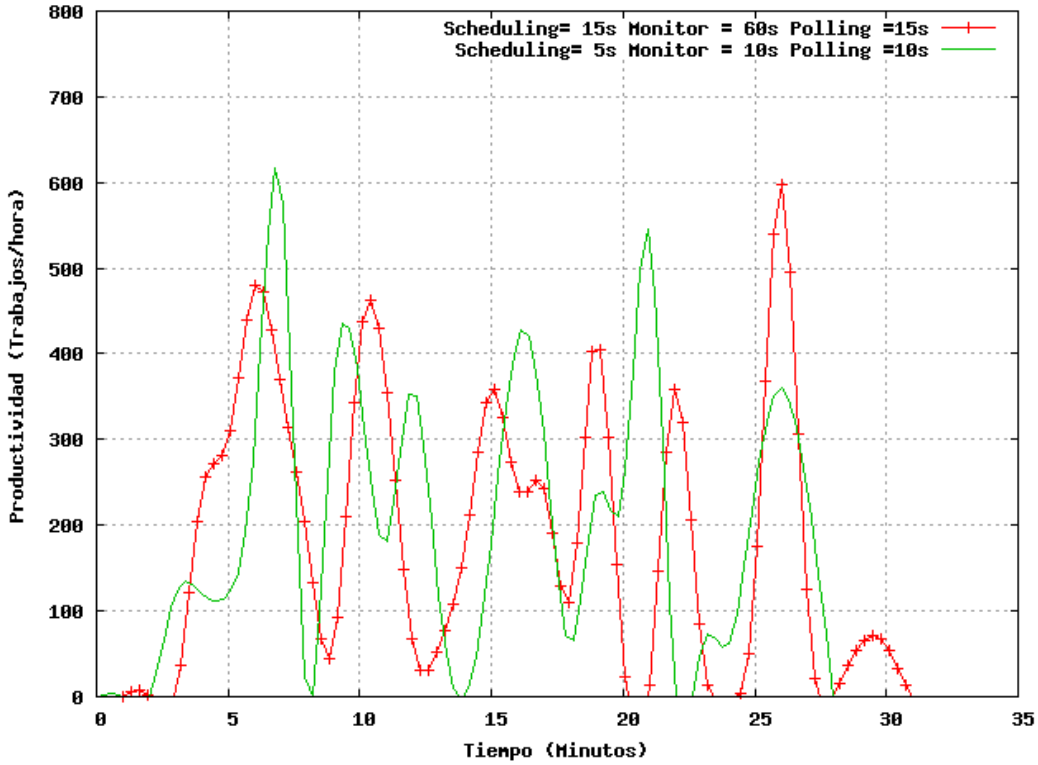


Figura 10.9: Mejorando la productividad del Grid disminuyendo intervalos de planificación.

Para realizar esta prueba se han lanzado 90 trabajos ED de forma concurrente desde Valencia y desde Murcia, primero desde un sitio y luego desde el otro. Los metaplanificadores locales para la Universidad de Murcia y la de Valencia estaban configurados para únicamente utilizar el recurso GridGateWay de la UCLM. Por lo tanto, es como si estuvieran accediendo a los recursos totales de las tres Universidades a través de una interfaz GGW, que es lo que se pretendía. En la Figura 10.10 se observan las curvas de productividad de las tres configuraciones: lanzando trabajos desde Murcia, desde Valencia y desde el metaplanificador bajo GGW de forma directa. La curva de productividad con mejor comportamiento es sin duda la de la configuración directa. Siempre es la más rápida, ya que no se van a realizar transferencias secundarias hasta el recurso GGW, y además, no se produce sobrecarga al acceder de forma directa a los recursos del Grid. Cabe destacar que la curva de productividad conseguida al lanzar trabajos desde Murcia a través de la interfaz GGW de la UCLM es muy similar a la curva accediendo directamente, e incluso bastante más alta que la curva de Valencia. Esto es debido a la latencia introducida por los enlaces WAN que hay que atravesar para llegar hasta los recursos. En el momento en que se hicieron las pruebas el ancho de banda de la comunicación con Murcia era mayor que para Valencia, pero esto puede variar con el tiempo dependiendo del estado de saturación de las líneas. En las Figuras 10.11 y 10.12 pueden observarse las gráficas de comparativa entre el acceso directo a recursos y el acceso a ellos a través de GGW desde Valencia y Murcia respectivamente. La productividad media conseguida lanzando trabados desde Murcia para las pruebas es de **170.62 trabajos/hora**, y para Valencia **160.58 trabajos/hora**.

Todos los metaplanificadores de la federación Grid tienen la misma configuración que la segunda configuración del apartado anterior (intervalo de planificación 15 s y monitorización de recursos 15 s). El acceso directo da únicamente **2.5 trabajos/hora** más de media que el acceso desde Murcia a través de GridGateWay para el tipo de pruebas utilizado. Cuánto menos requerimientos computacionales tengan las pruebas, más penalizará GridGateWay a la productividad de éstas, ya que en este caso, las transferencias de información ocuparán un papel más importante en el tiempo de vida de la tarea.

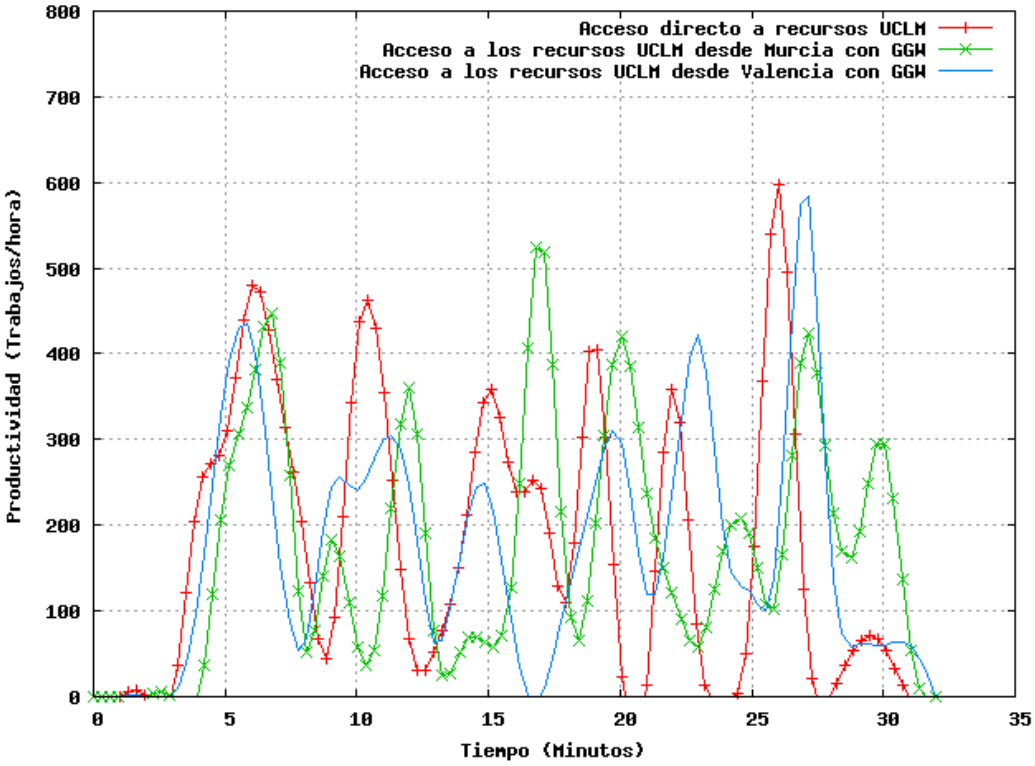


Figura 10.10: Gráfica comparativa entre el acceso directo a recursos y el acceso a través de GGW.

10.3. Pruebas de Latencia

Estas pruebas se realizan para evaluar las prestaciones del Grid frente a la ejecución en un único procesador. Se usará la prueba *VP* que ejecutará un conjunto de nueve tareas con dependencias entre sí de tal modo que la salida de una es utilizada como entrada en otra tarea. Es necesario realizar transferencias de datos entre los distintos recursos, por lo tanto se evaluarán las prestaciones del Grid frente a la ejecución monoprocesador. Aquí, la velocidad de la red juega un papel importante a la hora de calcular la latencia total de la ejecución de la prueba, ya que los ficheros generados por las tareas de la prueba *VP* son de centenares de megabytes. Para la ejecución de la prueba *VP* en el Grid es necesario separar en tareas el código original

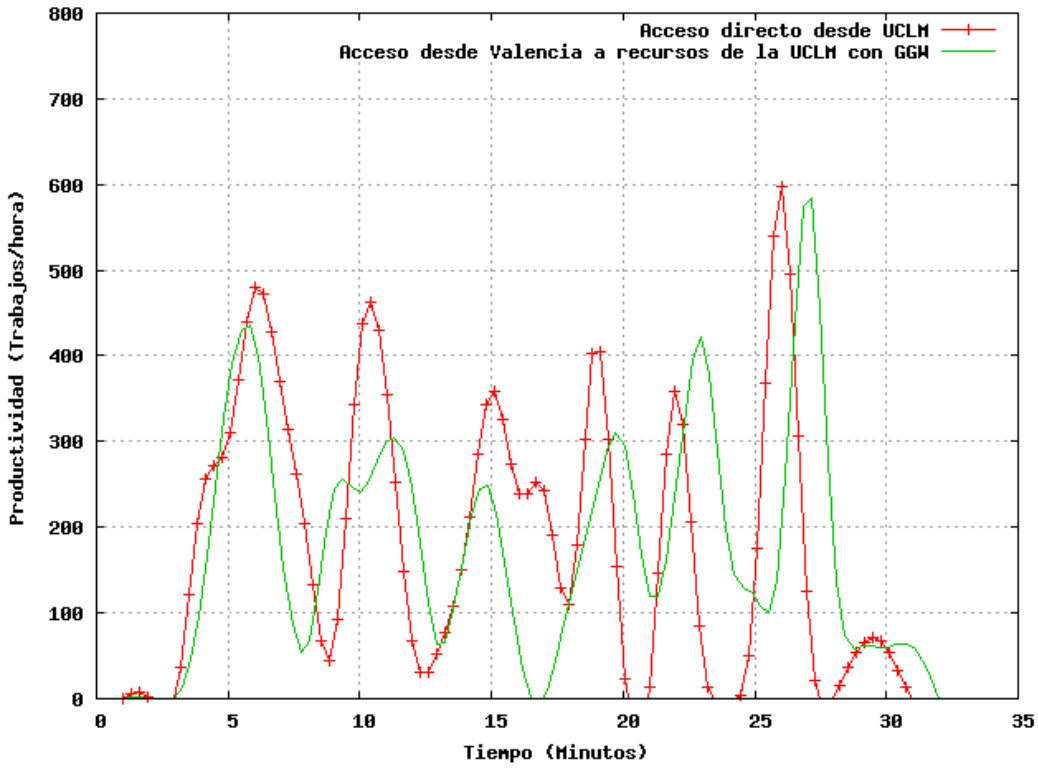


Figura 10.11: Acceso desde Valencia a GGW.

de las pruebas *NGB*. Para más información sobre la implementación de las pruebas consultar el Apendice A. En la Figura 10.13 pueden verse los tiempos de ejecución de la prueba VP en distintos recursos:

1. Configuración no paralela, ejecutada en un único recurso, en este caso *cipriano.uv.es*, una máquina del Grid de la Universidad de Valencia.

2. Ejecución paralela en el Grid RAAP del I3A. Redes de interconexión de área local entre recursos Gigabit Ethernet.

3. Ejecución paralela desde Valencia utilizando el recurso GGW de la Universidad de Castilla-La Mancha, pasando por cada trabajo dos veces por enlaces WAN.

La elección de un recurso Grid foráneo como candidato a ejecutar una prueba *VP* es una mala decisión, ya sea a través de GridGateWay o accediendo directamente al recurso. La ejecución de esta prueba necesita grandes ficheros que tienen que ser transferidos del nodo del cliente al nodo de cálculo y luego de vuelta de nuevo a otro nodo para otra parte de la prueba que necesite del fichero generado anteriormente. En la Figura 10.13 puede apreciarse perfectamente que la prueba tarda más del doble que la misma prueba ejecutada en un único procesador. La prueba no presenta tanto paralelismo como la prueba *Embarrasinly Distributed*, aunque presenta paralelismo de grano grueso que es explotado por los recursos locales del Grid del I3A conectados por redes LAN Gigabit

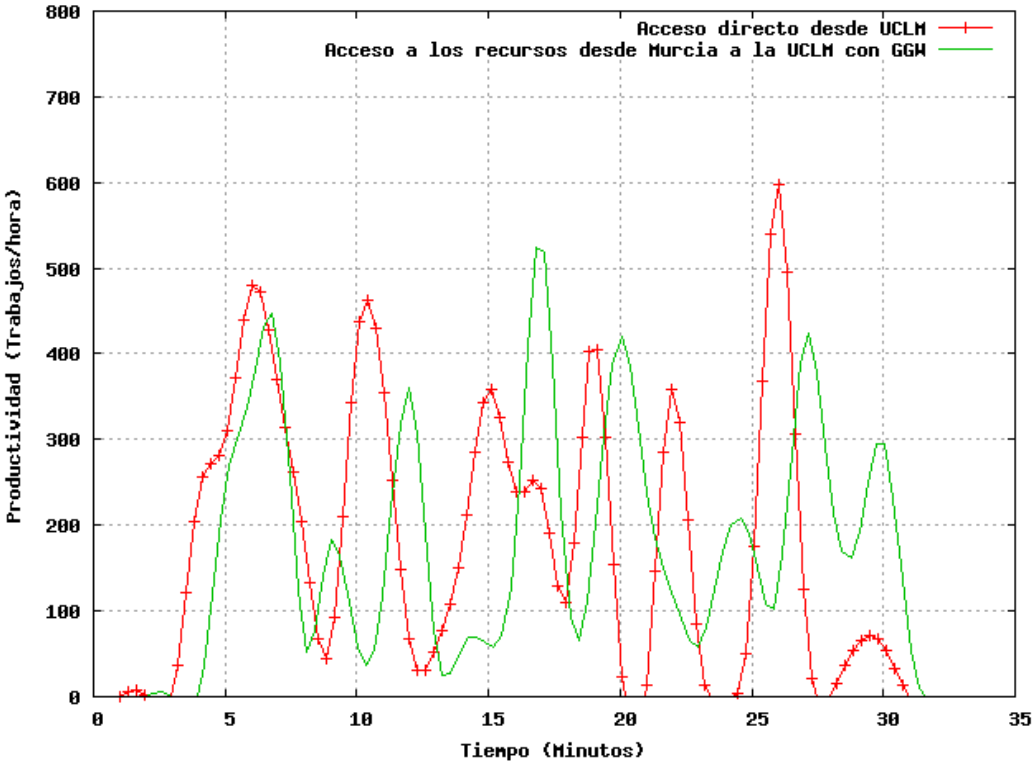


Figura 10.12: Acceso desde Murcia a GGW.

Ethernet, con capacidad superior a los enlaces WAN presentes entre la Universidad de Valencia y la de Castilla-La Mancha. La prueba ejecutada en paralelo en los recursos Grid locales supera en un **13 %** a la misma prueba no paralela ejecutada en el recurso de Valencia *cipriano.uv.es*.

10.4. Conclusiones

Las infraestructuras Grid abren las puertas a los científicos e investigadores a un ingente número de recursos de forma transparente, que de otra forma no podrían acceder, ya que una organización no puede afrontar los costes de muchos nodos de cómputo dedicados, ni la energía que estos gastan. Pero si éstos se encuentran en lugares geográficos distintos, e incluso pertenecen a organizaciones diferentes, el gasto se reparte entre el conjunto y la unión hace la fuerza. Estos recursos Grid que pueden ser encontrados en una infraestructura Grid pueden tener arquitecturas y sistemas operativos dispares, además de distintas capacidades como espacio de almacenamiento y memoria disponible, así como número de procesadores etc... Se ha visto en el apartado anterior que no siempre es más rápida la ejecución paralela que la ejecución en serie en un único recurso, y menos en un Grid, donde el hardware y las redes de interconexión no son "dedicadas", es decir, no son de "altas prestaciones", y por lo tanto hay que decidir en función del problema computacional a tratar si se va a diseñar de cara a la ejecución en un Grid, donde las tareas deberían carecer de dependencias para que se ejecuten

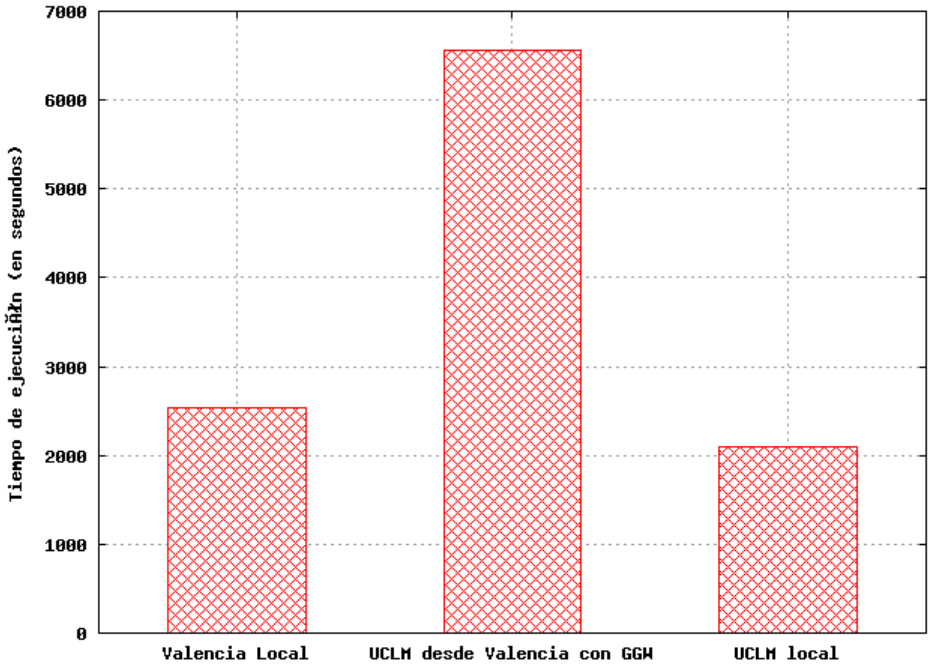


Figura 10.13: Tiempos de ejecución de la prueba VP en diferentes recursos del Grid.

bien en él, o por el contrario existen dependencias de grano fino que deben tratarse implementando el problema para una arquitectura paralela, como es un multiprocesador o un multicomputador, pero no para un Grid. GridWay como metaplanificador del Grid ha logrado una gran optimización de los recursos locales de un Grid, además de los recursos foráneos si el planificador está correctamente configurado dando menos prioridad a éstos recursos. Queda en manos del usuario final que ejecutará trabajos en el Grid, la productividad de éste. La productividad que alcanza GridWay del Grid es dependiente del tipo de tareas que se ejecuten en él.

Capítulo 11

GWGUI: Herramienta de diseño de WorkFlows para el metaplanificador GridWay

11.1. Introducción

Las ventajas que ofrece a los usuarios finales el metaplanificador GridWay explicado en el capítulo 8, son la implementación de su propia API DRMAA compatible con OGF, y por lo tanto compatible con la totalidad de las aplicaciones desarrolladas compatibles con la API DRMAA de OGF 1.4.0; y la interfaz *Command Line Interface* que provee un pequeño pero potente juego de comandos de terminal a los clientes que ejecuten trabajos vía SSH de forma remota al servidor de GridWay, sin necesidad de tener ninguna biblioteca instalada, ni acceso a ella. La ventaja de la interfaz CLI frente a utilizar la API DRMAA es la posibilidad de expresar dependencias entre tareas de forma estática, es decir, no es necesario que una aplicación esté conectada al servidor GridWay durante toda la ejecución de los trabajos para controlar las dependencias entre tareas, como ocurre con la API DRMAA. Aunque se programe en C o en Java (las dos implementaciones de la API DRMAA), en las dos se expresan las dependencias de forma dinámica.

En la interfaz de línea de comandos de GridWay es posible expresar de una sola vez y de forma estática, todas las dependencias entre todas las tareas (véase capítulo 8 apartado 8.5.1). El problema radica en que usuarios que quieran probar la potencia de esta CLI deben aprender este lenguaje de comandos y traducir sus aplicaciones a lenguajes de script. Usuarios que tenían desarrollados flujos de trabajos desde Java o C ahora tienen que empezar generar plantillas de trabajo a mano, scripts que definan dependencias entre ellas, subir todo esto al servidor de GridWay, y posteriormente, ejecutar el trabajo. O por el contrario, sería posible generar todos los scripts remotamente conectado a la máquina con GridWay y allí diseñar los WorkFlows de forma, mediante los comandos que proporciona *GridWay*.

Debería existir una herramienta gráfica sencilla para diseñar WorkFlows de tareas de forma intuitiva, que generara scripts y plantillas de trabajo de forma au-

tomática para las tareas creadas. De esta manera , los científicos e investigadores de cualquier campo, que necesiten utilizar recursos del Grid a través del metaplanificador GridWay, no tengan que centrarse en programar scripts para controlar sus trabajos, sino intentar centrarse en sus problemas de campo y dedicar un tiempo mínimo a escribir código para poder comunicarse con GridWay, en este caso no escribir código sino diseñar gráficamente lo que quieren hacer.

Se pretende desarrollar una herramienta que introduzca las mejoras anteriormente citadas a la interfaz de línea de comandos de GridWay. La herramienta deberá ser capaz de generar scripts para ejecutar plantillas de trabajo que también creará la aplicación a partir de flujos de trabajo diseñados por el usuario. El usuario podrá crear nuevos flujos de trabajo, crear tareas y dependencias en ellos. También editar todo lo relacionado con la ejecución de las tareas, es decir, ficheros de entrada, salida, entrada estándar, salida estándar, errores, etc... También será posible almacenar flujos de trabajo diseñados o cargar éstos desde ficheros portables. Por último sería recomendable que la aplicación se encargara de subir los trabajos al servidor de GridWay y lanzarlos a ejecución. Los usuarios se desconectarían y los trabajos seguirían su curso hasta que éstos terminaran, ya que las dependencias entre los trabajos se definen de forma estática como se ha citado anteriormente.

11.2. Desarrollo de la aplicación

11.2.1. Metodología de Software

Para desarrollar esta aplicación se va a aplicar una metodología ágil que siga un ciclo de vida iterativo e incremental, es decir, una metodología de ingeniería del software que permite realizar varias iteraciones a lo largo del ciclo de vida del proyecto software. En cada iteración se realiza de nuevo el análisis de requisitos, diseño, implementación y pruebas. El sistema va completándose a medida que avanzan las iteraciones, de tal forma que al terminar una iteración se procuran corregir todos los defectos posibles descubiertos a partir de errores en las pruebas durante esa iteración, pero que pueden pertenecer al código desarrollado en esa iteración, o de defectos sin corregir de anteriores iteraciones. No se seguirá ninguna metodología en particular dentro de las englobadas en este tipo de procesos ágiles de desarrollo como *Rational Unified Process (RUP)* [50] o *xTREME Programming (XP)* [16], ya que el proyecto ha desarrollar no es de gran envergadura como los que se pueden afrontar con RUP, no tendrá tantos requisitos que exijan muchas iteraciones, ni tantas fases en cada ciclo de vida, ya que los requisitos a analizar, diseñar y codificar son muy pocos. Las actividades que se desarrollarán en cada ciclo de desarrollo son:

1. **Requisitos y análisis.** En esta fase se añadirán requisitos nuevos al software. Los requisitos se modelarán siguiendo la notación del *lenguaje de modelado unificado (UML)* [37]. Se modelarán casos de uso a partir de los requisitos según los objetivos planteados para mejorar las necesidades en la interfaz de línea de comandos de GridWay. Se diseñarán diagramas de actividad que reflejen como se realizarán los casos de uso que se descubran en esta actividad.

2. **Diseño**. En esta fase se interpretará la funcionalidad requerida y se generarán los correspondientes diagramas de clases que mostrarán la vista estática del sistema.
3. **Implementación**. A partir de estas clases definidas en la actividad anterior se comenzará a implementar los casos de uso de la aplicación.
4. **Pruebas.** Se realizarán pequeñas pruebas de unidad a los casos de uso implementados. Estas pruebas las lleva a cabo el programador, y ayudan a corregir la mayoría de los fallos que se producen durante la codificación, que son erratas en la sintaxis del lenguaje o pequeños fallos en la semántica de lo que se esperaba hacer con el código escrito. Estas pruebas no se suelen documentar puesto que las diseña el programador y no un equipo de pruebas designado a tal efecto.

   En la última iteración del ciclo de vida se realizarán las pruebas de aceptación por un usuario final, es decir, un usuario del Grid que vaya a diseñar WorkFlows con la ayuda de la aplicación desarrollada. En esta prueba se demostrará si la aplicación ofrece la funcionalidad especificada en los casos de uso en condicionales normales de uso.

El ciclo de vida constará de 2 iteraciones. En cada iteración se realizarán las actividades descritas en el apartado anterior. Además en la segunda se realizarán las pruebas de aceptación.

11.2.2. Primera iteración

Requisitos y análisis

Al analizar los requisitos de alto nivel de la aplicación aparecen tres escenarios:

1. **Gestión de WorkFlows.**
2. **Gestión de diseño.**
3. **Gestión de trabajos.**

Gestión de WorkFlows:
En el escenario *Gestión de WorkFlows* se realizarán tareas realicionadas con los WorkFlows. En esta primera iteración aparecen dos casos de uso llamados:

- **UC-1 Crear WorkFlow**. El caso caso de uso comienza cuando el usuario del sistema se dispone a crear un nuevo WorkFlow. Se creará en la interfaz gráfica un nuevo espacio de trabajo para comenzar a diseñar el nuevo WorkFlow recién creado.
- **UC-2 Cerrar WorkFlow.** El caso de uso comienza cuando el usuario del sistema se dispone a cerrar un WorkFlow existente. El sistema preguntará al usuario si está seguro de cerrarlo. Si el usuario decide que no, el caso de uso termina. Si el usuario decide proseguir, la aplicación se encargará de limpiar de la interfaz gráfica todo rastro del WorkFlow.

Aparece un actor llamado *Usuario*, que será el usuario del Grid. El diagrama de casos de uso del escenario puede verse en la Figura 11.1.

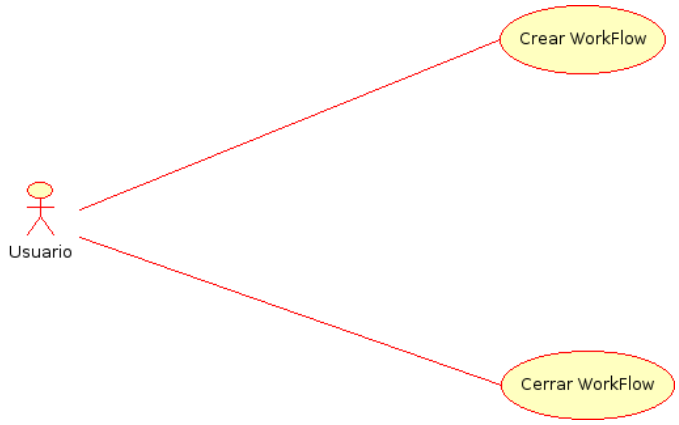


Figura 11.1: Escenario *Gestión de WorkFlows*.

Gestión de diseño

En este escenario se realizarán tareas relacionadas con la edición de tareas y dependencias. El usuario podrá diseñar sus propias tareas y dependencias entre tareas. Además el usuario será capaz de editar todos los parámetros relacionados con la tarea como son los ficheros, de entrada, salida... etc (véase 8 apartado 8.5.1). Se descubren los siguientes casos de uso:

- **UC-3 Crear tarea**. El caso de uso comienza cuando el usuario se dispone a crear una nueva tarea en un WorkFlow previamente creado. Se crea una nueva tarea en la interfaz gráfica.

- **UC-4 Editar tarea.** El caso de uso comienza cuando el usuario se dispone a editar una tarea previamente creada. Podrá editar todos los parámetros relacionados con las plantillas de trabajo para el metaplanificador GridWay.

- **UC-5 Eliminar tarea**. El caso de uso comienza cuando el usuario se dispone a eliminar una tarea previamente creada. Se preguntará al usuario por su decisión. Si el usuario decide borrar la tarea está se eliminará del sistema con todas las dependencias asociadas a ella ejecutando el caso de uso UC-7 Eliminar dependencia. Si no, el caso de uso terminará sin ningún efecto a posteriori.

- **UC-6 Crear dependencia.** El caso de uso comienza cuando el usuario se dispone a crear una dependencia entre dos tareas del sistema previamente creadas. El usuario seleccionará crear nueva dependencia. El sistema entonces solicitará al usuario seleccionar la primera tarea. Si el usuario realiza una selección errónea entonces el caso de uso termina. Si el usuario selecciona bien la tarea se le pedirá seleccionar la segunda tarea. Si se produce un error en la selección el caso de uso terminará, de lo contrario se creará una nueva dependencia entre las dos tareas seleccionadas. Si al seleccionar la segunda tarea se descubre que ya era

hija de la primera tarea, es decir, existía un camino en el grafo dirigido, entonces el caso de uso finalizará sin ningún efecto.

- **UC-7 Eliminar dependencia.** El caso de uso comienza cuando el usuario se dispone a eliminar una dependencia previamente creada entre dos tareas del sistema. Se preguntará al usuario por su decisión. Si decide proseguir con su decisión se eliminará la dependencia del sistema. Si no, el caso de uso terminará.

El diagrama de casos de uso puede verse en la Figura 11.2.

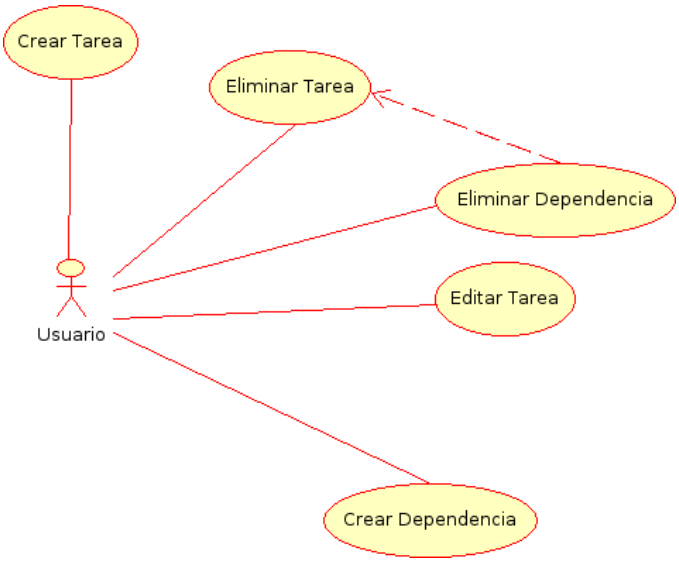


Figura 11.2: Escenario *Gestión de diseño*.

Gestión de trabajos

En este escenario se realizarán tareas relacionadas con la generación de scripts y plantillas de trabajo a partir de un WorkFlow. Se descubre el siguiente caso de uso:

- **UC-8 Generar scripts**. El caso de uso comienza cuando el usuario decide generar scripts a partir de un WorkFlow existente en el sistema. Se le preguntará al usuario por la localización en la que se generarán éstos, y a continuación se generarán las plantillas de trabajo para cada tarea del sistema y el script para el flujo de tareas.

El diagrama de casos de uso puede verse en la Figura 11.3.

Diagramas de actividad

Los diagramas de actividad más representativos, descubiertos en esta primera iteración pueden verse en las Figuras 11.4, 11.6 y 11.5.

Figura 11.3: Escenario *Gestión de trabajos.*

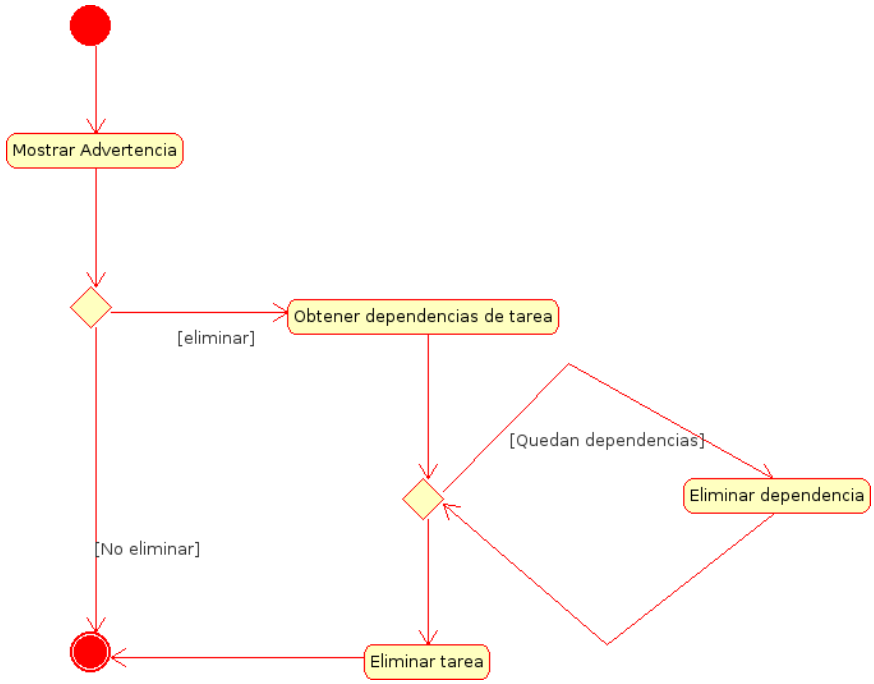


Figura 11.4: Diagrama de actividad de *UC-5 Eliminar tarea.*

Diseño

A partir de los casos de uso encontrados para los escenarios definidos puede definirse la arquitectura candidata del sistema. En la Figura 11.7 puede verse el diagrama de clases del sistema.

Para la implementación de la aplicación sería recomendable utilizar un lenguaje orientado a objetos, para poder generar una aplicación que pueda ejecutarse en varios sistemas operativos. La aplicaciones que se realizan para ejecutarse en paralelo utilizando recursos como clusters o Grids son implementadas para sistemas operativos tipo Unix, como Solaris o Linux. Un lenguaje candidato para desarrollar la aplicación tanto por sus cualidades como lenguaje orientado a objetos y multiplataforma, como por la amplia variedad de componentes de código abierto disponibles para utilizar, es Java.

Clases del sistema

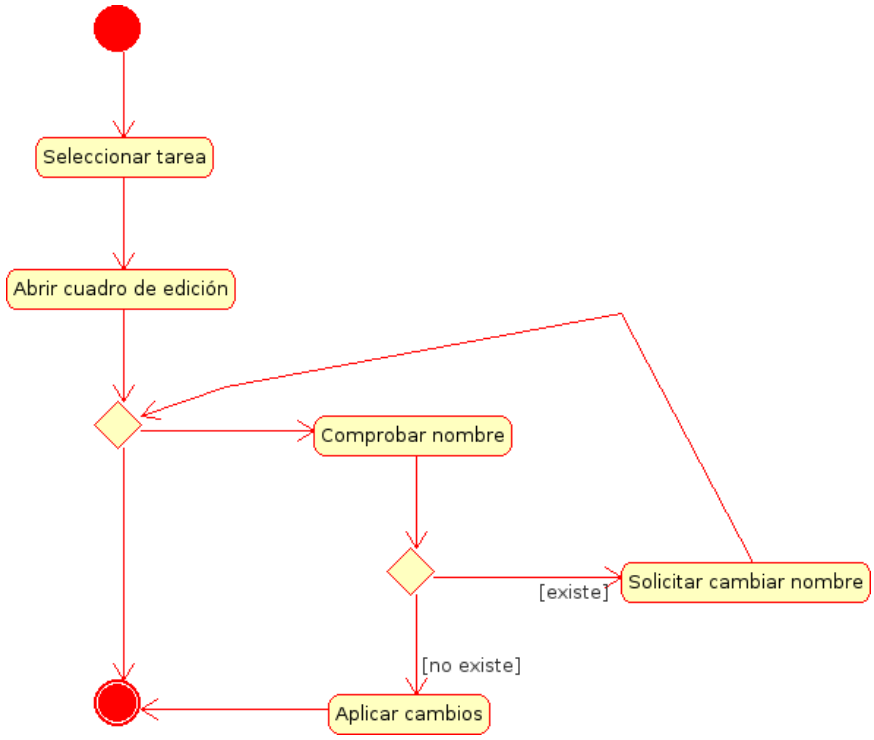


Figura 11.5: Diagrama de actividad de *UC-4 Editar tarea*.

- **GUI**. Es la clase principal de la interfaz gráfica. Se encargará de soportar las entradas del usuario y redireccionarlas a las diferentes clases del sistema.
- **WorkFlow**. Implementará la lógica de los flujos de trabajo albergando tareas y dependencias. Pueden verse los atributos y métodos de la clase en la Figura 11.8.
- **Tarea**. Esta clase servirá de contenedor para toda la información relacionada con las plantillas de trabajo de las tareas. Además de la lógica de generación de plantillas de trabajo. Pueden verse los atributos y métodos de la clase Tarea en la Figura 11.9.
- **Dependencia**. Esta clase contendrá información sobre las clases implicadas en una dependencia. En posteriores iteraciones puede que se amplíe el cometido de esta clase añadiendo más información sobre la dependencia entre tareas.
- **Fichero**. Esta clase implementa un tipo abstracto de dato nuevo para definir los ficheros que se definen en las tareas. Contiene información sobre si es fichero de entrada, salida, remoto, etc... (véase capítulo 8 apartado 8.5.1).

Implementación

Para el desarrollo de la interfaz gráfica, en la que poder visualizar y editar flujos de trabajo, se ha escogido la versión libre de las bibliotecas JGraph [32]. Para la

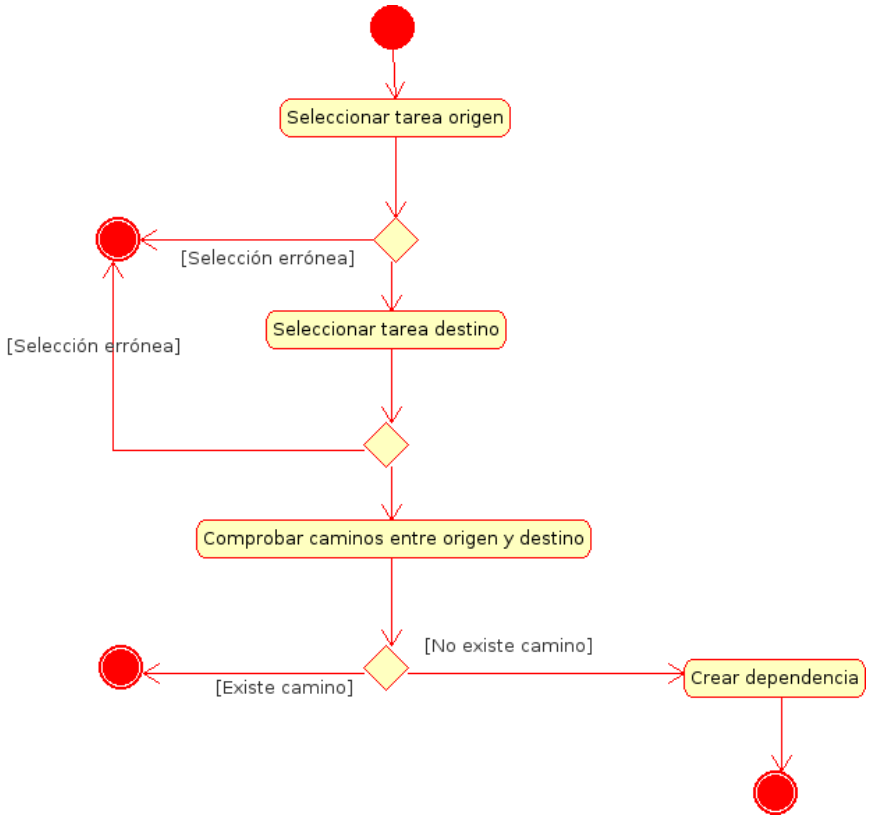


Figura 11.6: Diagrama de actividad de *UC-6 Crear dependencia*.

implementación del caso de uso *UC-4 Editar Tarea* es necesario desplegar un calendario para editar la *fecha de deadline*, un campo de la plantilla de trabajo para GridWay. Se ha escogido la biblioteca jCalendar [31] para mostrar el calendario. Como entorno de desarrollo para el proyecto que se desarrollará en Java se ha escogido el entorno de desarrollo integrado *Netbeans 5.5.1 IDE* [42] para Linux y JDK 1.5. En la Figura 11.10 puede verse el diagrama de paquetes del sistema.

- **paqInterfaz**. Este paquete contendrá todas las clases relacionadas con la interfaz gráfica. El diagrama de clases del paquete puede verse en la Figura 11.7. Este paquete además utiliza las bibliotecas JGraph, para visualizar grafos y jCalendar. Además usa el paquete paqSintactico.

- **paqSintactico**. Este paquete contendrá las clases de dos analizadores sintácticos para determinar si los campos *RANK* y *REQUIREMENTS* de la plantilla de trabajo son reconocidos por las gramáticas que las definen (véase Figuras 8.10 y 8.11).

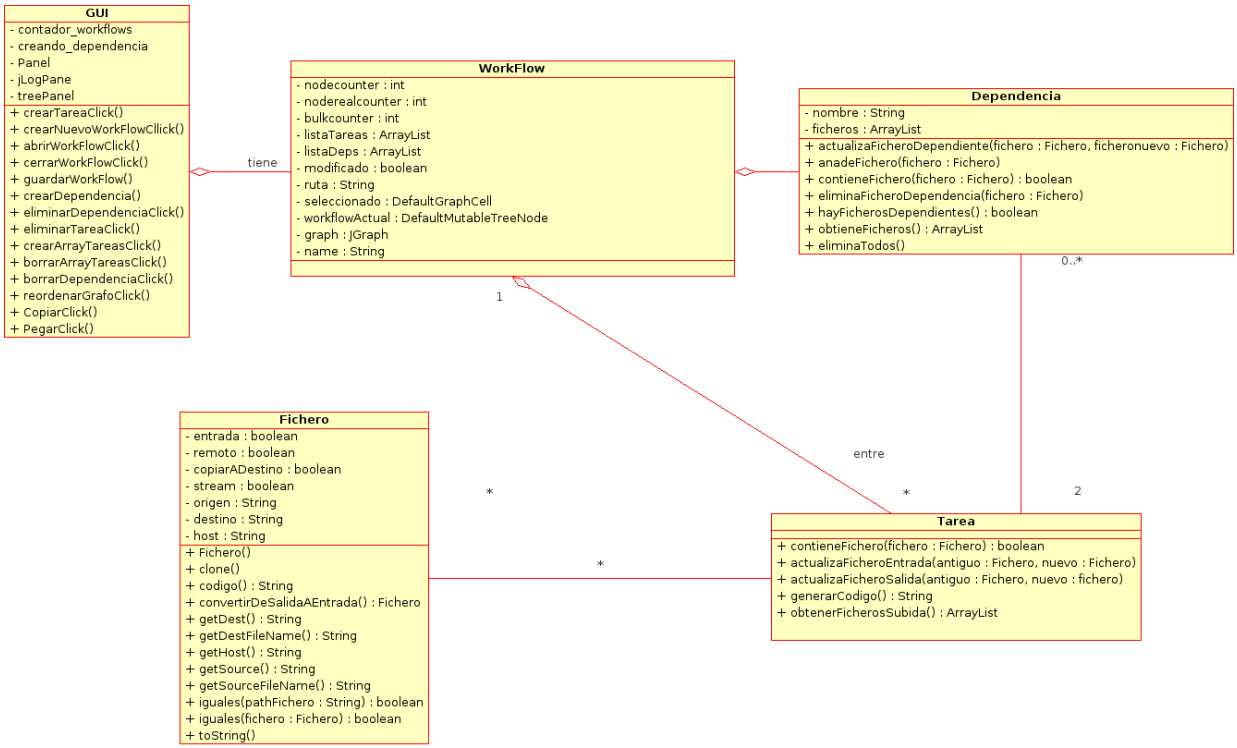


Figura 11.7: Diagrama de clases 1º Iteración.

Resultado de la primera iteración

Se han implementado los casos de uso especificados para los escenarios *Gestión de WorkFlows*, *Gestión de trabajos* y *Gestión de diseño*. La aplicación todavía no tiene toda la funcionalidad expresada en la introducción de este capítulo aunque es funcional como **prototipo**. El prototipo nos ayuda a comprender mejor la funcionalidad que le falta a la aplicación y lo que debería cambiarse de ésta.
La pantalla principal de la aplicación prototipo desarrollada consta de varias partes que pueden verse en la Figura 11.11. Estas partes son:

1. **Panel principal**. Permite la creación y destrucción de tareas y dependencias. La edición puede invocarse realizando dos clicks de ratón sobre el elemento a editar o mediante el menú contextual seleccionando el objeto y presionando el botón derecho.

2. **Log**. La mayoría de las acciones llevadas a cabo por la interfaz gráfica quedan reflejadas en este log.

3. **TreeView**. El árbol muestra los WorkFlows abiertos actualmente, así como sus tareas y dependencias creadas. Las tareas y dependencias se muestran agrupadas en subcarpetas. Es posible realizar la edición de los elementos directamente sobre el árbol mediante menús contextuales accesibles con el botón derecho del ratón.

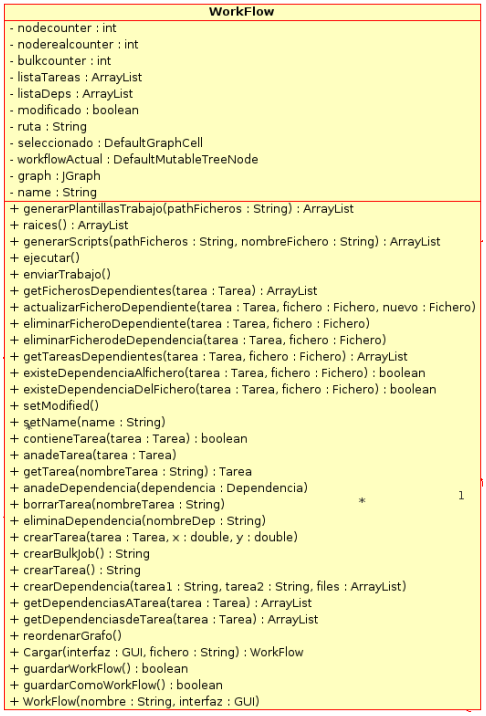


Figura 11.8: Vista de la clase WorkFlow con sus atributos y métodos.

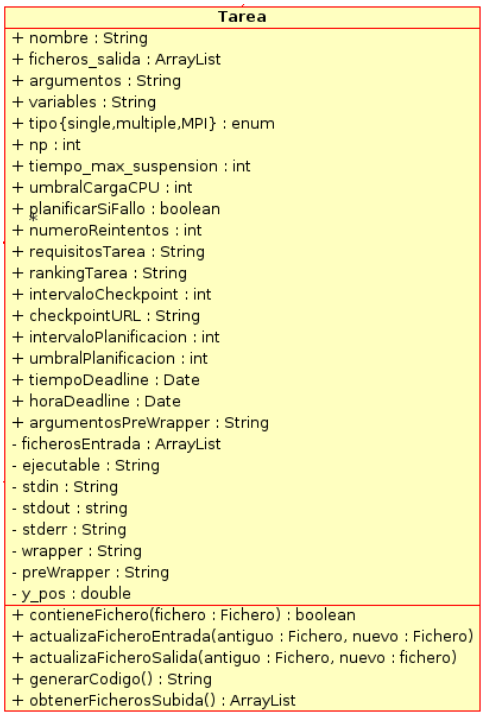


Figura 11.9: Vista de la clase Tarea con sus atributos y métodos.

En la Figura 11.12 puede verse el árbol de flujos de trabajo. Si se seleccionan

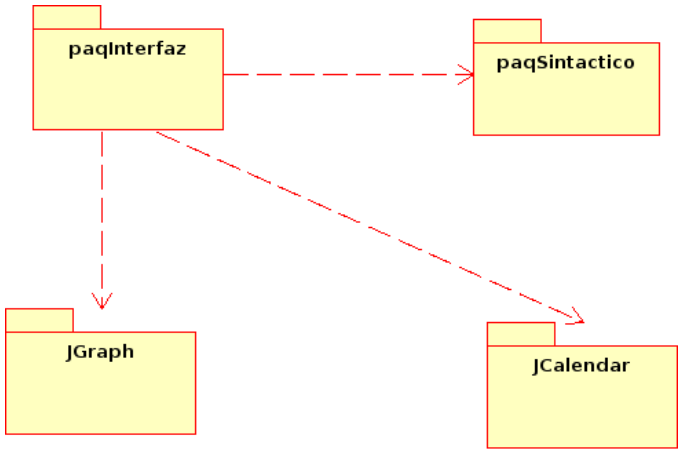


Figura 11.10: Diagrama de paquetes del sistema.

elementos de un workflow, automáticamente en el panel principal se muestra el workflow sobre el que se han seleccionado elementos.

4. **Barra de herramientas**. Aquí se emplazan los botones para la creación y destrucción de tareas y dependencias.

5. **Barra de menús**. En el menú *Archivo* se encuentran las opciones para crear y cerrar los WorkFlows. Además para salir de la aplicación.

11.2.3. Segunda iteración

Requisitos y análisis

Realizando de nuevo en esta segunda iteración la actividad de Elicitación de Requisitos, aparecen nuevos requisitos que no se habían tenido en cuenta en la primera iteración, pero que pueden ser integrados de forma suave a lo que ya había implementado de ésta. Aparecen nuevos casos de uso para el escenario *Gestión de WorkFlows*. En la Figura 11.13 puede verse su diagrama de casos de uso, añadiendo los nuevos casos de uso encontrados en esta segunda iteración.

- **UC-8 Guardar WorkFlow.** El caso de uso comienza cuando el usuario se dispone a guardar un workflow abierto y seleccionado en disco, es decir, almacenar el diseño del workflow en un fichero en disco. El usuario debe dar la localización y el nombre con el que se quiere almacenar el WorkFlow. Como **requisito no funcional**, el fichero debe estar almacenado en un lenguaje portable como XML, el lenguaje de etiquetas que se definirá para almacenar se diseñará de forma exclusiva para la aplicación.

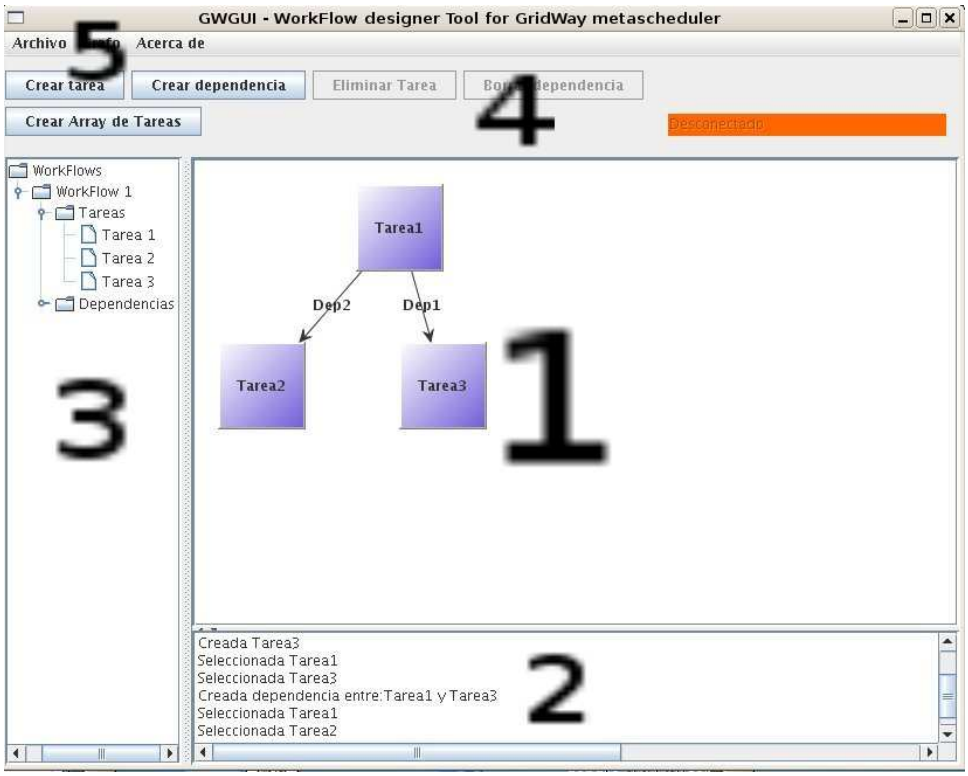


Figura 11.11: Partes de la aplicación.

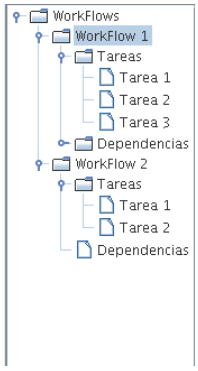


Figura 11.12: Vista del árbol de WorkFlows.

- **UC-9 Abrir WorkFlow.** El caso de uso comienza cuando el usuario se dispone a cargar en la aplicación un workflow previamente almacenado en disco. Se le solicita al usuario la localización exacta de dónde se encuentra éste. Si el fichero no existe, es ilegible o tiene datos inconsistentes se cancelará el caso de uso. De otro modo, se carga el WorkFlow y se muestra por pantalla. Como flujo de ejecución alternativo se prevee comprobar si el nombre del WorkFlow que se carga es idéntico a un WorkFlow abierto actualmente en la aplicación. En ese caso, se le nofiticará al usuario y éste se dispondrá a cambiar el nombre por otro que no haya actualmente en el sistema. Se ejecuta el caso de uso *UC-10 Reordenar WorkFlow*,

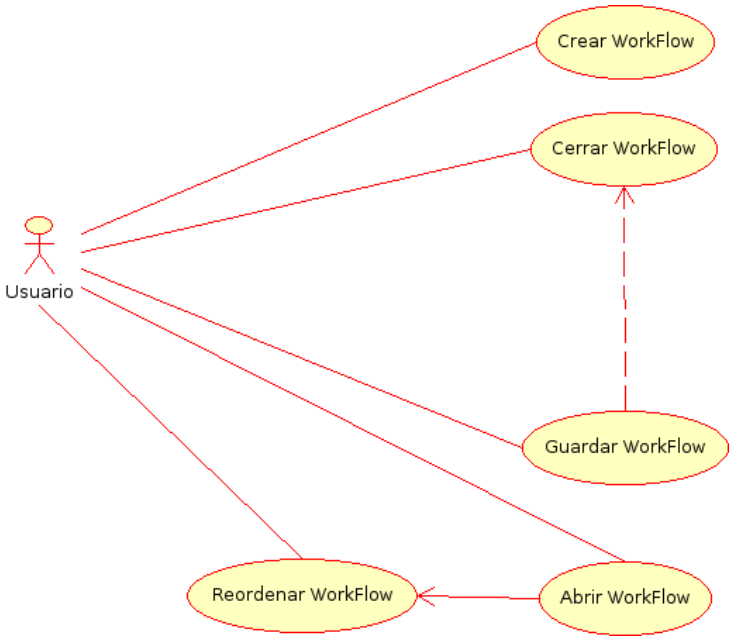


Figura 11.13: Escenario *Gestión de WorkFlows*. Segunda iteración.

que reordena el grafo de tareas y dependencias recién creado. Finalmente el caso de uso termina. El WorkFlow seleccionado será ahora el recién cargado.

- **UC-10 Reordenar WorkFlow**. El caso de uso comienza cuando el usuario se dispone a reordenar un grafo de tareas previamente creado o cuando se ha cargado un workflow desde disco y se prepara para ser visualizado. Las tareas se alinean por niveles. Las tareas que no tienen dependencias son raíces y se colocan en la cima. Las demás tareas van colocándose en niveles por debajo de la raíz. El resultado del caso de uso es un grafo de tareas ordenado.

Al escenario de *Gestión de diseño* se han añadido cinco casos de uso nuevos. Su diagrama de casos de uso puede verse en la Figura 11.14.

- **UC-11 Editar Dependencia.** El caso de uso comienza cuando el usuario se dispone a editar una dependencia. El usuario selecciona la dependencia y mediante un menú contextual elige editarla. Se mostrará al usuario información sobre los ficheros que entrarán a formar parte de la dependencia, es decir, se seleccionarán los ficheros de salida de la tarea de origen que se desea formen parte de los ficheros de entrada de la tarea destino. El caso de uso finaliza y los cambios se aplican en la tarea destino.

- **UC-12 Crear Array de Tareas.** Aparece un nuevo concepto de tarea y es el de *array de trabajos*. El caso de uso comienza cuando el usuario se dispone a crear un nuevo *array de trabajos* dentro de un Workflow previamente seleccionado. El array de trabajos se crea y se muestra en la interfaz gráfica y en árbol lateral, dentro del subárbol del workflow al que pertenece. El caso de uso termina.

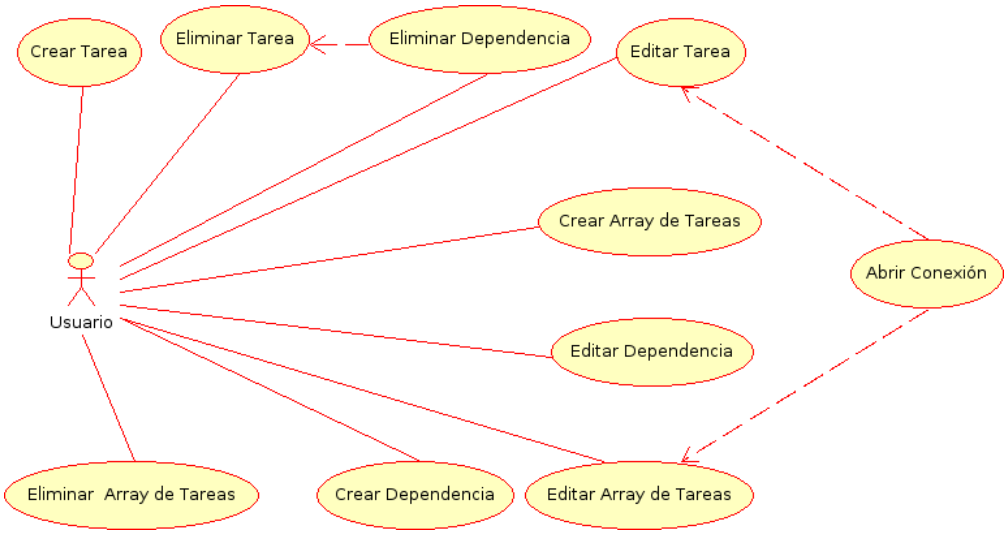


Figura 11.14: Escenario *Gestión de diseño*. Segunda iteración.

- **UC-13 Editar Array de Tareas.** El caso de uso comienza cuando el usuario se dispone a editar un array de tareas previamente seleccionado. Podrá editar todos los parámetros relacionados con las plantillas de trabajo para el metaplanificador GridWay.

- **UC-14 Eliminar Array de Tareas.** El caso de uso comienza cuando el usuario se dispone a eliminar un array de tareas previamente seleccionado. Se le solicitará al usuario una confirmación. Si el usuario acepta el objeto desaparecerá del sistema, en caso contrario el caso de uso finalizará.

Al escenario de *Gestión de trabajos* se han añadido dos casos de uso nuevos. Su diagrama de casos de uso puede verse en la Figura 11.15.

- **UC-16 Enviar Trabajo.** El caso de uso comienza cuando el usuario se dispone a enviar un trabajo al servidor de GridWay, pero no a ejecución, sino subirlo a su sistema de archivos. Se utiliza la funcionalidad proporcionada por el caso de uso *UC-8 Generar Scripts*. Una vez se han creado los scripts, se comprueba si existe una conexión actualmente creada en el sistema que se pueda utilizar. Si no existe ninguna se le notifica al usuario y se ejecuta la funcionalidad del caso de uso *UC-15 Crear conexión*. Si la conexión se ha realizado se envían los ficheros al servidor. Si ocurre algún error en el envío se reintentará un número dado de veces. Si sigue habiendo un error en el envío se le notificará al usuario y el caso de uso terminará deshaciendo todo lo anterior salvo borrar ficheros que se han copiado en el servidor remoto antes de que se produjera el error. Si todo ha ido bien, se le informa al usuario del éxito de la operación y el caso de uso termina correctamente. La conexión creada para el envío permanecerá abierta en el sistema.

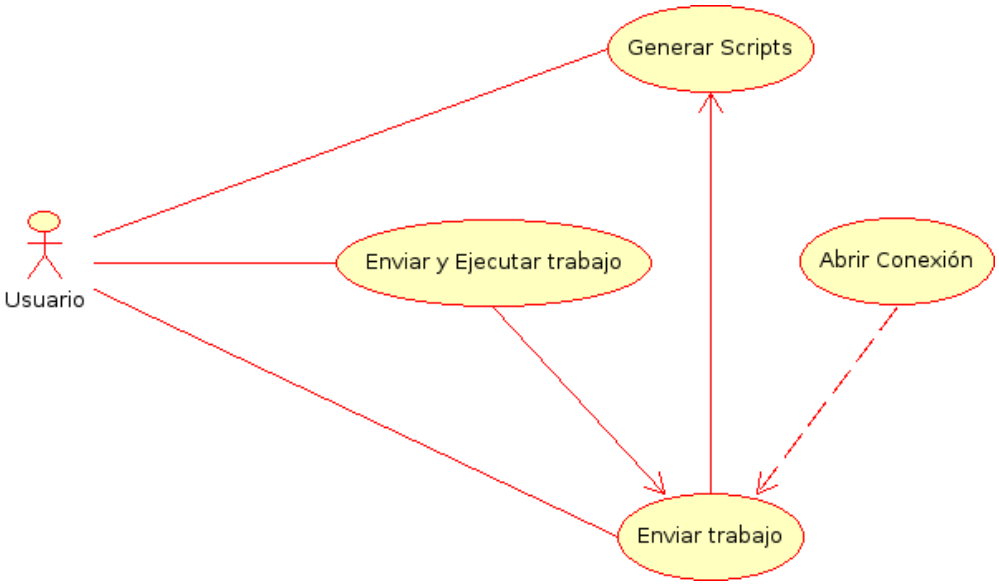


Figura 11.15: Escenario *Gestión de trabajos*. Segunda iteración.

- **UC-17 Enviar y ejecutar Trabajo.** El caso de uso comienza cuando el usuario se dispone a enviar y ejecutar un trabajo en el servidor de GridWay, es decir, cuando se decide ejecutar el array de tareas diseñado por la aplicación. El caso de uso ejecutará la funcionalidad proporcionada por el caso de uso *UC-16 Enviar trabajo*. Si el envío se ha realizado correctamente, se procederá a comprobar el entorno remoto. Se comprobará si las variables de entorno de Globus y GridWay han sido correctamente configuradas, y si no lo están, se le notificará al usuario finalizando el caso de uso. Si las variables de entorno están configuradas se comprobará si existe un proxy para utilizar la arquitectura GSI de Globus en el servidor remoto. Si es así, se comprobará su validez. Si el proxy es válido se procederá dando permisos al script que lanzará el workflow y se ejecutará. Posteriormente se ejecutará un comando de monitorización para las tareas lanzadas por el usuario y se mostrará por pantalla a éste únicamente esa vez. El caso de uso terminará.

Se ha encontrado un nuevo escenario que se llamará *Gestión de conexiones*. El nuevo escenario contendrá dos casos de uso. Puede verse en la Figura 11.16.

- **UC-15 Crear conexión.** El caso de uso comienza cuando el usuario se dispone a crear una conexión SSH al servidor GridWay. O desde los casos de uso UC-13 y UC-4 cuando el usuario desea especificar para la tarea ficheros de entrada que se encuentran subidos en el servidor GridWay, y no en la máquina local donde se está ejecutando la aplicación de diseño. El caso de uso solicita información sobre la conexión: (dirección remota, usuario y contraseña). Si el usuario acepta, se prosigue con la conexión, si cancela, el caso de uso termina sin ningún efecto. Si la conexión prosigue y falla, se muestra el error al usuario, y puede editar de

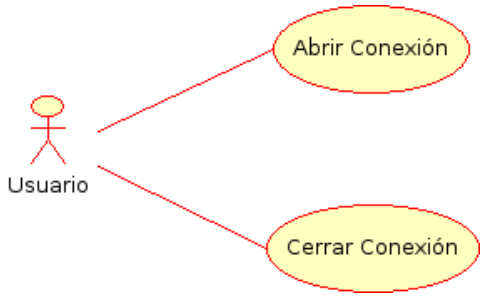


Figura 11.16: Escenario *Gestión de conexiones*.

nuevo la conexión. Una vez que la conexión se establece correctamente se notifica del éxito al usuario y el caso de uso termina.

- *UC-18 Cerrar conexión.* El caso de uso comienza cuando el usuario se dispone a cerrar una conexión abierta previamente. Se le pregunta al usuario por su decisión al respecto. Si decide continuar se cierra la conexión, si no, el caso de uso termina sin efecto.

Diagramas de actividad

En las Figuras 11.17 y 11.18, pueden verse los diagramas de actividad de los casos de uso más representativos descubiertos en esta iteración.

Diseño

A partir de los casos de uso encontrados en esta iteración se generan nuevas clases que se añaden al diagrama de clases de la Figura 11.7. El diagrama de clases resultante puede verse en la Figura 11.20.

Clases nuevas del sistema

- **SecureFTP**. Esta clase implementará la lógica de conexión al servidor GridWay para el envío de ficheros vía SFTP o de comandos por SSH para la ejecución de trabajos y comandos de monitorización de GridWay.
- **DisplayEstadoConexión**. Esta clase de tipo *Thread* creará un hilo de ejecución y monitorizará el estado de la conexión remota, es decir, el estado de la clase *SecureFTP*. El estado de la conexión quedará reflejado en la interfaz gráfica por un indicador que estará en **ROJO** si se está desconectado y en **VERDE** en caso contrario.
- **TareaMultiple**. Esta clase hereda de la clase Tarea. Añade un atributo más a la clase llamado *numeroTareas*, que especifica el número de tareas del array de tareas.

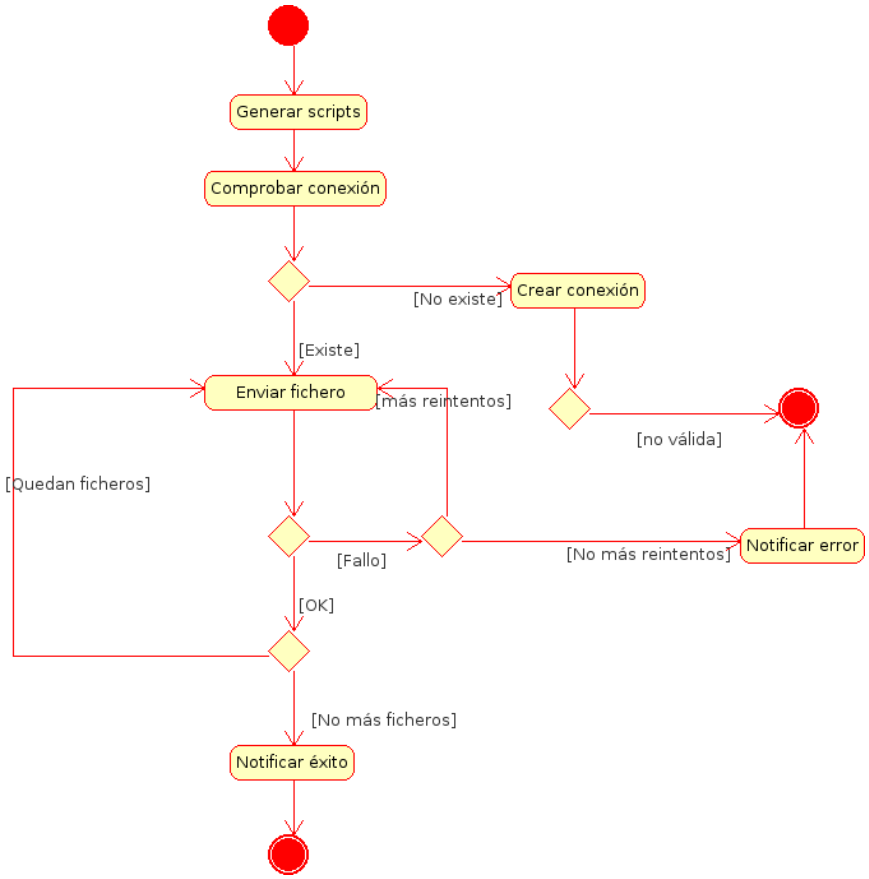


Figura 11.17: Diagrama de actividad de *UC-16 Enviar Trabajo*.

Clases que han sido modificadas

- **Dependencia.** La clase ha añadido una relación con multiplicidad *muchos a muchos* a la clase *fichero*. Esto es debido a que para implementar el caso de uso *editar dependencia* es necesario disponer en dependencia de los ficheros que van influir en la dependencia. De este modo se crea una relación entre estas dos clases: *Dependencia* y *Fichero*.

  En la Figura 11.21 puede verse el diagrama de estados de la clase *SecureFTP*.

Implementación

Se han añadido componentes al diagrama de paquetes del sistema. El diagrama ahora es el que aparece en la Figura 11.22.

- **La biblioteca JDom** ofrece un parser de XML para cargar y almacenar documentos en XML. Es necesario por la implementación de los casos de uso *Cargar* y *Guardar* del escenario *Gestión de WorkFlows*.

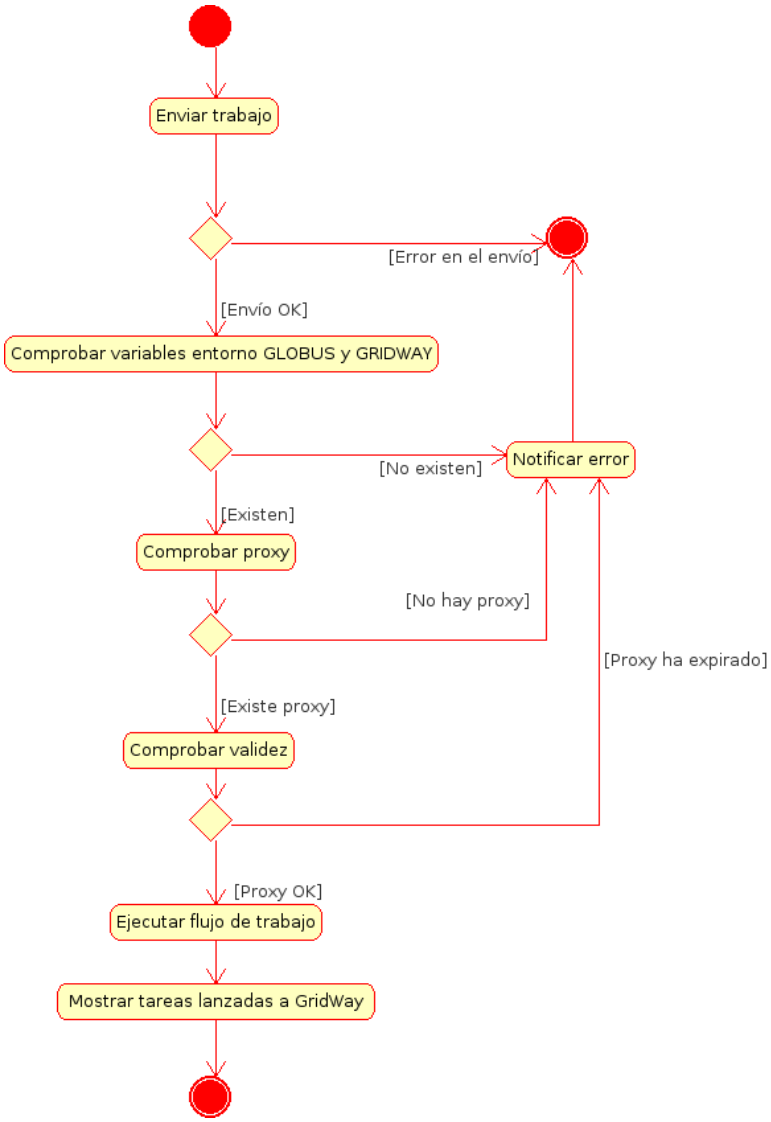


Figura 11.18: Diagrama de actividad de *UC-17 Enviar y Ejecutar Trabajo.*

- El **applet Java sftpApplet** es un componente Java libre para uso no comercial. No se incluye el código fuente pero puede importarse dentro del proyecto y utilizarse. Es un applet desarrollado por la empresa *JScape* [33]. Proporciona una interfaz sencilla para conectarse a un servidor remoto vía SSH y realizar operaciones sobre el sistema de archivos remoto como subir archivos, cambiar de carpeta, borrar, etc... Es necesario por la implementación de los casos de uso de los escenarios *Gestión de conexiones* y *Gestión de trabajos*. Esta biblioteca no permite el envío de comandos SSH.

- **Ganymed SSH-2** es una biblioteca libre con licencia BSD que implementa el protocolo SSH-2 en Java. Funciona con J2SE 1.4.2, 1.3.5 y seguramente con 1.6.X. Permite enviar comandos SSH a un terminal remoto. Es necesaria por la implementación del caso de uso *UC-17 Enviar y ejecutar trabajo.*

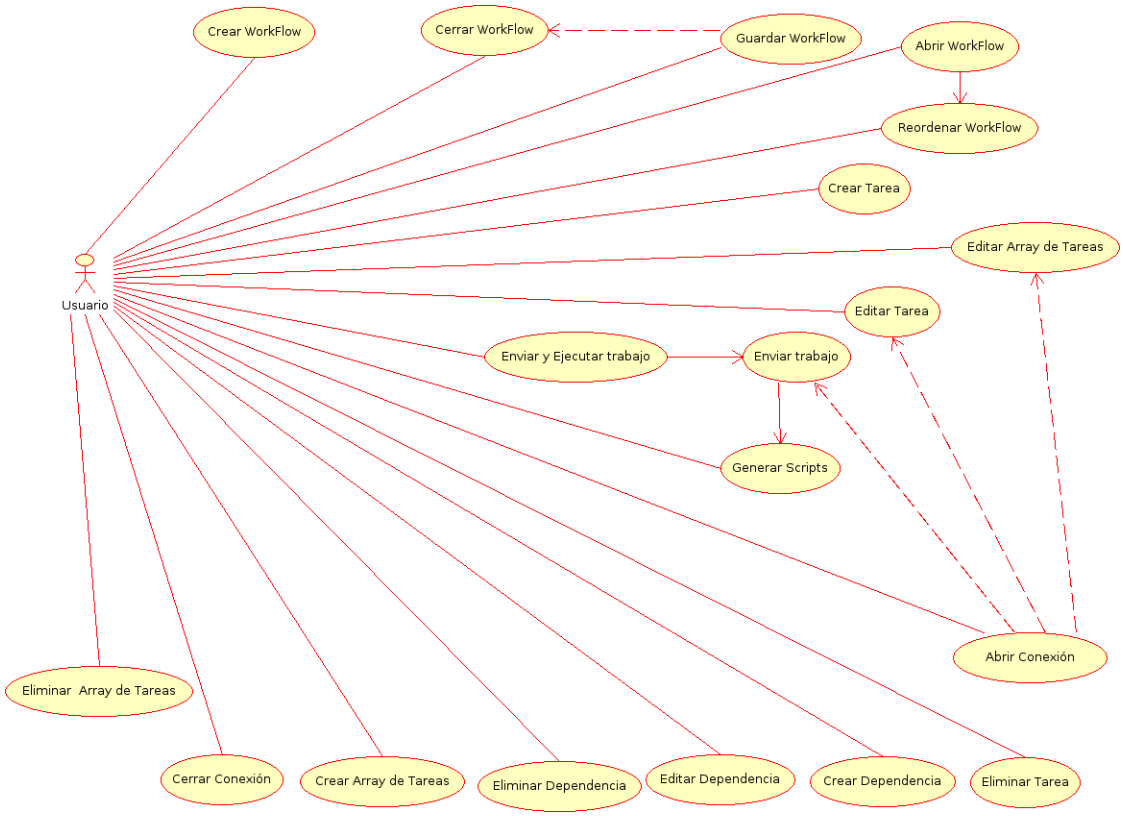


Figura 11.19: Diagrama de casos de uso de la aplicación al completo.

Resultado de la segunda iteración

Se han implementado los casos de uso descubiertos en esta iteración al igual que los casos de uso del nuevo escenario *Gestión de conexión*. El resultado de la iteración es una versión *beta* de la aplicación final. Es usable, pero es posible que aparezcan errores al ser utilizada por el usuario final (ellos siempre encuentran errores). Sobre esta versión, se realizarán las pruebas de aceptación por parte del cliente, que se llevarán a cabo durante el próximo año. De momento la aplicación está lista para ser probada *in situ* en el Grid desplegado en el Grid desplegado en el I3A (véase capítulo 9, apartado 9.7).

11.3. Requisitos Software

La aplicación puede ser ejecutada en cualquier sistema que disponga de la máquina virtual de Java Sun versión 1.5.0 en adelante. En el manual de usuario que se adjunta en el Apendice B se explica como ejecutar la aplicación desde consola.

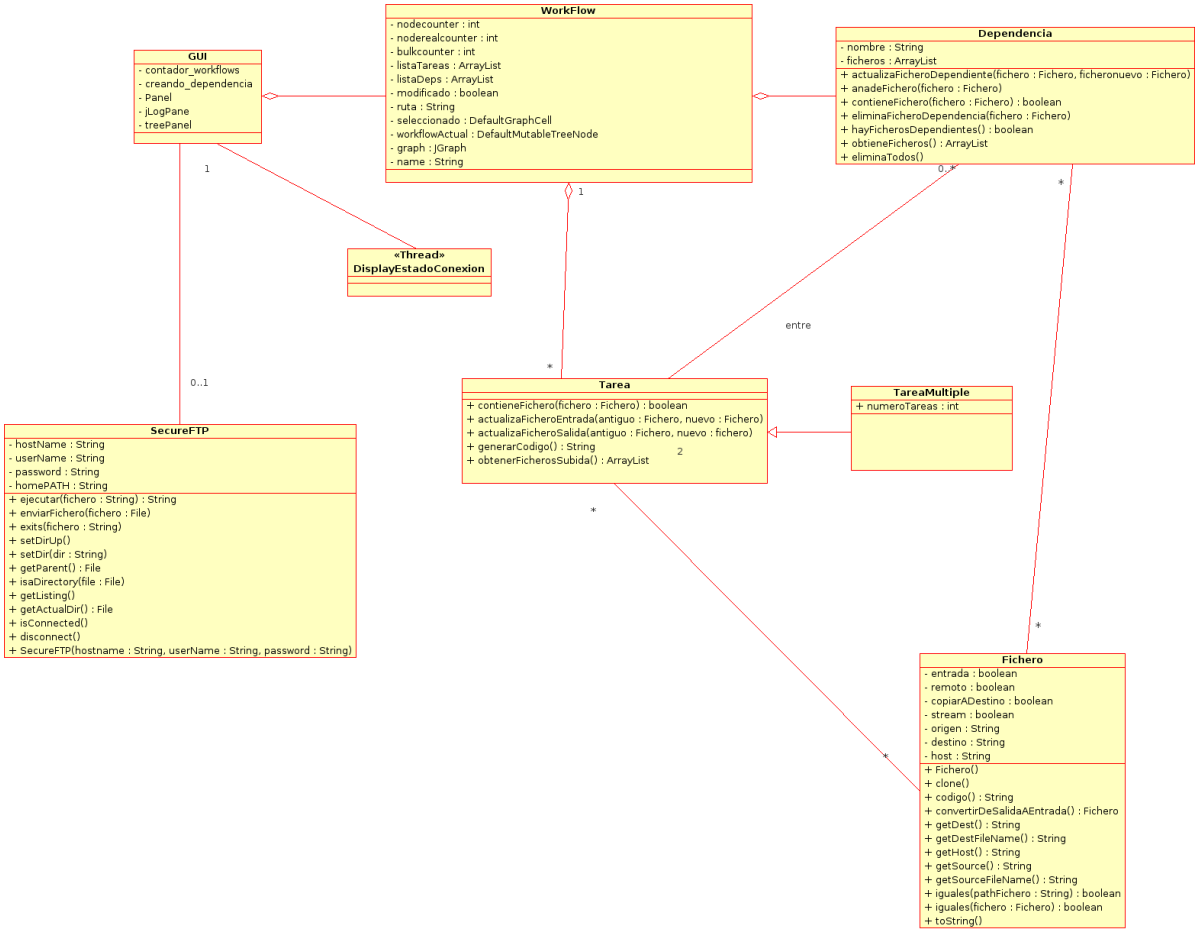


Figura 11.20: Diagrama de clases 2º Iteración.

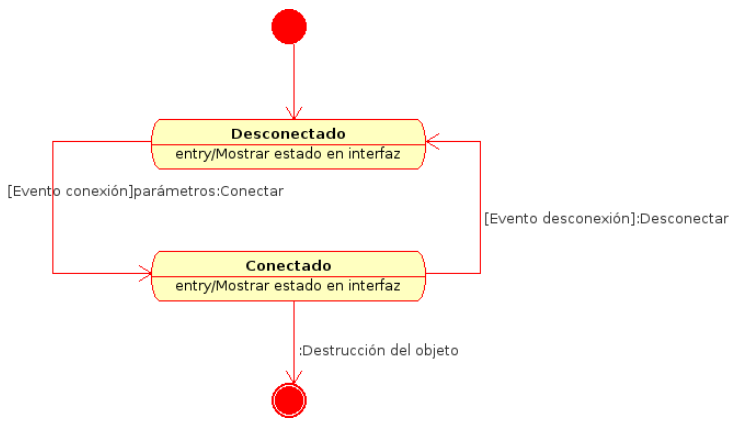


Figura 11.21: Diagrama de estados de la clase *SecureFTP*

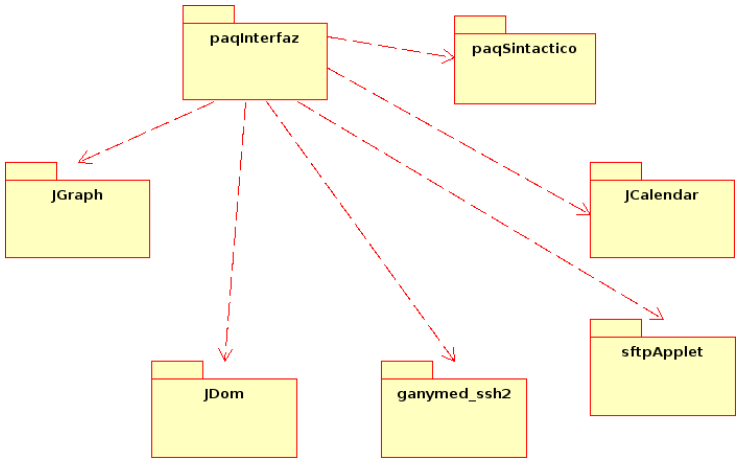


Figura 11.22: Diagrama de paquetes del sistema.

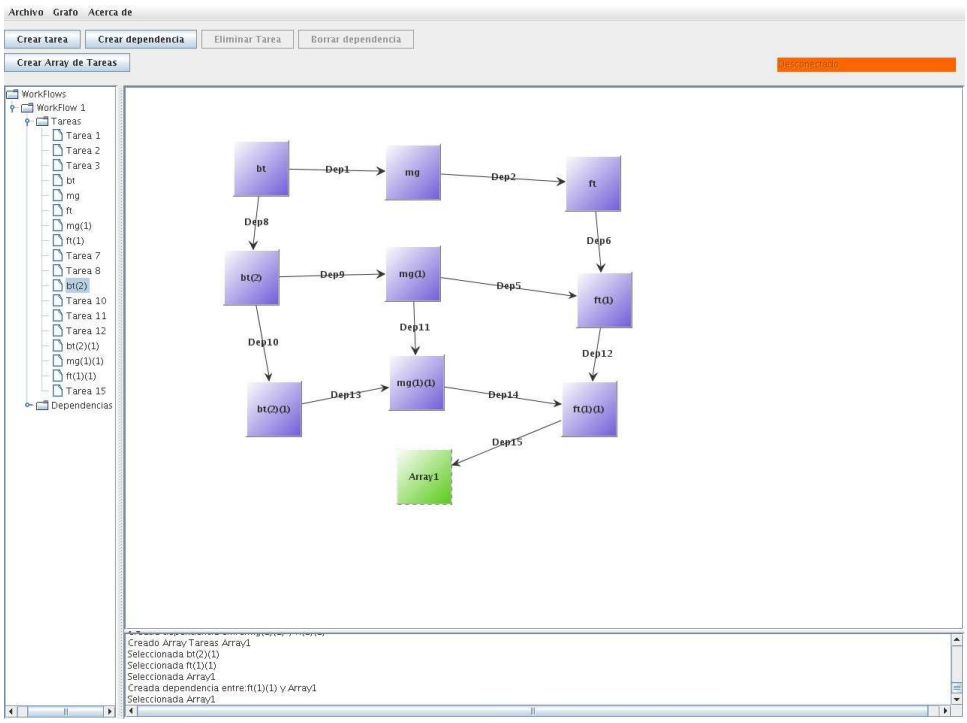


Figura 11.23: Vista de la aplicación principal. Segunda iteración

Capítulo 12

Conclusiones y trabajos futuros

Las conclusiones que pueden sacarse del trabajo realizado, desplegando una plataforma Grid, es que *Globus*, como middleware de capa base de Grid, tienen un alto nivel de sofisticación. Una vez instalado el middleware, y configurado correctamente, el usuario que accede sobre la capa de servicios de Globus, no tiene por qué tener conocimiento del recurso que existe por debajo. La desventaja de Globus, es que, al ser una plataforma de servicios de capa base de Grid, que puede ser instalado en cualquier sistema operativo tipo Unix, es complicado instalarlo y configurarlo en algunos recursos, debido a las versiones o implementaciones de las bibliotecas instaladas en éstos sistemas. Existen versiones ya compiladas de Globus listas para ser instaladas, pero son específicamente para distribuciones Linux concretas, versiones concretas. No se le ofrece al usuario una guía detallada de las bibliotecas que éste necesita, ni las versiones con las que trabaja sobre éstas. Se presuponen muchas cosas, y eso causa muchos problemas durante la instalación, que traen de cabeza al administrador del Grid.

El metaplanificador GridWay, como middleware de usuario, ha cumplido las expectativas proporcionando un entorno fiable para utilizar los recursos Grid. Ha funcionado correctamente durante las pruebas de validación del Grid, y durante las pruebas de rendimiento que se hicieron a éste. Se encargó de dar una visión al usuario final de un superordenador virtual con muchísimos procesadores, en el que el usuario puede lanzar sus tareas (bien definidas), y éstas se ejecutan de manera automática en los recursos que el metaplanificador considera.

Integrando en los recursos locales del metaplanificador del laboratorio RAAP, recursos de los distintos grupos de investigación, de las distintas Universidades, se encontró que GridWay soporta correctamente estos recursos. Es necesario *poner a punto* sus políticas de planificación, para que el uso de estos recursos se limite a cuando se producen saturaciones de los recursos locales. Las latencias de red entre los recursos locales y foráneos son muy diferentes y por lo tanto, los segundos deben tener menos prioridad a la hora de ser escogidos como recursos candidatos para la ejecución de un trabajo. Se han echado en falta en el metaplanificador *GridWay* de agentes de ancho de banda *bandwidth brokers*, como proporciona el metaplanificador *Gridbus*. Estos agentes de ancho de banda analizarían el ancho de banda de los enlaces proporcionando al metaplanificador la implementación de otras políticas de planificación basadas en el ancho de banda del enlace de red hasta el recurso, latencia del enlace, o confiabilidad

en el enlace.

La puerta de acceso al Grid *GridGateWay*, ofrece nuevas posibilidades al despliegue de Grids federados. Los recursos internos de un Grid se *ocultan* detrás de esta interfaz, cuyo acceso al exterior se realiza a través de un servicio Web GRAM de *Globus*. La imagen que se da al exterior es de un único recurso con muchísimos procesadores y muchísima memoria principal, la imagen de un superordenador. Hay que decir, que se han encontrado ciertas deficiencias en esta tecnología de puerta de acceso *GridGateWay*, debido fundamentalmente, a una documentación escasa e ineficaz. Parte de la documentación referencia a scripts de configuración que están mal documentados, además las rutas de estos scripts no son las correctas. En cuanto al modo en el que *GridGateWay* publica su información se ha echado en falta un método más sencillo de configuración, quizás mediante un fichero de configuración con variables a definir, o parámetros de entrada durante el arranque del servicio. Otro problema encontrado, desplegando esta puerta de acceso en los recursos escogidos de cada grupo de investigación, ha sido un problema de realimentación al unir varios *GridGateWays* en un Grid federado. Finalmente el problema se corrigió modificando los script de configuración, que hay que decirlo, fue con ayuda de sus autores, que se quedaron perplejos al descubrir el problema, y prometen introducir una modificación o una adaptación en la próxima versión.

La aplicación desarrollada para el diseño de flujos de trabajo, y su lanzamiento en el Grid, ha cumplido las expectativas de uso, y podría llegar a convertirse en habitual entre los usuarios del Grid desplegado, para enviar trabajos al metaplanificador. Se propone su mejora en futuros Proyectos Fin de Carrera.

Como conclusión final, es que el Grid es una tecnología floreciente, que va a seguir dando mucho de que hablar en los próximos años. Se verá su comportamiento a gran escala en su uso por el proyecto LHC, que entrará en funcionamiento el año próximo, en 2008.

Apéndice A

Pruebas NAS modificadas

A.1. Prueba ED

Para la realización de las pruebas ED se ha utilizado la API DRMAA y la API CLI.

Se ha dividido el script que ejecutaba la prueba de forma no concurrente de tal forma que admite un parámetro que es el número de iteración.

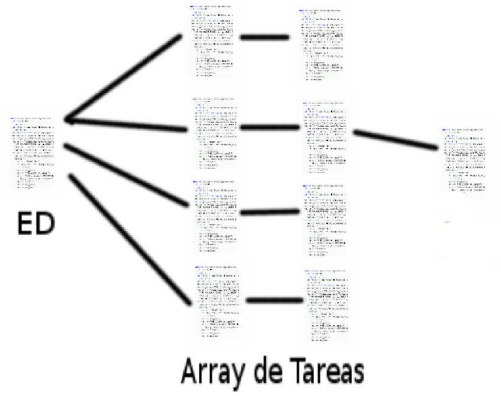


Figura A.1: Descomposición del script ED.

Las nueve tareas de la prueba se lanzan concurrentemente a partir de la misma plantilla de trabajo. Se lanza como array de trabajos de nueve elementos.

```
Array1=`gwsubmit -v -t Array1.jt -n 9| awk '$1 !~ /ARRAY/ {print}' \
| awk '$1 !~ /TASK/ {print}' | awk '{print $2}' |sed '/^$/d' | tr ' \
' ' ' `
```

A.2. Prueba VP

Para la realización de la prueba VP se ha descompuesto el script facilitado por las pruebas *NGP* en tareas diferentes e independientes que se ejecutan de forma concurrente aunque con dependencias entre unas y otras. Los ficheros que generaban una y otra deben ser establecidos como ficheros de entrada y salida para las tareas que se definen. Esta prueba se ha implementado en Java y por código CLI de GridWay.

```
#!/bin/sh

BT_12345=`gwsubmit -v -t  BT.jt | cut -f2 -d ':'`
BTw1x_12345=`gwsubmit -v -t BTw1x.jt -d "$BT_12345" | cut -f2 -d ':' `
BTw1xw1x_12345=`gwsubmit -v -t BTw1xw1x.jt -d "$BTw1x_12345" | cut -f2 -d ':' `
MG_12345=`gwsubmit -v -t MG.jt -d "$BT_12345" | cut -f2 -d ':' `
FT_12345=`gwsubmit -v -t FT.jt -d "$MG_12345" | cut -f2 -d ':' `
MGw1x_12345=`gwsubmit -v -t MGw1x.jt -d "$BTw1x_12345 $FT_12345" | cut -f2 -d ':' `
FTw1x_12345=`gwsubmit -v -t FTw1x.jt -d "$FT_12345 $MGw1x_12345" | cut -f2 -d ':' `
MGw1xw1x_12345=`gwsubmit -v -t MGw1xw1x.jt -d "$BTw1xw1x_12345" "$FTw1x_12345" | cut -f2 -d ':' `
FTw1xw1x_12345=`gwsubmit -v -t FTw1xw1x.jt -d "$FTw1x_12345 $MGw1xw1x_12345" | cut -f2 -d ':' `
```

Figura A.2: Código script de la prueba VP.

En la Figura A.2 puede verse el programa script que lanza esta prueba en GridWay. Éste código ha sido generado mediante la aplicación *GWGUI*. El código Java que ejecuta la prueba se encuentra en el CD-ROM adjunto a éste documento, al igual que el proyecto GWGUI para esta prueba.

Apéndice B

Manual de usuario de GWGUI

B.1. Descripción

La aplicación ha sido desarrollada con fines educativos dentro del contexto de un Proyecto Fin de Carrera, con el objetivo de disponer de una interfaz de usuario capaz de interactuar con el metaplanificador GridWay a un nivel básico, es decir, a través de su lenguaje específico de línea de comandos y no través de la API DRMAA de programación. Se pretende partir de un diseño sencillo para facilitar al usuario final del Grid el diseño de los scripts que se utilizarán para interactuar con el metaplanificador, y así como proporcionarle una herramienta visual para trabajar en el campo de los flujos de trabajo y matrices de tareas. En los siguientes apartados se explicarán las distintas partes de la aplicación.

B.2. Requisitos

Se necesita únicamente un sistema operativo con la máquina virtual de Java Sun 1.5.0 o superior. En sistemas operativos Windows no permite seleccionar ficheros locales para la edición de tareas, debido a que las forma de definir ficheros y rutas es distinta en Windows y en Linux.

B.3. Pantalla principal

En la Figura B.1 puede observarse la pantalla principal, que se divide en varias partes:

1. **Panel principal** donde se desarrollarán los diseños de flujo de trabajo. Será posible crear, seleccionar y borrar tareas y dependencias.
2. **Panel de log** donde se visualizarán en modo texto todas las acciones realizadas sobre la interfaz gráfica dando en caso de fallo, una herramienta de trazabilidad de errores.

3. **Panel lateral** visualiza de forma jerárquica todos los flujos de trabajo abiertos, y sus elementos: tareas y dependencias.

4. **Barra de herramientas** incluye botones para crear, y borrar tareas, dependencias y matrices de tareas.

5. **Barra de menús** incluye dos menús que opciones para abrir, guardar y cerrar flujos de trabajo; crear y destruir conexiones, y para salir de la aplicación. Además se ofrece la opción de reordenar el flujo de trabajos para que sea más inteligible por el usuario.

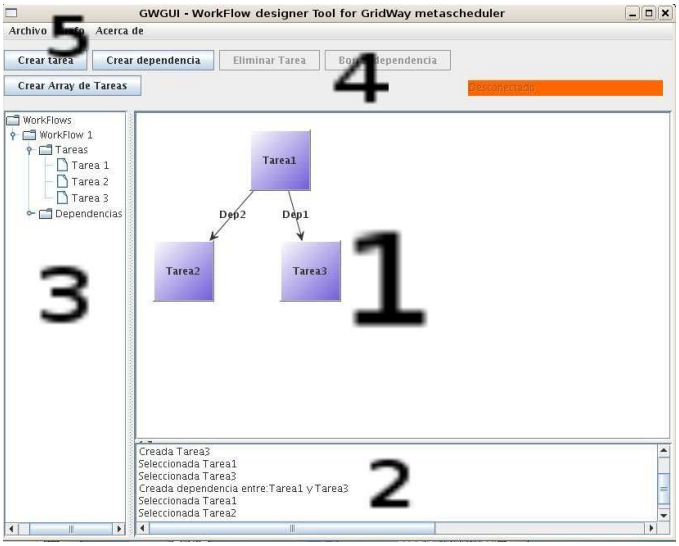


Figura B.1: Secciones de la pantalla principal.

En la Figura B.2 puede verse el menú *Archivo* desplegado.

Figura B.2: Menu Archivo.

B.4. Menu Archivo

B.4.1. Crear nuevo WorkFlow

Se creará un nuevo flujo de trabajo y será representando en el panel lateral y en el principal. Este nuevo flujo de trabajo pasará a tener el foco de la aplicación.

B.4.2. Abrir WorkFlow

Se mostrará una explorador de archivos que permitirá al usuario seleccionar un fichero que contenga definidos flujos de trabajo, previamente creado por la aplicación. El fichero puede ser almacenado en cualquier extensión, y por lo tanto, no se establece ninguna política de filtrado a la hora de visualizar los ficheros del sistema de archivos.

Si el fichero no era válido, o tenía información incompleta, se mostrará un error por pantalla y el flujo de trabajo no se cargará.

B.4.3. Cerrar WorkFlow

Se cerrará el flujo de trabajo seleccionado actualmente. Si éste ha sido modificado, se preguntará al usuario si desea guardarlo.

B.4.4. Guardar y Guardar Como

Se almacenará el flujo de trabajo seleccionado actualmente en la localización especificada por el usuario con el nombre indicado.

B.4.5. Conectar y Desconectar

El usuario podrá crear una conexión estable con un terminal remoto. Esta conexión se realizará por SSH, y se utilizará para enviar los trabajos definidos al nodo remoto. En la Figura B.3 puede verse la edición de los parámetros de conexión.

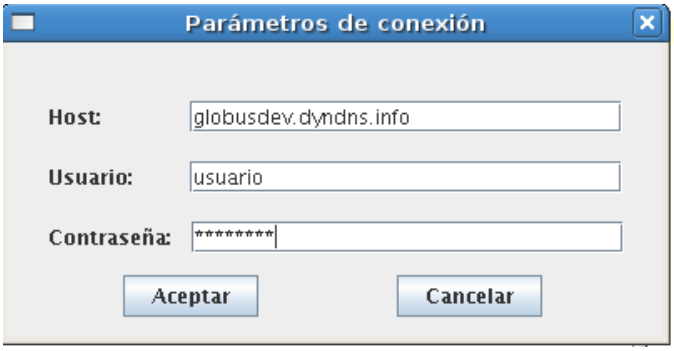


Figura B.3: Edición de parámetros de conexión

B.4.6. Generar scripts

El usuario podrá crear generar los ficheros de ordenes para *GridWay*. Se generarán las plantillas de trabajo para todas las tareas del flujo de trabajo seleccionado.

B.4.7. Enviar trabajo

Permite generar y enviar los scripts y plantillas de trabajos al nodo remoto donde se encuentra GridWay. Se copiarán todos los ficheros que hayan sido definidos de forma local a la máquina del usuario.

B.4.8. Enviar y ejecutar trabajo

El usuario podrá generar, enviar y ejecutar su trabajo de forma remota en el servidor de GridWay. Se comprobará si las variables de entorno están bien definidas en el servidor remoto, con el fin de determinar si está instalado *GridWay*.

B.5. Menu Grafo

Tiene la opción de *reordenar* el grafo de tal forma que sea fácilmente inteligible por el usuario.

B.6. Edición de Tarea

Se realizando doble click sobre una tarea en el panel principal, o mediante la selección de la opción editar desde el menú contextual, al seleccionar una tarea, se muestra una cuadro de diálogo en el que se pueden configurar todos los aspectos de la tarea, desde el punto de vista de las plantillas de trabajo de *GridWay*.

B.6.1. Pestaña entorno

Permite especificar el nombre del ejecutable, argumentos y demás información relacionada con el ejecutable del trabajo. En la Figura B.4 se muestra la pestaña *Entorno*.

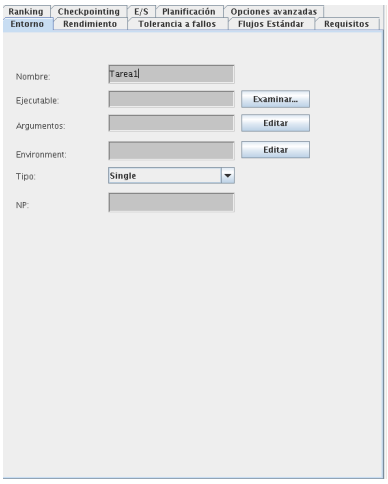


Figura B.4: Pestaña *Entorno*

B.6.2. Pestaña rendimiento

Permite especificar parámetros de la tarea relacionados con el rendimiento. Se utiliza para la migración de tareas. En la Figura B.5 se muestra la pestaña *Rendimiento*.

B.6.3. Pestaña tolerancia a fallos

Permite configurar si la tarea será tolerante a fallos, es decir, si se produce un fallo en un recurso se enviará a otro recurso. En la Figura B.6 se muestra la pestaña

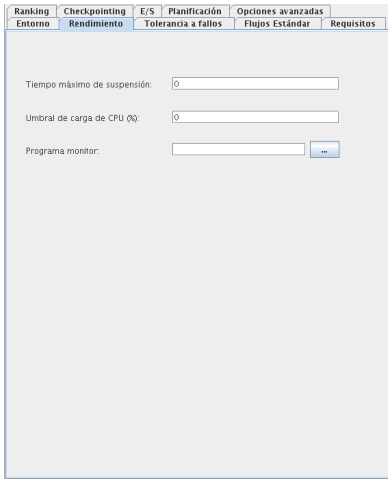


Figura B.5: Pestaña *Rendimiento*

Tolerancia a fallos.

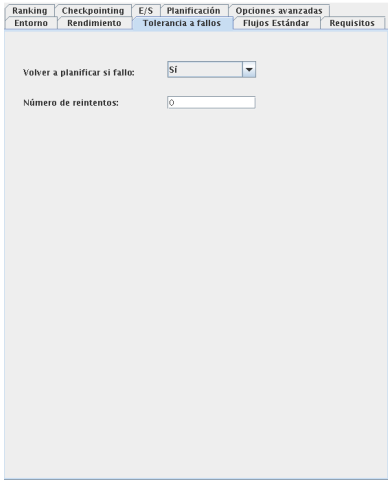


Figura B.6: Pestaña *Tolerancia a fallos*

B.6.4. Pestaña flujos estándar

Se permite configurar los ficheros que servirán como entrada estándar al ejecutable. Tanto de salida como de entrada, y salida de error. En la Figura B.7 se muestra la pestaña *Flujos estándar*.

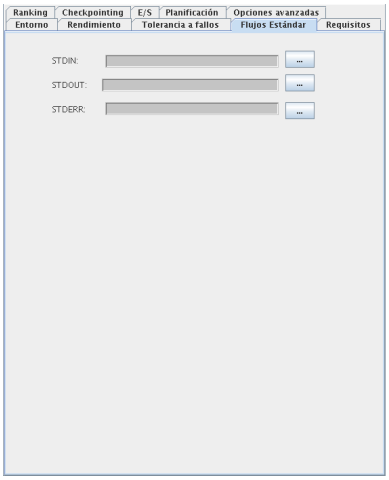


Figura B.7: Pestaña *Flujos estándar*

B.6.5. Pestaña Requisitos

Permite configurar los requisitos del trabajo en *GridWay*. Se incluye un analizador sintáctico que analiza el formato de los requisitos introducidos. En la Figura B.8 se muestra la pestaña *Requisitos*.

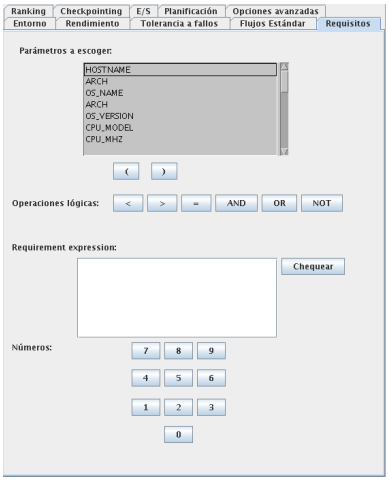


Figura B.8: Pestaña *Requisitos*

B.6.6. Pestaña de Ranking

Permite configurar los recursos más apetecibles por la tarea que se está definiendo, es decir, *GridWay* utilizará este ranking para reordenar la lista de recursos por

los que compite una tarea. Se incluye un analizador sintáctico para facilitar al usuario la edición de la cadena de *ranking*. En la Figura B.9 se muestra la pestaña *Ranking*.

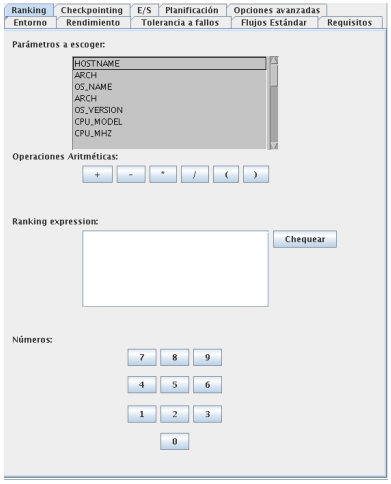


Figura B.9: Pestaña *Ranking*

B.6.7. Pestaña de Checkpointing

Permite configurar la localización de un fichero de *checkpoints*, en el que se crearán capturas de la ejecución del trabajo. En la Figura B.10 se muestra la pestaña *Checkpointing*.

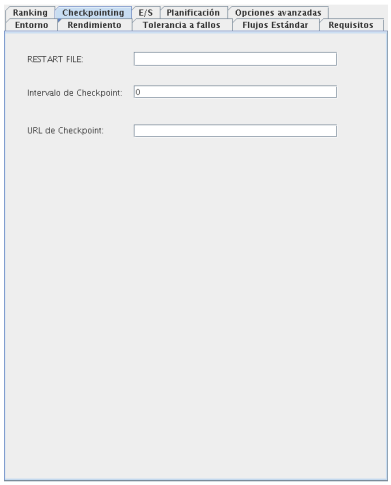


Figura B.10: Pestaña *Checkpointing*

B.6.8. Pestaña de E/S

Permite definir los ficheros que servirán como entrada a la tarea. Los ficheros de entrada podrán ser escogidos del sistema de archivos local del cliente, o del sistema de archivos remoto del servidor con GridWay, para esto, se solicitará al usuario crear una conexión con el servidor. Los ficheros de salida se definen siempre en el servidor remoto, ya que es allí donde se copiarán una vez se ejecute el trabajo en GridWay. En la Figura B.11 se muestra la pestaña *E/S*.

Figura B.11: Pestaña *E/S*

Durante la edición de ficheros de entrada y salida, se le notifica al usuario si esta realizando acciones que modifican ficheros que intervienen en las dependencias. En la Figura B.12 se el cuadro de diálogo para editar ficheros de entrada.

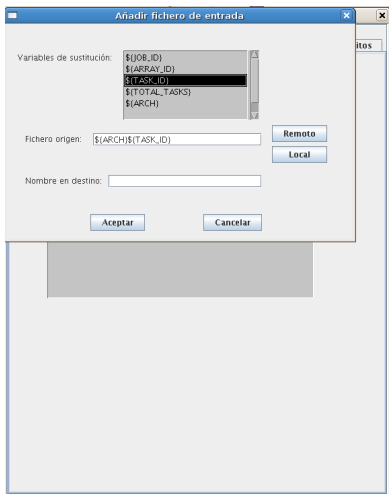


Figura B.12: Editor de ficheros de entrada

B.6.9. Pestaña de Planificación

Permite establecer la fecha de deadline y el intervalo de planificación en caso de migración de tarea. En la Figura B.13 se muestra la pestaña *Planificación*.

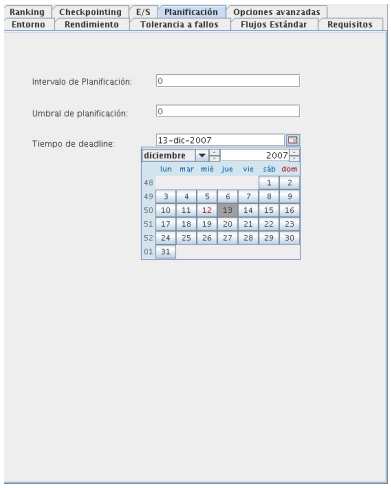


Figura B.13: Pestaña de *Planificación*

B.6.10. Opciones avanzadas

Permite establecer la utilización de un nuevo wrapper, y de un script de monitorización especial para la tarea. En la Figura B.14 se muestra la pestaña *Opciones avanzadas*.

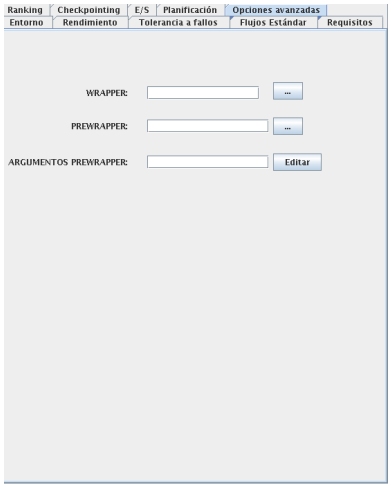


Figura B.14: Pestaña *Opciones avanzadas*

B.6.11. Pestaña de array de trabajos

Sólo aparece en las opciones de configuración de una tarea múltiple. Permite establecer el número de trabajos que tendrá la matriz de trabajos. En la Figura B.15 se muestra la pestaña *Array de trabajos*.

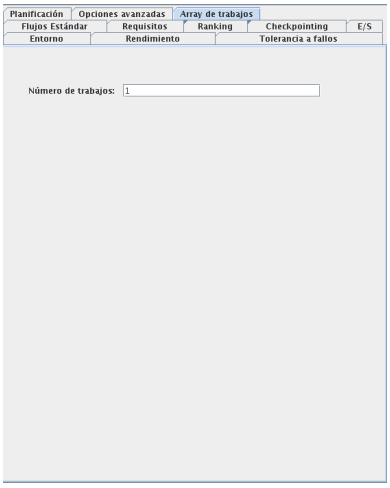


Figura B.15: Pestaña *Array de trabajos*

B.7. Edición de dependencias

Se muestra al usuario la información de los ficheros que van a participar en la dependencia, es decir, los ficheros de salida de la primera tarea que son necesario por la entrada en la segunda tarea. En la Figura B.16, puede verse el editor de dependencias que se muestra cuando se selecciona editar desde el menú contextual, al seleccionar una dependencia.

B.8. Ejecución

Desde consola hay que invocar a *Java* pasando como parámetro fichero *.jar*. Es necesario que la carpeta lib esté en el mismo directorio que el fichero *.jar*, ya que esta carpeta contiene las bibliotecas que necesita la aplicación Java principal. Para arrancar el programa ejecutar desde consola:

java -jar gwgui.jar

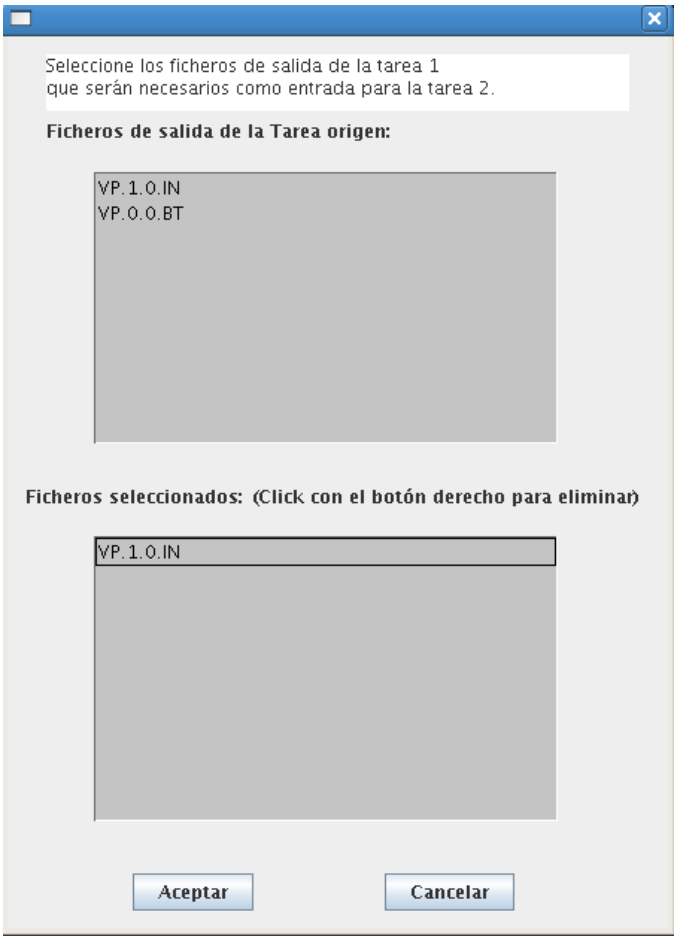


Figura B.16: Pantalla de edición de dependencias

Bibliografía

[1] Alchemi - Plug & Play Desktop Grid Computing. http://www.alchemi.net/.

[2] An Application Grid Laboratory for Science. http://www.ivdgl.org/grid2003/.

[3] APAC National Grid: Providing grid services for Australian eResearch. http://grid.apac.edu.au/.

[4] apt-get.org: Unofficial APT repositories. http://www.apt-get.org/.

[5] Centro Nacional de Supercomputación. http://www.bsc.es/.

[6] Climate Change Experiment. http://bbc.cpdn.org/.

[7] Cluster Resources. http://www.clusterresources.com/.

[8] Como mantener la hora exacta mediante el protocolo NTP. http://www.oxixares.com/ gbv/hora.html/.

[9] Cosmology at HOME. http://www.cosmologyathome.org/.

[10] CrossGrid project. http://www.crossgrid.org/.

[11] Distributed Resource Management Application API C Bindings v1.0.

[12] Distributed Resource Management Application API JavaTM Language Bindings 1.0.

[13] DRMAA compliance testsuite. http://www.drmaa.org/wiki/index.php?pagename=DrmaaTestsuite/.

[14] DRMAA Wiki. http://drmaa.org/wiki/.

[15] Einstein at HOME. http://einstein.phys.uwm.edu/.

[16] Extreme Programming:A gentle introduction. http://www.extremeprogramming.org/.

[17] Folding At HOME. http://folding.stanford.edu/.

[18] Gfarm file system. http://datafarm.apgrid.org/.

[19] GridBus Broker. http://gridbus.org/broker/.

[20] Grid For Utility. http://www.grid4utility.org/.

[21] Grid(Lab) Resource Management. http://www.gridlab.org/WorkPackages/wp-9/.

[22] GridLab technologies. http://www.gridlab.org/.

[23] Grid Scheduling Architecture. http://forge.gridforum.org/projects/gsa-rg/.

[24] GRID Superscalar.

[25] Guía rápida de Globus.
http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html/.

[26] Herramienta de Monitorizacián Ganglia. http://ganglia.sourceforge.net/.

[27] High Throughput Computing. http://www.cs.wisc.edu/condor/.

[28] Http Server Project. http://httpd.apache.org/.

[29] Infraestructura Abierta de Berkeley para Computación en Red.
http://boinc.berkeley.edu/.

[30] Instalacián y configuracián de Ganglia en Debian. http://wiki.lsc.dc.uba.ar/index.php/.

[31] JCalendar. http://www.toedter.com/en/jcalendar/.

[32] JGraph - The Java Open Source Graph Drawing Component.
http://www.jgraph.com/jgraph.html.

[33] JSCAPE SFTP APPLET. http://www.jscape.com/sftpapplet/index.html.

[34] K*Grid. http://www.gridcenter.or.kr/.

[35] Large Hadron Collider. http://lhc.web.cern.ch/lhc/.

[36] Lecture notes – Globus: The Grid Programming Toolkit.
http://www.cs.ucsb.edu/ rich/class/cs290I-grid/notes/Globus/index.html.

[37] Lenguaje de modelado unificado. http://www.uml.org/.

[38] Metascheduling Technologies for the Grid. http://www.gridway.org/.

[39] Middleware gLite. http://glite.web.cern.ch/glite/.

[40] Moab Workload Manager.

[41] National Research Grid Iniciative. http://www.naregi.org/.

[42] NETBEANS IDE. http://www.netbeans.org/.

[43] Nimrod: Tools for Distributed Parametric Modelling.
http://www.csse.monash.edu.au/ davida/nimrod/.

[44] Ninf: A Global Computing Infrastructure. http://ninf.apgrid.org/.

[45] Open Grid Forum. http://www.ogf.org/.

[46] Página oficial del proyecto Globus. http://www.globus.org/.

[47] Página web de EGEE. http://public.eu-egee.org/.

[48] Página web de IRISGrid. http://www.irisgrid.es/.

[49] RAAP website. http://www.i3a.uclm.es/.

[50] RATIONAL UNIFIED PROCESS. http://www.306.ibm.com/software/awdtools/rup/.

[51] Render Grid. http://rendergrid.net/.

[52] RRDtool. http://oss.oetiker.ch/rrdtool/.

[53] Second Life: Your World, your Imagination. http://secondlife.com/.

[54] Seti at Home.
http://setiathome.berkeley.edu/.

[55] TeraGrid Project. http://www.teragrid.org/.

[56] The Grid Computing and Distributed Systems (GRIDS) Laboratory, University of Melbourne. http://www.gridbus.org/.

[57] The Open Grid Services Architecture. http://www.globus.org/ogsa/.

[58] The largest Grid Experiment on Climate Prediction.
http://www.climateprediction.net/.

[59] The NAS Parallel Benchmarks (NPB).
http://www.nas.nasa.gov/Resources/Software/npb.html.

[60] Uniform Interface to Computing Resources. http://www.unicore.org/.

[61] Uniform Interface to Computing Resources. http://sourceforge.net/projects/gcsf/.

[62] Universidad de California. http://berkeley.edu/.

[63] Universidad Complutense de Madrid. http://www.ucm.es/.

[64] Universidad de Castilla-La Mancha. http://www.uclm.es/.

[65] Universidad de Murcia. http://www.um.es/.

[66] World of WarCraft Europe. http://www.wow-europe.com/es/index.xml.

[67] P. van Oorschot y S. Vanstone. A. Menezes. *Handbook of Applied Cryptography.* CRC Press, 1996.

[68] John Bresnahan Ann L. Chervenak Ian Foster Carl Kesselman Sam Meder Veronika Nefedova Darcy Quesnel Steven Tuecke Bill Allcock, Joe Bester. *Data Management and Transfer in High-Performance Computational Grid Environments.* Argonne National Laboratory, University of Southern California, Department of Computer Science & The Computation Institute The University of Chicago, 2001.

[69] Luis Tomás Bolívar. Evaluación de una plataforma grid mediante benchmarks específicos. *Proyecto Fin de Carrera Escuela Politécnica Superior de Albacete, Universidad de Castilla-La Mancha*, 2007.

[70] R. F. Van der Wijngaart y M. A. Frumkin. *NAS Grid Benchmarks Version 1.0*. NASA Advanced Supercomputing (NAS), NASA Ames Research Center, Moffett Field, 2002.

[71] Marian Bubak, Wlodzimierz Funika1 , and Rosa M. Badia. Performance evaluation of grid superscalar applications with ocm-g/g-pm. 2007.

[72] Marc de Palol Jorge Ejarque Jesás Labarta Josep M. Párez Raál Sirvent Rosa M. Badia, Pieter Bellens and Enric Tejedor. Grid superscalar: From the grid to the cell processor. 2007.

[73] Rubán S. Montero y Ignacio M. Llorente Tino Vázquez, Eduardo Huedo. Evaluation of a utility computing model based on the federation of grid infrastructures. *Euro-Par 2007 Parallel Processing*, pages 372–381, 2007.