Jesús María Latorre Canteli
Ingeniero Industrial del ICAI (2001) y Doctor Ingeniero Industrial del ICAI (2007). Es investigador en el Instituto de Investigación Tecnológica.

Rafael Palacios Hielscher
Ingeniero Industrial del ICAI (1990) y Doctor Ingeniero (1998). Es investigador del Instituto de Investigación Tecnológica y profesor propio del Departamento de Sistemas Informáticos. Imparte clases de programación y de seguridad informática en las titulaciones de Ingeniero Industrial y de Ingeniero en Informática del ICAI.

Andrés Ramos Galán
Ingeniero Industrial del ICAI (1982) y Doctor Ingeniero Industrial por la UPM (1990). Es Profesor Propio Ordinario y Director del Departamento de Organización Industrial de la ETS de Ingeniería (ICAI) de la Universidad Pontificia Comillas.

Comentarios a:
comentarios@icai.es

Entornos Grid. Sistemas distribuidos para cálculo masivo

Durante los últimos años, se ha producido un gran avance dentro del cálculo distribuido, de forma análoga a la evolución de la informática en general. El aumento de la capacidad de cálculo de los equipos de tipo personal y el auge de las comunicaciones digitales, ha llevado al desarrollo de los entornos grid como alternativa a los superordenadores, más potentes y eficientes aunque más costosos. En este artículo se presentan los entornos grid, encuadrándolos en primer lugar dentro de las clasificaciones de las arquitecturas de ordenadores. Con ese punto de partida se procede a discutir las características principales y la organización de los entornos grid. Posteriormente, se ofrece una panorámica de los proyectos más relevantes en este campo en constante avance.

Entornos grid

En el camino para obtener rendimientos más altos de los equipos informáticos pueden seguirse dos vías. Por un lado, pueden buscarse procesadores más rápidos individualmente. Éste es el camino seguido hasta hace poco por las principales compañías de fabricación de procesadores. La otra vía consiste en interconectar gran cantidad de equipos actuando en paralelo. Mediante la acumulación de recursos se pueden obtener incrementos considerables si se consigue coordinar la operación de los mismos. Dentro de esta opción están los procesadores con varios núcleos y los entornos grid, entre otros.

En este último caso, se habla de sistemas dispersos geográficamente formados por gran cantidad de equipos de diferentes características. En ocasiones se trata de grid diseñados específicamente con ese fin, que pueden denominarse grid dedicados. Sin embargo, no son infrecuentes los casos en que la infraestructura *hardware* no fue diseñada con el objetivo de convertirse en un grid. En lugar de eso, se aprovechan recursos ya existentes que están infrautilizados, para permitir que otros participantes en el grid empleen esos recursos. Un ejemplo de este tipo de sistemas que alcanzó gran popularidad, y demostró las posibilidades de estos enfoques, es el proyecto SETI@home [Korpela2001]. Este proyecto emplea la potencia de cálculo desaprovechada en millones de ordenadores para analizar las señales de radio recibidas en el radio-telescopio de Arecibo (Puerto Rico) en busca de señales con origen en vida inteligente.

Taxonomía de arquitecturas de ordenadores

Para ayudar a entender qué tipo de sistemas son considerados entornos grid, se va a comenzar revisando cuál es la arquitectura de dichos sistemas. Existe una gran variedad de arquitecturas con las que se diseñan los ordenadores, desde los equipos monoprocesador hasta los entornos grid, pasando por los *clusters* y los equipos multiprocesador. De entre las clasificaciones de las arquitecturas de ordenador, la más popular es la propuesta por Michael J. Flynn [Flynn1972]. En este esquema se catalogan los sistemas informáticos atendiendo a los dos flujos de información principales que intervienen en la operación de los mismos: el flujo de las instrucciones y el de los datos. El primero de ellos es empleado por la unidad de control del sistema (que puede ser única o múltiple) para dirigir la operación del procesador o los procesadores sobre el segundo de los flujos, el de los datos. En función de si esos flujos de información son únicos o múltiples, se pueden establecer las cuatro categorías que se describen a continuación.

1. ARQUITECTURA SIDS (SINGLE INSTRUCTION SINGLE DATA)

En esta categoría se encuadran las arquitecturas con un único flujo de instrucciones y otro flujo único de datos. Esta arquitectura es la empleada por todos los ordenadores monoprocesador, que se basan en la máquina propuesta en 1945 por John von Neumann y su equipo, que comprende los siguientes componentes:

- Una unidad de control central que se encarga de organizar el funcionamiento del resto del sistema para que se ejecuten las instrucciones.
- Una unidad de proceso encargada de llevar a cabo las instrucciones aritméticas y lógicas que constituyen el núcleo central del funcionamiento del ordenador.
- Una memoria que almacene tanto las instrucciones como los datos sobre los que se debe operar.
- Por último, los tres elementos anteriores deben intercambiar información con el exterior, tarea de la que se encargan los dispositivos de entrada y salida.

Este esquema teórico se ha complementado con el paso de los años con diversos elementos que permiten un mejor rendimiento del sistema o considerar nuevas posibilidades de interacción con el usuario. Entre estos elementos se encuentran unidades de cálculo específicas para coma flotante o gestión de gráficos, memoria caché más rápida y cercana a la unidad de proceso que la memoria convencional, el empleo de discos duros como almacenamiento persistente aunque más lento, y redes de comunicaciones, entre otros.

2. ARQUITECTURA MISD (MULTIPLE INSTRUCTION SINGLE DATA)

Los sistemas MISD aplican varios flujos de instrucciones al mismo flujo de datos. En la práctica no se ha implementado ningún ordenador con arquitectura MISD pura. Sin embargo, unos sistemas que podrían considerarse MISD son las matrices sistólicas, que están formadas por un conjunto de procesadores interconectados. En ellas cada procesador realiza diferentes operaciones sobre un único flujo de datos, que recibe de uno de sus procesadores vecinos y envía, tras procesarlo, a otro procesador vecino.

3. ARQUITECTURA SIMD (SINGLE INSTRUCTION MULTIPLE DATA)

En este caso, un solo flujo de instrucciones se aplica a diferentes conjuntos de datos. Este tipo de arquitecturas ha sido empleado durante muchos años en los ordenadores de altas prestaciones destinados a realizar cálculos científicos. Éstos son

supercomputadores con procesadores vectoriales, capaces de realizar operaciones matemáticas sobre varios datos de forma simultánea. En la actualidad, se emplean también en ordenadores de consumo y entretenimiento. En ellos, se encuentran bloques de arquitectura SIMD por ejemplo en las tarjetas gráficas, operando simultáneamente sobre varios píxeles de la pantalla, sobre los diferentes canales de color de las imágenes, las coordenadas de los vértices que componen los modelos 3D, … De esta forma se obtiene un rendimiento del procesador mucho más elevado que si los cálculos debiera realizarlos el procesador SISD.

4. ARQUITECTURA MIMD (MULTIPLE INSTRUCTION MULTIPLE DATA)

Las arquitecturas de tipo MIMD consisten en varios flujos de instrucciones que se ejecutan sobre sus correspondientes flujos de datos. Dentro de esta clase existen dos grandes grupos: los sistemas de memoria compartida y los sistemas de memoria distribuida.

Los sistemas de **memoria compartida** están formados por un conjunto de procesadores independientes (compuestos cada uno de ellos por su unidad de control y su unidad de proceso) que comparten la unidad de memoria. En principio, esa memoria debe tener un puerto de acceso para cada procesador y debe gestionar los conflictos de acceso. Eso complica y encarece el sistema, además de ser un potencial cuello de botella, pues los procesadores se bloquean unos a otros al acceder a la misma memoria. Por ello en esta arquitectura cobran especial importancia las memorias caché, que reducen el tráfico entre cada procesador y la memoria principal. Sin embargo, eso lleva implícito el problema de asegurar la coherencia de los datos de las memorias caché entre sí y con la memoria principal.

Desde el punto de vista del programador, ésta es la forma más sencilla de manejar un sistema MIMD. Esto es así porque toda la memoria se reparte el rango de direcciones accesible por todos los procesadores, de forma similar a como ocurre en un sistema monoprocesador. El *hardware* (y el sistema que lo opera) se encarga de ocultar al programador la complejidad de acceder a la memoria compartida.

Existen dos tipos principales de sistemas de memoria compartida:

- Sistema de acceso uniforme a la memoria *(Uniform Memory Access, UMA)*. En ellos, la memoria es compartida por todos los procesadores sin ninguna prioridad de acceso en caso de conflictos. Eso hace que el tiempo de acceso de cada procesador a una posición cualquiera de memoria sea uniforme en todo el rango de direcciones de memoria. Con este sistema se producen más conflictos de acceso a memoria, y por ello no es una arquitectura muy adecuada cuando el número de procesadores es elevado.
- Sistemas de acceso no uniforme a memoria *(Non-Uniform Memory Access, NUMA)*. En este caso, cada procesador tiene asignada una parte de la memoria, y puede acceder al resto a través de la red de comunicaciones. Esto hace que el tiempo de acceso a una posición de memoria dependa de si esa posición pertenece al bloque de memoria propio o al de otro procesador.

Por otro lado se encuentran los sistemas de **memoria distribuida**, en los que cada procesador (formado por su unidad de control y su unidad de proceso) tiene una memoria local. Todas las unidades de memoria están conectadas entre sí por medio de una red de comunicaciones. Los procesadores trabajan sólo sobre su memoria local y no tienen acceso directo al resto de la memoria del sistema. Los procesadores tan sólo intercambian información entre sí por medio de mensajes que envían a través de la red de comunicaciones. Éste es el principal cambio para el programador con respecto a

los esquemas tradicionales, aunque estos ajustes del código permiten simplificar el *hardware*. Al utilizar este esquema basado en memoria local y una red de comunicaciones, se evita la necesidad de una memoria común de gran tamaño y los elementos que gestionen los conflictos de acceso simultáneo y de sincronización de cachés. Estos sistemas suelen estar formados por equipos monoprocesador conectados por red, y por ello son más fácilmente escalables, es decir, puede aumentar el número de equipos en el sistema sin que ello afecte negativamente al rendimiento del mismo. Dentro de este tipo de sistemas se encuentran los *clusters* y los grid.

Los *clusters* son conjuntos de ordenadores conectados por una red que se comportan de cara al usuario como un solo sistema. Los *clusters* utilizan equipos idénticos que deben estar cercanos físicamente y conectados por una red de alta velocidad que permita obtener un buen rendimiento. Por otra parte, los grid son sistemas compuestos por equipos heterogéneos, dispersos geográficamente y con conexiones más débiles que en el caso de los *clusters*. Algunos sistemas grid están formados por ordenadores personales (de oficina o de uso doméstico) que aportan potencia de cálculo al grid cuando no se están utilizando por su usuario habitual. Por ello se denominan grids no dedicados o grids carroñeros *(scavenging grids)*. Como estos grids utilizan infraestructuras ya existentes, se pueden reducir los costes de implantación. Por otro lado, el empleo de grids dedicados evita interferencias en su funcionamiento y es el camino seguido por las grandes plataformas europeas.

Definición y características de los entornos grid

Los entornos grid toman su nombre de la red eléctrica *(electric grid)*, en un símil con su ubicuidad y facilidad de acceso. El objetivo, un tanto utópico, es que acceder a la capacidad de cálculo de estos sistemas sea tan sencillo como enchufar un electrodoméstico a la red y obtener suministro eléctrico. Las características que definen un grid son [Baker2002] heterogeneidad, disponibilidad dinámica, dispersión geográfica y escalabilidad:

- Son sistemas compuestos por **equipos heterogéneos**, y esta heterogeneidad puede referirse tanto a la arquitectura hardware de los equipos como al sistema operativo.
- Además, esos equipos están **disponibles de forma dinámica**, si están arrancados y conectados, y en el caso de equipos no dedicados, sólo si el propietario o usuario principal no está utilizándolo para otras tareas. Por otra parte, como red de comunicaciones se emplea habitualmente Internet, que no fue diseñada con requisitos de seguridad y fiabilidad, y por tanto puede fallar.
- Los equipos estarán **dispersos geográficamente**, ya que pueden pertenecer a instituciones en países distintos. Esto contribuye a disminuir la tasa de disponibilidad y a aumentar los tiempos de transmisión de datos. Además exige métodos de coordinación, y en su caso, de asignación de costes por uso de los recursos.
- En general, también se plantea un requisito de **escalabilidad** en los entornos grid, pues precisamente la unión de grandes conjuntos de recursos de diferentes organizaciones es uno de los objetivos de este tipo de plataformas, y ello no debe repercutir negativamente en el rendimiento del sistema.

Con lo que se acaba de presentar se puede deducir qué tipo de trabajos son adecuados para su resolución en entornos grid. Con aplicaciones de grano grueso (con mucho tiempo de cálculo respecto al tiempo de comunicaciones necesario), compuestas por tareas bastante independientes, lo más indicado es el uso de entornos grid. Pero en el caso de aplicaciones de grano fino, en las que se necesita comunicar frecuentemente datos entre tareas, las latencias en las comunicaciones introducen retrasos que pueden ser significativos en el tiempo total de ejecución. En ese caso sería mejor utilizar un ***cluster***, que cuida de manera especial ese aspecto.

De cara al usuario, debe proporcionarse un conjunto de funcionalidades que conviertan al grid en una plataforma útil. De manera breve, las necesidades del usuario que deben ser cubiertas por el grid son:

- Envío de trabajos al grid y monitorización de su estado de ejecución.
- Almacenamiento de datos y acceso a ellos, para el análisis de los resultados obtenidos.
- Calidad de servicio, que permita definir diferentes prioridades a los trabajos o garantizar unos niveles mínimos de utilización de los recursos.
- Medida del uso de los recursos, como forma de facturación de los costes del grid y para poder optimizar el rendimiento.

• Seguridad, que permita controlar el acceso a los recursos del grid.
• Registro de nuevos recursos cuando éstos estén disponibles, o notificación de la desaparición de los mismos del grid.

Para hacer frente a estas demandas, habitualmente se propone una estructura del grid basada en cuatro capas [Foster2003]: infraestructura, *middleware*, herramientas de gestión y desarrollo, y aplicaciones.

La infraestructura del grid *(grid fabric)* la componen los equipamientos físicos. Estos pueden ser los propios equipos de cálculo, los elementos de almacenamiento de información, las redes de comunicaciones o la instrumentación científica. Esta última opción resulta bastante interesante, pues permite, por ejemplo, la operación remota de instrumental o la adquisición de datos de forma remota.

El *middleware* es el *software* que se encarga de proporcionar los servicios que se han indicado más arriba, aislando el resto de capas del grid de las peculiaridades de la infraestructura. Para ello es muy importante el desarrollo de estándares que permitan interactuar con diferentes sistemas y ofrecer una interfaz común. Muy cercana a esta última capa se encuentra la de las herramientas de gestión y desarrollo, que controlan el acceso de las aplicaciones a los servicios del grid proporcionados por el *middleware*.

Por último, se encuentran las aplicaciones que hacen uso del grid, permitiendo a los usuarios emplearlo para resolver problemas con requisitos computacionales muy exigentes. Una particularización de las aplicaciones del grid son los portales, que permiten el acceso del usuario al grid a través de un entorno amigable, compuesto por páginas web desde las que se gestiona de manera sencilla la información necesaria para operar el grid.

Organización y algoritmos de asignación de tareas

La asignación de tareas a procesadores es un tema de gran importancia no sólo en los entornos grid, sino también en el cálculo paralelo y en los sistemas operativos multitarea. La eficacia de esta asignación condiciona el rendimiento general del sistema. De forma general, se habla de este último problema como el de asignación de recursos, en el que a las tareas en que se divide el programa principal se deben asignar a equipos disponibles en el grid. Este problema de asignación puede resolverse de dos formas principalmente [El-Rewini2005]:

• La decisión puede tomarse en **tiempo de compilación**, empleando el conocimiento previo que se tiene tanto del problema a resolver como del entorno en que se va a resolver. En este caso se habla de métodos estáticos, que pueden considerar o no las relaciones de precedencia entre las tareas que hay que asignar. Entonces se habla de planificación de tareas *(task scheduling)* y de distribución de tareas *(task allocation)*, respectivamente.
• En otros casos, la decisión de asignar las tareas a uno u otro equipo puede tomarse en **tiempo de ejecución**, buscando el balanceo de la carga. Se emplean entonces métodos dinámicos, en los que la asignación de recursos no utiliza ningún tipo de conocimiento a priori sobre el propio sistema distribuido.

A este último grupo pertenecen los métodos con que normalmente se asignan los recursos en los entornos grid, ya que por su propia naturaleza tienen una estructura dinámica que invalida los métodos estáticos. Para lograr un buen rendimiento del sistema, se lleva a cabo balanceo de la carga entre los procesadores del sistema, y en algunos casos se permite la migración de tareas de equipos más cargados a equipos más libres. En estos métodos se tienen en cuenta consideraciones adicionales como el retraso de comunicaciones, y la elección del mejor equipo para recibir tareas en función de su carga de trabajo y de su potencia de cálculo.

La toma de decisión para la planificación de tareas en un grid [Hamscher2000] se puede hacer de forma centralizada, descentralizada o jerárquica. Cada alternativa tiene sus ventajas e inconvenientes, que se comentan a continuación:

• Un **planificador centralizado** es el más sencillo de implantar porque cuenta con toda la información y no necesita coordinarse con nadie más para tomar sus decisiones. Sin embargo, al ser un elemento único para todo el grid, puede suponer un problema de escalabilidad, al convertirse en un cuello de botella en los grid de gran tamaño o cuando hay muchas tareas pequeñas.
• Un conjunto de **planificadores descentralizados**, por otra parte, permite resolver el problema de escalabilidad mencionado en el caso anterior. Cuando la decisión se toma descentralizada se puede crecer en tamaño añadiendo nuevos planificadores que se en-

carguen de los nuevos equipos y que se coordinen con los otros planificadores ya existentes. La estabilidad del conjunto depende de la eficiencia de los protocolos de coordinación entre planificadores. Además, en este enfoque se puede asignar un planificador por cada institución participante en el grid, permitiéndole tener el control sobre sus equipos y las reglas de asignación de los mismos.

- La **planificación jerarquizada** es un enfoque mixto entre los dos anteriores. Por un lado, retiene la sencillez de la planificación centralizada a alto nivel, considerando el grid en grandes bloques para que no suponga demasiada información. Por otro lado, tiene la flexibilidad y escalabilidad de los planificadores descentralizados, ayudando a su coordinación mediante la jerarquización.

Revisión de entornos grid existentes

En los últimos años ha habido un gran empuje en el desarrollo de los entornos grid. Esto se refleja en la gran cantidad de proyectos que han surgido y en el apoyo institucional que en algunos casos han recibido los desarrolladores.

Dentro de los desarrollos patrocinados por los estados, un caso muy destacado es el del proyecto europeo EGEE *(Enabling Grid for E-sciencE)*. Entre sus objetivos se encuentra proporcionar una infraestructura que permita analizar los resultados obtenidos con el acelerador de partículas LHC *(Large Hadron Collider)* que entrará en funcionamiento a finales de 2008. Con los desarrollos realizados hasta el momento se están llevando a cabo cálculos en disciplinas tales como la astrofísica, la bioinfórmática y la arqueología. Además del desarrollo en Europa, existen esfuerzos conjuntos con países de América Latina y Asia, con el fin de promocionar las tecnologías grid y la unión de recursos de cálculo en todo el mundo. Por su parte, en los Estados Unidos se ha desarrollado Teragrid, una infraestructura grid con financiación pública en la que actualmente participan varias instituciones de la industria y de la universidad. Y también en países de Asia se están desarrollando iniciativas de tipo público o con apoyo estatal.

En cuanto al *software* necesario para gestionar los grid, se van a presentar dos casos concretos. El primero *(Globus toolkit)* es bastante complejo pero cuenta con gran apoyo por parte de las instituciones e industria, y el segundo *(NetSolve)* es más sencillo y permite una implantación más directa de un grid. Pueden contemplarse como los dos extremos del espectro de posibilidades al construir un grid. Posteriormente se comentan brevemente otras opciones de *middleware* que cuentan con capacidades intermedias entre ambos.

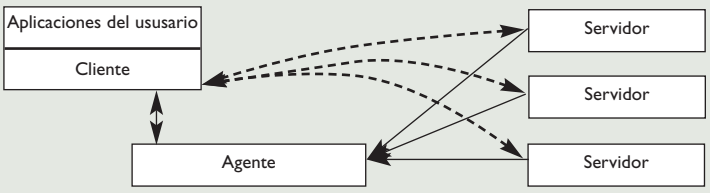
Globus Toolkit

Globus toolkit [Foster1997] es uno de los entornos grid más relevantes, con mayor base de usuarios. Ha sido desarrollado por la Globus Alliance, que reúne a universidades e instituciones de Estados Unidos y Europa, y recibe apoyo de grandes empresas del mundo de la informática.

La estructura de *Globus* contempla toda la complejidad para implantar un grid a nivel industrial. Sigue el camino marcado por el estándar *Open Grid Services Architecture*, aprobado por el *Open Grid Forum*, que es una agrupación de usuarios, desarrolladores y empresas relevantes en el mundo de la informática, con el objetivo de conseguir la estandarización de los grid y la creación de comunidades de usuarios. Globus tiene un diseño modular cuyos componentes pueden ser clasificados en las siguientes cinco categorías:

- **Núcleo común**, que permite la comunicación entre los equipos del grid por medio de servicios *web*, independientemente de la plataforma sobre la que se ejecute. Forma

Figura 1



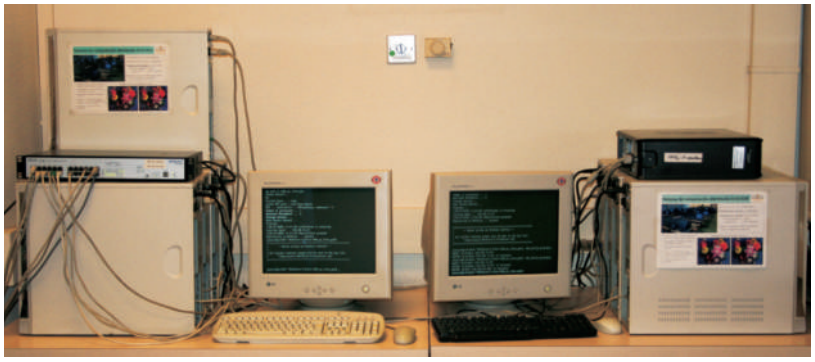


Figura 2

la base que interactúa con la red, proporcionando un estándar de comunicación entre equipos que permite que sobre él se lleven a cabo las tareas del resto de componentes.

- **Seguridad**, que incluye los componentes que permiten la autenticación de los usuarios y la autorización de éstos para lanzar tareas al grid, basado en un sistema de clave pública o de cifrado asimétrico. Esto resulta vital debido a que las organizaciones que participan en el grid están permitiendo a usuarios de otras organizaciones el acceso a sus equipos, lo que podría suponer un riesgo de seguridad en el caso no ser controlado.
- **Gestión de los datos**, que permite el uso de repositorios virtuales de datos en los que almacenar la información de forma descentralizada. En esta categoría se encuentran las herramientas de comunicaciones que permiten la transmisión de datos robusta, segura y eficiente, sobre todo en el caso de grandes cantidades de información. Asimismo, se dispone de un sistema de ficheros distribuido que se encarga de almacenar los ficheros y mantener una relación entre su ubicación física y el nombre lógico con el que el usuario lo referencia; y las herramientas de replicación de datos, que proporcionan mayor fiabilidad al mantener varias copias del mismo fichero en diferentes lugares.
- **Gestión de la ejecución**, que permite lanzar trabajos al grid, monitorizar su ejecución, cancelarlos en caso de que fuese necesario, sincronizar trabajos que formen parte de un cálculo coordinado, y por último recoger los resultados.
- **Servicio de información**, que permite registrar la evolución del grid y los recursos disponibles en el mismo.

NetSolve/GridSolve

NetSolve [Seymour2005] es un *middleware* desarrollado en el *Innovative Computing Laboratory*, de la Universidad de Tennessee en Knoxville (Estados Unidos). *NetSolve* está pensado para proporcionar un acceso muy sencillo al grid desde diferentes entornos: a bajo nivel, desde lenguaje C o Fortran, y a alto nivel, desde sistemas de cálculo matemático como Matlab, Mathematica u Octave. Se encuentran en desarrollo extensiones que permiten lanzar trabajos al grid desde Excel, así como una nueva versión denominada *GridSolve* que proporciona mayor control e interoperabilidad con otros grid.

La estructura de *NetSolve* es más sencilla que en el caso de *Globus*, y está formado por tres componentes principales que se pueden ver en la Figura 1:

- El **servidor**: Es la parte encargada de hacer los cálculos en cada uno de los ordenadores del grid. Con la funcionalidad por defecto se pueden realizar operaciones matriciales básicas, pero también es posible incorporar nuevas opciones de cálculo más específico.
- El **cliente**: Es la parte encargada de proporcionar al usuario la interfaz con el grid. Este bloque debe formar parte de la aplicación del usuario, y mediante el empleo de un reducido número de funciones permite enviar trabajos al grid y recoger los resultados de su ejecución.
- El **agente**: Es el elemento central del grid, que mantiene información de los servidores disponibles y las capacidades de cada uno. A él accede el cliente, de forma transparente para el usuario, para conocer a qué servidores puede mandar los trabajos. Una vez se ha asignado un servidor a un determinado trabajo, la comunicación de datos de entrada y resultados se realiza directamente entre el cliente y los servidores, para no sobrecargar el agente y evitar un doble tráfico.

En el Instituto de Investigación Tecnológica se está empleando un prototipo de grid, que se muestra en la Figura 2, que está formado por 10 servidores de cálculo y otro equipo que actúa como cliente y agente. Para mane-

jar ese grid se ha decidido emplear *NetSolve* por su sencillez de uso e implantación. Este prototipo se ha empleado en el Instituto para llevar a cabo cálculos de optimización estocástica [Latorre2007]. Además, en las aulas de informática de la Universidad, donde hay disponibles una gran cantidad de equipos, se han realizado análisis de cálculo intensivo en grid [Palacios2007], sobre la base de *NetSolve*, utilizando más de 200 ordenadores.

Otros entornos

Además de los gestores de grid anteriormente mencionados, existen otros proyectos grid muy activos, de entre el gran número de ellos que han surgido en los últimos años. Entre estos proyectos se puede destacar Alchemi, BOINC, Condor, G-Farm, Nimrod, Ninf o PVM. En este grupo, resulta especialmente relevante BOINC, o *Bekeley Open Infrastructure for Network Computing* [Anderson2004], *middleware* desarrollado en la Universidad de California Berkeley (Estados Unidos) que se originó a partir del proyecto SETI@home, al que sirvió de plataforma y del que en 2002 se independizó. Actualmente hay multitud de proyectos activos basados en BOINC a los cuales se puede asociar cualquier usuario para aportar tiempo de CPU. Además, es interesante resaltar PVM [Geist1994] que cuenta con muchos años de desarrollo y una amplia base de usuarios. Se trata de un sistema multiplataforma basado en paso de mensajes y ejecución remota de procesos, aunque no puede considerarse un grid desde un punto de vista estricto. Y Condor [Thain2005] es un grid no dedicado con estructura similar a NetSolve, cuya característica diferencial es que permite la migración de procesos dentro del grid cuando los equipos son empleados por sus propietarios.

Conclusiones

Se ha visto que los entornos grid son sistemas MIMD formados por gran cantidad de equipos heterogéneos, posiblemente dispersos geográficamente, que al trabajar de forma coordinada pueden alcanzar grandes capacidades de procesamiento. Además, tienen la ventaja de emplear equipos ordinarios, con lo que se abaratan los costes frente a las alternativas que emplean superordenadores o arquitecturas complejas. Sin embargo, es necesario tener en cuenta la disponibilidad dinámica de los equipos, que afecta a los grids formados por equipos no dedicados en exclusiva a esos grids.

Los entornos grid son especialmente indicados en el caso de cálculos de paralelismo de grano grueso, en los que hay una gran carga computacional frente a las necesidades de comunicaciones, o dicho de otra forma, si están compuestos por tareas independientes entre sí en gran medida. En ese caso, el esfuerzo adicional de adaptación del código a su ejecución en grid puede reportar grandes beneficios en tiempos de ejecución. ▮

Bibliografía

[Anderson2004] David P. Anderson. *BOINC: A System for Public-Resource Computing and Storage*. 5th IEEE/ACM InternationalWorkshop on Grid Computing. Pittsburgh, USA, 2004.

[Baker2002] Mark A. Baker, Rajkumar Buyya y Domenico Laforenza. *The Grid: International Efforts in Global Computing*. International Journal of Software Practice and Experience, 32, 2002.

[El-Rewini2005] Hesham El-Rewini y Mostafa Abd-El-Barr. *Advanced Computer Architecture And Parallel Processing*. JohnWiley & Sons, 2005.

[Flynn1972] Michael J. Flynn. *Some Computer Organizations and Their Effectiveness*. IEEE Transactions on Computers, C-21, n. 9, 1972.

[Foster1997] Ian Foster y Carl Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. International Journal of Supercomputer Applications, 11, 1997.

[Foster2003] Ian Foster y Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2003.

[Geist1994] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek y Vaidy Sunderam. *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[Hamscher2000] Volker Hamscher y Uwe Schwiegelshohn, Achim Streit y Ramin Yahyapour. *Evaluation of Job-Scheduling Strategies for Grid Computing*. GRID 2000: First IEEE/ACM International Workshop, Bangalore, India, 2000.

[Korpela2001] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, Matt Lebofsky. *SETI@home-Massively distributed computing for SETI*. Computing in Science & Engineering, enero-febrero 2001.

[Latorre2007] Jesús M. Latorre. Resolución distribuida de problemas de optimización estocástica. Aplicación al problema de coordinación hidrotérmica. Tesis doctoral. Universidad Pontificia Comillas, 2007.

[Palacios2007] Rafael Palacios. *Using a grid to evaluate the threat of zombie networks against asymmetric encryption algorithms*. Proceedings of the Spanish Conference on e-Science Grid Computing. Madrid, España 2007.

[Seymour2005] Keith Seymour, Asim Yarkhan,Sudesh Agrawal y Jack Dongarra. *NetSolve: Grid Enabling Scientific Computing Environments*. Grid Computing and New Frontiers of High Performance Processing, Elsevier, 2005.

[Thain2005] Douglas Thain, Todd Tannenbaum y Miron Livny. *Distributed computing in practice: the Condor experience. Concurrency - Practice and Experience*, 17, 2005.