UNIVERSIDAD POLITÉCNICA DE VALENCIA
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

TESIS DOCTORAL
Requisito para la Obtención del Grado de Doctor en Informática por la Universidad Politécnica de Valencia

Técnicas de monitorización y diferenciación de servicios para la asignación de recursos en entornos de computación Grid, en base a indicadores de nivel de servicio

Autor
Erik Torres Serrano

Directores
Dr. Ignacio Blanquer Espert
Dr. Vicente Hernández García

Valencia, 16 de marzo de 2010

AGRADECIMIENTOS

Quiero empezar agradeciendo a mis padres y a mi hermano, porque sin su apoyo nada de esto hubiese sido posible. Agradezco también a Yadira por su apoyo, pero sobre todo por su paciencia.

Agradecer especialmente a Vicente Hernández y a Ignacio Blanquer, que más que directores de esta tesis, son artífices de la misma. Muchas gracias por vuestra confianza y por vuestra ayuda.

Agradecer también especialmente a mis amigos, los de siempre y los de no hace tanto. Ellos saben el porqué, pero aún así me gustaría mencionar a MaCarmen y Damián, Silvia y Jose, y a todos los demás, por haberme brindado un lugar en su amada Valencia. A Carmen, Geovan, Víctor, Alfredo, Yudith, René, por todo lo vivido y lo por venir. A Liczandro, Noriel, Trujillo, Liesner, por los buenos momentos.

Agradecer también a todos los miembros del Grupo de Grid y Computación de Altas Prestaciones (GRyCAP), porque ha sido un placer compartir con vosotros. Aprovechar para agradecer a Carlos, Migue, Elizabeth, Cristina, y a todos los que están y los que no, por estos años en los que hemos sido compañeros dentro y fuera del trabajo.

En fin, a todos los que de una forma u otra han dejado una marca positiva en mi vida. A todos vosotros, muchas gracias.

RESUMEN

El fortalecimiento de las infraestructuras de computación Grid, como plataforma para el acceso a recursos de cómputo y de almacenamiento en forma de servicios en Internet, ha dado lugar a una nueva necesidad: vincular las solicitudes de los usuarios con los servicios más apropiados, con el objetivo de mejorar la eficiencia de utilización de los recursos que forman parte del Grid, a la vez que mejorar las prestaciones de los trabajos ejecutados en el mismo. Esta necesidad se traduce, en la práctica, en la demanda de nuevos mecanismos de planificación de trabajos y de gestión de recursos, que permitan dotar a las infraestructuras Grid con nuevas estrategias para proporcionar niveles de servicio diferentes a diferentes perfiles de usuarios, proyectos y aplicaciones, y todo ello sobre la base de los requerimientos de calidad de servicio (Quality of Service o QoS) de cada perfil, formalizados en un contrato de nivel de servicio (Service Level Agreements o SLA).

Por estas razones, proporcionar QoS en entornos de computación Grid es un área de investigación muy activa, a la vez que muy importante para la evolución del Grid hacia una infraestructura de propósito general que soporte modelos de negocio complejos.

Sin embargo, a pesar de los avances en las técnicas de planificación y de gestión de recursos, el soporte para QoS en entornos de computación Grid es todavía muy limitado y, hasta el momento, no existe una solución definitiva para el problema.

En este trabajo proponemos un nuevo modelo para la asignación de recursos en el Grid, en base a requerimientos de QoS. Como parte de este modelo, los servicios Grid son evaluados periódicamente a través de casos de prueba representativos, que son ejecutados en los recursos y que sirven para determinar la capacidad de los mismos para funcionar con unas prestaciones y una disponibilidad determinadas. A la vez, los recursos deben ser continuamente monitorizados para conocer su estado. De todo lo anterior se obtiene una clasificación que se utiliza para diferenciar los servicios en el momento de planificar la ejecución de un nuevo trabajo en el Grid. La segunda parte de este trabajo está enfocada a demostrar la aplicabilidad del modelo propuesto a la solución de un problema de planificación complejo: la asignación de recursos en el Grid sobre un modelo de optimización de costes de ejecución.

El trabajo desarrollado en la presente tesis cubre todas las etapas necesarias para la asignación de recursos para la ejecución de trabajos en el Grid. Se ha estudiado la utilización de indicadores de QoS para describir los recursos del Grid y expresar los requerimientos de las solicitudes de trabajo enviadas al mismo, centrándose para ello en la base conceptual que proporciona la OGSA (Open Grid Services Architecture). En este contexto, la QoS se define como una medida del nivel de los servicios prestados, con una lista precisa de parámetros que la caracterizan, entre los que destacan la seguridad, el ancho de banda, el tiempo medio de respuesta, la disponibilidad del servicio, la potencia de cálculo, la memoria y capacidad de almacenamiento. Sobre esta base, se ha propuesto un algoritmo de asignación de recursos que permite optimizar la selección de los mismos a nivel global en la infraestructura Grid. Además, se ha presentado un sistema de monitorización que permite recoger los indicadores de carga de los recursos, y propagarlos, de forma eficiente, a un conjunto de nodos distribuidos por todo el Grid.

Asimismo, se presentan resultados de todo lo anterior. Una parte de los mismos fueron obtenidos en simulaciones en entornos controlados, y otra parte en entornos de computación Grid reales, utilizando middleware Grid actual. De la misma forma, se estudia la aplicabilidad de los resultados a casos de estudio y aplicaciones reales.

RESUM

La fortalesa de les infraestructures de computació Grid, com a plataforma d'accés a recursos de còmput i emmagatzemament en forma de serveis d'Internet, ha donat lloc a una nova necessitat: vincular les sol·licituds dels usuaris amb els serveis mes apropiats, amb l'objectiu de millorar l'eficiència d'utilització dels recursos que formen part del Grid, a més de millorar les prestacions dels treballs executats. Aquesta necessitat es tradueix, en la pràctica, en la demanda de nous mecanismes de planificació de treballs i de gestió de recursos, que permeten dotar les infraestructures Grid amb noves estratègies, que proporcionen diferents nivells de servei segons els perfils d'usuaris, projectes i aplicacions, i tot això basant-se en els requeriments de qualitat de serveis (Quality of Services o QoS) de cada perfil, formalitzats mitjançant un contracte de nivell de servei (Service Level Agreements o SLA).

Per totes aquestes raons, proporcionar QoS en entorns de computació Grid es una àrea d'investigació molt activa, i alhora molt important per a l'evolució del Grid cap a una infraestructura de propòsit general que suporte models de negoci complexos.

Per contra, a pesar dels avanços en les tècniques de planificació i de gestió de recursos, el suport per a QoS en entorns de Computació Grid es encara molt limitat i, fins ara, no hi ha una solució definitiva per al problema.

En aquest treball es proposa un nou model per assignar recursos en el Grid, basat en requeriments de QoS. Com a part del model, els serveis Grid són avaluats periòdicament a través de casos de prova representatius, que són executats als recursos i que serveixen per a determinar la capacitat d'aquests per a funcionar amb unes prestacions i una disponibilitat determinada. Al mateix temps, els recursos han de ser contínuament monitorats per a conèixer-ne l'estat. De tot això, obtenim una classificació que s'utilitza per diferenciar els serveis en el moment de planificar l'execució d'un nou treball en el Grid. La segona part d'aquesta tesi està enfocada a demostrar l'aplicabilitat del model proposat a la solució d'un problema de planificació complexa: l'assignació de recursos en el Grid sobre un model d'optimització de costos d'execució.

El treball desenvolupat en aquesta tesi cobreix totes les etapes necessàries per a l'assignació de recursos i execució de treballs al Grid. S'ha estudiat la utilització d'indicadors de QoS per descriure els recursos del Grid i expressar els requeriments de les sol·licituds de treballs enviats, centrant-se en la base conceptual que proporciona OGSA (Open Grid Services Architecture). Es aquest context, la QoS es defineix com una mesura de nivell de serveis prestats, com una llista precisa de paràmetres que la caracteritzen, entre els quals destaquen la seguretat, l'amplada de banda, el temps de resposta, la disponibilitat del servei, la potencia de càlcul, la memòria i la capacitat d'emmagatzemament. Sobre aquesta base s'ha proposat un algorisme d'assignació de recursos que permet optimitzar la selecció d'aquests a nivell global en la infraestructura Grid. A més a més, s'hi ha presentat un sistema de monitoratge que permet recollir els indicadors de càrrega dels recursos, i propagar-los, d'una manera eficient, a un conjunt de nodes distribuïts per tot el Grid.

Finalment, es presenten els resultats de tot el que hem ressenyat. Una part d'aquests foren obtinguts en simulacions en entorns controlats, i un altra part, en entorns de

computació Grid reals, utilitzant programari intermediari Grid actual. Anàlogament, hi estudiem l'aplicabilitat dels resultats a casos d'estudi i aplicacions reals.

ABSTRACT

The strengthening of Grid computing infrastructures as a platform that makes available computational resources and storage capabilities in the form of Internet services with standardized interfaces, has given rise to a new necessity: matching requests with the most appropriate services, with the aim of improving the efficiency in the utilization of the resources in the Grid, and at the same time improving the performance of the jobs. In practice, this need can only be met by developing new scheduling and management mechanisms that achieve a specific differentiation level between services in the Grid, on the basis of their readiness to deliver a specific level of service. Moreover, clients should be enabled to manage the Quality of Service (QoS) requirements of their jobs, within the terms specified in the Service Level Agreements (SLA).

For these reasons, providing support for QoS in Grid computing environments is an active area of research, while very important for the development of general-purpose Grid systems that supports complex business processes.

However, despite the advances made in improving meta-scheduling and resource management in the Grid, the support for QoS in this environment is still incomplete, and at present there is no unique solution for this problem.

In this work, we aim at developing a new model for the allocation of resources in the Grid, on the basis of requirements of QoS. As part of this model, the services in the Grid are periodically evaluated through representative use cases that are executed on the resources with the objective of determining the capability of them to deliver a specific performance and availability. At the same time, the resources must be continuously monitored in order of knowing their status. Using these two measures, we obtain a rank that differentiates the services by their suitability for executing a job with the required QoS. This classification can be used in conjunction with a meta-scheduler to schedule the execution of new jobs in the Grid. The second part of this work is focused to demonstrate the applicability of our model to the solution of a complex problem: the allocation of resources in the Grid for the execution of jobs on the basis of an overall strategy for cost optimization.

The work embodied in the present thesis covers all the phases necessary for the allocation of resources for the execution of jobs in the Grid. We have studied the use of QoS indicators to improve the information that describes the resources in the Grid and to define the requirement of the jobs sent to them, focusing in the conceptual basis provided by the Open Grid Services Architecture (OGSA). In this context, QoS is a measure of the level of service attained, with a group of indicators that characterize the service. These indicators include, but are not limited to: security, bandwidth, average response time, availability of the service, computational power, and memory and storage capability. On this basis, we have proposed an algorithm for the allocation of resources in the Grid that allows the optimization of this process at a global level. Moreover, we present a distributed monitoring system that provides the necessary up-to-date information, including the workload of the resources.

Finally, we present results that support our work. A part of them are computational simulations, but the majority was obtained in real Grid environments, using current

Grid middleware. In the same way, we have studied the applicability of our results to use cases and real applications.

ÍNDICE GENERAL

Capítulo 1

1. INTRODUCCIÓN Y OBJETIVOS

En este primer capítulo se muestra la motivación del trabajo desarrollado, dando una visión general de la tesis doctoral, planteando los objetivos que se han cubierto en la presente tesis, así como la estructuración del resto de capítulos.

1.1. MOTIVACIÓN

El aumento en complejidad de las simulaciones y modelado de problemas científicos, así como el incesante incremento de información obtenida en las grandes instalaciones científicas, están demandando de capacidades de proceso y almacenamiento no triviales. Las infraestructuras de computación Grid están siendo utilizadas, con muy buenos resultados, para hacer frente a estos retos. Tanto es así, que muchos proyectos han desarrollado infraestructuras Grid de investigación internacionales, y muchos países (42 países, hasta Junio de 2009) han desarrollado sus propias Iniciativas de Grid Nacionales o NGIs [1]. Un ejemplo de esto son los proyectos BalticGrid [2], NorduGrid [3], NAREGI [4], TeraGrid [5], y especialmente, EGEE [6] y EELA [7], en los cuales participa España. Bajo el acrónimo de EGEE se han desarrollado una serie de proyectos financiados por la Comisión Europea que han tenido como objetivo principal desarrollar una infraestructura de servicios Grid, en calidad de producción y disponible permanentemente para que los investigadores europeos resuelvan grandes problemas científicos, como la construcción del LHC (Large Hadron Collider), la síntesis de nuevos fármacos, o la predicción de clima. En Octubre de 2009, había unos 260 centros (más de 13 en España), en 55 países, conectados a EGEE. En esa fecha, el número de núcleos de procesamiento disponible las 24 horas del día, los 7 días de la semana era de unos 150,000, y las capacidades de almacenamiento en disco y en cinta era de 28 y 41 petabytes, respectivamente [8]. Esta infraestructura es utilizada por unos 14,000 usuarios registrados en unas 200 Organizaciones Virtuales o VOs, para ejecutar un promedio de 330,000 trabajos diarios.

Las aplicaciones de EGEE se enmarcan en más de 15 disciplinas científicas y tecnológicas, entre las cuales se encuentran la Arqueología, la Astronomía, la Protección Civil, las Ciencias de la Vida, las Finanzas, las Ciencias de los Materiales, la Química Computacional, y varias ramas de la Física, entre otras.

En el marco latinoamericano, bajo el acrónimo de EELA se han desarrollado una serie de proyectos conjuntos entre la Unión Europea y América Latina que han tenido como objetivo principal crear y operar una infraestructura Grid común entre Europa y América Latina. En Agosto de 2008, había unos 13 centros de recursos conectados a EELA en Europa (varios de ellos en España) y unos 28 en América Latina. En conjunto, la infraestructura cuenta con 3,000 núcleos de procesamiento y más de 700 terabytes de espacio de almacenamiento. El 70% de las aplicaciones de EELA se enmarcan en el dominio de la Biomedicina, la Física de las Altas Energías y las Ciencias de la Tierra, y el 30% restante en la Previsión Meteorológica, la Educación, la Industria y el Gobierno.

De este fortalecimiento de las infraestructuras de computación Grid como plataforma para el acceso a recursos de cómputo y de almacenamiento en forma de servicios sobre en Internet, surge una nueva necesidad: vincular las solicitudes de los usuarios con los servicios más apropiados, con el objetivo de mejorar la eficiencia de utilización de los recursos que forman parte del Grid, a la vez que mejorar las prestaciones de los trabajos ejecutados en el mismo. Esta necesidad se traduce en la práctica en la demanda de nuevos mecanismos de planificación de trabajos y de gestión de recursos, que permitan dotar a las infraestructuras Grid con nuevas estrategias para proporcionar niveles de servicio (Level of Service o LoS) diferentes a diferentes perfiles de usuarios, proyectos y aplicaciones, y todo ello sobre la base de los requerimientos de calidad de servicio (Quality of Service o QoS) de cada perfil, formalizados en un contrato de nivel de servicio (Service Level Agreement o SLA).

Por estas razones, proporcionar QoS en entornos de computación Grid es un área de investigación muy activa, a la vez que muy importante para la evolución del Grid hacia una infraestructura de propósito general que soporte modelos de negocio complejos. A pesar de los avances en las técnicas de planificación y de gestión de recursos, el soporte para QoS en entornos de computación Grid es todavía muy limitado y, hasta el momento, no existe una solución definitiva para el problema.

Una forma de ilustrar la importancia de proporcionar QoS en los entornos Grid es comparar las tecnologías Grid con otro entorno que también ofrece recursos virtualizados como servicios sobre Internet: el "cloud computing". Aunque las tecnologías Grid y cloud son diferentes, se complementan entre sí y tienen varios puntos comunes. En primer lugar, ambas tecnologías permiten disminuir costes a la vez que mejorar la eficiencia y la escalabilidad de las aplicaciones. En segundo lugar, el Grid y el cloud requieren de un esfuerzo económico y un número de recursos importantes para su puesta en marcha y mantenimiento posterior. La diferencia principal está en que el Grid está orientado a compartir complejos servicios con diferentes dominios administrativos, mientras que el cloud expone recursos más básicos y desde un único dominio.

Esta discrepancia en la orientación de ambas tecnologías trae aparejada una diferencia en la complejidad de su gestión. El Grid tiene una complejidad adicional bastante importante, que está relacionada, principalmente, con la distribución geográfica de los recursos y con la participación de múltiples dominios administrativos. En consecuencia, los modelos de explotación de ambas tecnologías son diferentes.

Desde sus inicios, el cloud computing ha estado estrechamente vinculado a un objetivo económico y a modelos de negocios muy bien definidos, como por ejemplo Amazon Elastic Compute Cloud (Amazon EC2), que se orientan a disminuir costes. En contraste, la promesa del Grid es mucho más amplia porque busca la integración de recursos en dominios administrativos diferentes para facilitar la investigación colaborativa y favorecer la solución de grandes retos de investigación, pero a la vez menos concreta en cuanto al tema económico porque se orienta más a compartir recursos. De aquí que, de alguna forma, el potencial de negocio del Grid queda relegado a un segundo plano por la incapacidad de los proveedores de servicios para "vender" y de los usuarios para "comprar" servicios en el Grid. El primer paso para recuperar la capacidad de negocio del Grid es la diferenciación de servicios para que los proveedores puedan fijar las bases para la competencia por los potenciales "clientes".

La diferenciación de servicios es una forma de proporcionar QoS a los servicios Grid, que se propaga a través de las Organizaciones Virtuales (VOs) estableciendo SLAs que definen la clasificación y las normas para el etiquetado de los recursos en el Grid. Los SLAs también especifican los diferentes perfiles para la ejecución de trabajos y las acciones que los proveedores deben llevar a cabo cuando un trabajo no se ajusta a ninguno de los perfiles posibles. Durante el tiempo que tarda la ejecución de un trabajo en el Grid, el mismo recibe servicios, con una cierta calidad, de los recursos de la VO.

Como regla general, la medida de la QoS recibida por un trabajo se define como la probabilidad de que cierta métrica asociada al servicio, como por ejemplo la tasa de error y el número de relanzamientos del trabajo, no excede unos límites acordados previamente. Todos los servicios Grid involucrados en la ejecución del trabajo deben contribuir para asegurar que estos límites no sean sobrepasados.

En el sentido tradicional, la QoS se refiere al conjunto de tecnologías y técnicas utilizadas para garantizar que las operaciones de una red tengan resultados predecibles. En el ámbito de aplicación de las normas de redes, se definen como atributos principales de QoS: la disponibilidad, el ancho de banda, la latencia y la tasa de error. De forma semejante, también se definen técnicas de priorización de tráfico de red, que son especialmente importantes para aplicaciones en tiempo real sobre Internet.

En contraste, la QoS para los servicios Web (y por consiguiente, también para los servicios Grid) se refiere a los atributos que caracterizan la calidad de un servicio Web. Además de los atributos de QoS relacionados con la red, estos pueden incluir: las prestaciones, la productividad, la fiabilidad, la escalabilidad, la robustez, la gestión de excepciones, la exactitud, la integridad, la accesibilidad, la disponibilidad, la interoperabilidad, y la seguridad. Todo ello tiene como objetivo dar soporte a técnicas que permitan a las aplicaciones seleccionar múltiples servicios equivalentes en funcionalidad, pero diferentes desde el punto de vista de los niveles de QoS que proporcionan.

El nivel de servicio o LoS define un comportamiento esperado para las prestaciones de un servicio. En general, hay tres parámetros que deben ser considerados para describir este comportamiento: el tiempo de respuesta promedio, la productividad esperada y la disponibilidad del servicio en el tiempo [9].

Al tratarse de un tema tan importante, existe un número considerable de trabajos, en diferentes estados de madurez, que tienen como objetivo proporcionar QoS en entornos de computación Grid (consultar la Sección 3.1 para una breve reseña de los trabajos previos). Todos estos trabajos tienen en común su intención de etiquetar y gestionar los recursos Grid para proporcionar diferentes niveles de servicio, y que todos ellos deben afrontar de alguna forma el problema de la escalabilidad para producir resultados consistentes.

La escalabilidad es una propiedad que puede estar relacionada con un sistema, una red, o un proceso, y que expresa la capacidad de la entidad en cuestión para gestionar cantidades crecientes de trabajo, manteniendo una tasa favorable de productividad, eficiencia o velocidad, y que también puede expresar la capacidad de la entidad para ser ampliada fácilmente. Así, se dice que un sistema distribuido es escalable cuando al añadir nuevos participantes al mismo, las prestaciones o el rendimiento total del sistema aumentan en proporción a las capacidades añadidas. De forma semejante, se dice que un algoritmo escala si éste puede ser aplicado, de forma convenientemente

eficiente y práctica, en problemas de gran dimensión (por ejemplo, en el caso de sistemas distribuidos, si el conjunto de datos de entrada, el número de nodos participantes, o ambos, son suficientemente grandes). En cambio, si el algoritmo falla cuando la dimensión del problema aumenta, entonces el algoritmo no escala.

El hecho es que, para cumplir con los SLA, los proveedores de servicio necesitan almacenar información de los recursos. Esto consume una cantidad considerable de procesamiento y, lo que es aún más importante y más difícil de resolver, adquirir la información en los recursos incrementa de forma significativa las comunicaciones. Al mismo tiempo, hay dos factores que se dan con frecuencia en el Grid y que también incrementan la complejidad y la cantidad de comunicaciones, ellos son: la presencia de un número grande de nodos y la dispersión geográfica de los mismos. Todo ello hace que diseñar algoritmos para adquirir la información en los recursos sea muy difícil: o bien los algoritmos dificultan el crecimiento del Grid, o bien las dimensiones del Grid dificultan la escalabilidad de los algoritmos.

El problema de la escalabilidad está estrechamente vinculado con el problema de obtener resultados consistentes. No se puede encontrar la asignación de recursos que minimice el coste computacional en un sistema distribuido sin que la solución incluya algún tipo de aproximación [10]. Este hecho, unido a la distribución geográfica, el dinamismo y la dimensión impuestos por el Grid hacen que las herramientas de clasificación tradicionales, tales como el análisis de clústeres, no resulten efectivas. Los algoritmos de análisis de clústeres no escalan bien en los sistemas distribuidos de gran dimensión, y su adaptación a estas condiciones, si no se tiene en cuenta el contexto descrito anteriormente, puede producir inconsistencias entre los resultados obtenidos en diferentes nodos del sistema.

De la observación de estos hechos surge la base sobre la cual se plantea la hipótesis que motiva este trabajo: una forma de abordar estos problemas puede ser acercar la información de los recursos a los servicios de la infraestructura que hacen la asignación de recursos, eliminando de esta forma el problema de la escalabilidad, lo que haría posible la utilización de métodos tradicionales de clasificación.

En este trabajo proponemos un nuevo modelo para la asignación de recursos en entornos de computación Grid que proporciona QoS a nivel de servicio, centrándose para ello en solucionar el problema de la escalabilidad. Como parte de este modelo, los servicios Grid son evaluados periódicamente a través de casos de prueba representativos, que son ejecutados en los recursos y que sirven para determinar la capacidad de los servicios para funcionar con unas prestaciones y una disponibilidad determinadas. A la vez, los recursos deben ser continuamente monitorizados para conocer su estado. De los resultados de estas pruebas y del estado de los recursos se obtiene una clasificación de los servicios que se utiliza en el momento de planificar la ejecución de un nuevo trabajo en el Grid. La segunda parte de este trabajo está enfocada a demostrar la aplicabilidad del modelo propuesto a la solución de un problema de planificación complejo: la asignación de recursos en el Grid sobre un modelo de optimización de costes de ejecución.

El problema de la escalabilidad se resuelve aislando las operaciones del Grid de las operaciones del sistema de monitorización. Estas últimas tienen lugar en una capa de componentes diseñada específicamente para beneficiarse de las características de la red, y de este modo organizar las comunicaciones de forma eficiente. Con el problema de la

escalabilidad resuelto, el resto del trabajo describe un nuevo algoritmo de asignación de recursos basado en un método de análisis de clústeres tradicional (el algoritmo de k-medias sin modificación [60]), y su aplicación para la asignación de recursos en base a costes. Para ello hemos desarrollado un modelo económico simple, pero a la vez muy flexible, que nos permite evaluar la aplicabilidad del modelo de asignación a este problema. El algoritmo de asignación de recursos, junto con el modelo económico y el sistema de monitorización, fueron implementados en una arquitectura software basada en Globus Toolkit [17]. Posiblemente, el resultado más prometedor de este trabajo es la consistencia que se obtiene entre los diferentes casos de prueba utilizados para evaluar el modelo de asignación.

A pesar de que no en todos los casos se cumplen todos los requerimientos, los resultados obtenidos demuestran el potencial del modelo propuesto para proporcionar niveles de servicio satisfactorios, fundamentalmente en aquellos casos de uso donde obtener tiempos de ejecución uniformes en diferentes ejecuciones de trabajos similares puede ser relevante para las prestaciones de una aplicación. Tal es el caso de aplicaciones que necesitan de cierto nivel de predictibilidad y consistencia, como por ejemplo en algunos problemas de optimización y simulación en los que el problema objeto de estudio se divide en varios trabajos más pequeños, y el resultado final depende del resultado de todos los trabajos, o en las simulaciones utilizadas en las predicciones meteorológicas o ante situaciones de emergencia. En estos casos, contar con una estimación, con ciertas garantías, del tiempo de ejecución de los trabajos en los diferentes recursos del Grid ayudaría a automatizar los procesos de monitorización y relanzamiento de los mismos (no sólo en caso de fallo, sino también por la predicción temprana de un fallo o la imposibilidad de cumplir un requerimiento), a la vez que mejoraría el tiempo total de ejecución en el Grid al ajustar los recursos a las aplicaciones más priorizadas.

1.2. OBJETIVOS

El objetivo general de esta tesis es desarrollar un nuevo modelo para la asignación de recursos en el Grid en base a requerimientos de QoS, utilizando atributos de servicios Grid. Para cumplir este objetivo general nos trazamos los siguientes objetivos específicos:

- Estudiar el problema de la asignación de recursos en entornos de computación distribuida, haciendo énfasis en el Grid.
- Estudiar los modelos planteados en la literatura que abordan el problema enunciado, especialmente aquellos que utilizan información de estado del sistema para evaluar las condiciones de selección de recursos.
- Diseñar un modelo para la asignación de recursos en el Grid, y sobre esta base desarrollar una arquitectura de software mediante la cual una aplicación Grid pueda ser organizada para prestar diferentes niveles de servicio a diferentes perfiles de usuario. La arquitectura debe incluir soporte para la diferenciación de servicios Grid en base a requerimientos de QoS a nivel de servicio, programas y bibliotecas software para la monitorización de recursos

en el Grid, y además debe proporcionar una estructura y una lógica de trabajo que pueda ser extendida por las aplicaciones Grid.
- Desarrollar una aplicación concreta para la arquitectura de software implementada para ser utilizada en la validación, y caracterizar el comportamiento de la aplicación en un entorno de computación Grid real.

Adicionalmente, es necesario realizar un estudio del middleware Grid y de las infraestructuras de computación Grid actuales. Sobre la base de las generalidades y particularidades encontradas en este estudio con respecto a la organización y gestión de los recursos, la planificación de trabajos y la organización de los sistemas de información Grid, identificamos el conjunto de procesos y tecnologías que fueron utilizados para implementar la arquitectura software de este trabajo. Este es el soporte sobre el cual integramos los diferentes componentes del modelo de asignación de recursos. El resultado final es un sistema poco intrusivo, que puede ser desplegado (en su mayor parte) en un par de servicios Grid, sin perjuicios para el resto de los servicios y las aplicaciones Grid.

1.3. ESTRUCTURA DEL DOCUMENTO DE TESIS DOCTORAL

Este documento de tesis está estructurado de la siguiente forma:

- El Capítulo 2 presenta una descripción del middleware y los entornos Grid actuales, haciendo especial énfasis en aquellos que han sido utilizados en la presente tesis para el desarrollo y despliegue de las aplicaciones finales.
- El Capítulo 3 presenta una revisión del estado del arte de la asignación de recursos en entornos de computación distribuida. Se plantea la formulación del problema desde el punto de vista matemático y se demuestra la imposibilidad de obtener una solución exacta. Adicionalmente, se hace un estudio de las soluciones aproximadas, haciendo especial énfasis en aquellas que han sido validadas en entornos de computación Grid.
- El Capítulo 4 describe el trabajo realizado con el objetivo de modelar, desde el punto de vista computacional, el problema de la asignación de recursos en entornos de computación Grid y su solución, específicamente utilizando algoritmos locales y técnicas de análisis de clústeres. Además, se aborda la descomposición del dominio de búsqueda de recursos en dominios de radio acotado por cierta distancia de red, se formula un método de diferenciación y asignación de recursos basado en la clasificación de servicios por sus capacidades y por el estado de los recursos que los soportan, y se propone un algoritmo de asignación de recursos en base a una función de costes.
- El Capítulo 5 describe GRIDIFF, una arquitectura de software diseñada para dar soporte de QoS a las aplicaciones Grid. Además de la estructura global de la misma, se describen los pormenores de la implementación.
- El Capítulo 6 describe los procesos de validación y evaluación de la arquitectura GRIDIFF y el resultado de los mismos, así como los detalles del

despliegue en un entorno Grid real. Por último se presenta un estudio de evaluación de la experiencia de usuario con datos obtenidos con un prototipo de uso libre.

- El último capítulo está dedicado a plantear las conclusiones del trabajo, así como las nuevas líneas de trabajo en desarrollo y futuro.

Capítulo 2

2. MIDDLEWARE GRID

Este capítulo presenta una descripción del middleware y los entornos Grid actuales, haciendo especial énfasis en aquellos que han sido utilizados en la presente tesis para el desarrollo y despliegue de las aplicaciones finales.

2.1. COMPUTACIÓN EN GRID

El concepto de Grid apareció por primera vez en los inicios de la década de 1990, como una propuesta para el desarrollo de infraestructuras computacionales avanzadas de soporte para la ciencia y la ingeniería. Esta idea estaba inspirada por la analogía con las infraestructuras de distribución energética, especialmente la red eléctrica, donde la ubicación y procedencia de las fuentes de energía es completamente irrelevante para el consumidor.

Todas estas características de las redes eléctricas son imperceptibles para los consumidores, que entienden la energía eléctrica como un recurso que consumen a través de equipos básicos con interfaces estándar, como neveras, televisores, videoconsolas, etc. De esta forma, la primera idea de Grid se dio a conocer a través de una visión de computación bajo demanda donde "la energía computacional" se convertiría en un recurso básico, con una forma de compra-venta semejante a la utilizada en la comercialización de la electricidad, el agua corriente, el gas, etc.

Sin embargo, las tendencias actuales de la computación en Grid se han alejado de esta noción inicial, moviéndose en una dirección más enfocada a un modelo orientado a servicios, fundamentalmente sobre la base de utilizar estándares de servicios Web (WSDL, SOAP, etc.). Mediante estas tecnologías se accede a recursos computacionales de almacenamiento y cálculo, y otros dispositivos más especializados, como sensores e instrumentos científicos, en forma de servicios Grid. Esta evolución ha sentado las bases para un nuevo concepto de Grid como plataforma de recursos virtuales, en forma de servicios sobre Internet.

La nueva definición de Grid se ha construido a partir de la Arquitectura orientada a servicios (service-oriented architecture o SOA) y los estándares y tecnologías ampliamente utilizados en los servicios Web, como el Protocolo Simple de Acceso a Objetos (Simple Object Access Protocol o SOAP), el Lenguaje de Descripción de Servicios Web (Web Services Description Language o WSDL), y el protocolo de Descripción, Descubrimiento e Integración Universal (Universal Description Discovery and Integration o UDDI). A estas tecnologías se ha sumado una nueva familia de especificaciones, con las que se definen y construyen la Arquitectura Abierta de Servicios Grid (Open Grid Services Architecture u OGSA), la Infraestructura Abierta de Servicios Grid (Open Grid Services Infrastructure u OGSI), el Marco de Recursos de Servicios Grid (Web Services Resource Framework o WSRF), y la WS-ResourceLifetime. Estos grupos de especificaciones extienden a los servicios Web tradicionales con una definición estándar para nuevas características, como el estado y

el tiempo de vida, haciéndolos más apropiados para gestionar y compartir recursos en entornos de computación en Grid.

En cambio, la computación basada en cloud sí que responde, por el momento, a esta noción de computación bajo demanda. En este modelo, se accede a "horas CPU" suministradas por un proveedor, generalmente comercial, y que se facturan según el consumo y calidad. En cambio, el modelo de computación en Grid presenta limitaciones que condicionan las aplicaciones que pueden beneficiarse.

Normalmente, para que puedan ser ejecutadas de forma eficiente en el Grid, se considera que las aplicaciones tienen que cumplir con el conjunto de requisitos que se describe a continuación [20]:

- La aplicación debe poder ser dividida en varias sub-tareas, que puedan ser ejecutadas en paralelo sin que haya comunicación, o bien, que esta sea muy escasa, de forma tal que las sub-tareas sean independientes entre sí, o tengan muy pocas dependencias con otras sub-tareas.
- Si al retirar algunos nodos, el coste por continuar un trabajo que ejecuta una sub-tarea en el Grid, es muy bajo; y si, además, el trabajo puede utilizar nodos adicionales que son añadidos al sistema.

Sin embargo, los sistemas Grid existentes no permiten explotar la segunda característica al máximo. Precisamente, la elasticidad es una característica de los sistemas de computación bajo demanda, que proporciona una forma eficiente de utilizar más o menos recursos en dependencia de las necesidades de las aplicaciones. Por ejemplo, la computación basada en cloud permite utilizar más recursos si se produce un pico en la demanda de un servicio, y apagar recursos que no están siendo utilizados. En consecuencia, el coste por migrar un trabajo en el Grid es, generalmente, elevado. Esta es otra de las razones para mejorar los procesos de planificación y asignación de recursos.

Por otra parte, la necesidad de disponer de e-infraestructuras ha favorecido el desarrollo de middleware Grid, destacando Globus Toolkit [17], UNICORE [18] y gLite [19], que constituyen un núcleo muy sólido sobre el cual se soportan la mayor parte de los sistemas Grid más importantes de la actualidad.

2.2. GLOBUS TOOLKIT

Globus Toolkit se distribuye libremente bajo una licencia del estilo Apache, y proporciona un conjunto de herramientas, protocolos y bibliotecas software de seguridad, gestión de recursos, manejo de información y transferencia de datos en el Grid. Una buena parte del middleware Grid actual (incluyendo gLite) está basado en la pila de componentes de Globus Toolkit. En consecuencia, las investigaciones en el campo de las aplicaciones Grid están enfocadas, principalmente, al diseño de modelos que integran las nuevas tecnologías del middleware Grid en la solución de problemas.

Una de las áreas de investigación más activas en el tema del middleware Grid está enfocada a mejorar la interoperabilidad entre los componentes del mismo. En este

sentido, la OGSA proporciona una especificación libre de la arquitectura de servicios Grid, garantizando que los componentes de software que se adhieran a las especificaciones de la OGSA sean interoperables entre sí, aún si estuvieran basados en implementaciones diferentes. Para ello, la OGSA necesita una infraestructura que proporciones los medios para poder desarrollar los componentes del Grid. Esta infraestructura es la WSRF, que se implementa extendiendo los servicios Web.

Globus Toolkit 4 (GT4), la última versión de Globus Toolkit, proporciona un amplio conjunto de herramientas software que dan soporte para el desarrollo de componentes con interfaces de servicios Web. Asimismo, GT4 también proporciona un conjunto de componentes middleware que se ocupan, entre otras cosas, del manejo de mensajes, la gestión de recursos y de la seguridad, lo cual permite a los desarrolladores de aplicaciones Grid centrar su atención en implementar la lógica de la aplicación. Además, GT4 proporciona un entorno de ejecución que incluye contenedores de servicios Web, específicos para GT4, que permiten desplegar y gestionar servicios, escritos en Java, C y Phyton. Para ello, GT4 proporciona una implementación de WSRF sobre la cual están implementados la mayoría de estos servicios.

Un servicio Web es un sistema de software diseñado para dar soporte a la interacción entre máquinas sobre una red, que está enfocado, principalmente, a facilitar la interoperabilidad entre aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma. Los servicios Web se basan, en gran medida, en las tecnologías Web (protocolos HTTP/HTTPS, etc.), aunque el objetivo de un servicio Web no es publicar información en un sitio Web, sino intercambiar información entre aplicaciones. La aplicación que quiere acceder a la información, contacta con el servicio Web y envía una solicitud, en una forma muy semejante a como lo haría un usuario con un sitio Web, desde su navegador. Luego, el servicio Web envía una respuesta a la aplicación.

Una característica muy importante que heredan los servicios Web de las tecnologías Web es la falta de estado. Esto significa que un servicio Web no puede mantener el estado de una invocación a otra. Una buena parte de las aplicaciones Web funcionan bajo este paradigma, porque no necesitan conocer el estado, más allá del momento de la ejecución de las mismas. En cambio, en la mayoría de los casos, las aplicaciones Grid necesitan mantener el estado. Esta es la razón por la que GT4 proporciona una implementación de la especificación WSRF, para permitir que los servicios Web mantengan información de estado. No obstante, WSRF mantiene la estructura carente de estado de los servicios Web, incorporando el concepto de recurso como la entidad que gestiona el estado. El enlace entre servicio y recurso lo proporciona el cliente y el gestor de recursos. Cada recurso tiene un identificador único, que permite vincular el estado de un servicio Web con una sesión particular, entendiendo como sesión un flujo de solicitudes desde una aplicación cliente.

De esta forma, para implementar un servicio Web con GT4, son necesarios seis pasos:

1. Definir la interfaz de servicio, utilizando una versión extendida del lenguaje de especificaciones de servicio Web WSDL.
2. Implementar el servicio, utilizando la API de Java GT4.
3. Implementar el recurso, utilizando la API de Java GT4.
4. Definir los parámetros de despliegue del servicio, utilizando un Descriptor de

Despliegue de Servicio Web (Web Service Deployment Descriptor o WSDD) y la Interfaz de Nombrado y Directorio de Java (Java Naming and Directory Interface o JNDI).

5. Compilar todo y generar un archivo GAR, que contiene lo necesario para realizar el despliegue del servicio en el contenedor de servicios Web de GT4, utilizando ant, una herramienta software que permite automatizar los procesos de compilación y construcción de software.
6. Desplegar el servicio, utilizando las herramientas proporcionadas por GT4.

De acuerdo con la implementación de los recursos, los servicios Web de GT4 pueden ser: recursos singleton y recursos múltiples. El recurso singleton divide la implementación en recurso, gestor de recursos y servicio, utilizando clases diferentes para el servicio y el recurso. Cuando se trabaja con múltiples recursos a la vez, la especificación de WSRF recomienda utilizar un patrón factoría-instancia, que consiste en dividir el servicio en un servicio factoría, encargado de crear los recursos, y un servicio instancia, encargado de acceder a la información contenida en los recursos.

La información de estado del servicio es almacenada en un recurso, específicamente en una clase WSResourceProperties, definida por WSRF. Además, WSRF define un conjunto de interfaces (*PortTypes*) para interactuar con las propiedades de los recursos de un servicio. En WSDL, cada "PortType" identifica a un conjunto de operaciones abstractas y de mensajes, también abstractos, involucrados en estas operaciones. En general, los "PortTypes" definidos por WSRF permiten recuperar y modificar las propiedades de uno o varios recursos a la vez, ya sea accediendo por el nombre completo de los recursos, que no es más que una combinación de un nombre local y un identificador uniforme de recurso (Uniform Resource Identifier o URI), o bien interrogando a los recursos, utilizando el lenguaje XPath.

Adicionalmente, es posible gestionar el tiempo de vida de los recursos, utilizando la especificación WS-ResourceLifetime. La forma más simple es la destrucción inmediata, que consiste en una operación que permite destruir un recurso desde el servicio instancia. Además de esta forma, WS-ResourceLifetime proporciona otra forma más elaborada de gestión del tiempo de vida de los recursos, que consiste en planificar el momento específico en que ocurrirá la destrucción de los mismos.

Por último, GT4 soporta varios mecanismos de notificación, a través de la especificación WS-Notification. De esta forma, es posible exponer las propiedades de un recurso a través de las subscripciones de los consumidores, recibiendo notificaciones cada vez que cambia el valor de las propiedades.

GT4 proporciona varios mecanismos de seguridad que permiten proteger a los contenedores de servicios Web, sin modificar las interfaces de los servicios. Estos mecanismos forman parte de la Infraestructura de Seguridad Grid (Grid Security Infrastructure o GSI), que consisten en un conjunto de herramientas para gestionar certificados digitales y un conjunto de clases de Java que permiten integrar los mecanismos de seguridad en los servicios Web. Específicamente, GSI[1] aporta las

[1] Se refiere a la versión de GSI para GT4 porque otras versiones anteriores (Globus Toolkit 2) no utilizaban servicios Web.

siguientes características, basadas principalmente en el modelo de la infraestructura de clave pública (Public Key Infrastructure o PKI):

- Seguridad a nivel de transporte, de sesión y de mensaje.
- Autenticación a través de certificados digitales X.509.
- Varios esquemas de autorización.
- Delegación de credenciales e inicio de sesión único (*single sign-on*).
- Diferentes niveles de seguridad: contenedor, servicio y recurso.

GSI da la posibilidad de activar la seguridad a tres niveles: el nivel de transporte, el nivel de sesión, o el nivel de mensaje. La seguridad a nivel de transporte consiste en cifrar toda la comunicación que se intercambia entre una aplicación cliente y el servicio Web. Por otro lado, la seguridad a nivel de mensaje consiste en cifrar solamente el contenido de los mensajes SOAP, mientras que el resto del mensaje viaja por la red en texto plano. Finalmente, la seguridad a nivel de sesión (Secure Conversation) permite mantener la misma estrategia de cifrado y clave durante todos los mensajes de la sesión, reduciendo el sobrecoste de la negociación. La utilización de un nivel de seguridad u otro depende de los requerimientos de cada aplicación, pero en general la seguridad a nivel de mensaje, o sesión, tiene mejores prestaciones que la seguridad a nivel de transporte, porque la cantidad de datos cifrados en menor, mientras que la seguridad a nivel de transporte es más compatible, porque se basa en la pila de protocolos de Seguridad de la Capa de Transporte (Transport Layer Security o TLS) para proporcionar comunicaciones seguras por la red.

El estándar X.509 para la PKI, es una de las bases principales que soportan a GSI. En consecuencia, el método principal de autenticación de GSI es a través de certificados digitales X.509, aunque también soporta autenticación basada en usuario y contraseña, así como autenticación anónima. Asimismo, GSI soporta varios mecanismos de autorización del lado del servidor y del lado del cliente, algunos de ellos basados en certificados X.509. Adicionalmente, es posible añadir nuevos mecanismos de autorización a GSI. Del lado del servidor, es posible utilizar los siguientes mecanismos de autorización:

- Ninguna autorización: No se realiza autorización.
- Propia: Un cliente será autorizado a acceder a un servicio, si la identidad del cliente coincide con la identidad del servicio.
- Gridmap: Se especifica mediante un fichero de texto una lista de nombres distintivos (del campo sujeto de los certificados X.509), similar a una lista de control de acceso (ACL), que están autorizados para acceder a un servicio.
- Autorización de identidad: Un cliente será autorizado a acceder a un servicio, si la identidad del cliente coincide con una identidad específica. Este método es muy similar al anterior, con la diferencia de que el nombre distintivo del cliente autorizado se especifica de forma programática y no en un archivo gridmap.
- Autorización de host: Un cliente será autorizado a acceder a un servicio, si

presenta una credencial de host que coincida con un nombre de host específico.

- Autorización de llamadas SAML: Es posible delegar las decisiones de autorización a un servicio de autorización OGSA compatible, a través del módulo de autorización de llamadas en Lenguaje de Marcas de Afirmación de Seguridad (Security Assertion Markup Language o SAML).

Del lado del cliente, es posible utilizar los siguientes mecanismos de autorización:

- Ninguna autorización: No se realiza autorización.
- Propia: Un cliente autorizará una invocación, si la identidad del servicio coincide con la identidad del cliente.
- Autorización de identidad: Un cliente autorizará una invocación, si la identidad del servicio coincide con una identidad específica.
- Autorización de host: Un cliente autorizará una invocación, si el servicio presenta una credencial de host que coincida con un nombre de host específico. Además, el cliente tiene que ser capaz de resolver la dirección de Internet del nombre de host especificado en la credencial de host.

La autorización del lado del servidor es bastante común, pero la autorización del lado del cliente se utiliza en casos muy específicos y por ello es menos conocida. Por ejemplo, el "phishing" es un tipo de estafa cibernética caracterizado por intentar adquirir información confidencial de forma fraudulenta (como puede ser una contraseña o información detallada sobre tarjetas de crédito). En una de las modalidades de este delito, el estafador se hace pasar por una empresa de confianza en una aparente comunicación oficial, por lo común un correo electrónico, y utiliza algún engaño para mostrar un enlace que parezca la organización por la cual se hace pasar el impostor. En el enlace, se pedirán datos personales y números de cuenta bancaria, que luego podrán ser utilizados por el estafador.

El caso de "phishing" descrito anteriormente puede evitarse si el cliente utiliza un mecanismo de autorización. La autorización del lado del cliente permite validar la identidad del servicio al que se conecta el cliente, por ejemplo en una autorización de host, el servidor tiene que presentar una credencial de host, firmada por una autoridad certificadora de confianza, que certifique que su nombre de host coincide con un nombre de host conocido por el cliente (por ejemplo, www.mibanco.com).

La delegación de credenciales y el inicio de sesión único son dos de las características más importantes de GSI, y son posibles mediante el uso de certificados proxy X.509. Un certificado proxy permite que un servicio actúe en nombre de un usuario. Esto se conoce como delegación de credenciales, porque los certificados proxy permiten delegar, de forma efectiva, un conjunto de credenciales de un usuario (la identidad del usuario) a otro usuario o servicio. Además, los certificados proxy hacen posible el inicio de sesión único. En la práctica, un usuario tiene asignados un par de claves pública y privada que le sirven para identificarse en el Grid. Sin embargo, en lugar de utilizar directamente la clave privada para autenticarse con todas las organizaciones que

lo requieran, el usuario utilizará la clave privada para firmar un certificado proxy, que utilizará para autenticarse. De esta forma, el usuario utiliza la clave privada solamente una vez, para crear un certificado proxy que autoriza al usuario a actuar en nombre propio, durante un tiempo limitado (normalmente 12 horas). Este mecanismo asegura que, durante el tiempo de validez del certificado proxy, el usuario dispondrá de una especie de pase que le permitirá acceder al Grid, sin necesidad de utilizar su clave privada. De esta forma, los usuarios no tienen que exponer o transferir la clave privada fuera del entorno seguro donde activan el proxy.

Por último, GSI permite configurar la seguridad a diferentes niveles. Por ejemplo, es posible utilizar un mecanismo de autorización para un servicio y otro diferente para sus recursos, de forma tal que las operaciones sin estado puedan ser ejecutadas sin autorización, pero las operaciones con estado requieran autorización de algún tipo, como autorización de identidad o de host.

2.2.1. SERVICIOS DE GT4

GT4 proporciona otros componentes middleware que desempeñan tareas muy importantes, como son los servicios de información, la gestión de datos, y los servicios de gestión de ejecuciones.

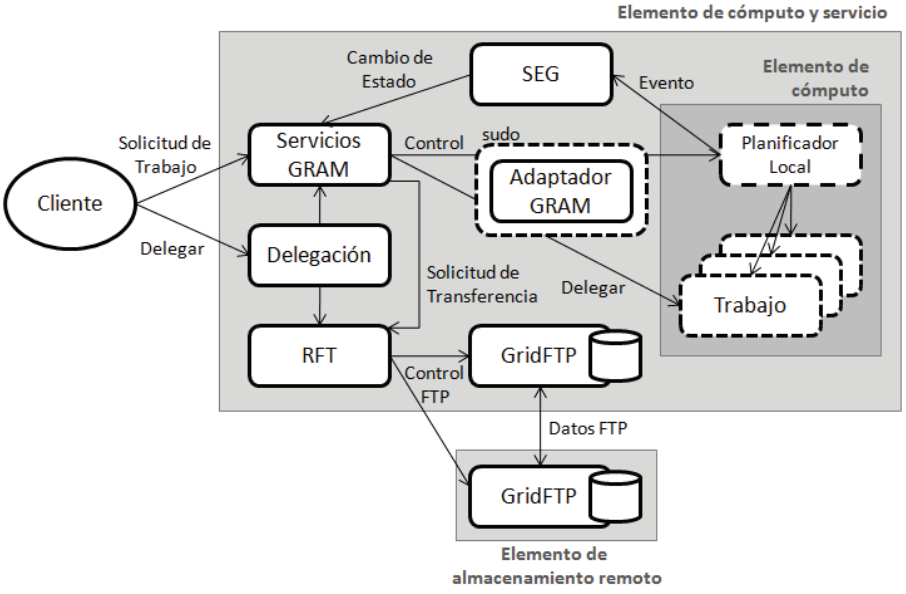
El sistema de monitorización y descubrimiento (Monitoring and Discovery System o MDS) es una interfaz colectiva para los diferentes servicios de información proporcionados por GT4. Cada uno de los servicios de información puede ser accedido de forma independiente, pero MDS proporciona un mecanismo estándar que los integra, y que permite publicar y recuperar información de configuración y de estado de los recursos computacionales en general (no sólo de los recursos de los servicios Web). Adicionalmente, MDS permite suscribir temas nuevos. Para ello, MDS proporciona varios mecanismos de suscripción y recogida de información que pueden ser utilizados por los servicios Web para suscribir sus recursos con el sistema de información. El mecanismo más simple consiste en utilizar encuestas programadas, a través de un colector que utiliza los "PortTypes" definidos por WSRF para recuperar, periódicamente, las propiedades de un recurso de un servicio. Otro mecanismo más complejo consiste en utilizar notificaciones, a través de un componente que se suscribe a un tema que contiene las propiedades de un recurso de un servicio, y que notifica cuando ocurren cambios en las mismas.

En cuanto a la gestión de datos, el componente principal de GT4 es el protocolo GridFTP, que es una extensión del protocolo FTP que tiene soporte para GSI y que añade algunas características al protocolo original, como son la transferencia de datos en paralelo, la transferencia parcial de archivos, y la replicación de datos, que es un conjunto de componentes que posibilitan la gestión de réplicas de archivos, distribuidas entre los componentes de almacenamiento del Grid, para dar soporte a aplicaciones que necesitan acceder, de forma eficiente, a un conjunto de datos remotos y aplicaciones que necesitan trabajar con conjuntos de datos de gran tamaño, que no pueden ser almacenados localmente en un único componente de almacenamiento.

Los servicios para la asignación y gestión de recursos Grid (Grid Resource Allocation and Management o GRAM) constituyen el núcleo de la gestión de ejecuciones en GT4. Estos servicios proporcionan los medios para lanzar y monitorizar trabajos en el Grid.

GRAM utiliza un componente de GSI llamado Gatekeeper, que valida las credenciales y cambia el identificador de usuario al usuario local asignado en el nodo, antes de ejecutar el trabajo. Los requerimientos de los trabajos son especificados en un documento XML de descripción de trabajo. Además de describir el trabajo, este documento permite copiar el ejecutable de la aplicación al nodo de trabajo, mover la entrada estándar, la salida estándar, la salida de error estándar y otros archivos, hacia los elementos de almacenamiento remotos, etc. Por último, GRAM crea un gestor vinculado al trabajo (JobManager), que monitoriza el mismo durante todo su tiempo de vida y envía esta información al sistema de información. De esta forma, GRAM mantiene un interfaz común para los diferentes servicios locales de ejecución de trabajos (fork, gestores de colas, MPI, etc.), cuyas particularidades son tenidas en cuenta por el JobManager.

La **Figura 2-1** muestra un diagrama de la arquitectura de componentes de GRAM en GT4. Un adaptador se ocupa de enviar las solicitudes de trabajo al planificador local, que gestiona la ejecución de los mismos. El Generador de eventos del planificador (Scheduler Event Generator o SEG) monitoriza el estado de los trabajos. Para ello, el cliente delega sus credenciales a través del servicio de Delegación de GT4. GRAM utiliza las credenciales de usuario para ejecutar trabajos y transferir archivos. El servicio para la transferencia fiable de archivos (ReliableFileTransfer service o RFT) sirve de intermediario entre GridFTP y los servicios GRAM, y se ocupa de gestionar las transferencias fallidas.



Fuente: http://www.globus.org/toolkit/docs/4.2/4.2.1/execution/key/

Figura 2-1: Arquitectura de componentes de GRAM en GT4.

En resumen, GRAM proporciona un conjunto de herramientas (globusrun-ws, etc.) que permiten, a los usuarios autenticados por un proxy válido y debidamente autorizados, lanzar trabajos a varios tipos de planificadores de trabajos (Condor [53], PBS [56], etc.), utilizando un descriptor de trabajo con un formato estándar.

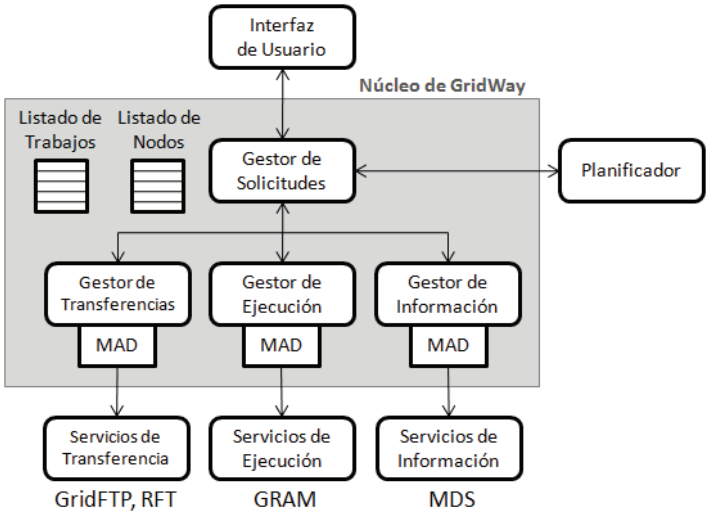
Sin embargo, GRAM únicamente proporciona un acceso estandarizado a uno a más recursos locales gestionados a partir de los JobManager. La elección del recurso GRAM constituye una tarea de alto nivel que se aborda desde la perspectiva de la meta-planificación.

2.2.2. META-PLANIFICACIÓN SOBRE GT4

Un meta-planificador es un supervisor de gestores de recursos locales, que controlan la utilización de recursos individuales como clústeres o supercomputadoras. Por ello, la meta-planificación es un componente clave del Grid porque es responsable de optimizar la utilización de los recursos. GT4 no incorpora, en sí mismo, ningún meta-planificador propio, aunque sí incluye enlaces a desarrollos externos certificados por Globus, como GridWay. Además, otros proyectos han desarrollado sus propias soluciones de meta-planificación sobre GT4, tal es el caso de Condor-G, el meta-planificador de Condor.

GridWay [21] es un gestor de carga de trabajo que permite federar la ejecución de trabajos en un conjunto de recursos computacionales gestionados por diferentes sistemas (PBS, SGE [54], LSF [55], Condor, etc.). Desde el punto de vista computacional, esta herramienta es una implementación de la API para las aplicaciones de gestión de recursos distribuidos (Distributed Resource Management Application API o DRMAA). Esta especificación del Open Grid Forum (OGF) describe una API para el lanzamiento y control de trabajos en uno o más sistemas de gestión de recursos distribuidos (Distributed Resource Management System o DRMS), en una arquitectura Grid. Adicionalmente, GridWay proporciona una interfaz de usuario para la consola (Command Line Interface o CLI) que sustituye a GRAM en la mayoría sus tareas (lanzar y controlar trabajos, monitorizar trabajos, etc.), pero a diferencia de éste, GridWay utiliza el lenguaje de descripción para el lanzamiento de trabajos (Job Submission Description Language o JSDL), que es un estándar del OGF. Con respecto a la usabilidad, GridWay mejora algunos aspectos de la ejecución, porque implementa mecanismos de detección de fallos que monitorizan los sistemas locales de gestión para detectar trabajos cancelados, fallos de sistema y desconexiones de red, y relanzan automáticamente los trabajos afectados. Para ello, GridWay se apoya, principalmente, en la migración de trabajos.

La **Figura 2-2** muestra un diagrama de la arquitectura de planificación de GridWay. Un gestor de solicitudes se encarga de gestionar las transferencias de datos, la ejecución de trabajos y la información de los servicios. Para ello, se conecta con los servicios de GT4 necesarios, a través de un conector de middleware (Middleware Access Driver o MAD).



Fuente: http://www.globus.org/toolkit/docs/4.2/4.2.1/execution/gridway/admin/

Figura 2-2: Arquitectura de planificación de GridWay.

Además de gestionar la ejecución de trabajos, GridWay participa como intermediario en la asignación de recursos. Para este fin, el proceso de planificación combina información de los recursos, recogida en diferentes momentos. Básicamente, GridWay utiliza los últimos valores de las tasas de migración y fallo, así como estadísticas de ejecución (tiempos de transferencia, ejecución y de espera en las colas), y los combina con valores históricos de los recursos. Asimismo, el planificador de GridWay soporta un conjunto de políticas que permiten dar prioridades diferentes a los trabajos. Sin embargo, GridWay no garantiza que los recursos del Grid cumplan estas políticas, porque el orden en que ocurre la ejecución de trabajos depende, entre otras cosas, del usuario que envía el trabajo y de que cada usuario del Grid tiene sus propios derechos de acceso y su historial de utilización de los recursos. Por último, GridWay intenta mejorar la ejecución y la utilización de los recursos utilizando técnicas de migración de trabajos. La migración de trabajos consiste en cancelar trabajos que ya han sido planificados en un recurso, y relanzarlos en un recurso diferente. GridWay soporta cuatro tipos de migración: migración oportunista (ocurre cuando el planificador encuentra un recurso mejor), migración de trabajos suspendidos durante mucho tiempo en la cola de un recurso, migración de trabajos porque las aplicaciones han cambiado sus requerimientos, y migración de trabajos por degradación en las prestaciones del recurso. Excepto la migración oportunista, el resto son medidas de recuperación, porque tienen como objetivo corregir un comportamiento incorrecto del sistema. En consecuencia, nada garantiza que los recursos tengan un nivel mínimo para cumplir con los requerimientos específicos de cada trabajo. Por otra parte, la migración oportunista puede mejorar, en ciertos casos, las prestaciones conseguidas en la ejecución de un trabajo, pero a la vez puede crear problemas de sobrecarga en las redes y los recursos, así como infrautilización de algunos recursos.

Finalmente, GridWay se instala a nivel local, compitiendo entre sí diferentes instancias desconectadas del mismo (no existe un servicio). Esto, sumado a la escasa actualización de la información del sistema dificulta aún más la toma de decisiones en la planificación de recursos.

Condor [53] es un sistema de computación de alta productividad, que permite gestionar trabajos en un entorno de computación distribuida. Este sistema proporciona un planificador de trabajos que implementa varias políticas de planificación, como por ejemplo planificación por prioridades. Además, proporciona mecanismos de monitorización y gestión de recursos.

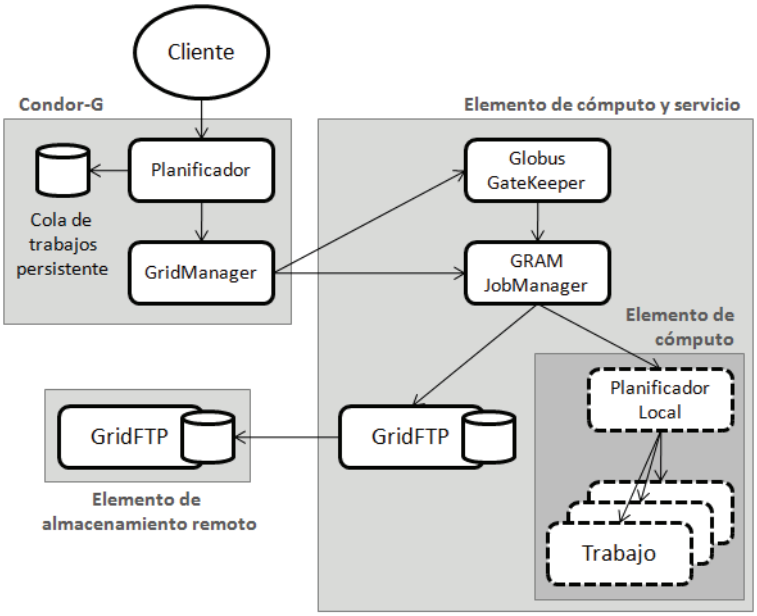
Sin embargo, Condor está enfocado a ejecutar trabajos dentro de un único dominio administrativo. Para ello mantiene un registro dinámico de los recursos de la organización que se encuentran inactivos. Este registro es utilizado por el planificador de Condor para asignar recursos para la ejecución de trabajos.

Condor-G [22] es un meta-planificador que combina Condor con GT4. Mientras que Condor aporta los métodos de gestión de recursos locales, GT4 contribuye con los métodos para establecer canales de comunicación seguros entre diferentes dominio administrativos, y proporciona acceso estandarizado a diferentes sistemas de colas. De esta forma, es posible utilizar Condor para gestionar recursos en uno o varios dominios administrativos, que utilizan GT4 como puente de comunicación entre ellos.

La **Figura 2-3** muestra un diagrama que representa la forma en que Condor-G implementa la gestión de trabajos. Condor-G proporciona una API y una CLI que permiten a los usuarios enviar trabajos, interrogar al sistema para conocer el estado de los trabajos, y acceder a las trazas de ejecución. Para esto, Condor-G crea un gestor vinculado a los trabajos de cada usuario (GridManager). El GridManager se encarga de enviar las solicitudes de trabajo al Gatekeeper de GT4, y además recibe las actualizaciones de estado de los trabajos, a través del JobManager de GRAM, y se las envía al planificador de Condor. Por su parte, el JobManager transfiere el ejecutable, a través de GridFTP, y se ocupa de proporcionar las E/S estándares, y además envía el trabajo a la cola local de los recursos (PBS, Condor, etc.).

Además de gestionar la ejecución de trabajos, Condor-G se ocupa de renovar las credenciales caducadas durante la ejecución de un trabajo. Cuando un trabajo tarda mucho tiempo en la cola de espera o en la ejecución, es posible que expiren las credenciales proxy. En ese caso, es necesario generar un proxy nuevo y enviarlo a los nodos de trabajo. Condor-G se ocupa de analizar periódicamente las credenciales de todos los usuarios que tienen trabajos en las colas. En caso de que alguna de las credenciales haya caducado, Condor-G se ocupa de hacer la renovación, y de enviar el nuevo proxy a los sitios remotos que estén relacionados con la ejecución del trabajo.

Sin embargo, la visión de los recursos que proporciona Condor-G es más apropiada para un clúster heterogéneo que para un Grid. El planificador de Condor-G mantiene una lista de todos los recursos disponibles, y utiliza GRAM para enviar trabajos a los sistemas de gestión de colas de los recursos locales. Esta visión de un conjunto de colas agrupadas para gestionar la ejecución de trabajos en un conjunto de recursos distribuidos es apropiada para trabajos tipo "batch" (por lotes), pero no tiene en cuenta otros requerimientos como la capacidad de almacenamiento o de la red, que son necesarios para trabajos interactivos, trabajos que acceden a grandes cantidades de datos distribuidos, etc.



Fuente: http://www.cs.wisc.edu/condor/manual/v7.3/5_3Grid_Universe.html

Figura 2-3: Gestión de trabajos en Condor-G.

Por el contrario, se espera que un meta-planificador Grid, además de gestionar la ejecución de trabajos en los nodos, se ocupe de optimizar la utilización de los recursos. Esta tarea incluye la responsabilidad de consultar los sistemas de información Grid para descubrir las características de los servicios disponibles y fijar a qué recursos se deben enviar los trabajos, así como de de trazar un plan para mover la menor cantidad de datos posibles por la red, por ejemplo, consultando el servicio de gestión de réplicas (Replica Location Service o RLS), que proporciona un registro de las réplicas en los sistemas de almacenamiento. Además de ello, Condor-G no tiene un sistema central de gestión de trazas que registre la ejecución de trabajos en el Grid, por lo que tampoco puede utilizar información histórica para la planificación de nuevos trabajos.

A modo de conclusiones de esta parte, Globus Toolkit es un middleware Grid, de código abierto, que permite construir sistemas de computación en Grid, además de dar soporte para el desarrollo de aplicaciones Grid. Además, Globus Toolkit 4 o GT4 es una implementación de un conjunto de estándares de servicios Web y de Grid, entre los que destacan: OGSA, WSRF, JSDL, DRMAA, WS-Management, WS-BaseNotification, SOAP, WSDL y GSI, careciendo de un sistema eficiente para la planificación de trabajos.

2.3. OTROS MIDDLEWARE GRID

gLite es el middleware Grid desarrollado como parte del proyecto EGEE. Basado parcialmente en Globus Toolkit 2, aún mantiene compatibilidad con algunos componentes de este middleware, aunque gLite es actualmente un sistema completamente independiente, que en algunos aspectos supera a GT4, como por ejemplo en la gestión de identificadores lógicos y réplicas de datos o metadatos, y en la planificación de trabajos. Se distribuye libremente bajo una licencia del estilo Apache, aunque su instalación está limitada, por aspectos técnicos, a plataformas muy específicas, de las que destaca Scientific Linux, una distribución de Linux que tiene un modelo de desarrollo estrechamente vinculado a la Organización Europea para la Investigación Nuclear (European Organization for Nuclear Research o CERN). Posiblemente, el punto de contacto más importante entre Globus Toolkit y gLite se encuentra en la utilización de GSI como infraestructura de seguridad, aunque gLite incluye importantes modificaciones que le permiten utilizar VOMS (Virtual Organization Membership Service), un sistema de autorización basado en afirmaciones incluidas como extensiones en los certificados digitales X.509.

Precisamente, el entorno VOMS implementa un concepto básico en Grid, la organización virtual (VO). Las organizaciones virtuales proporcionan un entorno que permite a cada proveedor de servicios especificar a nivel global de la VO, los recursos quiere compartir, quiénes están autorizados a utilizar estos recursos y los términos en los cuáles se produce la utilización de los recursos compartidos.

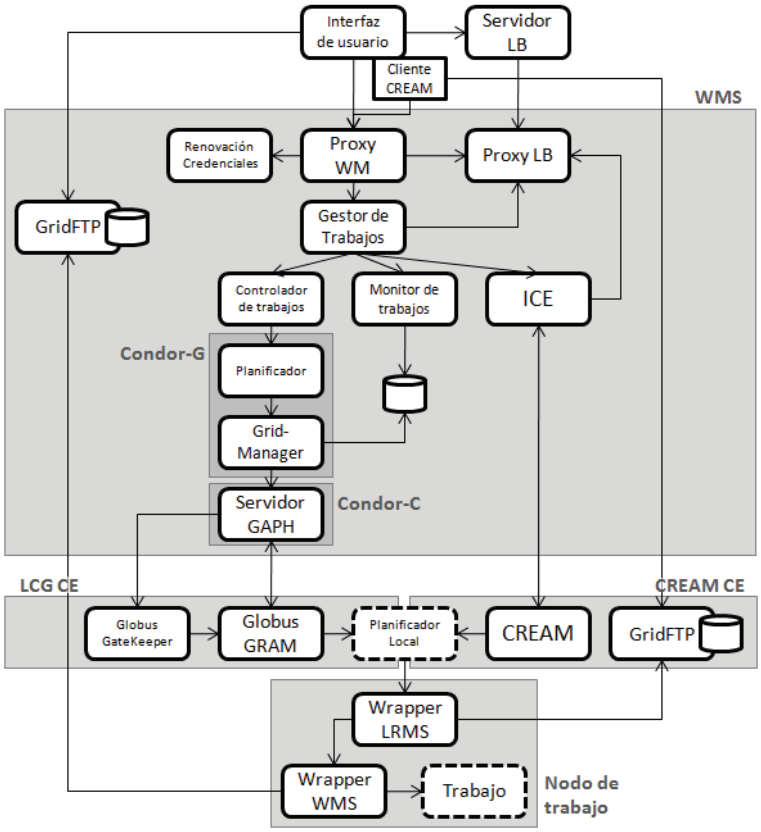
De esta forma, se gestiona la autorización a nivel global y centralizado, mejorando la escalabilidad y sin perjuicio de la independencia de gestión de los proveedores, que siempre tienen la última palabra.

Al conjunto de individuos y/o organizaciones que establecen las normas de dicho reparto se le conoce colectivamente como organización virtual (VO). VOMS proporciona el entorno que soporta el modelo de seguridad de las organizaciones virtuales. En esencia, el mecanismo de inicio de sesión único proporcionado por GSI, en conjunto con el mecanismo de gestión de la información de autorización proporcionado por VOMS, permiten que un usuario se autentique en una VO, sea autorizado por los servicios de la misma, y pueda descubrir y operar con recursos distribuidos, coordinados por organizaciones virtuales multi-institucionales.

El sistema de gestión de la carga de trabajo (Workload Management System o WMS) es un conjunto de componentes middleware que se ocupan de distribuir, gestionar y monitorizar la ejecución de trabajos en los recursos Grid. WMS soporta varios tipos de trabajos (batch, workflow, MPI, interactivos, etc.). Las características de cada trabajo son definidas en JDL (Job Description Language), un lenguaje desarrollado a partir del conjunto de etiquetas (ClassAds) utilizadas por Condor para representar las características de los trabajos. Para describir los recursos se utiliza GLUE (Grid Laboratory Uniform Environment), un esquema que estandariza la forma de describir los recursos Grid y facilita la interoperabilidad entre diferentes infraestructuras.

La **Figura 2-4** muestra un diagrama que representa la gestión de trabajos con gLite WMS. Teniendo en cuenta que el proyecto gLite ha previsto la sustitución de gLite CE por CREAM CE, sólo han sido representados LCG CE y CREAM CE. WMS envía trabajos al sistema de gestión de recursos local (Local Resource Management System o

LRMS) de los CE disponibles en el Grid. En cada CE, el planificador local se encarga de asignar los nodos para ejecutar los trabajos.



Fuente: https://twiki.cern.ch/twiki/bin/view/EGEE/EGEEgLiteJobSubmissionSchema

Figura 2-4: Gestión de trabajos en gLite WMS.

En cuanto a seguridad, WMS utiliza GSI para la autenticación y para establecer canales de comunicación seguros. Al igual que Condor-G, proporciona un componente (Proxy Renewal) que se encarga de renovar automáticamente las credenciales para los trabajos de larga duración.

Para decidir qué recurso es el adecuado para ejecutar un trabajo, WMS compara los requerimientos de la solicitud con las características de los recursos disponibles. La disponibilidad de un recurso, para un trabajo en particular, depende del estado del recurso y de las políticas de utilización de la VO. Es posible que varios recursos a la vez cumplan los requerimientos para la ejecución de un trabajo. En este caso, WMS organiza los mismos utilizando una función de evaluación, que puede ser suministrada

en el JDL, y utiliza el recurso con mayor puntuación. Por ejemplo, en la función de evaluación es posible indicar que se envíe el trabajo al recurso con el menor tiempo estimado de permanencia en la cola. Para evitar la sobrecarga de los recursos con mejor puntuación, WMS utiliza un algoritmo estocástico que tiene como objetivo alternar en el tiempo entre los recursos mejor puntuados. En la comparación, también es posible definir atributos para los recursos de almacenamiento. Para ello, WMS consulta el catálogo de datos para obtener un listado de posibles recursos de almacenamiento, teniendo en cuenta que estos tengan los datos de entrada necesarios para el trabajo, y además, espacio suficiente para almacenar los datos de salida.

Una de las tareas más importantes de WMS consiste en meta-planificar la ejecución de los trabajos. En este sentido, WMS proporciona varias políticas de planificación. En un extremo está la política de asignar los recursos para la ejecución de un trabajo tan pronto como sea posible, y enviar el trabajo al recurso de cómputo inmediatamente que éste sea seleccionado. En el extremo opuesto está la política de suspender los trabajos en la cola de espera del WMS hasta que algún recurso se encuentre disponible, en cuyo caso las características del recurso serán comparadas con los requerimientos de los trabajos suspendidos, y el trabajo que mejor se ajuste será enviado al recurso. Estas dos políticas tienen una relación de simetría porque en el primer caso, se compara un trabajo con una lista de recursos disponibles, y en el segundo se compara un recurso disponible con una lista de trabajos a la espera de ser ejecutados. Sin embargo, la segunda política requiere mantener comunicaciones asíncronas con los proveedores de servicio, y necesita más cuidado a la hora de gestionar notificaciones de varios WMS para garantizar la escalabilidad del sistema. Por esta razón, solamente la primera política ha sido llevada a producción.

Para monitorizar el estado de los recursos, WMS utiliza un componente de información (Information Super-market o ISM) que puede ser consultado por el componente encargado de hacer la comparación de los requerimientos de los trabajos con las características de los recursos disponibles. El ISM es actualizado a través de notificaciones de los recursos y también a través de la exploración directa de los mismos.

Una vez asignado el recurso, WMS crea una interfaz (Job Submission Service o JSS) se encarga de establecer una comunicación autenticada con el recurso asignado, de enviar el trabajo y de monitorizar la ejecución del mismo. Esta capa se implementa sobre Condor-G, que se ocupa de gestionar la ejecución de los trabajos: enviar el trabajo al recurso de cómputo, monitorizar la ejecución de los trabajos, eliminar los trabajos, etc. JSS se encarga de monitorizar las trazas de Condor, y de interceptar los eventos referentes a los trabajos activos. Para ello, es necesario instalar un CE (Computing Element) en los recursos de cómputo (clústeres). Este componente proporciona una interfaz al recurso, y un planificador local que forma parte del LRMS que gestiona la ejecución de trabajos en el recurso.

Un servidor GAHP (Grid ASCII Helper Protocol) sirve de puente entre WMS y los diferentes CE. Como gLite ha tenido varios cambios de diseño, en los cuales han sido utilizados diferentes componentes software de diferentes proyectos, el tema de la instalación de los CE resulta confuso. En la actualidad coexisten tres CE: LCG CE (pre-WS Condor-G y GRAM), gLite CE (pre-WS Condor-G y Condor-C), y CREAM (Computing Resource Execution And Management) CE. Aunque a mediados de 2009,

solamente LCG CE y gLite CE se encontraban en producción, las nuevas versiones de WMS incorporan varios cambios, como resultado de un trabajo de reestructuración con el objetivo de aligerar el núcleo de componentes de WMS y mejorar la capacidad de respuesta del mismo. Precisamente uno de estos cambios, está relacionado con la sustitución de gLite CE por CREAM CE. Para ello, WMS incorpora ICE (Interface to CREAM Environment), un nuevo componente que proporciona una interfaz para la gestión de los CREAM CE. En consecuencia, la interfaz de usuario también incorpora un cliente para CREAM.

CREAM proporciona un conjunto de servicios para la gestión de trabajos, que pueden ser accedidos a través de una interfaz OGSA-BES (OGSA - Basic Execution Services), que estandariza las operaciones de gestión de trabajo a nivel de CE. Como se aprecia en la **Figura 2-4**, estos servicios sustituyen la mayor parte de los componentes de WMS (controlador de trabajos, Condor, monitor de trabajos, etc.).

En cuanto a la tolerancia a fallos, WMS puede relanzar automáticamente los trabajos que no han podido completarse con éxito, permitiendo un número máximo de relanzamientos que puede ser fijado de forma individual para cada trabajo, y teniendo en cuenta si el fallo se produce antes o después de que ocurra la asignación de un recurso para la ejecución del trabajo.

Por último, gLite dispone de un servicio centralizado de trazas (Logging and Bookkeeping service o L&B), que inicialmente formaba parte de WMS, pero que en la actualidad es un componente independiente. Este servicio permite consultar el estado de los trabajos enviados al Grid. Para ello, mantiene un registro de los eventos importantes relacionados con los trabajos (envío, asignación de un CE, ejecución, etc.) que es la suma de la información de todos los WMS y de otros servicios del Grid.

2.4. CONCLUSIONES DEL CAPÍTULO

La computación en Grid se ha consolidado como un entorno distribuido para la ejecución de trabajos, lo cual ha motivado el desarrollo de varios middleware Grid, entre los que destacan Globus Toolkit y gLite. En este contexto, la meta-planificación constituye un elemento clave que tiene como objetivo la selección de recursos, a nivel global, para planificación de trabajos en el Grid. Debido a la importancia que tiene, existen varios meta-planificadores que porporcionan funcionalidades avanzadas para la asignación de recursos.

Capítulo 3

3. ASIGNACIÓN DE RECURSOS EN ENTORNOS DE COMPUTACIÓN DISTRIBUIDA

Este capítulo presenta una revisión del estado del arte de la asignación de recursos en entornos de computación distribuida. Se plantea la formulación del problema desde el punto de vista matemático y se demuestra la imposibilidad de obtener una solución exacta. Adicionalmente, se hace un estudio de las soluciones aproximadas, haciendo especial énfasis en aquellas que han sido validadas en entornos de computación Grid.

3.1. ESTADO DEL ARTE

La idea de vincular trabajos con recursos y planificar el orden de ejecución de los mismos en base a sus requerimientos de QoS ha sido abordada con anterioridad en diferentes contextos. En este sentido, es necesario mencionar que en la actualidad existen varios sistemas que ofrecen soporte al respecto. Sin embargo, en cuanto a complejidad computacional, se conoce que el problema de seleccionar entre un grupo de recursos distribuidos el conjunto de recursos que minimice el coste computacional de ejecutar un trabajo determinado es NP-completo [10]. Por esta razón, la mayor parte de las investigaciones de la última década, en el campo de la selección de recursos distribuidos para completar una tarea, se han centrado en el desarrollo de algoritmos efectivos de selección de recursos. Un número considerable de estos proyectos de investigación han estado enfocados a resolver dos grandes retos: 1) encontrar el conjunto de recursos que minimizan el tiempo que un trabajo permanece en el sistema. En este sentido hay tres grandes objetivos: minimizar el tiempo de ejecución, minimizar la varianza del tiempo de ejecución (Completion-Time Variance o CTV), y minimizar la varianza del tiempo de espera (Waiting-Time Variance o WTV); y 2) proporcionar, tanto de forma anticipada como bajo demanda, los recursos que garanticen un nivel mínimo de disponibilidad para cumplir con los requerimientos del trabajo en cuestión.

La QoS ha sido una prioridad para los sistemas de computación Grid desde que aparecieron los primeros trabajos sobre Grid, en la década de los 90. Tal es así, que una de las primeras contribuciones que aparecen en la literatura se remonta al año 1999, y la firman dos de los investigadores más influyentes en el tema Grid: Ian Foster y Carl Kesselman. En este trabajo, Foster y colaboradores presentan GARA [23], una arquitectura para la gestión de recursos distribuidos construida sobre Globus Toolkit. El enfoque de esta arquitectura trataba de evitar, en la medida de lo posible, los problemas relacionados con la monitorización de los recursos. Para esto, GARA se apoyaba fundamentalmente en información *a priori* para reservar un conjunto de recursos anticipadamente a la ejecución de un trabajo, con el objetivo de garantizar un nivel mínimo de recursos disponibles para la ejecución del mismo.

GARA también sentó las bases para la discusión en torno a la efectividad de las estrategias de asignación de recursos en el Grid. A partir de la publicación de este trabajo, muchos investigadores apoyaron la tesis de que la reserva anticipada de recursos era la única forma de alcanzar unos niveles de fiabilidad y de calidad de servicio razonables en el contexto de la computación Grid. En cambio, otros investigadores seguían apoyando el aprovisionamiento bajo demanda, a pesar de los problemas que presentaba en situaciones de sobrecarga. Esta discusión se ve reflejada en varios trabajos que apoyan la superioridad de la reserva anticipada sobre la reserva bajo demanda [24] y [25].

Una versión posterior de GARA [26] exploró la posibilidad de utilizar información del estado del sistema en tiempo de ejecución para mejorar la selección de recursos. Este enfoque recae en la monitorización de los recursos y las aplicaciones, para determinar si las prestaciones de una aplicación están dentro de sus requerimientos de QoS o no, y para emprender acciones correctivas cuando las prestaciones sean inaceptables. De esta forma quedaba zanjada la discusión en torno a la asignación de recursos: primero, se logró el consenso en torno al hecho de que era necesario utilizar información real de los sistemas, obtenida mediante monitorización; y segundo se reconoció la validez del aprovisionamiento de recursos bajo demanda, que luego demostró su utilidad al combinarse con la reserva anticipada para resolver situaciones en las que, una vez comenzada la ejecución de un trabajo, se podía prever la imposibilidad de cumplir con los requerimientos de QoS.

Este enfoque ya había sido utilizado con éxito en otros sistemas. Los trabajos de Al-Ali y colaboradores [27][28] fueron los primeros intentos por mejorar la efectividad de la selección de recursos en el Grid utilizando monitorización. Los autores de este trabajo propusieron una solución heurística que tenía como objetivo maximizar la ganancia económica global, manteniendo un nivel de servicio aceptable de cara al usuario final. Para esto introdujeron G-QoSM, un sistema de gestión de recursos que era capaz de ajustar el comportamiento de las aplicaciones para responder a ciertos cambios de contexto. Sin embargo, G-QoSM estaba limitado por un conjunto predefinido de posibles configuraciones y por la precariedad de los sistemas de monitorización en los que se apoyaba para conocer el estado de los recursos.

Varios años después, Doulamis y colaboradores [29] presentaron un enfoque diferente que consiste en utilizar un modelo no lineal de la complejidad computacional de una aplicación para predecir la carga que supone cada trabajo antes de que sea enviado al Grid. Esta predicción es utilizada luego para seleccionar el recurso que, por sus características, tenga la mayor probabilidad de completar el trabajo en el menor tiempo posible. Este enfoque, aunque en principio puede resolver varios de los problemas de los sistemas anteriores, tiene una aplicación muy limitada porque la creación de modelos de complejidad computacional que sirvan para predecir la carga de una aplicación es un problema tan complejo en sí mismo que, en la mayoría de los casos, es imposible contar con estos modelos. Sin embargo, la posibilidad de contar con una estimación de la carga que producirá un trabajo antes de enviarlo al Grid es una muy buena idea de la cual se puede sacar mucho provecho. De ahí que, como veremos más adelante, otros autores han explorado otras formas para conseguir estas estimaciones.

Otra propuesta relevante se puede encontrar en el trabajo de Lu y colaboradores [30]. Estos autores se trazaron como objetivo encontrar el conjunto de recursos que

minimizan el tiempo que un trabajo consume en el sistema. Para ello, clasifican los sitios en cuanto al deseo que manifiestan los usuarios de utilizar los recursos de cada sitio, formando varios grupos que van desde sitios poco deseados a sitios muy deseados. Luego, los trabajos son asignados a los sitios más deseados. Los grupos son ajustados dinámicamente en tiempo de ejecución para corregir la clasificación de los sitios, reemplazando los sitios sobrecargados de los grupos más deseados con los sitios menos cargados del resto de los grupos. Este trabajo nos presenta una forma de diferenciación que permite abordar el problema de la QoS en el Grid. Sin embargo, la forma de clasificación que utilizan los autores de este trabajo se puede mejorar utilizando herramientas que permitan ampliar los parámetros de clasificación, así como establecer relaciones complejas entre ellos. Además, la reconfiguración dinámica se basa en la carga total del sitio, cuando en realidad los recursos de los sitios están agrupados (por ejemplo, en clústeres de ordenadores), por lo cual se podría ganar en efectividad a la hora de reorganizar los sitios si se pudiera conocer la carga de cada grupo de recursos.

Subrata y colaboradores [31] propusieron un algoritmo basado en la teoría de juegos cooperativos para seleccionar recursos Grid en base a requerimientos de QoS. El enfoque innovador de este trabajo posibilita evaluar casos muy reales donde se tienen en cuenta detalles de los sitios que hasta el momento no podían ser estudiados. Por ejemplo, este trabajo permite diferenciar los sitios por sus intereses y prioridades. Sin embargo, el algoritmo tiene un nivel de centralización considerable, lo cual es un inconveniente importante para su aplicación en entornos distribuidos de gran tamaño. Además, los enfoques cooperativos son susceptibles a sufrir problemas asociados con jugadores hostiles, que se dan cuando alguno de los participantes decide poner en práctica alguna estrategia que perjudique a los demás. Precisamente, las características de los entornos de computación Grid pueden llegar a favorecer la aparición de este tipo de situaciones. En particular, debido a la heterogeneidad, es probable que en algún momento se lleguen a contraponer los intereses de dos o más proveedores de servicio, y que alguno de estos decida dejar de cooperar.

Algunos trabajos van más allá de proponer algoritmos de propósito general, y además de abordar la selección de recursos en base a requerimientos de QoS, dan solución a problemas vinculados a esta. Tal es el caso del trabajo de Middleton y colaboradores [32]. Los autores de este trabajo presentan un sistema que permite reservar recursos, de forma anticipada, para aplicaciones de simulación médica en un Grid comercial. Este trabajo es muy importante para entender las limitaciones que impone el entorno de trabajo sobre la solución de un problema específico. En este caso, la política de gestión de recursos se ve forzada a cumplir dos limitaciones impuestas por el entorno. La primera limitación consiste en que los modelos de fijación de precios están definidos por el entorno. La fijación de precios es el procedimiento por el cual una entidad (fabricante, vendedor, etc.) establece un precio para los productos o servicios que vende [33]. Este procedimiento refleja una estrategia de planificación que tiene en cuenta varios factores, como los objetivos económicos de la entidad, la demanda de los consumidores, los atributos del producto, los precios de los competidores, y las tendencias económicas y mercantiles. La forma más sencilla de establecer el precio de un producto consiste en aplicar el mismo precio por cada unidad del mismo. Sin embargo, esta política, conocida como fijación de precios uniforme, es la que menos ganancia proporciona. Otras políticas consiguen mejor rentabilidad aplicando

diferentes precios a diferentes compradores. Por ejemplo, la diferenciación de precios se puede implementar en base a la ubicación del comprador (a mayor dificultad para hacer llegar el producto o servicio al comprador, mayor es el precio que se aplica), o en base a la cantidad de productos o servicios que son incluidos en una compra (a mayor número de productos o servicios, ya sean iguales o similares, menor es el precio que se aplica). La segunda limitación expresa una necesidad de mantener la utilización de los recursos al máximo, con el objetivo de maximizar las ganancias. Esta limitación se traduce en una necesidad de establecer un equilibrio entre el tiempo de espera en el sistema y los costes, confinando el espacio posible de exploración a resolver este problema específico. Por otra parte, los autores de este trabajo utilizan modelos computacionales para estimar el tiempo de ejecución y los requerimientos computacionales de los trabajos que van a ser lanzados al Grid. Ante el número de aplicaciones que no pueden ser evaluadas con este enfoque, por los motivos expuestos anteriormente, los autores proponen un mecanismo de evaluación alternativo que consiste en mantener una base de datos con los requerimientos de cada aplicación. Además, proponen la utilización de casos de prueba representativos y registros históricos para obtener dicha base de datos. Aún así, la habilidad del sistema para seleccionar los recursos para ejecutar un trabajo de forma exitosa recae, en gran medida, en la exactitud de los modelos computacionales. Este hecho tiene un efecto secundario no deseado dado que el sistema, para protegerse de las posibles inexactitudes de los modelos, sobrestima el número de recursos necesarios para ejecutar un trabajo. De esta forma, el sistema asegura que, de ocurrir un error en la estimación de los recursos necesarios, habrá un conjunto de recursos adicionales que pueden ser utilizados por el trabajo. El problema de utilizar este enfoque es que los recursos adicionales permanecerán bloqueados, y en muchos casos desaprovechados, hasta que el trabajo termine su ejecución y el sistema los libere.

Zheng y colaboradores [34] presentan tres algoritmos para encontrar la distribución de la carga que minimice el tiempo de respuesta o el coste computacional en sistemas Grid. Los autores de este trabajo vuelven a utilizar la diferenciación de recursos para resolver problemas de QoS. En este caso, proponen separar los nodos en diferentes grupos de acuerdo a dominios administrativos o a tiempos de comunicación, y mantener entonces el intercambio de información dentro de estos grupos, utilizando un nodo como puente entre los nodos de su grupo y el resto de los nodos en otros grupos. Este nodo recibe el nombre de nodo central. Los nodos centrales de todos los grupos se conectan entre sí y deciden, en conjunto, a qué grupo será redirigida una solicitud de trabajo para su ejecución. Además de ello, los autores presentan un estudio del efecto de la fijación de precios en la distribución de la carga. Para este fin, introducen en los algoritmos el precio de utilizar un nodo. Sin embargo, esto los conduce a un problema de optimización muy complejo, que se simplifica considerando que todos los nodos de un grupo tienen el mismo precio. Este modelo es aplicable a infraestructuras centralizadas, pero no se puede trasladar a infraestructuras distribuidas de gran dimensión, donde no es posible tener una visión global del sistema. En tal caso, los precios calculados de esta forma no siempre reflejan las prestaciones reales de los nodos. Efectivamente, se puede dar el caso de que nodos con precios muy altos estén sobrecargados o simplemente estén ubicados en grupos donde el resto de los nodos son mucho mejores, o tienen mejores características de red, en cuyo caso, esos

nodos estarán operando con un nivel de prestaciones inferior al de otros nodos de menor precio.

Stuer y colaboradores [35] presentan un modelo económico que trata los recursos Grid como bienes sustituibles. La propuesta de estos autores consiste en agrupar los nodos en diferentes categorías, en base a una combinación difusa de sus propiedades (por ejemplo, tipo de CPU, cache). De esta forma, el precio por utilizar un recurso puede ser determinado utilizando principios de mercado de productos básicos. Los autores de este trabajo presentan un algoritmo de fijación de precios que resuelve varias situaciones de mercado. Sin embargo, el algoritmo carece de una evaluación de escalabilidad. De hecho, los autores avisan sobre la posibilidad de que la cantidad de comunicaciones que produce el algoritmo puede ser excesiva para entornos de computación Grid reales. Adicionalmente, el modelo de comunicación que utiliza el algoritmo no tiene en cuenta la localización de los recursos y los costes asociados a la utilización de redes con latencias y anchos de banda heterogéneos.

Hughes y colaboradores [36] presentan una perspectiva muy diferente a todas las vistas hasta el momento. Estos autores presentan un enfoque centrado en el componente humano para predecir el comportamiento de servicios compuestos, en base a la información de QoS de los componentes individuales. Los autores de este trabajo presentan una herramienta visual que puede ser utilizada para evaluar, de forma interactiva, el impacto que tiene la composición de servicios sobre las prestaciones de un sistema. La tesis que defiende este trabajo consiste en que un experto, auxiliado con esta herramienta, puede decidir de entre un grupo de candidatos caracterizados previamente, qué servicios podrán combinarse con éxito en un flujo de trabajo.

Por último, el proyecto BOINC [37] presenta un punto de vista radicalmente diferente. BOINC ofrece una plataforma software que permite hacer computación intensiva utilizando recursos voluntarios. Este enfoque prescinde de la gestión de QoS asumiendo que el número de recursos disponibles en el sistema es virtualmente ilimitado, por lo cual siempre será posible sumar más recursos a la solución de un problema[2]. Esta consideración de que el número de recursos es infinito tiene la ventaja adicional de que para garantizar que se cumplan los requerimientos de QoS de un trabajo, solamente es necesario ejecutar varias réplicas del mismo trabajo en diferentes nodos de diferentes proveedores. De esta forma se asegura que, eventualmente, uno de ellos terminará el trabajo en tiempo y forma. Esta estrategia ha demostrado su eficiencia para resolver algunos tipos de problemas. Sin embargo, presenta unos problemas técnicos y sociales muy particulares, derivados de la necesidad de conseguir un número de recursos suficientemente grande para abordar un proyecto. Para conseguir estos recursos, un proyecto tiene que convencer a un número grande de individuos para que cedan sus recursos de forma voluntaria, y luego tiene que preparar los binarios, bibliotecas y demás herramientas para que puedan ser utilizados en estos sistemas, que normalmente no están preparados para realizar el trabajo que se espera que hagan. De esta forma, los entornos de computación voluntaria se caracterizan por la indeterminación del número de recursos, la volatilidad y la heterogeneidad. A la par, este enfoque también presenta problemas de seguridad y escalabilidad.

[2] No sólo BOINC, sino realmente la estrategia de EGEE también carece de soporte para QoS.

3.2. ESTÁNDARES PARA LA GESTIÓN DE QOS EN EL GRID

El Open Grid Forum (OGF) es una comunidad de usuarios, desarrolladores y vendedores, que lideran el proceso de estandarización de las tecnologías Grid. Desde su fundación en 2006, el OGF ha desarrollado varios estándares, como OGSA, OGSI y JSDL, que son ampliamente utilizados, y sobre los cuales se implementa una buena parte del middleware Grid disponible en la actualidad.

El OGF utiliza un modelo de estandarización semejante al que utiliza la Internet Engineering Task Force (IETF), y al igual que ésta desarrolla estándares abiertos. En este sentido, el OGF publica las recomendaciones, que es un tipo de documento análogo al "Internet Standards track document" que utiliza la IETF para documentar una especificación. Las recomendaciones del OGF pasan por un proceso de confirmación, que comienza con una propuesta que es evaluada en la práctica y, según los resultados, puede llegar a convertirse en una recomendación.

En cuanto a la QoS, la actividad del OGF está orientada a estandarizar los resultados que han sido obtenidos en varios proyectos, como GRIDCC [38], G-QoSM [27], BEinGRID [39], SmartLM [40], GridEcon [41], SORMA [42], BREIN [43], SLA@SOI [44], AssessGrid [45] y RESERVOIR [46], y a completar la especificación WS-Agreement, para reflejar las necesidades del Grid.

Teniendo en cuenta que actualmente, la computación Grid tiene una superposición considerable con la arquitectura SOA basada en servicios Web, y que los avances conseguidos en las tecnologías de servicios Web han proporcionado las bases para la estandarización de la arquitectura Grid orientada a servicios, el interés por desarrollar el estándar WS-Agreement, dentro del marco del OGF, está más que justificado. WS-Agreement consiste en un lenguaje y un protocolo, que han sido diseñados con el objetivo de exponer la capacidad de los proveedores de servicio, y sobre la base de la oferta inicial, crear acuerdos entre proveedores y consumidores, que luego pueden ser monitorizados, en tiempo de ejecución, para verificar su cumplimiento. La motivación para el diseño de la WS-Agreement parte de la necesidad de garantizar los niveles de QoS que necesitan algunas aplicaciones, fundamentalmente del sector empresarial, y que no pueden ser conseguidos con las tecnologías de servicios Web tradicionales. Por esta razón, el protocolo está definido en términos generales, y proporciona las operaciones necesarias para llevar a cabo la negociación de QoS en cualquier sistema distribuido desarrollado sobre una infraestructura de servicios Web.

Actualmente, la especificación WS-Agreement se encuentra en estado de propuesta, que necesita ser confirmada por la industria [47]. Por otra parte, la propuesta actual no contempla la posibilidad de cambiar un acuerdo en tiempo de ejecución. Esta característica de la misma constituye un punto crítico que necesita ser reevaluado, para conseguir acuerdos robustos. Con este cambio se busca mejorar la capacidad de un acuerdo para hacer frente a variaciones impredecibles del entorno, con el menor impacto posible sobre los procesos mediados por el acuerdo en cuestión. Específicamente, se quiere disponer de un mecanismo para renegociar el acuerdo cuando se producen violaciones individuales en tiempo de ejecución, lo que permitiría reajustar puntualmente el acuerdo y continuar con la ejecución. Conseguir este tipo de mecanismos es fundamental para el Grid, como se verá a continuación.

Conjuntamente con WS-Agreement, el SLA es un instrumento fundamental para la gestión de la QoS, que proporciona un método para describir los requerimientos de la misma. Sin embargo, la gestión de SLA en el Grid es muy compleja, debido a que éste agrupa recursos de varias organizaciones diferentes, que mantienen el control sobre los recursos locales. Estas características dificultan las operaciones relacionadas con la especificación de los SLA, la negociación y renegociación de los SLA, y la monitorización de la QoS. Aunque hay mucho interés entre los miembros de la comunidad por resolver estos problemas, hay muy pocos puntos de contacto entre los diferentes trabajos que se han llevado a cabo hasta la fecha. El OGF ha intentado mejorar este aspecto a través de dos reuniones realizadas para intercambiar experiencias entre los investigadores y desarrolladores de las organizaciones involucradas en proyectos relacionados con la gestión de QoS y SLA en el Grid. El OGF19, llevado a cabo en enero de 2007, y posteriormente el OGF23, llevado a cabo en junio de 2008, sirvieron para estos objetivos.

Como resultado de estas reuniones quedó definido que la QoS, en el contexto de la OGSA, es una medida del nivel de los servicios prestados, con una lista precisa de parámetros que la caracterizan, entre los que destacan la seguridad, el ancho de banda, el tiempo medio de respuesta y la disponibilidad del servicio [48]. Esta definición resalta el hecho de que la QoS, en el contexto del Grid, va más allá de la definición de QoS para la red (que se centra en el ancho de banda, la latencia, el retardo y la pérdida de paquetes, y el jitter), y además de ello, tiene en cuenta la QoS a nivel de middleware (potencia de cálculo, memoria y capacidad de almacenamiento, etc.) y la QoS a nivel de aplicación (percepción del usuario, tiempo de respuesta, seguridad, etc.).

Aunque la gestión de la QoS había sido identificada antes como una necesidad en la especificación de la OGSA [49], y también había sido tenida en consideración para plantear los casos de uso de los entornos de computación Grid comerciales [50], estas reuniones sirvieron para recalcar la importancia que ha adquirido la QoS, y en especial la gestión de SLA, en el contexto de las infraestructuras de computación bajo demanda, como el cloud computing. Para operar, estas estructuras necesitan de mecanismos que les permitan asegurar que los proveedores puedan ofrecer un nivel de calidad aceptable.

3.3. MODELO DE COLAS PARA LA PLANIFICACIÓN GRID

El problema que motiva esta tesis es la selección de recursos para la ejecución de trabajos en entornos de computación Grid, particularmente atendiendo a criterios que garanticen un nivel de servicio apropiado para que los trabajos enviados al Grid consigan terminar con éxito, y mejorar la utilización de los recursos. Estos objetivos se suscriben a un área de investigación específica dentro de las tecnologías Grid: la planificación de trabajos y la gestión de la carga. El primer paso para la solución del problema consistirá en obtener una definición del mismo, para lo cual vamos a obtener un modelo.

Un modelo de un sistema es una representación de la interacción de las variables y funciones que gobiernan al sistema original. La precisión de un modelo depende del conocimiento que tenemos del sistema, pero también de la necesidad de contar con

una forma simple para evaluar ciertas características del mismo. Es por esto que, muy pocas veces es posible trabajar con un modelo que represente, de forma precisa, todas las características reales del sistema modelado. En la mayoría de los casos, los modelos son aproximaciones precisas de varias características reales del sistema modelado, que simplifican otros detalles menos importantes para el estudio de un caso determinado.

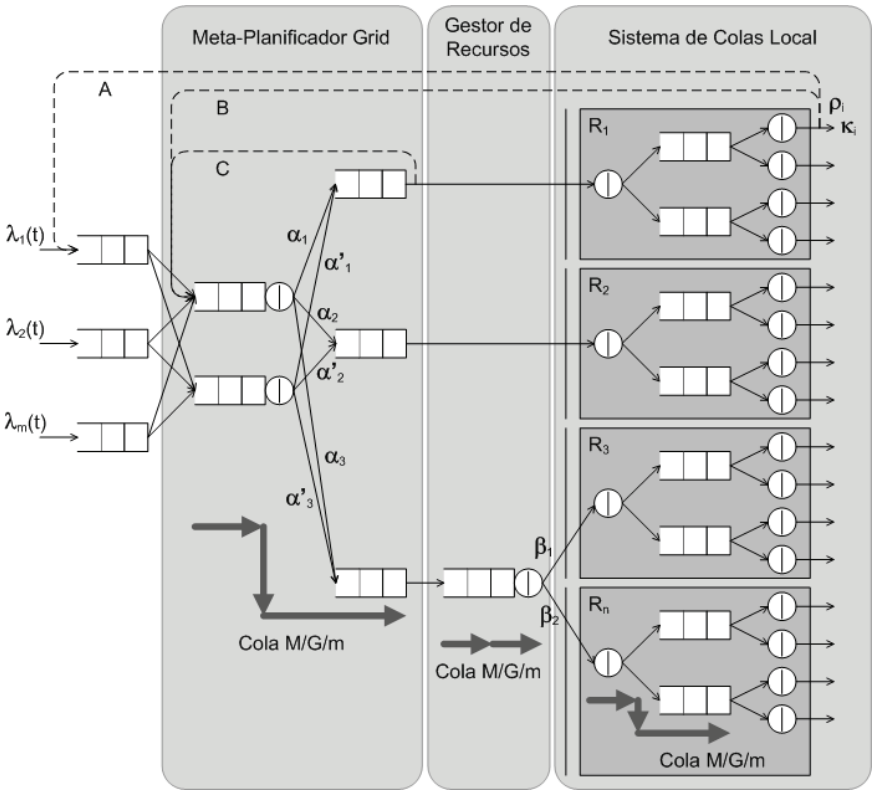


Figura 3-5: El sistema de planificación Grid representado en un modelo de colas.

Uno de los enfoques más comunes para abordar la representación de problemas de planificación y de selección de recursos consiste en utilizar modelos de colas. Este modelo ha sido utilizado tradicionalmente con éxito en la planificación de recursos en redes y sistemas de almacenamiento, con los objetivos de mantener la utilización de los recursos al máximo, al mismo tiempo que cumplir con las garantías de QoS para todos los clientes [51]. En este sentido, el modelo más general considera la utilización de múltiples servicios concurrentes en el mismo sistema. El caso más simple consiste en considerar que el sistema está formado por múltiples servicios uniformes, que son idénticos en términos del servicio que prestan. Un ejemplo de este sistema es un clúster de ordenadores que ejecuta transacciones para un conjunto de clientes, donde cualquier procesador del clúster puede ser asignado para ejecutar un trabajo. Una situación más

compleja se da cuando los servicios son dispares y, por tanto, la asignación de recursos depende de subconjuntos del conjunto total de servicios. Los sistemas Grid constituyen un buen ejemplo de este tipo de sistemas, al estar formados por recursos heterogéneos, que ejecutan trabajos para una organización virtual. En este caso, los trabajos tienen que ser asignados al servicio apropiado, dependiendo de los requerimientos particulares de cada trabajo.

La **Figura 3-5** muestra una representación gráfica de un sistema de planificación Grid utilizando modelos de colas. Las colas han sido representadas utilizando una notación estándar [52]: un círculo con una barra vertical representa un área de servicios, y un rectángulo con el extremo izquierdo abierto y dos barras verticales representa una cola de espera. Las colas de espera reciben entradas por la izquierda y las transfieren a las áreas de servicio, que a su vez transforman las entradas en salidas. Las barras verticales representan trabajos en espera cuando se encuentran en una cola de espera y trabajos activos cuando se encuentran en un área de servicio. Cada área de servicio puede procesar un trabajo a la vez. Por otra parte, las entradas y las salidas pueden presentar bifurcaciones. Una bifurcación en la entrada es un punto de combinación, y una bifurcación en la salida es un punto de separación. En los puntos de separación se distingue entre salidas exitosas y salidas fallidas. Estas últimas pueden seguir un camino de retroalimentación que las lleve nuevamente a una de las colas de espera, o pueden ser descartadas. En los puntos de combinación se combinan varias entradas, por ejemplo, se puede combinar la entrada que procede de una cola de espera, con salidas fallidas que han sido retroalimentadas desde un área de servicio, o simplemente pueden unirse las entradas procedentes de varias colas de espera.

En general, un sistema de planificación Grid puede estar compuesto de hasta tres subsistemas de planificación diferentes. El primer subsistema consiste en un meta-planificador Grid, que se ocupa de recibir las solicitudes de trabajo de los clientes y de planificar la ejecución a nivel global (meta-planificación), utilizando para ello todos los recursos del Grid. El segundo subsistema es opcional, y consiste en un sistema de gestión de recursos que se ocupa de recibir las solicitudes del meta-planificador Grid y de planificar la ejecución en un conjunto de recursos de una organización particular. El último subsistema consiste en un sistema de colas local, que recibe las solicitudes del meta-planificador Grid, o del sistema de gestión de recursos, y planifica la ejecución de trabajos en un recurso particular (usualmente clústeres de ordenadores).

El modelo de la **Figura 3-5** utiliza varias colas conectadas entre sí para representar los tres subsistemas de planificación. Los subsistemas están delimitados por recuadros con los bordes redondeados, y las colas están indicadas con flechas gruesas. Este modelo se puede aplicar a casos donde un meta-planificador Grid (por ejemplo, gLite WMS) recibe solicitudes de trabajos de varios clientes. Usualmente, los clientes se agrupan por algún criterio que reúne varios trabajos pertenecientes al mismo proyecto o tarea. Por ejemplo, es bastante común que una tarea específica de un proyecto implique diversos trabajos, y que estos sean enviados al mismo tiempo para su ejecución en el Grid, utilizando para ello algún automatismo. Este hecho ha sido representado en la figura por varias colas de espera, cada una con una tasa de llegada de trabajos λ_i, combinándose con un camino de retroalimentación que envía trabajos fallidos desde los recursos, con una tasa de llegada ρ_i. Las colas de espera de los clientes convergen en una cola M/G/m que representa al meta-planificador Grid. El tiempo entre llegadas

sigue una distribución exponencial (M en la notación de Kendall). Esta distribución describe el patrón de probabilidad según el cual los clientes generan solicitudes. En cambio, el tiempo de procesamiento de cada solicitud es impredecible, en parte porque los meta-planificadores Grid suelen combinar varias políticas de servicio, por ejemplo, una política por prioridades y otra FCFS (First-come, first-served). Por esta razón, el tiempo de servicio puede seguir cualquier distribución estadística (G en la notación de Kendall). Además, la cola dispone de varios servicios idénticos (m en la notación de Kendall) que pueden atender las solicitudes. Por ejemplo, en un entorno como gLite, los recursos se ofertan en forma de servicios independientes. De ahí que, una vez que se hace la asignación de recursos para la ejecución de un trabajo, la solicitud pasa a la cola de espera de un componente del meta-planificador que se ocupa de reenviar la solicitud al recurso asignado.

Es también habitual combinar varios meta-planificadores para evitar la sobrecarga del servicio, enviando los trabajos de forma alternativa, utilizando, por ejemplo, una planificación por turnos con un algoritmo round-robin. Este hecho ha sido representado en la **Figura 3-5** por varios meta-planificadores. La tasa de salida α_i de cada meta-planificador dependerá del tiempo de espera en las colas, o lo que es lo mismo, de la latencia del algoritmo de planificación. Suponiendo que los recursos no se saturan o que hay un número infinito de recursos, el único requerimiento que debe cumplir este algoritmo es que las solicitudes pendientes tienen que ser despachadas antes de que cualquier cola del siguiente subsistema se quede vacía, para evitar que el resto de las colas se ralenticen.

El gestor de recursos ha sido representado en la **Figura 3-5** por una cola M/G/m. Este subsistema recibe solicitudes del meta-planificador Grid y las reenvía a los sistemas de colas de los recursos, aunque es más frecuente que el meta-planificador envíe las solicitudes directamente a los sistemas de colas. A diferencia del meta-planificador, que es el mismo para todo el Grid, cada sitio puede utilizar un gestor de recursos diferente, por ejemplo Condor [53] y LSF [55]. Es por eso que, cuando participa un sistema de gestión de recursos, las tasas de entrada β_i de los sistemas de colas de los recursos dependerán del algoritmo de planificación que utilice el sistema de gestión de recursos. En cambio, cuando el meta-planificador envía las solicitudes directamente al sistema de colas, la tasa de entrada de los sistemas de colas de los recursos es igual a la tasa de salida α_i del meta-planificador Grid.

El tercer subsistema es el sistema de colas de los recursos, que puede diferir de un recurso a otro. Normalmente, la utilización de un clúster de ordenadores está organizada en varias colas gestionadas por un sistema de colas, por ejemplo PBS [56] o TORQUE [57], que se ocupa de gestionar los trabajos enviados al clúster. En la **Figura 3-5**, los recursos están delimitados por recuadros, por ejemplo, R_1 y R_2. Dentro de los recursos, una cola M/G/m representa al sistema de colas local, y las áreas de servicio a la derecha de las colas, representan a los nodos de los clústeres. Las colas utilizan las áreas de servicio en paralelo para procesar trabajos de forma concurrente. De esta forma, las salidas están caracterizadas por la tasa de servicio por recurso. Además de ρ_i, que describe el patrón de probabilidad según el cual los trabajos fallan en un recurso, κ_i es la tasa de salidas exitosas de un recurso particular.

Algunos sistemas Grid incluyen la opción de que sea el usuario quien elija un recurso específico (un clúster) de una organización específica, para cada trabajo. Por ejemplo,

en un entorno gLite, cuando se hace el *matchmaking* (emparejamiento) de la solicitud con los recursos disponibles, se obtiene una lista con los *endpoints* de las colas de los recursos. En este caso, es posible especificar una lista de recursos, con cada trabajo que se envía al Grid, para definir cuáles recursos tienen más preferencia. Sin embargo, esta forma de utilizar los recursos del Grid dificulta cualquier intento de planificación global, posibilitando la aparición de cuellos de botella en los recursos más "conocidos", al mismo tiempo que otros recursos pueden permanecer infrautilizados. Aunque existen varios mecanismos para distribuir la carga entre los recursos, este comportamiento dificulta la modelación del sistema, por lo cual solamente se tuvo en cuenta el caso donde los clientes no pueden elegir los recursos a los que envían los trabajos.

Adicionalmente, los sistemas de colas de los clústeres implementan algún mecanismo de contención que les permite limitar el número de nodos que son destinados a los trabajos del Grid, con el objetivo de mantener el control sobre los recursos locales de cada organización. En la mayoría de los casos, estos mecanismos de contención permiten variar, de forma dinámica, el número de nodos que reciben trabajos de cada cola o la prioridad de la cola, como también pueden limitar el número de trabajos concurrentes de determinados grupos de usuarios. En consecuencia, la tasa de salidas exitosas x_o, además de depender de la tasa de servicio por recurso, podrá verse afectada por cambios de configuración imprevisibles, que puedan suceder como parte del mecanismo de contención. Este hecho ha sido representado en la figura por un medio de contención (una barra vertical) ubicado delante de cada recurso. Normalmente, los trabajos se envían al Grid con una credencial de usuario que está vinculada con un usuario local en el recurso. Por esta razón, la contención comienza en el momento en que la credencial Grid es convertida en un identificador de usuario local.

A, B y C representan tres caminos de retroalimentación diferentes. B y C representan dos caminos de retroalimentación que normalmente se producen sin la intervención del usuario. Por ejemplo, si el gestor de colas falla, el meta-planificador se ocupa de relanzar los trabajos, cambiando de recurso. Lo mismo sucede si falla la ejecución de un trabajo en un recurso específico. En cambio, A es un camino de retroalimentación que no se produce sin la intervención del usuario. Por ejemplo, si el meta-planificador relanza el mismo trabajo un número de veces, y ninguna de ellas se completa con éxito, entonces el trabajo es descartado. En ese caso, lo mismo si el trabajo falla y el cliente decide cancelarlo y reenviarlo manualmente, el trabajo es reenviado desde el cliente.

Si se considera que un trabajo con requerimientos de garantías de servicio se completa con éxito solamente cuando se cumplen todos los indicadores de calidad de servicio, por ejemplo, si el trabajo consigue terminar dentro de un plazo de tiempo definido de antemano, entonces el problema de seleccionar los recursos en el Grid para ejecutar un trabajo con garantía de servicio se puede resolver con un algoritmo que minimice la probabilidad P de que la salida tome un camino de retroalimentación (A o B) en el punto de separación del último subsistema del modelo.

Por otra parte, el tiempo total que un trabajo permanece en el sistema es el tiempo de respuesta o ejecución del mismo. Este tiempo es la suma de todos los tiempos de espera en las colas y de los tiempos de servicio. Por esta razón, un trabajo que tome un camino de retroalimentación (A, B o C) tardará más de lo esperado.

3.3.1. FORMULACIÓN DEL PROBLEMA

Existen varias formas de abordar el problema de encontrar un algoritmo que minimice la probabilidad de retroalimentación en el modelo de colas propuesto en la sección anterior. Sin embargo, en todas ellas hay que tener en cuenta las restricciones impuestas por el contexto de trabajo. En primer lugar, el middleware Grid se soporta sobre una capa de componentes independientes que no puede ser reemplazada ni modificada (al menos sin una enorme inversión de tiempo y de recursos, y aún así, en muchos casos es incompatible con los requerimientos locales de los proveedores de recursos). De ahí que, gran parte de la funcionalidad del Grid se dedique a interconectar sistemas que existían mucho antes que éste, y que sirven para gestionar recursos distribuidos. Específicamente en el caso que nos ocupa, el sistema de gestión de recursos y el sistema de colas local no son accesibles, al depender de políticas locales. Por tanto, la capacidad de actuación se limita principalmente al meta-planificador Grid, donde se podrá tomar una decisión que luego ayude a decidir cuáles recursos participarán en la ejecución de un trabajo. Esto nos deja con muy pocas posibilidades para insertar un algoritmo de toma de decisiones con respecto a los recursos. La principal consecuencia de esta restricción es que nos obliga a tomar decisiones muy precisas a la hora de seleccionar los recursos. Esto es así porque la retroalimentación tiene una latencia muy alta, considerando que los trabajos fallidos se vuelven a enviar varias veces desde el meta-planificador y luego desde los clientes, lo cual tiene el inconveniente de que los sistemas de monitorización que informan a los clientes del estado de los trabajos suelen tener intervalos de refresco muy grandes (del orden de decenas de minutos en algunas Organizaciones Virtuales de EGEE).

En la **Figura 3-5** hay dos puntos de separación, que son los puntos donde se pueden producir salidas fallidas. Sin embargo, el camino de retroalimentación C está más controlado porque ocurre dentro del meta-planificador, y además, es el que menos demora ocasiona. Así, el punto crítico se encuentra en las áreas de servicio que representan a los nodos, que es cuando termina el proceso de planificación y comienza la ejecución del trabajo. En este punto se pueden producir varios fallos que aumentan la probabilidad de retroalimentación, aunque algunos de ellos se pueden prever de antemano, incluso desde el meta-planificador Grid. Por ejemplo, un trabajo que tenga unos requerimientos específicos en cuanto a los recursos que necesita o en cuanto al tiempo máximo que puede tardar en conseguir resultados, es posible que no se disponga de información completa y actualizada que permita evitar planificar el trabajo sin posibilidad de éxito. Mejorar este tipo de decisiones contribuye a minimizar la probabilidad de retroalimentación, a la vez que favorece la utilización racional de los recursos del Grid.

Antes de comentar las posibles soluciones, necesitamos formalizar el problema, y para ello simplificaremos el modelo combinando las variables y funciones que describen a los componentes que no pueden tomar parte en la decisión, dejando solamente a los que sí pueden mediar en la solución.

Para el modelo de la **Figura 3-5**, se establecen las siguientes relaciones:

$$\lambda = \sum_{i=1}^{m} \lambda_i + \sum_{j=1}^{n} \rho_j \qquad \textbf{Ecuación 3-1}$$

Donde λ es la tasa de llegada de las solicitudes, que combina las tasas de entrada de los clientes λ_i y de retroalimentación ρ_j, m es la cantidad de clientes enviando trabajos al Grid, y n es la cantidad de recursos procesando trabajos del Grid.

Los trabajos pasan por los diferentes subsistemas de planificación a las áreas de servicio en los nodos de ejecución, además de pasar por el mecanismo de contención de los recursos. La salida de las áreas de servicio se dividen en salidas exitosas y salidas fallidas que se retroalimentan a las colas. Los recursos no son uniformes, en consecuencia cada uno está caracterizado por su propia tasa de salidas exitosas y de retroalimentación. La **Ecuación 3-2** representa la fórmula de "conservación de los trabajos":

$$\sum_{i=1}^{m} \lambda_i = \sum_{i=1}^{n} (\rho_i + \kappa_i) \qquad \textbf{Ecuación 3-2}$$

La probabilidad de retroalimentación P es la suma de las probabilidades de los recursos:

$$P = \sum_{i=1}^{n} P_i = \sum_{i=1}^{n} \frac{\rho_i}{\kappa_i + \rho_i} \qquad \textbf{Ecuación 3-3}$$

Donde P_i, es la probabilidad de retroalimentación de un recurso particular. Además, la **Ecuación 3-2**, debe cumplir con la **Condición 3-4**:

$$\forall i = 1 \ldots n, \rho_i + \kappa_i > 0 \qquad \textbf{Condición 3-4}$$

Esta condición asegura que $P \neq \infty$, si al menos 1 trabajo se planifica en cada servicio, o se excluye el servicio.

3.3.2. ANÁLISIS DE LAS ECUACIONES

En el modelo de colas de la **Figura 3-5**, la **Ecuación 3-3** expresa la probabilidad de que la salida tome un camino de retroalimentación. En consecuencia, el valor mínimo

que puede tomar P es cero, en el caso de que todos los trabajos enviados al Grid terminen con éxito:

$$\forall i = 1 \dots n, P_i = 0 \qquad \textbf{Condición 3-5}$$

Teniendo en cuenta que la tasa de salidas exitosas y la tasa de de retroalimentación de un recurso se contraponen, de la **Ecuación 3-3** se deduce que para minimizar la probabilidad de que la salida tome un camino de retroalimentación es necesario maximizar la tasa de salidas exitosas.

3.4. SELECCIÓN DE RECURSOS EN UN SISTEMA DISTRIBUIDO

La sección anterior describe un modelo de colas para el sistema de planificación Grid, y muestra, a través de las ecuaciones que describen el sistema, que minimizar la probabilidad de que la salida tome un camino de retroalimentación implica maximizar la tasa de salidas exitosas. En esta sección analizaremos qué problemas sufren los sistemas distribuidos para seleccionar los recursos de forma tal que la tasa de salidas exitosas sea la máxima posible. Al estudiar el estado del arte se hizo referencia al trabajo de Fernandez-Baca [10] que demuestra la imposibilidad de resolver, en tiempo polinómico, el problema de seleccionar los recursos que minimicen el coste computacional de ejecutar un trabajo determinado en un sistema distribuido. La importancia de este trabajo es que demuestra que la complejidad computacional del problema general de selección de recursos para minimizar el coste total de ejecución y comunicación es NP-completa, a la vez que prepara las bases para la discusión de otros problemas más específicos, en los cuales las variables a optimizar son la utilización de memoria, el balance de cargas, el tiempo de espera en las colas, etc.

El trabajo de Fernandez-Baca plantea que para seleccionar el conjunto de recursos distribuidos que minimiza el coste total de ejecución y comunicación de un trabajo es necesario explorar todo el espacio de los recursos, utilizando para ello alguna representación que permita ordenar la búsqueda de los recursos. Los sistemas distribuidos introducen una complejidad adicional en la búsqueda, que es la imposibilidad de conocer el estado real de todos los nodos que participan en el sistema. Esta característica viene dada por la distribución de los nodos, que pueden encontrarse a miles de kilómetros de distancia unos de los otros, interconectados mediante redes heterogéneas con características muy diferentes, lo cual hace imposible conocer el estado real de cada nodo en cada momento, principalmente cuando el número de nodos es muy grande. De aquí, que la búsqueda se restringe a un espacio limitado, específicamente al entorno más próximo al nodo que realiza la búsqueda. Esta consideración permite encontrar soluciones aproximadas para este problema en tiempo polinomial.

De forma semejante al problema analizado por Fernandez-Baca, el caso de querer maximizar las salidas exitosas implica analizar todo el espacio de los recursos, con el objetivo de vincular los trabajos con los recursos que tengan las mejores condiciones

para completar cada trabajo con éxito. Como en el caso del coste total de ejecución y comunicación, esta exploración solamente puede ser llevada a cabo en un contexto limitado, y por lo tanto la solución a este problema requiere de algún tipo de aproximación que permita acortar el tiempo de búsqueda.

3.5. ANÁLISIS DE LAS CONDICIONES DE SOLUCIÓN

La Sección 3.1 discute algunos de los trabajos más relevantes en el tema de la gestión de recursos en el Grid de acuerdo a requerimientos de QoS. Mientras que en la asignación de recursos tradicional en los sistemas multiprocesadores los indicadores estáticos tienen más peso que los indicadores dinámicos, en los sistemas Grid ocurre que los indicadores dinámicos predominan sobre los estáticos [51]. En general, los sistemas de información Grid, como WS-MDS, MDS y BDII (Berkeley Database Information Index), son servicios con un alto nivel de centralización, que proporcionan información acerca de los indicadores estáticos y dinámicos. A esto se contrapone el hecho de que para conseguir la escalabilidad necesaria, las organizaciones virtuales se ven obligadas a utilizar tiempos de refresco bastante altos. En consecuencia, es muy improbable que en el Grid se puedan conseguir niveles de calidad de servicio como los que se obtienen en un entorno local controlado, como un clúster. Una forma de afrontar esta dificultad consiste en reservar un conjunto de recursos anticipadamente a la ejecución de un trabajo. Sin embargo, la reserva anticipada tiene una desventaja muy grande al implicar la cooperación de los diferentes gestores de recursos locales, que operan cada cual bajo sus propias políticas, y no siempre pueden soportar este tipo de técnicas avanzadas de planificación. No obstante, la imposibilidad de controlar los recursos locales (debido a que en el Grid, los sitios locales no ceden el control de los recursos) limita la aplicabilidad de esta solución. Un segundo enfoque consiste en predecir las prestaciones de las aplicaciones y los recursos en el Grid. Si se consigue predecir el nivel de servicio de los recursos con suficiente exactitud, es posible mejorar la asignación de los recursos y la planificación de la ejecución de trabajos en el Grid. Para esto es necesario disponer de un sistema de monitorización eficiente, que permita detectar los cambios que puedan producirse en el Grid.

De este planteamiento se deducen dos conclusiones, por un lado, es imprescindible contar con información del estado real de los recursos para realizar la asignación. En consecuencia, cualquier algoritmo debe contar con información actualizada de los recursos, y para ello debe utilizar un sistema de monitorización que además de eficiente, sea efectivo.

Por otro lado, se hace necesario contar con un mecanismo preciso para evaluar los requerimientos de un trabajo antes de que éste sea enviado al Grid. De los mecanismos desarrollados en trabajos anteriores, la utilización de casos de uso significativos en combinación con resultados históricos parece ser la forma más factible para conseguir estas estimaciones.

Finalmente, y contando con soluciones para los dos aspectos anteriores, es importante analizar la diferenciación de servicios, porque es la solución utilizada por la mayoría de los autores. En consecuencia, hay un número importante de beneficios y limitaciones descritas para este caso. La limitación fundamental se encuentra en el proceso de

clasificación de los servicios. De la elección del método de clasificación dependen luego la eficiencia y la consistencia de los resultados. Además, algunos métodos de clasificación son de usabilidad limitada, como por ejemplo los métodos cooperativos, que dependen de que todos los participantes del Grid se comporten correctamente. Un problema asociado a la clasificación de servicios en entornos distribuidos son las dificultades que imponen estos entornos para la utilización de métodos tradicionales de clasificación, como por ejemplo el análisis de clústeres.

Muchos de los trabajos enfocan la gestión de recursos como parte de un mecanismo más amplio orientado a resolver problemas económicos. Los problemas más abordados son la minimización del coste económico para la ejecución de trabajos en el Grid, y el efecto de la fijación de precios en la utilización de los recursos del Grid. En este caso, la elección del modelo económico es un factor crítico para obtener resultados consistentes.

Por último, todos los autores coinciden en que la escalabilidad, la localización de los recursos y los costes asociados a la utilización de redes con latencias y anchos de banda heterogéneos son factores decisivos para conseguir algoritmos con buenas prestaciones en el Grid.

3.6. CONCLUSIONES DEL CAPÍTULO

Existen varios sistemas que permiten vincular trabajos con recursos distribuidos, y planificar el orden de ejecución de los mismos en base a sus requerimientos de QoS. Las características del Grid hacen que este problema sea de difícil solución, de ahí que existan diversos enfoques y varios proyectos de estandarización.

La asignación de recursos en el Grid puede ser abordada en un modelo de colas. Desde este punto de vista, la solución del problema consiste en minimizar la probabilidad de que la salida de las colas tome un camino de retroalimentación, para lo cual es necesario maximizar la tasa de salidas exitosas. Para conseguir esto, es imprescindible contar con información del estado real de los recursos para realizar la asignación. Además, es necesario contar con un mecanismo preciso para evaluar los requerimientos de un trabajo antes de que éste sea enviado al Grid. De todos los factores a tener en cuenta, la escalabilidad, la localización física de los recursos y la heterogeneidad de las redes son claves para mejorar la asignación.

Capítulo 4

4. ASIGNACIÓN DE RECURSOS EN ENTORNOS DE COMPUTACIÓN GRID UTILIZANDO ALGORITMOS LOCALES Y TÉCNICAS DE ANÁLISIS DE CLÚSTERES

Este capítulo describe el trabajo realizado con el objetivo de modelar, desde el punto de vista computacional, el problema de la asignación de recursos en entornos de computación Grid y su solución, específicamente utilizando algoritmos locales y técnicas de análisis de clústeres. Además, se aborda la descomposición del dominio de búsqueda de recursos en dominios de radio acotado por cierta distancia de red, se formula un método de diferenciación y asignación de recursos basado en la clasificación de servicios por sus capacidades y por el estado de los recursos que los soportan, y se propone un algoritmo de asignación de recursos en base a una función de costes.

4.1. MODELO COMPUTACIONAL DE ASIGNACIÓN DE RECURSOS EN ENTORNOS DE COMPUTACIÓN GRID

El middleware Grid actual está preparado para soportar, principalmente, aplicaciones de alta productividad (High-Throughput Computing o HTC) y aplicaciones de alto rendimiento (High-performance computing o HPC). Entre estas, las aplicaciones más comunes son las que utilizan un solo núcleo para hacer procesamiento de trabajos por lotes, y en menor extensión las que utilizan varios núcleos y bibliotecas software de paso de mensajes para implementar la comunicación entre los procesos que se ejecutan en el mismo clúster de ordenadores.

Estos dos grupos de aplicaciones utilizan recursos computacionales gestionados centralmente (por ejemplo, un clúster de ordenadores con un sistema de colas y una lista de nodos donde los trabajos son ejecutados). En consecuencia, la mayoría de los despliegues Grid están organizados en sitios donde los proveedores tienen físicamente agrupados sus recursos computacionales. Cada uno de los sitios hospeda a uno o más clústeres de ordenadores, donde los nodos del mismo clúster están interconectados a través de una red dedicada, de baja latencia y gran ancho de banda, y donde cada clúster está equipado con uno o varios nodos que gestionan y controlan los trabajos que son enviados al mismo.

Así, el primer paso para construir un modelo de asignación de recursos en el Grid consiste en definir un grupo de indicadores que permita caracterizar a los nodos y a los clústeres de ordenadores, en cuanto a disponibilidad y prestaciones. La literatura define un número de indicadores que pueden utilizarse para estos fines [58]. En esta tesis hemos seleccionado dos conjuntos de indicadores cuantitativos que caracterizan, por un lado, de forma estática a un grupo de servicios, y por otro, el estado de un recurso

particular de forma dinámica (por ejemplo, un nodo o un conjunto de nodos, como un clúster o un grupo de nodos que reciben trabajos de una cola) en un momento dado de tiempo. El primer conjunto está formado por la capacidad total de memoria física y swap instalada en un recurso, el tiempo de respuesta del mismo en el mejor escenario (con toda la capacidad de procesamiento y de memoria disponible, y las colas vacías), la disponibilidad, el rendimiento (número de solicitudes completadas con éxito por unidad de tiempo) y la seguridad (características que una entidad debe tener para ser considerada segura). El segundo grupo está formado por la carga de la CPU (tomando los promedios de los últimos 1, 5 y 15 minutos) y la utilización de la memoria.

Los indicadores estáticos cambian como consecuencia de interacciones fuertes, que son interacciones poco frecuentes que implican una actuación importante, como la reconfiguración de los nodos, actualizaciones de software, etc., mientras que los indicadores dinámicos cambian debido a la utilización normal de los nodos. Como los sistemas computacionales se comportan de forma impredecible bajo la influencia de interacciones fuertes, es posible asumir que, en condiciones normales, un sistema estable (por ejemplo, un sistema en producción) tiene pocas probabilidades de sufrir este tipo de cambios. De ahí que en esta tesis consideramos que los indicadores estáticos no constituyen un problema para la monitorización del Grid, y que por tanto, los indicadores estáticos pueden ser colectados, evaluados y distribuidos utilizando los sistemas de información Grid combinados con unos pocos servicios diseñados para este fin. Los cambios de estado en los indicadores estáticos también pueden ser detectados a través de puntos de control convenientemente ubicados en el código de los servicios Grid, que utilizarán las mismas vías explicadas anteriormente para difundir el nuevo estado a los servicios del Grid encargados de tomar las decisiones de asignación.

En cambio, los indicadores dinámicos representan un reto para la monitorización del Grid, que no puede ser resuelto utilizando el middleware Grid actual. Estos indicadores cambian con demasiada frecuencia, por lo que no es posible recogerlos en los nodos de un sitio y propagarlos, de forma eficiente, al resto de los sitios. Más aún, el nivel de centralización de los sistemas de planificación Grid puede hacer que se conviertan en cuello de botella para otros servicios. La forma de evitar que esto suceda consiste en disminuir la carga de los sistemas de planificación, y para ello los administradores de las VO se ven obligados a aumentar los tiempos de refresco de la información. Por ejemplo, las últimas versiones de gLite actualizan la información del sistema de gestión de trabajos (WMS) cada 20 o 30 minutos, mientras que la información del subsistema de registro y contabilidad (LB) es actualizada con mayor frecuencia. En consecuencia, la información en los distintos componentes del Grid es inconsistente y contradictoria, y por tanto, no permite tomar decisiones que soporten una estrategia de asignación de recursos enfocada a optimizar los costes, u otro indicador de nivel de servicio.

Una vez definidos los indicadores, el resto de la sección describe una metodología basada en computaciones locales, en un entorno distribuido, para diferenciar los servicios en el Grid. Cada servicio puede representar un nodo, un conjunto de nodos o un clúster de ordenadores. La metodología está basada en un modelo computacional que representa cada servicio como un conjunto de coordenadas en un espacio de N dimensiones, donde N es el número de indicadores estáticos a tener en cuenta para la

asignación de recursos. Mediante un algoritmo de análisis de clústeres se divide el conjunto de servicios en *k* clústeres, de forma tal que los servicios en el mismo clúster tienen prestaciones semejantes. Sobre esta base se establece un sistema de prioridades para la asignación de servicios, que también puede utilizarse para fijar los precios por utilizar los servicios. Los usuarios pagarían un precio mayor por utilizar los servicios en los clústeres con mayor puntuación que por los servicios en clústeres con menor puntuación.

El primer paso en la metodología consiste en evaluar, de forma asíncrona, los cambios en la disponibilidad y en las prestaciones del sistema, y propagarlo a los nodos que participan en la asignación de recursos, en cada sitio Grid. En un segundo paso, estos nodos utilizarán la información disponible en su vecindad local (de la que forman parte todos los nodos que se encuentran en un segmento de red definido de antemano) para calcular los clústeres. El paso final consiste en que dichos nodos encontrarán, de forma independiente, la distribución de recursos que resuelve un problema de asignación definido. Cada nodo iterará sobre estos pasos, provocando nuevos eventos y cambiando el comportamiento del algoritmo de asignación en caso de que fuese necesario. Por ejemplo, un nodo puede ajustar la cantidad de mensajes que envía al resto del Grid, y su contenido, para reflejar el hecho de que no se han producido cambios importantes en las prestaciones del sistema estudiado.

4.2. DESCOMPOSICIÓN DEL DOMINIO DE BÚSQUEDA

Analizaremos primero cómo se lleva a cabo la descomposición del dominio de búsqueda, y luego describiremos cómo es posible realizar esta operación en un entorno de computación distribuida, donde por motivo de eficiencia y escalabilidad no es posible utilizar algoritmos que exploran todo el dominio de búsqueda.

La **Figura 4-6** muestra un gráfico con los resultados de una simulación llevada a cabo para estudiar el comportamiento de un sistema distribuido. El entorno de simulación consiste en una red de máquinas virtuales creadas con VMware [59] en un servidor Linux. Un puente de red conecta las máquinas virtuales entre sí. Cada máquina virtual ejecuta Linux, y sobre éste ejecuta un servicio Web que toma por entrada un número entero, genera un conjunto de números aleatorios que contiene tantos números como han sido especificados en la entrada, los ordena en orden ascendente, y calcula la suma de todos los números del conjunto. Varios clientes envían solicitudes a los servicios, siguiendo una política coordinada globalmente, de planificación por turnos rotatorios. Para cada nueva solicitud, el cliente en cuestión selecciona un número al azar de un listado de posibles tamaños de trabajo. Al final de la solicitud, el cliente recibe una respuesta del servicio, que consiste en un conjunto de indicadores de ejecución, como el rendimiento y el tiempo de respuesta, para cada tamaño de problema. Esta respuesta es inmediata, y se produce antes de que el servicio ejecute la solicitud. Además, cada cliente lleva un registro de las veces que ha enviado un trabajo a un servicio y no ha recibido respuesta de éste, en un cierto plazo de tiempo fijado para todo el sistema. Este registro caracteriza a cada servicio en cuanto a disponibilidad.

El gráfico de la **Figura 4-6** muestra el estado del sistema, en un cierto instante de tiempo. En cuanto a funcionalidad, todos los servicios son equivalentes e intercambiables entre sí. Sin embargo, la figura muestra que, en cuanto a características no funcionales (como carga de trabajo, disponibilidad de recursos, etc.), los servicios son diferentes.

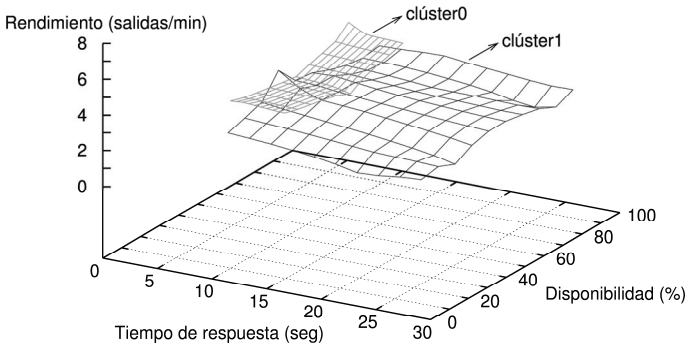


Figura 4-6: El subconjunto de servicios que superan unos valores umbrales definidos de tiempo de respuesta (en segundos), disponibilidad (en tanto por ciento) y de rendimiento (en solicitudes completadas con éxito por minuto), han sido clasificados en dos clústeres, utilizando para ello el algoritmo de k-medias con k=2.

Las dos superficies representadas en la **Figura 4-6** contienen una parte de los servicios del sistema distribuido. El resto de los servicios (que no aparecen en la figura) fueron filtrados utilizando unos valores umbrales para cada una de las coordenadas representadas en el gráfico. Las superficies fueron calculadas utilizando el algoritmo de k-medias [60]. El algoritmo de k-medias tiene como objetivo descomponer un conjunto de n puntos de un espacio en k clústeres, en los cuales cada punto pertenece al clúster con la media más próxima a las coordenadas del punto (la siguiente sección proporciona más detalles acerca de este algoritmo). De esta forma, los servicios forman parte, o bien del clúster 0 o del clúster 1, o de los servicios filtrados antes de proceder con el análisis de clústeres. Con el objetivo de homogenizar la nomenclatura, nos referiremos a los servicios filtrados como clúster de servicios no asignables, o simplemente clúster 2. El hecho de que los servicios que forman parte del clúster 2 no sean asignables se debe a que los mismos no cumplen con uno o varios de los requerimientos, es decir que estos servicios no sobrepasan los valores umbrales requeridos para una aplicación específica.

La **Figura 4-7** representa una vista parcial del sistema representado en la **Figura 4-6**. Esta vista muestra una clasificación alternativa de los servicios utilizando solamente uno de los planos de la **Figura 4-6**, específicamente el plano de la disponibilidad y el tiempo de respuesta.

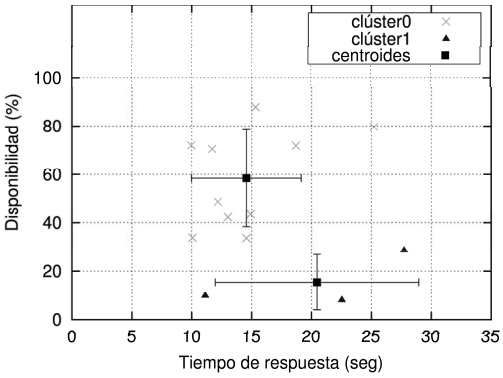


Figura 4-7: Una vista particular del análisis de clústeres, tomando en cuenta solamente dos de las coordenadas originales. En este caso los servicios son representados como puntos en el gráfico, junto con los centroides de los clústeres y las desviaciones estándares.

En esta vista, los mejores servicios son los que se encuentran más cerca de la esquina superior izquierda del gráfico porque tienen una disponibilidad alta y un tiempo de respuesta pequeño. En el caso particular representado en la **Figura 4-7**, el clúster 0 es, sin duda alguna, el mejor de todos los clústeres. De forma analítica se podría utilizar la posición de los centroides en el espacio de coordenadas como criterio de clasificación de los clústeres en cuanto a la calidad de los servicios que contienen. Sin embargo, en la práctica hay un factor adicional a tener en cuenta: la distribución de los elementos en los clústeres.

La importancia de los centroides en la selección del mejor clúster se puede inferir directamente del análisis de la **Figura 4-7**. Como el análisis de clústeres minimiza las desviaciones estándares de las distancias entre los elementos en el mismo clúster, la posición del centroide refleja los valores promedio de los valores de los indicadores de los servicios en el mismo clúster. De forma similar, la influencia de la distribución de los elementos en el clúster puede ser inferida de la **Figura 4-8**.

La **Figura 4-8** representa un gráfico de densidades de distribución de los elementos en el clúster 0. Este gráfico muestra la distancia desde el centroide a cada una de las coordenadas representadas en la **Figura 4-7**, para cada uno de los servicios del clúster 0. En este gráfico se puede observar que los servicios del clúster 0 están agrupados en torno al tiempo de respuesta, pero dispersos con respecto a la disponibilidad.

Normalmente, la posición de los centroides debería ser suficiente para diferenciar los clústeres entre sí. Solamente cuando se dan clústeres muy dispersos, el valor de la distribución de los elementos en cada clúster debería tener un peso significativo en este análisis. Sin embargo, los clústeres dispersos son más frecuentes de lo que podría esperarse. Esto se debe a que los algoritmos que fijan un número de clústeres de antemano (como el algoritmo de las k-medias) imponen una restricción en la clasificación que hace que todos los servicios sean clasificados en uno de los k clústeres definidos. Esta restricción mejora la velocidad de convergencia respecto a los

algoritmos que además de clasificar los elementos tienen que calcular el número óptimo de clústeres, pero a la vez produce clústeres más dispersos.

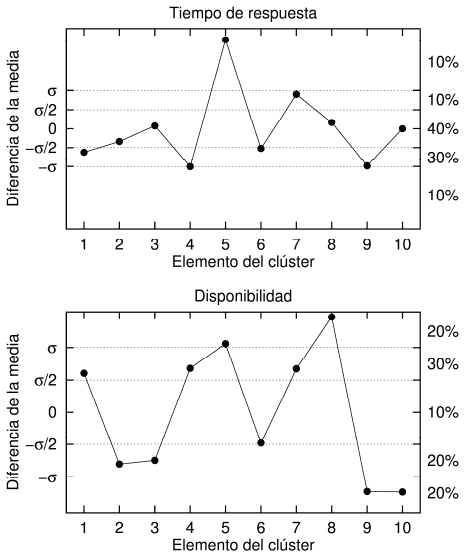


Figura 4-8: Densidad de distribución de los elementos en el clúster 0 con respecto a las coordenadas del centroide. Cada gráfico muestra el tanto por ciento de elementos del clúster 0 que se encuentran a una distancia del centroide comprendida en uno de los siguientes intervalos: $(-\infty, -\sigma]$, $(-\sigma, -\frac{\sigma}{2}]$, $(-\frac{\sigma}{2}, \frac{\sigma}{2})$, $[\frac{\sigma}{2}, \sigma)$ y $[\sigma, +\infty)$, donde σ es la desviación estándar.

4.2.1. DESCOMPOSICIÓN DEL DOMINIO DE BÚSQUEDA EN ENTORNOS DISTRIBUIDOS

En el año 2007, Krivitski y colaboradores [61] propusieron una solución al problema de localización de instalaciones en sistemas distribuidos de gran escala. El problema de localización de instalaciones (facility location), también conocido como análisis de la ubicación, es una rama de la investigación de operaciones que consiste en modelar y solucionar problemas relacionados con la colocación de instalaciones, con el fin de minimizar los costes de transporte, alejar los materiales peligrosos de las zonas pobladas, superar a la competencia, etc.

El enfoque presentado por estos autores tiene un interés especial para el Grid porque demuestra la posibilidad de utilizar algoritmos locales, en los cuales cada nodo computa localmente un resultado utilizando la información de sus vecinos más próximos, para resolver un problema de descomposición de datos en entornos donde la escalabilidad y el dinamismo son problemas críticos. En estas condiciones, la carga computacional y de comunicaciones de cada nodo es independiente del número de nodos del sistema.

La próxima sección propone un algoritmo de asignación de recursos, basado en esta idea, que permite la descomposición del dominio de búsqueda en subdominios para resolver el problema de la asignación de recursos a un trabajo de acuerdo a requerimientos de QoS.

4.3. ALGORITMO DE ASIGNACIÓN DE RECURSOS

La **Figura 4-9** presenta un diagrama del algoritmo de asignación de recursos, en el que participan los nodos más próximos agrupados en grupos llamados vecindarios. El algoritmo ha sido dividido en varios pasos. Estos pasos son: 1) Alineamiento: Vincula las solicitudes de trabajo con los servicios que podrían ejecutar con éxito los trabajos, 2) Análisis de clústeres: Clasifica los servicios en diferentes grupos de posibles candidatos para la ejecución de trabajos, 3) Corrección de pesos: Estima la capacidad de un servicio para ejecutar un trabajo causando el mínimo impacto posible al resto de los procesos mediados por el mismo recurso, y 4) Asignación: Asigna el servicio más apropiado para ejecutar una solicitud.

Como se observa en la **Figura 4-9**, el algoritmo se acopla al proceso de meta-planificación. Una solicitud de trabajo consiste en un conjunto de especificaciones que son enviadas al meta-planificador Grid utilizando algún formato conocido por el mismo, por lo general un documento JSDL (Job Submission Description Language). Una solicitud normal no contiene requerimientos de QoS, y es procesada directamente por el meta-planificador, que asigna el trabajo a los recursos más apropiados para su ejecución, planifica el mismo en el CE que haya sido seleccionado y, por último, monitoriza su ejecución. En cambio, una solicitud que además tenga requerimientos de QoS es procesada por el algoritmo de asignación, que filtra los servicios disponibles por su nivel de servicio (Level of Service o LoS), luego los clasifica utilizando un algoritmo de análisis de clústeres y, por último, los ordena según su carga de trabajo. Opcionalmente, el algoritmo de asignación puede optimizar el listado de servicios disponibles utilizando una función de costes (Cost Function o CF). En cualquier caso, el resultado del algoritmo de asignación consiste en un documento XML, como el mostrado en el ejemplo de la **Figura 4-10**, que contiene un listado de servicios clasificados según su idoneidad para ejecutar el trabajo enviado al Grid, con las garantías de QoS que se requieren.

El listado de este ejemplo contiene tres servicios, y para cada uno de ellos proporciona una referencia al *endpoint* del servicio, formada por el identificador uniforme de recurso o URI (acrónimo del inglés Uniform Resource Identifier) del recurso (por ejemplo, un clúster de ordenadores), también conocido URI del espacio de nombre (del inglés namespace), y la parte local de la referencia, que en este caso es el nombre del servicio. Los valores de los elementos *service-suitability*, *cluster-score* y *container-load* se refieren, respectivamente, al índice de idoneidad del servicio, al índice de calidad del clúster donde el algoritmo ha clasificado al servicio y a la carga del recurso donde el servicio está desplegado, definidos en la Sección 4.3.4, cuando se explica la **Ecuación 4-7**. El elemento *offer* expresa si el servicio cumple con los requerimientos del trabajo, o no.

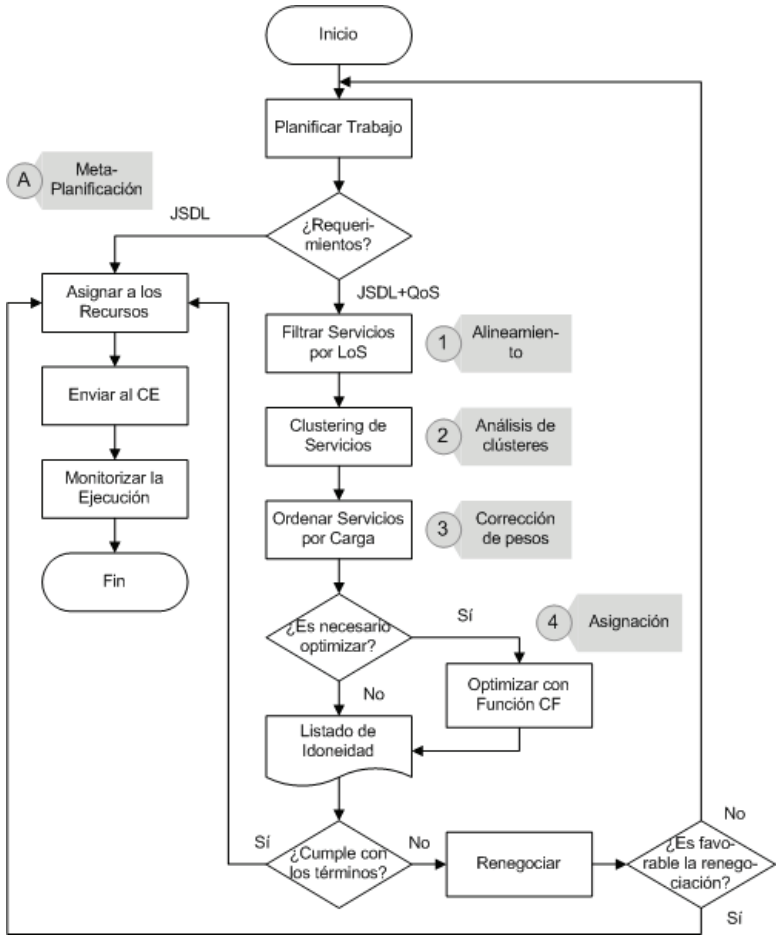


Figura 4-9: Algoritmo de asignación.

Finalmente, puede ocurrir que ninguno de los servicios disponibles cumpla con los requerimientos de QoS que tiene el trabajo. En este caso, se procede a renegociar los términos con el cliente que envió el mismo y si la negociación es favorable, se envía el listado de servicios disponibles al meta-planificador, utilizando un formato conocido por el mismo, por ejemplo una listado con los recursos organizados por la preferencia que tienen para la asignación. Esta salida también puede ocurrir en el paso anterior, si el algoritmo encuentra servicios apropiados para la ejecución. Hay una última salida posible, que ocurre cuando la renegociación no es favorable, en cuyo caso el algoritmo de asignación termina sin reenviar la solicitud al meta-planificador, pero antes envía una notificación al cliente que envió el trabajo, para que este pueda tomar una decisión al respecto, que por lo general consiste en esperar un tiempo y enviar nuevamente la solicitud, o bien modificar los requerimientos de la misma antes de reenviarla.

```
<?xml version="1.1" encoding="UTF-8"?>
<bic:best-in-class version="1.1"
    xmlns:bic="http://upv.org/adgrid/domain/best-in-class_service"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://upv.org/adgrid/domain/best-in-class_service
        ../xmltypes/schemas/best-in-class_service.xsd ">
    <bic:best-suitable-service>2</bic:best-suitable-service>
    <bic:service id="1">
        <bic:ns-uri>http://uri1.org:80/wsrf/services</bic:ns-uri>
        <bic:local-part>TestService</bic:local-part>
        <bic:offer>true</bic:offer>
        <bic:quality-description>
            <bic:cluster-score>0.0</bic:cluster-score>
            <bic:container-load>0.12</bic:container-load>
            <bic:service-suitability>0.0</bic:service-suitability>
        </bic:quality-description>
    </bic:service>
    <bic:service id="2">
        <bic:ns-uri>http://uri2.org:80/wsrf/services</bic:ns-uri>
        <bic:local-part>TestService</bic:local-part>
        <bic:offer>false</bic:offer>
        <bic:quality-description>
            <bic:cluster-score>0.77</bic:cluster-score>
            <bic:container-load>0.75</bic:container-load>
            <bic:service-suitability>0.12</bic:service-suitability>
        </bic:quality-description>
    </bic:service>
    <bic:service id="3">
        <bic:ns-uri>http://uri3.org:80/wsrf/services</bic:ns-uri>
        <bic:local-part>TestService</bic:local-part>
        <bic:offer>false</bic:offer>
        <bic:quality-description>
            <bic:cluster-score>0.23</bic:cluster-score>
            <bic:container-load>0.71</bic:container-load>
            <bic:service-suitability>0.07</bic:service-suitability>
        </bic:quality-description>
    </bic:service>
</bic:best-in-class>
```

Figura 4-10: Ejemplo de un documento XML que contiene un listado de idoneidad en el formato utilizado por el algoritmo de asignación. El elemento *offer* expresa si el servicio cumple con los requerimientos del trabajo (*offer=false*), o no (*offer=true*). En caso de que ninguno de los servicios pueda alcanzar el nivel de servicio requerido por el trabajo, el algoritmo iniciará una renegociación con el cliente que envió el mismo. Si el cliente acepta uno o varios de los servicios ofrecidos, entonces el proceso de asignación continuará, en caso contrario el algoritmo descartará la solicitud.

4.3.1. FASE DE ALINEAMIENTO

El primer paso del algoritmo consiste en alinear los requerimientos de una solicitud de trabajo con el conjunto de indicadores estáticos monitorizados en el sistema. La **Figura 4-9** representa este paso en el algoritmo en un bloque llamado "*Alineamiento*". *Alineamiento* diferencia los servicios en tres conjuntos: a) servicios que cumplen con los requerimientos necesarios para ejecutar la solicitud, b) servicios no asignables porque no tienen uno o más indicadores estáticos, ya sea porque en la declaración de servicio no han suscrito los indicadores al sistema de monitorización o bien porque el sistema de monitorización aún no ha podido recoger suficientes valores de estos indicadores para obtener un estado del servicio, y c) servicios que no cumplen con alguno de los requerimientos.

4.3.2. FASE DE ANÁLISIS DE CLÚSTERES

En un segundo paso, representado como "*Análisis de clústeres*" en la **Figura 4-9**, los servicios son clasificados en tres clústeres: C_0, C_1, y C_2.

Tomando como coordenadas para el análisis de clústeres, los indicadores estáticos de los servicios que cumplen los requerimientos, se aplica el algoritmo de las k-medias [60] para construir los clústeres C_0 y C_1. En dependencia de los requerimientos evaluados para cada trabajo, el análisis de clústeres puede incluir un número más o menos grande de servicios. Por esta razón, el análisis de clústeres pone más énfasis en la velocidad de convergencia del método de clasificación, que en encontrar la mejor clasificación. En general, el algoritmo de las k-medias es lento porque itera sobre todo el conjunto de datos hasta que alcanza la convergencia. El algoritmo "*Farthest First Traversal*" [62] proporciona un método de inicialización simple, rápido y eficiente para el algoritmo de las k-medias. Utilizando este algoritmo es posible encontrar un conjunto inicial de clústeres que acelera la convergencia del algoritmo de las k-medias.

Con el objetivo de contrarrestar el efecto que puede tener sobre la clasificación el hecho de restringir el número de clústeres producidos a dos, el algoritmo hace un pre-procesamiento de los datos antes del análisis de clústeres. Este pre-procesamiento consiste en transformar las coordenadas de los servicios utilizando para ello un criterio de amortiguamiento que suaviza las coordenadas, a la vez que reduce la influencia que tendrán algunos indicadores estáticos en la clasificación. En concreto, el pre-procesamiento consiste en multiplicar las coordenadas por un conjunto de funciones continuas con crecimiento diferente. Como los indicadores estáticos tienen límites finitos bien definidos en el conjunto de los reales no negativos, es posible utilizar funciones *ad-hoc* para este fin. El Anexo 2 describe las funciones de amortiguamiento que fueron utilizadas en la implementación del prototipo descrito en el Capítulo 5.

Por ejemplo, el criterio de amortiguamiento puede ajustar el algoritmo para que, en el análisis de clústeres, el tiempo de ejecución y la disponibilidad tengan un peso mayor que cualquier otro indicador. La utilización de este criterio de amortiguamiento puede mejorar los resultados producidos en aquellos casos donde es necesario diferenciar los servicios por su capacidad para ejecutar un trabajo en un plazo de tiempo determinado.

De esta forma, el criterio de amortiguamiento puede ajustar los resultados de la clasificación poniendo más importancia en unos indicadores que en otros.

A continuación, los clústeres C_0 y C_1 son clasificados por la calidad de los servicios que contienen, y se les asigna un valor numérico que sirve para evaluar, de forma comparativa, esta diferencia. En este punto, además de este valor numérico de calidad se les puede asignar un precio utilizando una función de fijación de precios. Las puntuaciones de los clústeres son abordadas más adelante en esta sección, en relación con la **Ecuación 4-8**.

El bloque "*Análisis de clústeres*" termina con la construcción del clúster C_2, formado por la unión de los servicios no asignables con los servicios que no cumplen con alguno de los requerimientos.

4.3.3. FASE DE CORRECCIÓN DE PESOS

El tercer paso, que ha sido representado como "*Corrección de pesos*" en la **Figura 4-9**, consiste en calcular el índice de idoneidad para cada servicio de los clústeres C_0 y C_1. En este paso, la información obtenida en el análisis de clústeres es reevaluada de acuerdo a unas políticas de utilización de los recursos. Para ello, el algoritmo utiliza el conjunto de indicadores dinámicos para evaluar la disponibilidad de recursos computacionales en los nodos.

Aunque todos los pasos del algoritmo son importantes, el cálculo del índice de idoneidad es el factor clave de cara a obtener un algoritmo de asignación de recursos que permita mejorar las prestaciones globales del sistema, porque evita que los servicios con mejores indicadores estáticos (que serán los mejores servicios de todo el Grid) se sobrecarguen con trabajo mientras que el resto de los servicios permanecen infrautilizados. En efecto, este paso utiliza la carga de los recursos para modular la asignación de los mismos. Mientras sea mayor la probabilidad de que un recurso esté ocupado con otros trabajos, mayor será la penalización que impone la *Corrección de pesos* a la asignación de nodos de este recurso para ejecutar un trabajo nuevo. Por otra parte, las prestaciones y la precisión del algoritmo de asignación dependen en gran medida de los índices de idoneidad. Mientras que los pasos anteriores del algoritmo dependían solamente de indicadores estáticos, el cálculo de los índices de idoneidad depende además de los indicadores dinámicos. Por esta razón, los pasos *Alineamiento* y *Análisis de clústeres* pueden ser evitados en muchos casos, reutilizando resultados previos, pero el paso *Corrección de pesos* es necesario en todos los casos. Por ejemplo, cuando un planificador o un gestor de recursos reciben una solicitud para procesar un lote de trabajos, de forma tal que todos los trabajos son iguales y lo que varía es el conjunto de datos de entrada de cada uno de ellos, el alineamiento de los requerimientos con los indicadores estáticos y el análisis de clústeres hechos para el primer trabajo pueden ser reutilizados para todos los demás. En cambio, será necesario re-calcular los índices de idoneidad para cada trabajo, antes de asignar los recursos, porque esta es la única forma de obtener una visión actualizada de la disponibilidad de recursos en el sistema. Los índices de idoneidad son retomados más adelante en esta sección, en relación con la **Ecuación 4-7**.

4.3.4. FASE DE ASIGNACIÓN

El último paso del algoritmo, representado como "*Asignación*" en la **Figura 4-9**, selecciona varios servicios del conjunto de posibles candidatos y hace la asignación de recursos. Con el objetivo de garantizar que la asignación de un servicio no implique la penalización de las prestaciones del resto de los procesos ejecutados de forma concurrente en el mismo recurso, para la asignación solamente se tendrán en cuenta aquellos servicios desplegados en recursos cuyos valores de los indicadores dinámicos superen un valor umbral predefinido. De aquí que este paso comience con una reorganización de los clústeres C_0, C_1 y C_2 en tres nuevos clústeres llamados C'_0, C'_1 y C'_2, donde los servicios de C_0 y C_1 que no superan los valores umbrales requeridos son movidos a C'_2.

Los servicios en los clústeres C'_0 y C'_1 difieren en prestaciones, y también pueden diferir en el precio asignado por utilizar sus recursos. Si el coste de utilizar un servicio en el clúster con las mejores prestaciones es asequible para el presupuesto del perfil que solicita la ejecución del trabajo, entonces el trabajo será asignado a ese clúster. En cambio, si el perfil no puede permitirse la ejecución del trabajo en el clúster con las mejores prestaciones (o simplemente no quiere destinar más fondos a costear la ejecución del trabajo en cuestión), entonces los servicios utilizados serán los del segundo mejor clúster. De esta forma, los servicios en el clúster con las mejores prestaciones son utilizados por el algoritmo como una fuente primaria de asignación de recursos, mientras que los servicios en el segundo clúster son utilizados como respaldo para ejecutar trabajos que tengan menos prioridad (de ahí que se pague menos por ellos), y también para seguir asignando recursos a los trabajos cuando los recursos del clúster con las mejores prestaciones se hayan agotado.

En cambio, los servicios del clúster C'_2 serán utilizados solamente cuando no exista ninguna otra posibilidad de asignación, nunca su asignación será automática, ofreciendo al cliente que envía el trabajo dicha posibilidad y dejando en manos del usuario la decisión de ejecutar el trabajo en el servicio, o esperar a que se hayan liberado recursos en los clústeres primario y de respaldo para iniciar otro proceso de solicitud.

La selección de un servicio u otro en el clúster C'_2 tiene un componente de aleatoriedad. En cambio, en el caso de los clústeres C'_0 y C'_1 la selección se basa en un criterio de optimización. En caso de que se utilice una función de fijación de precios, entonces el criterio a optimizar será el coste por ejecutar el trabajo en un servicio, en concreto se tratará de encontrar el conjunto de recursos que minimicen el coste por ejecutar el trabajo. En este caso, a los clústeres C'_0 y C'_1 se les asigna una función de coste basada en el coste por asignar recursos en los servicios del clúster, el índice de idoneidad de los servicios y la carga de los recursos (expresada en los indicadores dinámicos). Adicionalmente, los precios de las funciones de coste se pueden ajustar dinámicamente a través de políticas de fijación de precios que reflejen la carga total del sistema.

Por tanto, las funciones de coste (Cost Function o CF) de los clústeres C'_0 y C'_1 tienen la siguiente forma:

$$\mathrm{CF}_0(x,y) = \mathrm{P}_0(y)/\mathrm{si}(x)$$

$$\mathrm{CF}_1(x,y) = \mathrm{P}_1(y)/\mathrm{si}(x)$$

Ecuaciones 4-6 (a y b)

Donde $si(x)$ representa el índice de idoneidad del servicio x, y $P_0(y)$ y $P_1(y)$ son funciones paramétricas que definen los precios por utilizar recursos en los clústeres C'_0 y C'_1, respectivamente, en las cuales y puede ser, por ejemplo, un parámetro de tiempo.

Como el objetivo final consiste en encontrar el servicio x que minimiza la función de coste del clúster en cuestión, las funciones $P_0(y)$ y $P_1(y)$ deben ser funciones continuas y derivables. Tanto $P_0(y)=P_0$ como $P_1(y)=P_1$, constituyen casos especiales de fijación de precios donde se fija un precio constante por la utilización de los servicios.

El índice de idoneidad de un servicio depende del índice de calidad cs del clúster donde éste se encuentra. El índice de calidad de un clúster es una medida de la calidad de los servicios del clúster, así como de la dispersión de los elementos en el mismo. En consecuencia, el coste por asignar un recurso depende de la pertenencia del servicio que proporciona el recurso a uno de los clústeres. De esta forma, el índice de idoneidad del servicio x, que pertenece al clúster C'_0 se define de la siguiente forma:

$$si(x) = cs_0/\bar{L}(x)$$

Ecuación 4-7

Donde cs_0 es el índice de calidad del clúster C'_0 y $\bar{L}(x)$ es la carga del recurso donde el servicio x está desplegado, normalizado por la disponibilidad máxima de recursos computacionales en los nodos del recurso. Por su parte, el índice de calidad del clúster C'_0 viene dado por:

$$cs_0 = \frac{c_0}{c_0 + c_1} \cdot \frac{1}{\mathrm{d}_0}$$

Ecuación 4-8

Donde c_0 y c_1 denotan la calidad de los elementos en los clústeres C'_0 y C'_1 respectivamente, y d_0 denota la dispersión de los elementos en el clúster C'_0. Estos están dados por:

$$c_0 = \sum_{i=0}^{m} \sum_{j=0}^{n} \frac{1}{\left[T_j - \bar{x}_0(i,j)\right] + 1}$$

Ecuación 4-9

$$c_1 = \sum_{i=0}^{m} \sum_{j=0}^{n} \frac{1}{\left[T_j - \bar{x}_1(i,j)\right] + 1} \qquad \textbf{Ecuación 4-10}$$

$$\frac{1}{d_0} = \frac{1}{m+1}\left\{\left[\sum_{i=0}^{m} \delta\left(\bar{\mathrm{x}}_0(\mathrm{i})\right)\right] + 1\right\} \qquad \textbf{Ecuación 4-11}$$

Donde *m* es el número de elementos en el clúster, *n* es el número de coordenadas de los elementos, T_j es el umbral para la coordenada *j*, $\bar{x}_0(i,j)$ y $\bar{x}_1(i,j)$ son las coordenadas *j* del elemento *i* en los clústeres C'_0 y C'_1 respectivamente, siendo $\bar{x} \in (-\infty, T_j]$, y $\delta(\bar{\mathrm{x}})$ es una función que evalúa si un elemento está en el intervalo $[\bar{\bar{x}} - \sigma, \bar{\bar{x}} + \sigma]$ o no, siendo $\bar{\bar{x}}$ las coordenadas del centroide del clúster y σ es la desviación estándar de los elementos del clúster.

La función $\delta(\bar{x})$ está dada por:

$$\delta(\bar{\mathrm{x}}) = \begin{cases} 1, & \bar{\mathrm{x}} \in [\bar{\bar{\mathrm{x}}} - \sigma, \bar{\bar{\mathrm{x}}} + \sigma] \\ 0, & \text{en otro caso} \end{cases} \qquad \textbf{Ecuación 4-12}$$

Además de los elementos del clúster, el centroide también ha sido añadido a las ecuaciones **Ecuación 4-9**, **Ecuación 4-10** y **Ecuación 4-11**. En el caso de las ecuaciones **Ecuación 4-9** y **Ecuación 4-10**, la contribución del centroide consiste en sumar 1 al denominador del término de la suma. Dado que $\bar{x} \leq T_j$, esto asegura que todas las raíces $\bar{x}_0(i,j)$ y $\bar{x}_1(i,j)$, de las ecuaciones **Ecuación 4-9** y **Ecuación 4-10**, respectivamente, son reales. En el caso de la **Ecuación 4-11**, la contribución del centroide consiste en sumar 1 al denominador del primer factor. Dado que puede darse el caso de que un clúster no contenga ningún elemento ($m = 0$), esto evitaría que el valor de $\frac{1}{d_0}$ pueda resultar indefinido.

Como describe previamente esta sección, las coordenadas $\bar{x}_0(i,j)$ y $\bar{x}_1(i,j)$ fueron obtenidas aplicando las funciones de amortiguamiento a las coordenadas reales, es decir que fueron obtenidas a partir de los indicadores estáticos $x_0(i,j)$ y $x_1(i,j)$, respectivamente. Esta transformación mejora el dominio de entrada al reducir la dispersión de los elementos en el clúster.

Por último, se pueden definir ecuaciones semejantes para el caso $\bar{x} \in [T_j, +\infty)$, que representa requerimientos con límite inferior (por ejemplo, disponibilidad por encima del 99%), aunque es más conveniente utilizar una transformación de coordenadas, y de esta forma utilizar las mismas ecuaciones para resolver ambos casos.

4.4. ANÁLISIS DEL ALGORITMO DE ASIGNACIÓN

Esta sección presenta un estudio que evidencia la corrección del algoritmo presentado en la **Figura 4-9**. La primera apreciación es que todos los pasos del algoritmo conducen a una afirmación. En particular, el paso *Análisis de clústeres* construye los clústeres utilizando el algoritmo de las k-medias. Está demostrado que este algoritmo termina en un número finito de pasos, dando como resultado una distribución de los datos de entrada en k clústeres disjuntos. Por otra parte, el paso *Asignación* hace una optimización de las **Ecuaciones 4-6**. Teniendo en cuenta que $P(y)$ y $si(x)$ son funciones continuas y derivables, se puede afirmar que siempre será posible encontrar al menos un mínimo para la función $CF(x,y)$. Los pasos restantes del algoritmo involucran procedimientos numéricos conocidos, cuyo comportamiento es correcto.

La complejidad computacional del algoritmo de las k-medias es $O(tnk)$, donde t es el número de iteraciones, n es el número de instancias y k es el número de clústeres deseados. De forma similar, el algoritmo "*farthest-first traversal*" tiene un coste computacional de $O(nk)$. Como el paso *Análisis de clústeres* clasifica los servicios en dos clústeres, y considerando el peor escenario donde $t = n/2$, el coste computacional de este paso es $O(n^2)$. Por otra parte, la minimización de la función $CF(x,y)$ tiene, en general, un coste computacional de $O(n^2)$ (por ejemplo, utilizando el algoritmo BFGS [63]). Por tanto, el coste computacional del algoritmo de asignación en el peor escenario es de $O(n^2)$, siendo n el número de servicios en el Grid.

4.5. UTILIZACIÓN DEL ALGORITMO DE ASIGNACIÓN EN ENTORNOS DISTRIBUIDOS

Este capítulo ha propuesto un algoritmo de asignación de recursos en base a una función de costes. El funcionamiento del algoritmo se sostiene sobre la idea de que es posible diferenciar los recursos a través de la clasificación, en base a sus capacidades, de los servicios que los proporcionan, y luego modular esta decisión utilizando información de la disponibilidad de los recursos computacionales de los nodos que soportan los servicios. En particular, todo ello depende de la utilización de dos técnicas. Por una parte, la utilización de técnicas de análisis de clústeres para clasificar a los servicios, y por otra, la utilización de algoritmos locales que hacen posible la utilización del análisis de clústeres en entornos de computación distribuida.

Como hemos visto, el análisis de clústeres permite descomponer el dominio de búsqueda de este problema en subdominios, donde los elementos que componen un subdominios particular tienen características semejantes. En esta sección veremos cómo los algoritmos locales permiten acotar el radio de los subdominios, utilizando para ello una cierta distancia de red.

Sin un límite, sería imposible realizar la búsqueda en un sistema distribuido. Sin embargo, no cualquier límite es válido porque se trata de acortar el radio de la búsqueda, sin acortar su alcance. Los límites que vamos a imponer tienen su base en acotar el número de comunicaciones que realiza el algoritmo de búsqueda, sin que por ello los recursos de algún sitio en particular sean excluidos de la búsqueda. Entonces, la primera condición que tienen que cumplir los límites es que ninguno de los sitios del

Grid quede excluido siempre de la búsqueda. La segunda condición es que en cualquier búsqueda exista al menos un conjunto de candidatos. La última condición, y no por ello menos importante, es que toda búsqueda termine en un tiempo finito.

Para cumplir estas tres condiciones es necesario considerar la localización de los recursos en el Grid. La localización de los recursos en un sistema distribuido se refiere a unos criterios de localización, que al ser aplicados por un algoritmo le permiten diferenciar entre un recurso y otro, en base a la distancia de red que separa al nodo que ejecuta el algoritmo de un recurso particular. Una forma de garantizar que cada búsqueda disponga de un conjunto de candidatos consiste en asegurar que los recursos más cercanos al punto de búsqueda sean considerados siempre, en cualquier búsqueda que se haga. Para conseguir esto último, se definen dos localizaciones para los recursos con respecto al punto de búsqueda: local y remoto. Toda búsqueda incluirá los recursos locales.

Luego, el hecho de que todas las búsquedas exploren el espacio de los recursos locales puede ser aprovechado para asegurar que la búsqueda termine en un tiempo finito. Esto se puede conseguir si en lugar de explorar el espacio de los recursos remotos, utilizamos un método que permita acercar la información de los recursos remotos, de forma asíncrona y eficiente, al servicio que se ocupa de realizar la búsqueda (en este caso el servicio que ejecuta el algoritmo de asignación). De esta forma, un servicio que realice una búsqueda en su entorno local encontrará, además de la información de los recursos locales, cierta información de los recursos remotos. Como el valor de esta información dependerá del momento en que fue producida, el mecanismo de propagación que la transporta desde el recurso que la produce hasta las cercanías del punto de búsqueda tiene que cumplir dos condiciones. En primer lugar tiene que ser lo suficientemente rápido y seguro como para asegurar que ninguno de los sitios del Grid quede excluido siempre de la búsqueda. Para entender esta condición, imaginemos un sitio Grid que proporciona información de sus recursos, pero ésta nunca llega al servicio que ejecuta el algoritmo de asignación porque caduca antes de que pueda ser utilizada. La segunda condición consiste en que el mecanismo utilizado para propagar la información no deteriore las operaciones del Grid.

4.6. MONITORIZACIÓN DE RECURSOS EN SISTEMAS DE COMPUTACIÓN GRID

La sección anterior plantea la necesidad de propagar los indicadores dinámicos a todos los sitios del Grid, utilizando para ello un método rápido y seguro, a la vez que ligero y eficiente. La monitorización de sistemas distribuidos es un problema muy difícil de resolver, y además se sabe que es imposible conocer el estado, en un instante de tiempo dado, de todas las entidades que forman parte de un sistema distribuido dinámico y grande, como el Grid. De aquí que esta sección tiene como objetivo describir el trabajo realizado con el objetivo de modelar un sistema de monitorización para el Grid que cumpla con las condiciones expuestas en la sección anterior.

Existen varios modelos para la difusión de información en el Grid. Los más extendidos son los modelos centralizados o jerárquicos, y los modelos de redes peer-to-peer (P2P). De ellos, los sistemas centralizados o jerárquicos han sido los más utilizados. Por

ejemplo, el sistema de monitorización y descubrimiento (MDS) [17] es el componente principal del sistema de información de Globus Toolkit 4, uno de los middleware Grid más extendidos en la actualidad. Tanto la implementación pre-WS de MDS, como la implementación WSRF (WS-MDS), utilizan una arquitectura jerárquica para difundir la información a los diferentes sitios del Grid. También existen numerosos estudios que han demostrado las ventajas de las redes P2P para el Grid, fundamentalmente para el descubrimiento de servicios en despliegues de gran escala [64].

Sin embargo, el modelo de difusión de información que mejor se adapta a las necesidades de un sistema de monitorización Grid es el modelo de redes súper-peer. En este modelo, el súper-peer es un nodo que juega un papel especial porque actúa como recurso centralizado para un número limitado de nodos regulares. Los nodos regulares utilizan a los súper-peers para acceder a la información en el sistema distribuido. A su vez, los súper-peers se interconectan unos con otros para formar una red de alto nivel, en la que intercambian información.

La **Figura 4-11** muestra un esquema de un modelo de red súper-peer con redundancia. La redundancia fue introducida en [65] como una forma de mejorar la disponibilidad y la fiabilidad de estas redes. Una red súper-peer es redundante si hay varios súper-peers en el mismo ámbito local. Como todos los súper-peers en el mismo ámbito local comparten los mismos roles y responsabilidades, todos de ellos pueden responder, de forma equivalente, a las solicitudes de los nodos regulares.

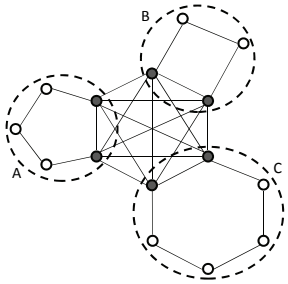


Figura 4-11: Esquema de una red súper-peer con redundancia. Los círculos sombreados representan súper-peers y los círculos claros representan nodos regulares. Las líneas discontinuas delimitan tres entornos locales: 'A', 'B' y 'C'. Un entorno local es un conjunto de nodos en un área geográfica común, normalmente definida por un área conveniente de latencia de red. Por ejemplo, T<2 milisegundos, para cualquier camino con hasta 2 saltos de red (network hops) que involucren dispositivos de red, como puentes de red, routers, switches, hubs, etc.

Mantener la redundancia en el modelo de redes súper-peer aumenta los costes en comunicaciones porque cada mensaje tiene que ser enviado desde el nodo regular hasta todos los súper-peers en el entorno local del nodo. Sin embargo, en el mismo trabajo [65] los autores demuestran que, en redes súper-peer con dos súper-peers por cada entorno local, la redundancia reduce la carga en cada súper-peer y mejora la fiabilidad y la disponibilidad de los mismos. Esta evidencia sugiere que el hecho de contar con

caminos redundantes entre los súper-peers en diferentes ámbitos puede contribuir a mejorar la fiabilidad de toda la red, especialmente cuando un número considerable de las comunicaciones que se producen en este contexto ocurren en redes físicas poco fiables, como por ejemplo redes WAN. Este hecho es relevante para poder construir un sistema de monitorización fiable para el Grid.

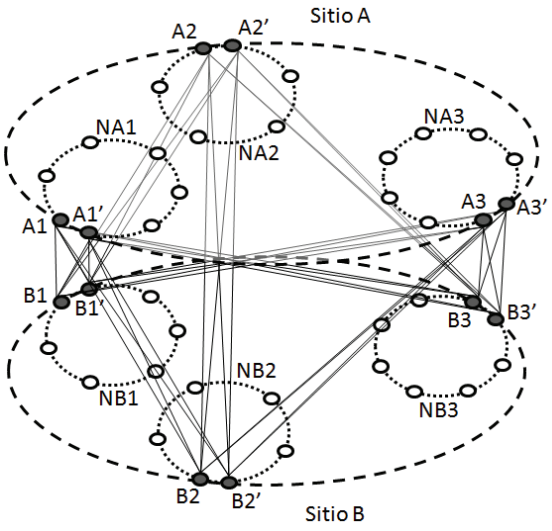


Figura 4-12: Red virtual organizada por encima del Grid para monitorizar el estado de los nodos. Las líneas de puntos representan los vecindarios locales (por ejemplo, NA1, NA2, y NB1), mientras que las líneas discontinuas representan los sitios regionales (Sitio A y Sitio B). Los círculos sombreados representan súper-peers (por ejemplo, A1, A1', B2 y B3'), y los círculos claros representan nodos regulares. Por conveniencia, los súper-peers del mismo vecindario local se representan agrupados, pero en la práctica pueden estar en cualquier posición en el anillo formado por los nodos locales.

La **Figura 4-12** muestra una topología de red virtual, organizada por encima del Grid, que tiene como objetivo difundir información hacia grupos virtuales de nodos. Esta red se organiza a nivel de aplicación, de lo que se deriva su carácter virtual. Los nodos que participan adquieren sus roles a través de directivas de configuración, y los grupos que se utilizan para difundir la información se crean en el momento de crear la red. Para que la comunicación a los diferentes grupos dentro de esta topología sea efectiva, los nodos del Grid han sido organizados en tres niveles diferentes: vecindarios locales, sitios regionales y dominios globales. Los vecindarios locales son los grupos más internos en la topología, representados como pequeños anillos en la **Figura 4-12**.

La redundancia de esta red es 2, porque este es el número de súper-peers equivalentes por cada vecindario. Los anillos de los vecindarios se unen a anillos mayores, que están interconectados entre sí por medio de una malla totalmente conexa que tiene sus vértices en los súper-peers. Los anillos se forman entre nodos del mismo segmento de

red. En cambio, la malla se forma entre nodos que pueden estar en segmentos de red diferentes. Los anillos menores se diferencian de los mayores porque sus nodos, además del segmento de red, comparten también el dominio de difusión (broadcast). Además, la comunicación entre los miembros del mismo anillo ocurre con mejores prestaciones en los anillos menores que en los mayores, porque la latencia en los anillos menores es menor, la fiabilidad es mayor y el coste por comunicación también es menor. Los anillos mayores están formados exclusivamente por súper-peers.

Cada una de las estructuras descritas juega un papel específico dentro del modelo. Como dijimos antes, los anillos menores representan el nivel de vecindario y hacen posible la comunicación entre los nodos regulares y los súper-peers del mismo vecindario local. Los anillos mayores representan el nivel de sitios regionales, y permiten que un súper-peer se comunique con todos los súper-peers que forman parte de su sitio, pero no de su vecindario. Por último, la malla representa el nivel de dominios globales y hacen posible la comunicación entre un súper-peer y todos los súper-peers que no forman parte de su sitio. Las comunicaciones que ocurren a nivel de dominios globales ocurren en la capa física que conecta dos o más segmentos de red diferentes. En consecuencia, un número considerable de las comunicaciones a estos grupos cruzará, al menos, un salto WAN. Este hecho hace que el coste de las comunicaciones a los dominios regionales sea muy elevado, y la fiabilidad incierta. La redundancia de la malla permite que los súper-peers envíen el mismo mensaje utilizando varios canales al mismo tiempo, mejorando la fiabilidad de la trasmisión de mensajes en estos grupos.

4.7. ORGANIZACIÓN DE LA INFORMACIÓN DE MONITORIZACIÓN

La sección anterior describe una red virtual diseñada para organizar la comunicación a grupos en el Grid. Esta red tiene una topología de red P2P estructurada, donde los súper-peers son nodos que operan a la vez como servidores para un conjunto de nodos y entre iguales (P2P) en una red de súper-peers. Las redes P2P se han utilizado fundamentalmente para almacenar, buscar y distribuir archivos entre un número grande de usuarios. Para cumplir este propósito, las redes P2P almacenan un índice distribuido entre todos los participantes de la red, de forma tal que ninguno de los participantes almacena una copia completa de la información compartida en la red. Para esto, un número considerable de los sistemas P2P que existen en la actualidad utilizan una tabla hash distribuida (DHT).

La DHT es una metodología para almacenar tablas hash en una red de nodos distribuidos, que proporciona un mecanismo de búsqueda tolerante a fallos. Aunque existen varias implementaciones diferentes, todas tienen en común que proporcionan un servicio de búsqueda semejante al de una tabla hash. Es decir, proporcionan las operaciones que permiten almacenar pares, donde el primer elemento del par es una clave que permite buscar y recuperar el segundo elemento. La característica principal que distingue a la DHT de una tabla hash normal, es que las operaciones de búsqueda y recuperación de la DHT permiten que cualquier participante de la red pueda acceder, de forma eficiente, al valor asociado con una clave particular. De esta forma, uno de

los patrones de uso más frecuentes consiste en utilizar una API de almacenamiento sobre la capa de búsqueda de la DHT para proporcionar una interfaz sobre la cual se construye una aplicación para compartir archivos, o para hacer copias de seguridad distribuidas.

En la DHT, la responsabilidad de gestionar la información que vincula a las claves con los valores se distribuye entre todos los nodos, de forma tal que cualquier cambio en la composición de la red de participantes causaría un daño mínimo. Esta característica de la DHT ha permitido desarrollar sistemas distribuidos de gran dimensión, que utilizan Internet como medio de comunicación, y que manejan los cambios dinámicos que se producen en la red, como consecuencia de la llegada de nuevos nodos, la salida de otros y los fallos.

Cada implementación de la DHT utiliza un dominio de claves, por ejemplo, el conjunto de cadenas de caracteres de 160-bits, y un esquema de distribución que permite dividir el dominio entre los nodos participantes. Una red de superposición (overlay network) conecta a los nodos entre sí, y proporciona las operaciones de búsqueda que permiten encontrar el nodo al que le ha sido asignado una clave particular del dominio. Para esto, cada nodo mantiene una tabla de enrutamiento, que consiste en un conjunto de referencias a un grupo determinado de nodos de la red, denominados vecinos. La forma en que los nodos seleccionan a sus vecinos depende de la topología de la red de superposición, que consiste en un conjunto de reglas definidas por la implementación de la DHT. Precisamente, la topología de la red de superposición es uno de los factores más importantes para conseguir la escalabilidad y la tolerancia a fallos. De la topología depende que las rutas sean cortas, lo cual es necesario para conseguir buenos tiempos de respuesta, y que las tablas de enrutamiento también sean pequeñas, lo cual es necesario para que el sobrecoste por el mantenimiento de las tablas sea pequeño, y el impacto de las llegadas, salidas y fallos sea mínimo. Generalmente, se utilizan topologías donde cada nodo tiene que coordinar con un número de vecinos del orden $O(\log n)$, siendo n el número total de participantes de la red.

Por otro lado, el objetivo del sistema de monitorización es que todos los participantes de la red mantengan una copia local del estado del Grid, que pueda ser utilizada en la asignación de recursos. Para esto, la red P2P almacena un índice replicado en todos los participantes. Esta estrategia puede ser descrita como una tabla hash replicada (RHT) porque cada nodo en la red P2P almacena una copia completa de los datos, que se va actualizando con la información que intercambian los nodos en la red. La ventaja principal de la RHT es que permite la replicación de la información en cada nodo participante. De esta forma, el sistema de monitorización transporta los indicadores dinámicos desde los sitios hasta los servicios que hacen la asignación en cada VO.

Como en el caso de la DHT, la topología de la red de superposición es fundamental para conseguir escalabilidad y tolerancia a fallos. En este caso, la separación de roles de la red súper-peer hace posible la separación de funciones en los nodos. Los nodos regulares (o simplemente peers) se encargan de monitorizar a los recursos Grid y de almacenar esta información. Por tanto, la tabla hash de la RHT es almacenada en los nodos regulares. Estos nodos utilizan una tabla de enrutamiento muy simple, que contiene referencias a los nodos del vecindario local, mientras que los súper-peers

almacenan una estructura de datos diferente que les sirve para encaminar la información en toda la red. Una de las ventajas principales de esta separación consiste en que la tabla de enrutamiento de los peers tiene un sobrecoste de mantenimiento muy bajo. Por otra parte, las tablas de enrutamiento de los súper-peers también son pequeñas, y las rutas que contienen son muy cortas porque enlazan a cada súper-peer con los peers del vecindario local, y con el resto de los súper-peers en el sitio y en la malla (**Figura 4-12**). En este sentido, la Sección 5.2.3 aporta más detalles sobre la topología, en relación con la implementación específica de la RHT utilizando Spread, donde las tablas de enrutamiento contienen referencias a grupos, en lugar de a nodos específicos.

La **Figura 4-13** muestra un esquema de la RHT implementada en dos tablas hash lineales extendidas (la Sección A.1.3 del Anexo 1, describe esta estructura de datos). La primera tabla almacena información que describe a los recursos, como la dirección de Internet del servicio que sirve de "front-end" al recurso, el número de CPUs físicos y lógicos (núcleos) del recurso y el total de memoria física y swap instalada en los nodos del mismo. La clave de cada tabla se forma combinando la dirección de Internet del peer que monitoriza al recurso con la dirección de Internet del servicio. Además, la información se indexa por la dirección de Internet del peer que monitoriza al recurso y por el sitio Grid donde se encuentra el mismo físicamente. Estas claves secundarias se almacenan de forma tal que los duplicados son organizados lexicográficamente, es decir, en el primer índice, todos los recursos monitorizados por el mismo peer se almacenan de forma consecutiva, y en el segundo índice, todos los recursos que pertenecen al mismo sitio se almacenan de forma consecutiva. Esta estrategia de organización acelera considerablemente el acceso a los datos. La segunda tabla almacena los indicadores dinámicos (en tanto por ciento) de los recursos, como la carga de CPU y la cantidad de memoria disponible en el mismo. Además, esta información se indexa por la dirección de Internet del peer que monitoriza al recurso y por la fecha y hora de la última modificación (LMT) del registro en la tabla.

Uno de los principales beneficios que aporta esta estructura de datos al sistema de monitorización es que permite separar la información por el tipo de acceso que esta tiene. La primera tabla almacena, principalmente, información de solo lectura, porque es esperable que una vez que han sido almacenadas las características de un recurso, esta información permanezca igual en el tiempo. En cambio, la segunda tabla almacena información que cambia con frecuencia en el tiempo. Ahora bien, es esperable que, debido al nivel de conexión que existe entre la carga de la CPU y la utilización de la memoria en un sistema, los cambios que se produzcan afecten toda la información perteneciente a un recurso. Esto facilita la actualización de la información, porque en lugar de comparar cada campo se sustituyen los paquetes completos, que por otra parte tienen el mismo tamaño para todos los recursos porque la información se almacena en tanto por ciento. Adicionalmente, esta forma de almacenar los indicadores dinámicos presenta ventajas a la hora de organizar la comunicación a grupos, como se verá en el capítulo siguiente. Además de producir paquetes de información más pequeños que los que se producirían de utilizarse los valores reales, como el tamaño de cada paquete es conocido de antemano antes de enviar o recibir información por la red, es posible establecer patrones de entrega y recepción muy eficientes.

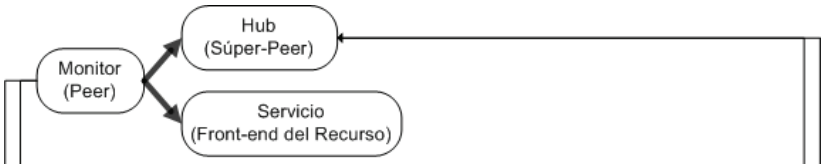


Tabla Hash Replicada (RHT)

Información Estática

Clave	Clave Secundaria ⟠		Valor			
Endpoint Servicio	Dirección IP Peer	Sitio	CPUs Físicas	CPUs Lógicas	Memoria Total (kB)	Swap Total (kB)
uri1/s1	10.0.0.2	1	1	1	516300	2031608
uri1/s2	10.0.0.2	1	1	1	516300	2031608
uri2/s3	10.0.0.3	1	1	2	1017668	2008084

Información Dinámica

Clave	Clave Secundaria ⟠		Valor		
Endpoint Servicio	Dirección IP Peer	Última Modificación	Carga CPU (promedios 1, 5 y 15 min)	Memoria Disponible	Swap Disponible
uri1/s1	10.0.0.2	1262102762	46% 27% 15%	57%	100%
uri1/s2	10.0.0.2	1262104835	53% 29% 16%	61%	100%
uri2/s3	10.0.0.3	1262102777	47% 38% 11%	32%	98%

Figura 4-13: Tabla hash replicada (RHT) implementada en dos tablas hash lineales extendidas.

La **Figura 4-14** muestra un esquema de la información almacenada en los súper-peers. Cada vez que un súper-peer recibe un mensaje de un peer o de un súper-peer, extrae del mismo la dirección de Internet del peer o el súper-peer que envía el mensaje, así como el grupo hacia el cual va dirigido el mensaje y una marca de tiempo que incluye cada nodo en sus mensajes. Además, esta información es indexada por la última fecha y hora de modificación del campo en la tabla hash. Un súper-peer puede utilizar esta información para redirigir el mensaje hacia un grupo en la red o para descartarlo, interrumpiendo una cadena de difusión. Por ejemplo, un súper-peer puede recibir un mensaje enviado a su vecindario local desde un sitio remoto, pero el mismo mensaje puede haber llegado a uno de los súper-peers redundantes por un camino alternativo. En este caso, el primero de los súper-peers que repita el mensaje en el vecindario local inhabilitará el reenvío del mensaje en el otro súper-peer. Esto sucede así porque cada súper-peer puede comparar la información de los mensajes que le van llegando con la información que almacena para decidir si tiene que hacer reenvío o no. El objetivo de esto es reducir las comunicaciones que puedan llegar a repetirse debido al mecanismo de redundancia.

Con las estructuras de datos representadas en la **Figura 4-13** y **Figura 4-14**, el sistema de monitorización puede rastrear los cambios en el Grid y actualizar los indicadores dinámicos en los sitios. En consecuencia, los servicios encargados de la asignación de recursos pueden consultar la información almacenada en la RHT de los peers de su entorno más cercano, y con ello tener una vista continua del estado del sistema.

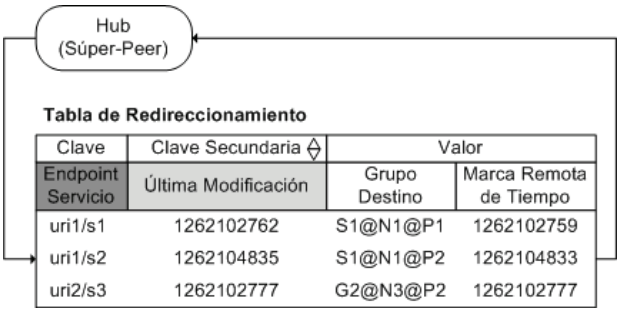


Figura 4-14: Información almacenada en los súper-peers en una tabla hash lineal extendida.

La utilización de la RHT es posible porque, normalmente, el número de recursos disponibles en el Grid está limitado (y por esa razón la cantidad de información compartida también es limitada) a una tarea específica, a un período de tiempo definido, o a los recursos de algunos proyectos o comunidades. Por ejemplo, una VO de entrenamiento puede existir solamente durante un tiempo y luego desaparecer. También es posible que una VO exista por tiempo indefinido, pero que solamente pueda utilizar los recursos de una parte del Grid. Este hecho contrasta con las redes P2P de intercambio de ficheros, como por ejemplo Gnutella, Freenet y Napster, donde el número de participantes es ilimitado.

Hasta aquí hemos descrito las bases donde se asienta GRIDIFF, un sistema de gestión de recursos en base a QoS, para el Grid. El próximo capítulo aborda los detalles de la implementación y puesta a punto de este sistema, y completa algunas de las ideas introducidas con los modelos descritos en este capítulo.

4.8. CONCLUSIONES DEL CAPÍTULO

Los indicadores de QoS cuantitativos caracterizan a los servicios Grid en cuanto a prestaciones y disponibilidad. Los mismos pueden ser utilizados para diferenciar a los servicios. Para esto, es necesario contar con algoritmos que funcionen en el Grid. Los algoritmos de análisis de clústeres son muy apropiados para clasificar conjuntos de servicios Grid, pero necesitan ser adaptados para entornos distribuidos. Una posible solución consiste en evaluar, de forma asíncrona, los indicadores de QoS y de carga del sistema, y propagarlos, de forma eficiente a los nodos que hacen, de forma independiente, la clasificación.

Capítulo 5

5. GRIDIFF: UNA ARQUITECTURA DE SOFTWARE PARA LA ASIGNACIÓN DE RECURSOS EN EL GRID EN BASE A REQUERIMIENTOS DE QOS A NIVEL DE SERVICIO

Este capítulo describe GRIDIFF, una arquitectura de software diseñada para dar soporte de QoS a las aplicaciones Grid. Además de la estructura global de la misma, se describen los pormenores de la implementación.

5.1. SISTEMA DE COMPONENTES Y SERVICIOS DE GRIDIFF

El nombre GRIDIFF tiene su origen en la combinación de las palabras Grid y diff. Ya que en informática, diff es una utilidad para la comparación de archivos, este nombre resalta la idea de que la diferenciación es un componente esencial de la arquitectura, pero en lugar de archivos, se comparan servicios Grid.

GRIDIFF se basa en un sistema de componentes y servicios que se despliegan sobre una infraestructura Grid. La mayoría de ellos se implementan en forma de servicios Grid. La excepción es el sistema de monitorización, que se despliega directamente en los sitios, sin importar qué middleware Grid está instalado en estos. La **Figura 5-15** muestra un gráfico con la arquitectura de componentes y servicios de GRIDIFF, y además muestra las relaciones que se establecen entre ellos.

La elipse grande en la izquierda de la **Figura 5-15** muestra al Sistema de Gestión de Servicios (SGS). Los servicios Grid se pueden suscribir a GRIDIFF a través del Proveedor de Servicio del SGS. Una vez que un servicio Grid se suscribe al SGS, este propaga la suscripción al Registro de Servicios del Repositorio Activo de Información de Servicios y Configuración (RAISC), que a su vez propaga la suscripción a todos los sitios de la VO. Además, el Proveedor de Servicio del SGS se encarga de dar respuesta a las solicitudes y de asignar los recursos para la ejecución de trabajos en el Grid.

El RAISC es, básicamente, un repositorio de información que almacena registros históricos de los indicadores estáticos de los servicios Grid. Adicionalmente, el RAISC almacena también las políticas de la VO, como las políticas de fijación de precios y los requerimientos de QoS de las aplicaciones soportadas por la VO. La naturaleza activa del RAISC viene dada porque, además de proporcionar una interfaz que permite realizar búsquedas en la información que almacena, también envía mensajes a los SGS de su sitio y a otros RAISC de todo el Grid para alertar de los cambios que modifican el estado de un servicio Grid. Por ejemplo, cuando un servicio gestionado por GRIDIFF se ve afectado por un cambio en los recursos que lo soportan, y de este cambio resulta que sus indicadores estáticos varían sus valores considerablemente, pasando de un estado en el cual superaban los valores umbrales para los

requerimientos de QoS a un estado en el cual no superan uno o varios requerimientos, entonces el RAISC envía una notificación que se propaga al resto de los RAISC de la VO, para alertar del nuevo estado en que se encuentra el servicio. Esta relación puede interpretarse como un proceso de retroalimentación que se produce entre el Registro de Servicios del RAISC y el Proveedor de Servicios del SGS (en la **Figura 5-15**, una línea discontinua terminada en flecha, en sentido contrario al registro): un SGS registra un servicio con el RAISC, pero luego este último descubre, por una vía que no incluye al SGS, que el estado del servicio ha cambiado, y entonces notifica el cambio al SGS.

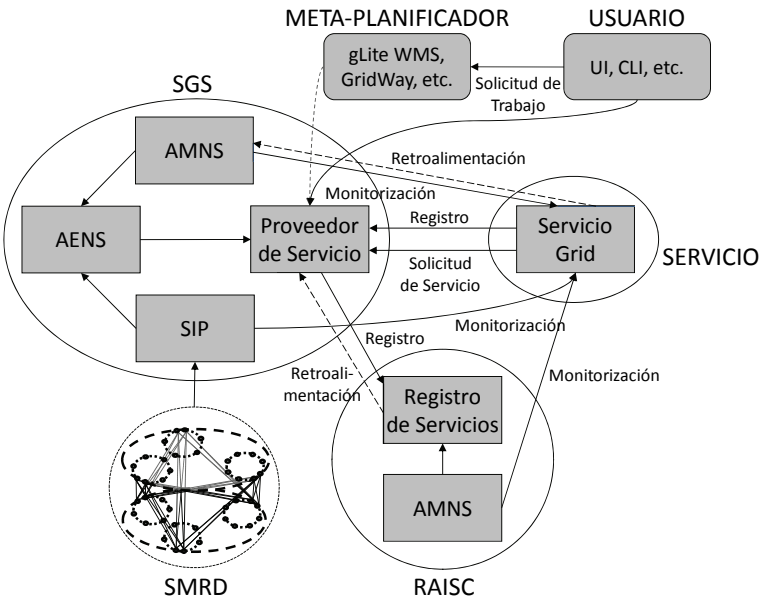


Figura 5-15: Arquitectura de componentes y servicios de GRIDIFF.

El Sistema de Información de Proveedores (SIP) es utilizado por el SGS para mantener un registro del estado de disponibilidad de los recursos computacionales y la carga de los elementos de cómputo (clústeres de ordenadores, nodos, etc.) de los proveedores de servicio. El SIP del SGS gestiona una pequeña base de datos que contiene información del estado global de los recursos de la VO en un instante de tiempo determinado, específicamente el SIP guarda los últimos valores de los indicadores estáticos y dinámicos. Para conseguirlo, el SIP obtiene información del Sistema de Monitorización de Recursos Distribuido (SMRD). Este sistema implementa el modelo de difusión de información descrito en la Sección 4.6, para distribuir los indicadores dinámicos a todos los sitios de la VO. El SMRD es el único componente de la arquitectura que no utiliza componentes del Grid. Por esta razón, la comunicación

entre el SIP y el SMRD ocurre mediante canales convencionales que no están protegidos por la infraestructura de seguridad Grid, como se verá en la Sección 5.2.9.

El Agente para la Monitorización del Nivel de Servicio (AMNS) es un componente, utilizado por el SGS y por el RAISC, que permite ejecutar pruebas en los servicios Grid y recuperar los indicadores de prestaciones y disponibilidad de los servicios. Adicionalmente, el AMNS puede recibir notificaciones de los servicios Grid. Por ejemplo, un servicio Grid que implementa un *checkpoint* para evaluar su estado como parte de la ejecución de un trabajo, puede utilizar las funciones que proporciona la API de GRIDIFF para enviar una notificación al AMNS avisando de un cambio en el estado de alguno de sus indicadores estáticos. Al igual que se producía entre el Proveedor de Servicio del SGS y el Registro de Servicios del RAISC, éste es un proceso de retroalimentación que se produce entre el Servicio Grid y el AMNS: un AMNS monitoriza el estado de un Servicio Grid, pero luego este último descubre que su estado ha cambiado, y entonces notifica al AMNS.

Por último, el Agente para la Evaluación del Nivel de Servicio (AENS) es un componente que forma parte del SGS, y permite evaluar los indicadores estáticos y dinámicos. Mientras que el resto de los componentes se ocupan de recoger y transmitir los indicadores, el AENS los evalúa, ya sea por separado o combinados en complejas reglas, utilizando para ello las políticas de la VO. Por ejemplo, la API de GRIDIFF proporciona métodos para que las aplicaciones cliente envíen solicitudes de trabajo al Grid. Estas solicitudes pueden ir acompañadas de varios documentos XML que especifican los requerimientos de nivel de servicios y de disponibilidad de recursos. De esta forma, la API de GRIDIFF permite especificar valores de disponibilidad, plazos de ejecución, mecanismos de seguridad, etc., que deben cumplir los servicios Grid asignados para la ejecución de un trabajo. El AENS puede combinar todos estos elementos para seleccionar un conjunto de servicios que cumplan con todos los requerimientos, por ejemplo, para encontrar en la VO un servicio que pueda ejecutar un trabajo crítico que requiera un cierto nivel de seguridad, como el procesamiento de una imagen médica de un paciente, o el análisis de datos de una empresa. Luego, los servicios seleccionados pueden ser utilizados como parte del proceso de meta-planificación. Para esto, GRIDIFF contempla dos opciones: la primera, y a la vez más evidente, consiste en enviar la solicitud de trabajo al meta-planificador Grid, acompañada de una lista blanca con los servicios que han sido seleccionados por GRIDIFF; la segunda, y más compleja, consiste en incorporar GRIDIFF al planificador Grid. Esto último es posible porque algunos middleware Grid, como gLite, permiten extender sus funcionalidades con *plugins* que pueden ser añadidos a los planificadores.

Es necesario señalar que el middleware Grid permite especificar varios aspectos de la ejecución de un trabajo. Para ello, existen especificaciones estándares, como el Job Submission Description Language (JSDL). Con GRIDIFF no pretendemos reemplazar estos mecanismos, sino extenderlos con información de QoS a nivel de servicio. En este sentido, la Sección 3.2 describe los esfuerzos hechos en el marco del OGF para estandarizar WS-Agreements, una especificación de servicios Web para la gestión de SLA. Sin embargo, al día de hoy el desarrollo de este estándar continúa, y no tenemos conocimiento de que exista un middleware Grid que proporcione una implementación de WS-Agreements. Aunque sí existen varios proyectos que abordan las tareas

relacionadas con la especificación, negociación y renegociación de SLAs, como parte de la estandarización de WS-Agreements (la Sección 3.2 presenta un listado de proyectos relacionados con la gestión de QoS y SLA en el Grid). Por estas razones, GRIDIFF utiliza una implementación propia, que si bien no es compatible con WS-Agreements, utiliza la base conceptual que proporciona la OGSA para la gestión de QoS. Esta capa de componentes se ha diseñado poniendo un cuidado especial en la interoperabilidad y la compatibilidad, previendo que en el futuro, cuando se disponga de una implementación middleware de WS-Agreements, se pueda reutilizar la mayor parte del código para migrar la arquitectura.

En GRIDIFF los indicadores dinámicos se propagan por el Grid utilizando la red súper-peer del sistema de monitorización, y los indicadores estáticos se propagan utilizando el sistema de información del Grid. Por ejemplo, en una implementación basada en GT4, el AMNS colecta los indicadores estáticos de los servicios Grid y los suscribe al WS-MDS de Globus. Luego, cualquier AENS de la VO puede interrogar al WS-MDS para recuperar esta información. Normalmente, estos indicadores pueden ser actualizados en intervalos de tiempo grandes (entre 30 minutos y 1 hora), lo cual hace posible la utilización de los sistemas de información del Grid sin crear sobrecarga. De cualquier forma, los cambios importantes se notifican entre servicios RAISC, como fue explicado anteriormente en esta sección.

5.2. SISTEMA DE MONITORIZACIÓN DE GRIDIFF

Comenzaremos describiendo el Sistema de Monitorización de Recursos Distribuido (SMRD) de GRIDIFF. La **Figura 5-16** muestra un gráfico con los componentes de este sistema.

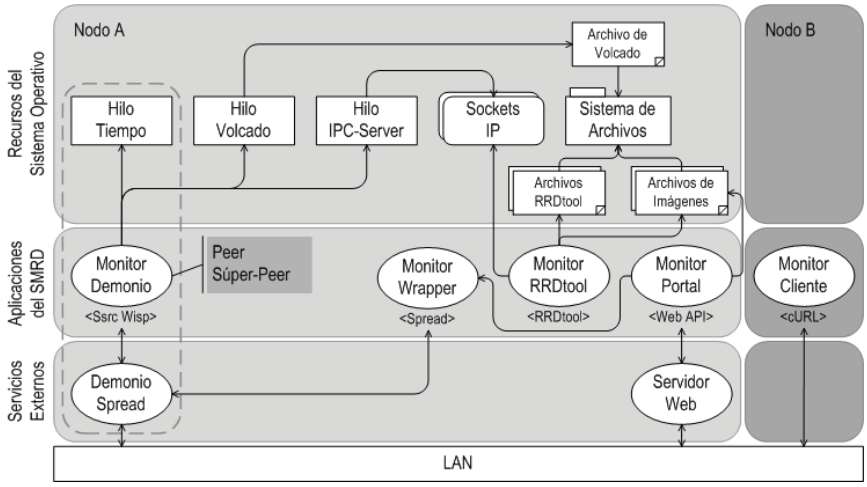


Figura 5-16: Componentes del Sistema de Monitorización de Recursos Distribuido (SMRD) de GRIDIFF.

En el gráfico de la **Figura 5-16** han sido representados dos nodos que hacen uso de dos sistemas de componentes diferentes. El Nodo A hace uso del sistema de componentes P2P, mientras que el Nodo B utiliza componentes Web. Las divisiones horizontales representan diferentes niveles de abstracción en los nodos. La más cercana a la red representa aplicaciones de red que funcionan como servicios. En este nivel están el Servidor Web y el demonio de Spread. Spread es un proyecto que proporciona un conjunto de herramientas de código abierto, que permiten articular un servicio de mensajería de altas prestaciones, en redes LAN y WAN, con una tolerancia a fallos muy alta [66]. La capa intermedia representa las aplicaciones que forman parte del SMRD, estas son:

- Monitor Demonio: Implementa el núcleo del sistema de monitorización. Entre otras cosas, proporciona los servicios que actúan como súper-peers y peer regulares en la red P2P. Esta aplicación se suspende en segundo plano, esperando a que lleguen datos al socket de red que escucha el demonio de Spread. Adicionalmente, cuando actúa como peer regular, lanza un planificador en un hilo auxiliar, que ejecuta acciones cada cierto tiempo. Entre estas acciones se encuentra monitorizar el estado actual de los recursos computacionales y enviar un mensaje, con esta información, a los nodos del entorno local. Para monitorizar los recursos, puede utilizar llamadas del sistema operativo, o en caso de que el nodo tenga acceso a un planificador, como Condor, LSF o PBS, también puede consultar la información del mismo.
- Monitor Wrapper: Esta aplicación forma parte de una vía alternativa de comunicación con el sistema de monitorización. La red P2P es la principal vía de intercambio de información del sistema de monitorización, donde se transporta el grueso de la información del SMRD. Sin embargo, no todos los recursos del Grid pueden utilizar esta vía de comunicación. Por ejemplo, algunos middleware Grid solamente pueden ser instalados en sistemas operativos muy específicos, que muchas veces tienen soporte limitado para aplicaciones y bibliotecas que son necesarias para instalar el Monitor Demonio. Por otra parte, muchos sitios tienen requerimientos de seguridad muy estrictos que les impiden abrir puertos de comunicación adicionales, más allá de los puertos estándares. Con el objetivo de monitorizar a estos sistemas, el SMRD implementa una segunda vía de comunicación que utiliza los protocolos HTTP y HTTPS sobre TCP/IP para comunicarse con el Monitor Wrapper, que se encarga de retransmitir los mensajes en la red P2P.
- Monitor RRDtool: Esta aplicación se comunica con el Monitor Demonio, a través de un socket IP, para interrogar regularmente a la RHT y actualizar una base de datos round-robin. Para esto, la aplicación utiliza RRDtool, una herramienta de código abierto que permite registrar y graficar series temporales de datos, de forma tal que el espacio necesario para almacenar en disco las series de datos es constante en el tiempo [67]. Monitor RRDtool permite crear tres series temporales para cualquier recurso monitorizado en el Grid: la primera contiene las mediciones de la última hora, la segunda las mediciones de la última semana, y por último, la tercera contiene las

mediciones históricas del último mes. Adicionalmente, Monitor RRDtool puede crear gráficos de cualquiera de las series temporales. Monitor RRDtool puede ser configurado para ejecutarse en segundo plano, actualizando las series temporales en intervalos regulares de tiempo.

- Monitor Portal: Junto con Monitor Wrapper y Monitor Cliente, forma la vía alternativa de comunicación con el sistema de monitorización. Monitor Portal publica un formulario en el Servidor Web, que puede ser utilizado por Monitor Cliente para enviar información a los peers de la red P2P (Monitor Demonio). Al recibir este formulario, Monitor Portal ejecuta Monitor Wrapper para repetir la información en la red P2P. Adicionalmente, Monitor Portal proporciona un conjunto de aplicaciones Web que permiten buscar y visualizar los gráficos de las series temporales mantenidas por Monitor RRDtool.
- Monitor Cliente: Esta aplicación se ejecuta en los nodos *legacy* para monitorizar los recursos computacionales locales y la carga del sistema, y enviar esta información a los peers de la red P2P (Monitor Demonio), a través de un camino que pasa por Monitor Portal y Monitor Wrapper. Se caracteriza por su simpleza, ya que utiliza llamadas al sistema operativo para obtener el estado de los recursos, y solamente tiene una dependencia externa: la biblioteca cURL [68]. Esta biblioteca software es una herramienta de código abierto que permite transferir archivos, de forma programática, utilizando protocolos de red estándares, como HTTP y HTTPS. Esta herramienta se distribuye prácticamente con todos los sistemas operativos UNIX y tipo UNIX. Monitor Cliente puede ejecutarse en segundo plano, monitorizando el estado del sistema y enviando esta información al Monitor Portal.

La última capa representa los recursos del sistema operativo. Forman parte de esta capa, los hilos auxiliares de Monitor Demonio, los sockets IP que utiliza Monitor Demonio para comunicarse con aplicaciones externas, por ejemplo Monitor RRDtool y el SIP del SGS de GRIDIFF, los archivos con las series temporales y sus imágenes, y un archivo de volcado de la RHT.

Los hilos auxiliares de Monitor Demonio son:

- Hilo Tiempo: Ejecuta la función del planificador de tiempo de Monitor Demonio, que sincroniza las tareas del mismo.
- Hilo Volcado: Ejecuta una función de volcado que crea o actualiza un archivo que contiene una copia de la RHT. Este archivo proporciona información sobre el funcionamiento del sistema de monitorización, que puede ser utilizada para depurar el código de las aplicaciones y para evaluar las prestaciones de las mismas. Por esta razón, la función de volcado utiliza llamadas asíncronas al sistema operativo, con el objetivo de interrumpir, lo menos posible, la ejecución de los procesos principales de Monitor Demonio. Adicionalmente, el Hilo Volcado puede ser utilizado para almacenar la información en un formato específico, que pueda ser interpretado

directamente por una herramienta externa, como por ejemplo un meta-planificador.

- Hilo IPC-Server: Ejecuta una función que proporciona un servidor de comunicación entre procesos (Inter-process communication o IPC). Este servidor proporciona una interfaz simple que permite interrogar la RHT y obtener los resultados de la búsqueda a través de un socket IP. Los métodos de búsqueda que proporciona el servidor son paginados, por lo que el servidor tiene un límite de conexiones para las cuales puede mantener una cache con las búsquedas. El tamaño de esta cache permite almacenar la información de un sitio, por lo cual si una búsqueda incluye varios sitios, el servidor ejecutará más de una búsqueda en la RHT.

Los únicos componentes obligatorios del SMRD, ya sean externos o proporcionados por GRIDIFF, son: el Demonio de Spread, el Monitor Demonio y el Hilo Tiempo. En la **Figura 5-16**, estos componentes han sido enmarcados en un recuadro con líneas de contorno discontinuas. El resto son opcionales, y desplegarlos o no depende de las funcionalidades del SMRD que se quiera utilizar. Por ejemplo, el Monitor Wrapper, el Monitor Portal y el Monitor Cliente, no son componentes obligatorios del SMRD, y pueden no ser considerados en el despliegue.

5.2.1. COMUNICACIONES DENTRO DEL SISTEMA DE MONITORIZACIÓN

La parte más difícil de construir de un sistema de monitorización en un entorno distribuido es la capa de comunicación. La **Figura 5-17** muestra un gráfico que relaciona a los participantes de una red P2P con los canales utilizados para difundir los mensajes a los diferentes grupos de distribución en la red, de acuerdo a la estructura mostrada en la **Figura 4-12**.

Los súper-peers se conectan a un demonio de Spread, que se conecta directamente a la red. En cambio, los peer regulares se pueden conectar de diferentes formas. La más recomendable consiste en desplegar un demonio de Spread con cada peer, y combinar en el mismo nodo, siempre que sea posible, un peer con un súper-peer. Otra forma consiste en conectar varios peer regulares a un único demonio Spread a través de la red. De esta forma, disminuye el número de demonios de Spread conectados a la red pero aumentan las comunicaciones. Por último, otra forma consiste en desplegar un peer regular con un demonio de Spread, sin incluir súper-peer, y utilizar un súper-peer de otro nodo del vecindario.

Los peer regulares y los súper-peers reconocen dos entornos diferentes en los que ocurren las comunicaciones: local y remoto. En el entorno local ocurren las comunicaciones con los nodos que forman el vecindario, mientras que en el entorno remoto ocurren las comunicaciones que involucran a los nodos que no forman parte del vecindario. Adicionalmente, los súper-peers distinguen entre los nodos que están en el sitio y los que están en el resto del Grid. Finalmente, como vimos en la **Figura 4-12** del capítulo anterior, los peer regulares se conectan solamente a su vecindario. Por lo tanto, un peer regular no puede enviar mensajes directamente al entorno remoto. En

su lugar, lo que hace es enviarlos en su vecindario, y los súper-peers se encargan de repetir los mensajes hacia los nodos remotos. A continuación, veremos cómo esta característica del protocolo de comunicación aumenta la eficiencia del sistema de monitorización.

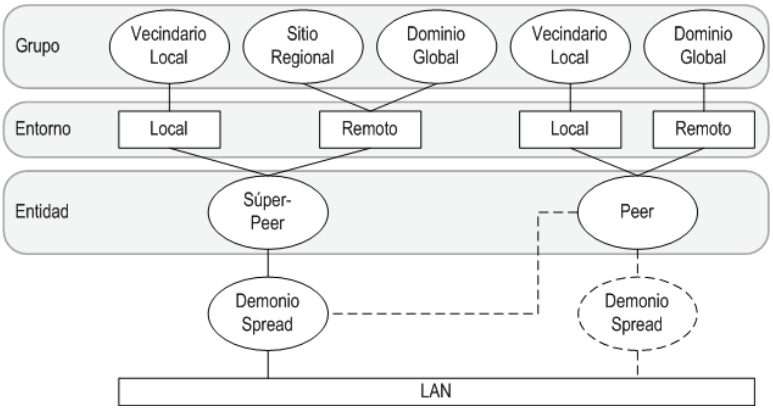


Figura 5-17: Los participantes de la red P2P con los canales utilizados para difundir mensajes a los diferentes grupos de distribución en la red.

En una red, cada peer regular actúa como una sonda que proporciona métodos para reunir información de un conjunto de nodos, que están localizados en un área geográfica relativamente pequeña, y conectados por una red común. Esta forma de organización responde a las necesidades del Grid, donde los proveedores de servicio agrupan recursos a nivel de centro y región. Agrupar recursos en un área geográfica definida facilita la operación de los mismos, a la vez que permite simplificar las tareas de mantenimiento. Por esta razón, la mayoría de los proveedores de servicio utiliza sistemas de monitorización de recursos a nivel de centro, y algunos a nivel de región. Los peer regulares implementan las operaciones necesarias para consultar a los sistemas de monitorización locales y reunir la información necesaria para realizar tareas de meta-planificación a nivel de Grid. Adicionalmente, los peer regulares implementan operaciones para reunir esta información de los sistemas de colas locales de los recursos, y directamente de los nodos de trabajo. Por último, los peer regulares también pueden recibir información que le es enviada desde un nodo remoto, utilizando un canal HTTP/HTTPS tradicional.

De esta forma, la única información que gestionan los peer regulares en la red P2P es la información de monitorización referente al nodo en el que están desplegados, y de otros nodos que pueden estar monitorizando. Cada peer envía los indicadores dinámicos de un conjunto de nodos, y recibe los indicadores dinámicos del resto de los peers en el Grid. Como en cualquier caso, un peer regular tiene que difundir esta información a todos los nodos de su vecindario, un súper-peer que se encuentre en el vecindario va a recibir la información como si fuese un peer más.

La mayoría de las redes LAN permiten hacer difusión a todos (broadcast) y difusión a grupos de multidifusión (multicast), dentro de límites definidos. Estas técnicas pueden mejorar drásticamente la eficiencia de la comunicación a grupos. A los dispositivos (nodos, dispositivos de red, etc.) a los que va dirigida la información se les conoce como destinos. La difusión a grupos de multidifusión consiste en enviar la misma información, simultáneamente, a un conjunto de destinos en una red LAN. Para ello, la información es encaminada en la red siguiendo una estrategia que consiste en enviar una vez los mensajes de multidifusión sobre cada enlace de la red, creando copias solamente en los puntos que conectan varios destinos con diferentes enlaces de red. En contraste, la difusión a todos consiste en enviar la misma información, simultáneamente, a todos los dispositivos de una red, aunque en la práctica las redes LAN implementan lo que se conoce como dominios de difusión o dominios de broadcast, que es una forma de agrupar a un conjunto de destinos que serán los que recibirán los mensajes de difusión a todos. De forma similar, la difusión a grupos de multidifusión se implementa a través de grupos de multidifusión, que es una forma de agrupar a los destinos. La forma más común de implementar los dominios de difusión y los grupos de multidifusión consiste en utilizar direcciones de Internet. Por ejemplo, la tecnología Ethernet y el protocolo IPv4 utilizan direcciones donde todos los bits son 1, como en 255.255.255.255, para indicar que un paquete va dirigido al dominio de difusión de la red. De esta forma, los dispositivos de la red que se encargan de encaminar los paquetes a diferentes niveles (a nivel de capa física, IP, etc.) pueden reconocer cuándo un paquete es de difusión. Estas dos tecnologías contrastan con la unidifusión (unicast), que consiste en enviar un paquete de un punto a otro en una red. En una comunicación a grupo basada en unidifusión, se envía una copia de cada mensaje a cada destino siguiendo una topología determinada, por ejemplo un anillo. En cambio, la difusión permite reducir el número de paquetes enviados, inundando la red con el mismo mensaje. La difusión a grupos de multidifusión es más eficiente que la difusión a todos porque inunda la red con menos mensajes.

Esta característica fue tomada en cuenta para el diseño del sistema de monitorización. Recordemos que los nodos que forman parte del mismo vecindario comparten un segmento de red física y un dominio de difusión (dominio de broadcast). Además, Spread ofrece la posibilidad de utilizar las dos técnicas de transmisión de información (difusión a todos y difusión a grupos de multidifusión), en dependencia de las características de la red. De esta forma, un peer regular envía los mensajes a los nodos de su vecindario utilizando la forma de transmisión más eficiente, que puede ser difusión a grupos de multidifusión o difusión a todos, en este orden. Solamente en el caso de que el nodo no pueda utilizar ninguna de estas técnicas (por ejemplo, si se encuentra formando parte de una red que no soporta ninguna de las dos técnicas), el peer enviará el mensaje a través del anillo de su vecindario.

En resumen, cada peer monitoriza, regularmente, los indicadores dinámicos de un conjunto de nodos, y envía un mensaje a todos los nodos de su vecindario, utilizando para ello el método más eficiente que soporte la red. Los demás peers van a utilizar el mensaje para actualizar la RHT con la información del remitente, y los súper-peers van a utilizar la información para difundirla al resto de la infraestructura. En primer lugar, los súper-peers envían la información al resto de los súper-peers del sitio, y luego la envían a los súper-peers en otros sitios de la infraestructura. Los súper-peers que

reciben la información desde un súper-peer que no forma parte de su vecindario, comprueban que ningún otro súper-peer del vecindario haya difundido la información en dicho ámbito y entonces la difunden, utilizando también la técnica más eficiente.

De esta forma, se consigue reducir el tráfico en los vecindarios. Teniendo en cuenta que un dominio de difusión puede tener, como mucho, 250 peer regulares (considerando un máximo de 250 nodos por subred, y por tanto un máximo de 250 nodos por dominio de difusión, que es el caso más general), en estas condiciones un sistema de monitorización con un refresco de 10 segundos estaría generando, como promedio, un tráfico en cada nodo de unos 25 mensajes por segundo. Este sería el máximo de tráfico que un vecindario podría llegar a generar. Ahora bien, esto sería en caso de utilizar un peer para monitorizar cada nodo del vecindario. En la práctica es más efectivo utilizar un peer para monitorizar varios nodos, un clúster de ordenadores o una parte de un clúster, siempre que las condiciones de la red y el software instalado en los clústeres lo permitan. En este caso, se reduce considerablemente el número de mensajes de difusión (a todos o a grupos de multidifusión) en el vecindario, a la vez que aumenta el tamaño de los mensajes, porque cada peer puede enviar la información de varios nodos a la vez, en el mismo mensaje. Las pruebas de validación de GRIDIFF demostraron que esta es la forma más eficiente de utilizar el SMRD.

5.2.2. RELACIÓN ENTRE LA RED FÍSICA Y LA RED DE SUPERPOSICIÓN

La **Tabla 5-1** muestra los resultados de un estudio realizado para explicar la relación que se establece entre la red física y la red de superposición (overlay network). Esta tabla muestra el tiempo promedio que tarda un nodo en recibir una respuesta a un paquete de eco ICMP, desde otro nodo en una red.

Tabla 5-1: Tiempo promedio, en milésimas de segundo, que tarda un nodo (*trencadisv04.itaca.upv.es*) en recibir respuesta a un paquete ICMP, desde otro nodo en la red.

Nombre del ordenador	IP	Salto 1 (ms)	Salto 2 (ms)	Salto 3 (ms)	Salto 4 (ms)
trencadisv01.itaca.upv.es	158.42.167.5	0,130	–	–	–
trencadisv02.itaca.upv.es	158.42.167.18	0,128	–	–	–
trencadisv03.itaca.upv.es	158.42.167.12	0,133	–	–	–
trencadisv04.itaca.upv.es	158.42.167.52	0,000	–	–	–
trencadisv05.itaca.upv.es	158.42.167.85	0,124	–	–	–
hekew.itaca.upv.es	158.42.167.67	0,188	–	–	–
loki.itaca.upv.es	158.42.167.1	0,233	–	–	–
ias.cc.upv.es	158.42.4.23	0,613	0,276	0,179	–
nova.alc.upv.es	158.42.130.109	2,322	2,591	9,342	7,755

Cada fila de la **Tabla 5-1** representa los saltos que realiza un paquete eco ICMP desde *trencadisv04.itaca.upv.es* hasta un nodo de la red. Dentro de las filas, cada columna representa el tiempo que tarda un paquete en alcanzar una puerta de enlace (gateway) o el ordenador de destino, y obtener un eco de respuesta. Los valores de la tabla han sido obtenidos con ayuda del comando *traceroute*. Esta herramienta está disponible en la mayoría de los sistemas UNIX y tipo UNIX, y permite enviar tres paquetes simultáneos, de 40 bytes cada uno, y medir el tiempo que tarda cada uno de ellos en llegar al ordenador de destino y producir una respuesta. Los valores mostrados en la **Tabla 5-1** son tiempos promedio de ida y vuelta de tres paquetes de eco ICMP generados con *traceroute*. Como regla general, la mitad de este tiempo da una idea aproximada del tiempo que tarda un paquete de información en llegar desde el nodo de origen hasta el nodo destino, en una red.

Todos los ordenadores *trencadisv* (*trencadisv01*, *trencadisv02*, *trencadisv03*, *trencadisv04*, *trencadisv05*), y los ordenadores *hekew* y *loki* se encuentran en el mismo segmento de red física, y comparten el mismo dominio de difusión. Además, los tiempos representados en la tabla muestran valores semejantes para los ordenadores *trencadisv*, y un poco mayores para *hekew* y *loki*. En cambio, el ordenador *ias* se encuentra en un segmento de red diferente, a dos saltos de red de *trencadisv04*. De forma semejante, el ordenador *nova* se encuentra en otro segmento de red, a tres saltos de *trencadisv04*.

Suponiendo que un proveedor de servicios tuviese recursos alojados en estos ordenadores, analicemos la forma de configurar los componentes del SMRD para desplegar un sistema de monitorización eficiente en esta red. Lo primero a tener en cuenta es que hay 3 segmentos de red diferentes, y por tanto serán necesarios, al menos, 3 vecindarios. Los ordenadores *ias* y *nova* forman, cada uno, su propio vecindario. Analicemos qué pasa con el resto de los ordenadores. Está claro que podemos poner a todos los ordenadores *trencadisv* en el mismo vecindario. Sin embargo, ¿qué hacer con *hekew* y *loki*? Si bien es cierto que el tiempo de comunicación de estos ordenadores a los ordenadores *trencadisv* es mayor que el tiempo que se consigue entre los ordenadores *trencadisv*, no es menos cierto que estas diferencias no son muy grandes. La decisión de formar un único vecindario que contenga a todos los ordenadores *trencadisv* junto con *hekew* y *loki*, o formar varios vecindarios, dependerá de las características de la red. De ser posible, la mejor opción consiste en formar un grupo de multidifusión (multicast) con estos ordenadores, y asociarlo con un único vecindario. La segunda opción consiste en utilizar el dominio de difusión (broadcast) con el mismo propósito. Sin embargo, en este caso hay que tener en cuenta si hay otros ordenadores formando parte del mismo dominio de difusión. En tal caso, la segunda opción no es viable, y entonces la única opción es utilizar comunicaciones de emisor a receptor (unicast) entre los miembros del mismo vecindario. Para esto, se puede utilizar un único vecindario o varios, y la elección dependerá de la configuración lógica de la red y de la experiencia acumulada en pruebas previas al despliegue del sistema de monitorización.

Por otra parte, el ordenador *ias* formará parte de un vecindario que comparte el mismo sitio regional (siguiendo la nomenclatura introducida en la Sección 4.6) con el vecindario (o los vecindarios) que contienen a los ordenadores *trencadisv*, *hekew* y *loki*. En cambio, *nova* formará parte de un vecindario en un sitio diferente. Esto es así porque algunos de los saltos que separan a *nova* del resto de los ordenadores ocurren en

redes WAN, lo cual se puede apreciar en el resultado de la salida del comando *traceroute* y en los tiempos de la **Tabla 5-1**, principalmente en los saltos 3 y 4.

5.2.3. GRUPOS DE MULTIDIFUSIÓN DE LA RED DE SUPERPOSICIÓN

La sección anterior proporciona un ejemplo de cómo configurar la red de superposición (overlay network) del SMRD, para garantizar que el mismo pueda sacar provecho de las características de la red física. Esta sección describe cómo se organiza la difusión a grupos de multidifusión sobre la red de superposición. Para ello, cada súper-peer mantiene una tabla de encaminamiento que contiene referencias a grupos de multidifusión, que son utilizados para enviar el mismo mensaje, simultáneamente, a un grupo de nodos de la red de superposición. Estos grupos se crean centralmente antes de crear la red de superposición, y cada nodo (peer o súper-peer) que se une a la red conoce de antemano a qué grupos pertenece.

Esta condición de que cada nodo tiene que conocer todos los grupos a los que pertenece, es una consecuencia de la necesidad de tomar en consideración la topología del Grid para organizar las operaciones de monitorización. Normalmente, los nodos del SMRD van a ser desplegados en un sitio o un recurso operado por un proveedor de servicios específico. Por esta razón, siempre será posible conocer cuáles son los destinos de los mensajes de difusión, y por tanto, conocer de qué destinos forma parte un nodo determinado. Por ejemplo, en un Grid hipotético compuesto por los sitios *S1* y *S2*, un súper-peer *P1* ubicado en el vecindario local *N3*, del sitio regional *S2* tendrá como destinos a los grupos de multidifusión formados por:

- Los peer regulares del vecindario *N3*, a los cuales retransmitirá los mensajes que lleguen desde el exterior del vecindario *N3*.
- Los nodos súper-peer del vecindario *N3*, a los cuales trasmitirá la información de control, que está formada principalmente por mensajes de pertenencia a grupos, roles en la red P2P, etc.
- Los nodos súper-peer que forman parte del sitio *S2*, pero que no forman parte del vecindario *N3*, a los cuales retransmitirá los mensajes que lleguen desde el vecindario *N3*.
- Los nodos súper-peer que no formen parte de ninguno de los grupos anteriores, a los cuales retransmitirá también los mensajes que lleguen desde el vecindario *N3*, pero después de trasmitirlos al grupo anterior.

Adicionalmente, el mismo súper-peer formará parte de varios grupos de multidifusión que lo convierten en destino en los siguientes casos:

- En la recepción de mensajes que lleguen desde cualquier peer regular del vecindario *N3*.
- En la recepción de mensajes que lleguen desde cualquier súper-peer del

vecindario *N3*.

- En la recepción de mensajes que lleguen desde cualquier súper-peer del sitio *S2*, que no sea del vecindario *N3*.
- En la recepción de mensajes que lleguen desde cualquier súper-peer que no sea del sitio *S2*.

La **Tabla 5-2** muestra los grupos de multidifusión que han sido descritos en el ejemplo anterior. Las tres primeras columnas representan el sitio regional, el vecindario local y el súper-peer. El resto de las columnas representan a los grupos de multidifusión. La nomenclatura utilizada para éstos organiza las entidades de izquierda a derecha, por orden de abstracción. Por ejemplo, el súper-peer 1 del vecindario 3 del sitio 2 se representa como *S2@N3@P1*, donde se utilizan las letras *S* para el sitio, *N* para el vecindario (neighborhood) y *P* para el súper-peer. Además, la letra *G* representa un dominio global y la letra *M* representa a un grupo especial, formado por los nodos súper-peer del mismo vecindario. Este grupo especial se llama grupo de control (representado en la tabla por *CTRL-CHANNEL*), y hay uno por vecindario. Los súper-peer del mismo vecindario utilizan el grupo de control para intercambiar información de gestión. Por ejemplo, cuando un súper-peer quiere unirse (ya sea por primera vez o después de una desconexión) al vecindario, envía un mensaje al grupo de control del vecindario, solicitando un identificador válido que le permita obtener luego la lista de sitios y grupos que debe gestionar.

La **Tabla 5-2** representa una red de superposición formada por 2 sitios, cada uno con 3 vecindarios y 2 nodos súper-peer por vecindario. Como habíamos visto anteriormente, los peer regulares se comunican solamente con los nodos (peer y súper-peer) de su vecindario. De aquí que, un peer regular utilizará el grupo vecindario (representado en la tabla por *NBHD-CHANNEL*) para enviar y recibir información en la red P2P. Para conseguir esto, cada nodo tiene un archivo de configuración que especifica a qué sitio y a qué vecindario pertenece. Un peer regular utiliza esta información para saber a qué canal *NBHD-CHANNEL* tiene que conectarse, y luego monitoriza la recepción de mensajes de otros nodos en este canal y envía sus propios mensajes por la misma vía.

En consecuencia, cada despliegue tiene que proporcionar un archivo de configuración global, que vincula a los sitios reales con los identificadores que son utilizados para nombrar los canales. Las Secciones A.3.1 y A.3.2 del Anexo 3 muestran los archivos de configuración utilizados por Monitor Demonio. El archivo de configuración global es único en el Grid para un entorno de trabajo específico, por ejemplo la VO, y para su distribución debe utilizarse algún mecanismo seguro, como por ejemplo, a través del componente RAISC de GRIDIFF. En este archivo, cada entorno de trabajo especifica un nombre que lo identifica dentro del sistema de monitorización. Este nombre será utilizado como prefijo en cada uno de los grupos de multidifusión de la red de superposición, asociados con el entorno. De esta forma es posible desplegar un único SMRD que sirva para monitorizar más de una VO.

Tabla 5-2: Grupos de multidifusión de la red de superposición. Las tres primeras columnas representan el sitio regional, el vecindario local y el súper-peer, el resto de las columnas representan a los grupos. Cada grupo permite intercambiar información, de forma eficiente, con un grupo de nodos de la red P2P.

Sitio	Vecin-dario	Súper-peer	CTRL-CHANNEL	NBHD-CHANNEL	SITE-OUTPUT	GRID-OUTPUT	SITE-INPUT	GRID-INPUT
1	1	1	S1@M1	S1@N1	S1@N1@P1	G1@N1@P1	S1@N2@P1 S1@N2@P2 S1@N3@P1 S1@N3@P2	G2@N1@P1 G2@N1@P2 G2@N2@P1 G2@N2@P2 G2@N3@P1 G2@N3@P2
1	1	2			S1@N1@P2	G1@N1@P2		
1	2	1	S1@M2	S1@N2	S1@N2@P1	G1@N2@P1	S1@N1@P1 S1@N1@P2 S1@N3@P1 S1@N3@P2	
1	2	2			S1@N2@P2	G1@N2@P2		
1	3	1	S1@M3	S1@N3	S1@N3@P1	G1@N3@P1	S1@N1@P1 S1@N1@P2 S1@N2@P1 S1@N2@P2	
1	3	2			S1@N3@P2	G1@N3@P2		
2	1	1	S2@M1	S2@N1	S2@N1@P1	G2@N1@P1	S2@N2@P1 S2@N2@P2 S2@N3@P1 S2@N3@P2	G1@N1@P1 G1@N1@P2 G1@N2@P1 G1@N2@P2 G1@N3@P1 G1@N3@P2
2	1	2			S2@N1@P2	G2@N1@P2		
2	2	1	S2@M2	S2@N2	S2@N2@P1	G2@N2@P1	S2@N1@P1 S2@N1@P2 S2@N3@P1 S2@N3@P2	
2	2	2			S2@N2@P2	G2@N2@P2		
2	3	1	S2@M3	S2@N3	S2@N3@P1	G2@N3@P1	S2@N1@P1 S2@N1@P2 S2@N2@P1 S2@N2@P2	
2	3	2			S2@N3@P2	G2@N3@P2		

Adicionalmente, cada entorno definirá un número de parámetros globales de configuración, como la utilización o no de sitios regionales (el sistema de monitorización utiliza los grupos sitio regional y dominio global para agrupar a los nodos remotos, de manera opcional también puede trabajar sin sitios, utilizando solamente los dominios globales), el intervalo de refresco de los indicadores dinámicos, y parámetros estructurales de la red de superposición, como el número máximo de sitios regionales, el número máximo de vecindarios locales por sitio y la redundancia de los nodos súper-peer en los vecindarios. Por último, cada entorno proporciona un mapa que vincula el nombre de los sitios regionales (acordado de antemano entre los participantes de las VOs, ya sea el nombre del proveedor de servicios del sitio, el

nombre de la organización, etc.) con un identificador numérico que es utilizado para nombrar los grupos de multidifusión de la red de superposición.

En resumen, la topología de la red de superposición se define estáticamente antes de crear la misma. Para ello, es necesario definir el número máximo de sitios regionales, de vecindarios locales por sitio y de nodos súper-peer por vecindario. Estos valores definen el número máximo de grupos de multidifusión (para más detalles ver las Ecuaciones **Ecuación 5-13**, **Ecuación 5-14**, **Ecuación 5-15**, **Ecuación 5-16** y **Ecuación 5-17**, al final de la sección), y por tanto, el número máximo de referencias que pueden llegar a contener las tablas de enrutamiento de los nodos peer y súper-peer de la red. Además, es necesario definir la identidad de cada sitio regional. Sin embargo, la configuración global de la red de superposición no dificulta el despliegue del SMRD porque los proveedores de servicio controlan la forma de operar la red a nivel local. Por ejemplo, en una VO determinada, la configuración global define que cada proveedor de servicios específico opera un conjunto de sitios regionales identificados globalmente, y que cualquiera de los sitios de cualquier proveedor puede agrupar hasta un máximo establecido de vecindarios locales, cada uno de ellos con un número máximo de nodos súper-peer. De esta forma, la entrada y salida de nodos en la red de superposición produce cambios en la pertenencia a grupos de multidifusión, pero no en las tablas de enrutamiento. Por ende, los proveedores de servicios pueden añadir y eliminar nodos a la red en función de sus necesidades, sin causar un impacto considerable sobre las prestaciones del SMRD.

Un ejemplo concreto consiste en un proveedor de servicios que amplía su infraestructura con una nueva LAN, compartimentada en varias redes lógicas. El proveedor comparte una parte de sus recursos en una infraestructura Grid, y tiene asignado globalmente un conjunto de identificadores de sitio regional, de los cuales utiliza una parte para gestionar los recursos instalados inicialmente y otra parte está sin asignar. Para añadir los recursos de la nueva LAN al sistema de monitorización Grid, el proveedor utiliza uno de los indicadores sin asignar para identificar a la misma, y despliega un conjunto de nodos peer y súper-peer, según la estructura lógica de la red física, asignando un identificador de vecindario local diferente por cada segmento lógico de la red. Los nuevos nodos solamente necesitan conocer la identidad del sitio y del vecindario donde están ubicados. Todo lo demás se configura de forma automática, como se verá en las secciones siguientes.

Por otra parte, cada nodo de la red P2P utiliza el archivo de configuración local para definir su comportamiento. Este archivo define si el nodo actuará como peer o como súper-peer, y si tendrá relación con un componente SIP (es decir, si dispondrá de una interfaz IPC para interrogar a la RHT). Luego, en caso de que el nodo actúe como peer regular, como súper-peer, o como ambos a la vez, el archivo de configuración local define parámetros de configuración que describirán el comportamiento del nodo en cada caso. En caso de actuar como súper-peer, este archivo definirá el comportamiento del nodo durante la formación de la red P2P. De todos los nodos súper-peer de un vecindario, solamente uno puede ser líder. El archivo de configuración local de un nodo define si el nodo debe reclamar el liderato del vecindario, esperar a que otro nodo lo reclame, o simplemente adaptarse a las condiciones que encuentre en el momento en que se une a la red. Además, los nodos que actúan como peer regulares conocen, a través del archivo de configuración local, si deben monitorizar el recurso

local en el que se encuentran desplegados, y en caso de hacerlo, reciben información del sistema local, como la dirección de Internet y el número de procesadores físicos y lógicos (núcleos) instalados en el mismo. De igual forma, conocerán una lista de nodos que no participan en la red P2P, de los cuales podrán recibir notificaciones de estado. A estos nodos se les denomina virtuales porque pierden su identidad, y su información se vincula con el peer que forma parte de la red P2P. Recordemos que los nodos virtuales utilizan una vía de comunicación alternativa, que consiste en trasmitir la información al peer a través de un canal tradicional (HTTP/HTTPS), y que el peer se encarga de difundir esta información al resto de los nodos en la infraestructura utilizando la red P2P. Adicionalmente, el archivo de configuración local contiene información que le dice al peer regular cómo gestionar los volcados a disco de la RHT, y al súper-peer si debe activar, o no, el soporte para gestionar un Monitor Wrapper a través de un Monitor Portal en el recurso local. Por último, el archivo de configuración local contiene la información de pertenencia del nodo, que es la que utiliza el Monitor Demonio para crear la lista de grupos de multidifusión de la red de superposición.

Los grupos de multidifusión, como los que se muestran en la **Tabla 5-2**, tienen funciones específicas. Los súper-peers utilizan el grupo *NBHD-CHANNEL* para comunicarse con los nodos del vecindario. Sin embargo, a diferencia de los peer regulares, los súper-peers pueden comunicarse, de forma selectiva, con grupos de súper-peers de la red. Para comunicarse con los súper-peers de su vecindario, un súper-peer utiliza el grupo *CTRL-CHANNEL*, que tiene la misma forma que el grupo *NBHD-CHANNEL*, cambiando la *N* por una *M* en el identificador del grupo. Como fue descrito anteriormente, este canal de comunicación se utiliza para intercambiar información de control, como por ejemplo información de pertenencia a grupos. En cambio, para comunicarse con los súper-peers que no forman parte de su vecindario, los súper-peers utilizan un mecanismo diferente al descrito hasta el momento. Las comunicaciones en los grupos *CTRL-CHANNEL* y *NBHD-CHANNEL* son bidireccionales, porque estos canales pueden ser utilizados lo mismo para enviar, que para recibir mensajes. Es decir, cada nodo en la red de superposición tiene un grupo de multidifusión *NBHD-CHANNEL* al cual envía mensajes, y a la vez también es destino de este grupo. Si el nodo es un súper-peer, tendrá además un grupo *CTRL-CHANNEL* con las mismas características. Sin embargo, los grupos *SITE-OUTPUT*, *GRID-OUTPUT*, *SITE-INPUT* y *GRID-INPUT* son unidireccionales, y por tanto sólo pueden ser utilizados para enviar o para recibir, pero no las dos cosas. Los grupos *SITE-OUTPUT* son utilizados por los súper-peers para enviar mensajes a los súper-peers que forman parte de su sitio, pero no de su vecindario. De forma semejante, los grupos *GRID-OUTPUT* son utilizados por los súper-peers para enviar mensajes a los súper-peers que forman parte de otros sitios. Los grupos *SITE-INPUT* y *GRID-INPUT* tienen una función muy similar, pero vinculada a la recepción de mensajes. Es decir que, cada nodo súper-peer tiene un grupo de multidifusión *SITE-OUTPUT* y otro *GRID-OUTPUT*, a los cuales envían mensajes. Además, cada nodo súper-peer es destino de un conjunto determinado de grupos de multidifusión, que son los grupos *SITE-INPUT* y *GRID-INPUT* del súper-peer en cuestión.

Este esquema permite difundir mensajes a un grupo específico de destinos en la red P2P, en una forma muy efectiva. Cuando se forma la red P2P, los nodos (peer regulares y súper-peer) se suscriben, selectivamente, a los grupos de multidifusión de la

red de superposición. Cada nodo conocerá, de su archivo de configuración local, a qué grupos deberá suscribirse para enviar y recibir mensajes. Luego, para enviar un mensaje, el nodo utiliza el nombre de un grupo para referirse al destinatario del mensaje, en lugar de enviarlo a un nodo en particular. El software de Spread se encarga de distribuir el mensaje a todos los nodos suscritos al grupo al cual está dirigido el mensaje, utilizando para ello los mecanismos de difusión más efectivos para cada nodo, dependiendo de los mecanismos proporcionados por la red física subyacente. De esta forma, solamente reciben los mensajes los nodos suscritos al grupo de multidifusión. Al recibir un mensaje, el nodo examina el grupo al que el mensaje fue enviado (en caso de que el receptor sea un súper-peer, además analiza la cabecera del mensaje para descartar la información redundante que se genera producto de la repetición de mensajes en los súper-peer redundantes), y solamente con esta información decide qué acción debe tomar con el mensaje recibido. Por ejemplo, si el súper-peer 1 del vecindario 1 del sitio 2 recibe un mensaje destinado al grupo *S2@N1@P1*, entonces, de no existir registros redundantes para este mensaje, la acción a tomar consiste en reenviar el mensaje al grupo *S2@N1*, en caso contrario descartar.

El número total de grupos de control, vecindario local, sitio regional y dominio global necesarios para gestionar la red de superposición del SMRD viene dado, respectivamente, por las ecuaciones:

$$\mathrm{nCTRL} = \mathrm{ns} \cdot \mathrm{nv} \qquad \textbf{Ecuación 5-13}$$

$$\mathrm{nNBHD} = \mathrm{ns} \cdot \mathrm{nv} \qquad \textbf{Ecuación 5-14}$$

$$\mathrm{nSITE} = \mathrm{ns} \cdot \mathrm{nv} \cdot \mathrm{r} \qquad \textbf{Ecuación 5-15}$$

$$\mathrm{nGRID} = \mathrm{ns} \cdot \mathrm{nv} \cdot \mathrm{r} \qquad \textbf{Ecuación 5-16}$$

Donde *ns* es el número máximo de sitios regionales en la infraestructura, *nv* es el número máximo de vecindarios locales por sitio, y *r* es la redundancia de la red súper-peer. La **Ecuación 5-17** agrupa todos estos totales, y permite calcular el número total de grupos de multidifusión que son necesarios para construir la red de superposición del SMRD, para una infraestructura específica:

$$\mathrm{ng} = 2 \cdot \mathrm{ns} \cdot \mathrm{nv} \cdot (1 + \mathrm{r}) \qquad \textbf{Ecuación 5-17}$$

Utilizando esta ecuación podemos calcular que para construir una red de superposición para monitorizar una VO con 2 sitios y 3 vecindarios por sitios, con una redundancia de súper-peer igual a 2, son necesarios 36 grupos. Este ejemplo coincide con el sistema

descrito en la **Tabla 5-2**. Si contamos los grupos mostrados en la **Tabla 5-2**, veremos que el total coincide con el número obtenido utilizando la **Ecuación 5-17**.

5.2.4. PROTOCOLO DE MONITORIZACIÓN

La **Tabla 5-3** muestra el formato de los mensajes transmitidos en la red P2P. Los mensajes están formados por una cabecera y un cuerpo. Las cabeceras contienen tres campos: el tipo de cabecera, el tipo de mensaje y la etiqueta de reenvío. El tipo de cabecera define la finalidad del mensaje, y en consecuencia, también define el tipo de nodos a los cuáles está destinado el mensaje. Por ejemplo, una solicitud de suscripción es un mensaje que se envía al grupo de control del vecindario, con una cabecera del tipo *SPEER_SUBS*, indicando a todos los súper-peers (activos y no activos) que deben atender dicha solicitud.

Adicionalmente, el tipo de mensaje indica si el nodo emisor espera una respuesta del receptor, o no. Este campo es muy importante porque permite a los receptores distinguir entre los mensajes de solicitud, respuesta y notificaciones. Por ejemplo, durante la fase de suscripción, un peer identifica los diferentes estados del proceso de suscripción de un peer remoto por la combinación del tipo de mensaje con el contenido del cuerpo del mensaje. El tipo *SYN* indica que el receptor espera una respuesta a la solicitud enviada en el cuerpo del mensaje, y el tipo *ACK* indica que el mensaje enviado contiene una respuesta o una notificación en el cuerpo del mensaje. Esta notación es similar a la utilizada en el protocolo TCP/IP, donde un mensaje *SYN* corresponde a un evento de sincronización (synchronization), mientras que un mensaje *ACK* corresponde a un evento de contestación o respuesta (acknowledgment).

El último campo en la cabecera del mensaje es la etiqueta de reenvío, que es necesaria para gestionar la información redundante que se produce debido a la replicación de los nodos súper-peer en la red P2P. Este campo indica si el receptor del mensaje debe, o no, reenviar el mensaje al próximo grupo de multidifusión. Este campo permite reducir las comunicaciones redundantes en la red P2P, y es utilizado, principalmente por los súper-peers. Los súper-peers modifican siempre la etiqueta de reenvío de la cabecera antes de reenviar un mensaje que han recibido del exterior (un mensaje cuyo origen sea un sitio que no coincide con el sitio regional del súper-peer, o un dominio global) al vecindario local, para indicar al resto de los súper-peers del vecindario que el mensaje ha sido reenviado localmente por un súper-peer del vecindario. Esto es, un súper-peer crea un mensaje para enviarlo a un sitio regional o a un dominio global, y marca con un *1* la etiqueta de reenvío en la cabecera del mensaje. Con ello consigue que todos los súper-peers que reciban el mensaje, vuelvan a reenviarlo en sus vecindarios. Sin embargo, antes de difundirlo en su vecindario, los súper-peers que reciben el mensaje cambiarán la etiqueta de reenvío, poniendo un *0* donde antes había un *1*, y además guardarán un acuse de recibo para saber que el mensaje ya ha sido procesado. En consecuencia, cualquier súper-peer que reciba el mensaje hará dos comprobaciones antes de reenviar el mensaje en su vecindario: primero comprobará que no existe un acuse de recibo para el mensaje en su registro local, y luego comprobará que la etiqueta de reenvío está activada (valor 1). Cualquiera de las dos condiciones que no se cumpla evitará el reenvío del mensaje en el vecindario. De esta forma, los súper-peers de un vecindario que reciban un mensaje con la etiqueta de reenvío desactivada, van a tratar

el mensaje como si ellos mismos fueran el destino final del mensaje, y no van a realizar ningún tipo de reenvío hacia ninguno de los grupos. Como el reenvío se produce en una dirección definida, los mensajes se identifican para guardar acuse de recibo utilizando campos específicos de cada tipo de mensaje. Por ejemplo, los mensajes de notificación de peer regular, que son la mayor parte de los mensajes transmitidos en la red P2P, se identifican utilizando una codificación específica que contiene el origen y destino del mensaje, y la marca de tiempo (timestamp) tomada en el emisor (representado por *RTIME* en la **Tabla 5-3**).

Tabla 5-3: Protocolo de mensajes. Las cabeceras se muestran en itálica, mientras que los campos del cuerpo se muestran en estilo regular. La descripción del contenido de proporciona a continuación en la sección.

Tipo de mensaje	**Formato del mensaje**
Suscripción de súper-peer	*SPEER_SUBS* | *SYN* | *0* | RND_CHAL *SPEER_SUBS* | *ACK* | *0* | RND_CHAL | SP_ID
Suscripción de peer regular	*RPEER_SUBS* | *SYN* | *1* | HOST_IP | RTIME | PHY&LOG_NCPUS | MEM&SWAP *RPEER_SUBS* | *ACK* | *0* | HOST_IP
Notificación de peer regular	*RPEER_LOAD* | *ACK* | *1* | HOST_IP | RTIME | L1&L5&L15 | AVAIL_MEM&SWAP
Notificación de nodo virtual	*VHOST_INFO* | *ACK* | *1* | VHOST_IP | PHY&LOG_NCPUS | MEM&SWAP | L1&L5&L15 | AVAIL_MEM&SWAP
Solicitud de información	*RPEER_INFO* | *SYN* | *1* | HOST_IP *RPEER_INFO* | *ACK* | *1* | HOST_IP | RTIME | PHY&LOG_NCPUS | MEM&SWAP
Señal	*SIGNAL* | *ACK* | *0* | SIGNAL_ID
Señal con identificador	*SIGNAL* | *ACK* | *0* | SIGNAL_ID | SESSION_ID

El cuerpo de los mensajes cambia de un caso a otro. Algunos mensajes tienen un cuerpo simple, mientras que otros pueden contener múltiples segmentos de datos. Este último es el caso de los mensajes de suscripción y de notificación de los peer regulares. En los dos, el emisor puede proporcionar información de varios nodos en el mismo mensaje. Después del campo *HOST_IP*, que contiene la dirección de Internet del peer, y del campo *RTIME*, que contiene la marca de tiempo tomada en el peer de origen, el emisor puede incluir varios bloques de información. El contenido de estos bloques corresponderá a cada uno de los nodos virtuales monitorizados por el peer (los nodos

que utilizan vías tradicionales para comunicarse con el peer regular). En el caso de las suscripciones, el cuerpo de los mensajes contiene el número de procesadores físicos y lógicos, así como la memoria física y la swap instalada en los nodos. En consecuencia, el mismo emisor producirá luego mensajes de notificación que incluyan los indicadores dinámicos (en tanto por ciento) para cada uno de estos campos. Es importante resaltar que los peers mantendrán, en cada mensaje, el mismo número de bloques y la ordenación de la suscripción. En caso contrario, se producirá una solicitud de información.

La descripción de los campos adicionales, como *RND_CHAL*, que se utiliza para enviar una cadena de texto aleatoria como parte de la suscripción de un nuevo súper-peer a la red P2P, y los campos *SIGNAL_ID* y *SESSION_ID*, que se utilizan para gestionar mensajes que contienen señales interpretables por los nodos, se proporciona en las secciones siguientes.

A continuación, se describe la utilización del protocolo de mensaje como parte del proceso de suscripción de un súper-peer a la red P2P. Este caso describe la mayoría de los detalles a tener en cuenta durante la operación del sistema de monitorización. El resto de los casos (suscripción de los peer regulares, notificaciones, búsqueda de información y envío de señales) tienen una complejidad similar o menor, que el caso de la suscripción de los súper-peers.

5.2.5. PROCESO DE SUSCRIPCIÓN DE UN SÚPER-PEER

La **Figura 5-18** representa un vecindario donde dos súper-peers inician un proceso de suscripción al mismo tiempo. Suponiendo que el mecanismo de transmisión utilizado en la red es el envío de mensajes del emisor al receptor (unicast), la flecha en la figura muestra la dirección de transmisión en el anillo. Adicionalmente, los anillos pueden ser unidireccionales. Normalmente, la dirección de transmisión no afecta el resultado final del proceso, que tampoco es afectado por el tipo de transmisión (unicast, multicast o broadcast). Por tanto, la dirección de transmisión y el tipo no son importantes para el ejemplo, y sólo se muestran con el fin de ejemplificar un caso particular que hasta ahora habíamos mencionado, pero no habíamos utilizado, y que ocurre cuando no es posible utilizar multidifusión ni difusión en la red que soporta al vecindario.

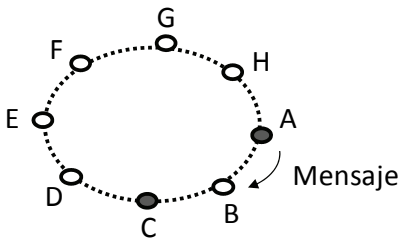


Figura 5-18: Proceso de suscripción de dos súper-peers en un anillo de vecindario. A y C representan súper-peers, mientras que B, D, E, F, G y H representan peer regulares.

El proceso de suscripción se inicia cuando un súper-peer envía un mensaje de presentación al vecindario. Después de esto, el súper-peer comprobará periódicamente la cola de mensajes para procesar los mensajes entrantes que tienen que ver con el proceso de suscripción. El resto de los mensajes son ignorados. Cada súper-peer escoge un número aleatorio de veces que la cola de entrada es comprobada como parte del proceso de suscripción. Este número es el período de espera del súper-peer. Normalmente, la cola de entrada es comprobada cada 1-2 segundos hasta que se alcanza el período de espera. Los mensajes recibidos durante este tiempo afectan el comportamiento y el estado del súper-peer, y desencadenan una secuencia de respuesta en el nodo. La **Tabla 5-4** muestra el tránsito del súper-peer *A* por todos los estados del proceso de suscripción. Las columnas de la tabla muestran las diferentes secuencias de respuesta por las que puede transitar *A* en cada estado, en dependencia de los mensajes que reciba del súper-peer *C*, que también se encuentra en el proceso de suscripción.

En la presentación, el súper-peer *A* envía un mensaje al vecindario, utilizando el canal de control, que contiene una cadena de texto aleatorio. Durante este estado de la suscripción, se pueden dar tres posibles escenarios. En primer lugar, puede ser que no haya ningún súper-peer suscrito al canal de control del vecindario local y que tampoco haya ningún súper-peer tratando de suscribirse, con lo cual, *A* no recibirá ninguna respuesta a su mensaje de solicitud y entonces, una vez agotado su período de espera, pasará al siguiente estado. En segundo lugar, puede que haya otro súper-peer tratando de suscribirse al mismo tiempo que *A*, lo cual está representado en la primera fila de la **Tabla 5-4** por un mensaje de solicitud que proviene del súper-peer *C*, y que es ignorado por *A*. En general, en este estado *A* ignorará todos los mensajes que no estén originados en un súper-peer activo. El último escenario consiste en que *C* ha conseguido suscribirse antes que *A*, y se ha convertido en un súper-peer activo. Este escenario está representado en la segunda fila de la **Tabla 5-4** por un mensaje de respuesta que proviene del súper-peer *C* y que contiene la misma cadena de texto aleatoria que envió *A* con su solicitud, para indicar que en efecto se trata de un mensaje de respuesta a la solicitud formulada por *A*, y además contiene el identificador de *C* en el vecindario, en este caso 2, lo cual quiere decir que *C* es el súper-peer 2. Luego, el resto de los estados representados en la **Tabla 5-4** describen dos posibles escenarios diferentes que se corresponden con los dos últimos descritos para la presentación.

Al agotarse el período de espera, *A* pasa del estado de presentación al de solicitud, y envía una propuesta para convertirse en el súper-peer identificado por el número más bajo disponible, con inicio en 1, que es el líder del vecindario. Otra vez se pueden encontrar tres posibles escenarios. Si no hay otro súper-peer suscrito al canal de control del vecindario local y tampoco hay ningún súper-peer tratando de suscribirse, entonces *A* agota su tiempo de espera y pasa al estado siguiente. En cambio, si hay otro súper-peer tratando de suscribirse al mismo tiempo que *A*, sucede lo que está representado en la primera fila de la sección de la **Tabla 5-4** que corresponde al estado de solicitud.

Tabla 5-4: Estados del proceso de suscripción de un súper-peer en un vecindario. La descripción del contenido de proporciona a continuación en la sección.

Estado	***A* envía**	***A* recibe**	**Reacción en *A***
Presentación	SPEER_SUBS | SYN | 0 | 80c33c56	SPEER_SUBS | SYN | 0 | 820e1ce8	*A* ignora la suscripción simultánea de *C*.
		SPEER_SUBS | ACK | 0 | 80c33c56 | 2	*A* almacena la información de *C*, previamente suscrito como el súper-peer 2.
Solicitud (completado el período de espera de *A*)	SPEER_SUBS | SYN | 0 | 80c33c56 | 1	SPEER_SUBS | SYN | 0 | 820e1ce8 | 1	Las suscripciones simultáneas de *A* y *C* colapsan. Ambos peers obtienen un nuevo período de espera y reinician el proceso de suscripción.
		SPEER_SUBS | ACK | 0 | 80c33c56 | 1	*C* fue suscrito, con anterioridad, como el súper-peer 1. *A* reinicia el proceso de suscripción, al final *A* será el súper-peer 2.
Reconocimiento (completado el período de espera de *A*)			*A* es el súper-peer 1.

En este caso, *A* recibe un mensaje de solicitud de otro súper-peer que está tratando de obtener el identificador 1 en el vecindario, que es el mismo que está solicitando *A* para sí mismo. En este escenario se produce una colisión que se resuelve cuando ambos nodos retoman el proceso de suscripción desde el principio, cambiando sus tiempos de espera por un tiempo aleatorio más largo, con el objetivo de disminuir la probabilidad de que se produzca una nueva colisión. El último escenario posible consiste en que hay un súper-peer activo en el vecindario que ya tiene asignado el identificador que está solicitando *A*. En este caso, representado en la segunda fila de la sección de la **Tabla 5-4** que corresponde al estado de solicitud, se produce una colisión que se resuelve cuando *A* retoma el proceso de suscripción desde el inicio, donde se supone que el súper-peer activo enviará un mensaje de respuesta durante la presentación de *A*, para que este pueda elegir un identificador válido.

Por último, si agotado nuevamente el período de espera de *A*, ninguno de los súper-peers del vecindario ha enviado una solicitud para obtener el mismo indicador que pretende *A*, y además ninguno de los súper-peers activos han vetado la propuesta de *A*, entonces *A* se convierte en el súper-peer identificado por el número indicado en la propuesta, con lo cual termina el proceso de suscripción y *A* se convierte en un súper-peer activo.

Un súper-peer puede repetir el proceso de suscripción en cualquier momento, si detecta que hay un identificador duplicado en su vecindario. Por ejemplo, si *A* es el súper-peer 1 del vecindario 1 del sitio 1, y recibe un paquete que está destinado a un grupo que coincide con su canal de salida *NBHD_CHANNEL* (para este ejemplo, *S1@N1@P1*), entonces *A* descartará su identidad e iniciará una nueva ronda de suscripción. Esta medida evita que se produzcan situaciones donde varios súper-peers comparten un mismo indicador tras producirse, por ejemplo un fallo en la red de comunicación.

5.2.6. PROCESO DE SUSCRIPCIÓN DE UN PEER REGULAR, NOTIFICACIÓN Y SOLICITUD DE INFORMACIÓN

La única diferencia a destacar en este caso es la utilización de la etiqueta de reenvío de la cabecera del mensaje, y del campo *RTIME*, del cuerpo del mensaje. La **Tabla 5-5** ilustra el uso de estos campos adicionales como parte del proceso de suscripción de un peer regular y también como parte de los proceso de notificación y de solicitud de información.

El proceso de suscripción de un peer regular consiste en obtener una respuesta a su solicitud de suscripción. Esta respuesta puede ser enviada por cualquier súper-peer del vecindario. Una vez que un peer regular ha conseguido suscribirse, quedará activo, enviando notificaciones regulares con el estado de los indicadores dinámicos de todos los nodos que monitoriza. Los formatos de mensaje de suscripción y de notificación de peer regular, así como el de solicitud de información, son los descritos en la **Tabla 5-3** de la Sección 5.2.4.

La solicitud de información se produce cuando un nodo de cualquier tipo recibe una notificación que contiene información de un peer regular que no conoce. Una notificación de peer regular es un mensaje que contiene un conjunto de indicadores

dinámicos en los campos *L1&L5&L15* y *AVAIL_MEM&SWAP*, específicamente el tanto por ciento de la carga de la CPU en el último minuto y en los últimos 5 y 15 minutos, además del tanto por ciento de la memoria física y swap disponibles en el recurso monitorizado. Sin embargo, esta información no es útil si además no se conocen los valores absolutos que describen al recurso, que son el número de procesadores físicos y lógicos (núcleos) de los que dispone el recurso, y la cantidad de memoria física y swap instalada en el mismo. Cuando un nodo recibe una notificación de este tipo, y al tratar de actualizar la RHT encuentra que no dispone de la información absoluta para el recurso, entonces envía un mensaje de solicitud de información que se propaga por la red P2P hasta que alcanza a un nodo (no necesariamente el peer regular del cual se solicita información) que puede proporcionar la información del nodo que ha enviado la notificación. Por ejemplo, la solicitud de información puede ser respondida por uno de los peer regulares de su vecindario, en caso de que alguno conozca la información del nodo, o sino la solicitud viaja hasta el sitio donde se encuentra dicho nodo, y allí se produce la respuesta. Este mecanismo aporta gran flexibilidad al sistema de monitorización, porque permite que nuevos nodos, vecindarios y sitios completos sean añadidos al sistema, minimizando la sobrecarga que esto causa.

Cuando un súper-peer recibe un mensaje de suscripción, notificación o una solicitud de información de un peer regular, entonces utiliza los campos adicionales del mensaje para determinar si debe reenviar el mensaje hacia el vecindario local, hacia el sitio regional o al dominio global, dependiendo del caso. Como ha sido descrito en las secciones anteriores, un mensaje enviado en el vecindario con la etiqueta de reenvío activada, será reenviado tantas veces como súper-peers haya en el vecindario. Es decir, cada súper-peer en el vecindario enviará el mismo mensaje en el anillo del sitio y en la malla del Grid.

Tabla 5-5: Utilización de los campos adicionales en el mensaje como parte de la suscripción de peer regulares, la notificación y la solicitud de información. Las columnas de la tabla representan el grupo remitente y destinatario del mensaje, y la acción que ejecuta el súper-peer que recibe el mensaje.

Desde	**Hacia**	**Acción**
NBHD	Sitio	Si la etiqueta de reenvío está activada en la cabecera del mensaje, entonces reenvía el mensaje.
NBHD	Grid	Si la etiqueta de reenvío está activada en la cabecera del mensaje, entonces reenvía el mensaje.
Sitio	NBHD	Si el campo *RTIME* del cuerpo del mensaje es mayor que el campo *RTIME* almacenado en la base de datos local para el tipo de cabecera del mensaje (*RPEER_SUBS*, *RPEER_LOAD* o *RPEER_INFO*), entonces desactiva la etiqueta de reenvío de la cabecera del mensaje y reenvía el mensaje. En caso contrario, ignora el mensaje.
Grid	NBHD	

El campo *RTIME* no es más que una marca de tiempo que crea el emisor del mensaje para indicar el orden en que han sido creados los mensajes. Esta marca, junto con el

tipo de cabecera del mensaje, son almacenados localmente por los nodos súper-peer, a modo de acuse de recibo. Los mismos comprueban sus registros locales antes de procesar cualquier mensaje, y descartan los mensajes con un *RTIME* menor que el último mensaje, con el mismo tipo de cabecera, recibido del mismo remitente.

5.2.7. SEÑALES

Las señales constituyen un tipo especial de mensaje que solamente se envían entre nodos del mismo vecindario. Normalmente, las señales son enviadas entre dos nodos o de un nodo a sí mismo. En la mayoría de los casos las señales forman parte de mecanismos de alarma que notifican la ocurrencia de un suceso importante, y que pueden ser utilizadas, por ejemplo entre dos súper-peers de un vecindario.

Estos mensajes pueden ser de dos tipos: los que no tienen identificador, y los que son identificados por una variable de sesión. Las variables de sesión posibilitan el uso de señales en patrones más complejos, donde los peers tienen que reconocer que una señal forma parte de un flujo de señales.

5.2.8. POLÍTICAS DE CONTROL DE ACCESO PARA LA GESTIÓN DE MENSAJES

Por último, la **Tabla 5-6** muestra las políticas de control de acceso a diferentes niveles. El primer nivel, llamado nivel de servicio, es un punto de control global por el que pasa cualquier mensaje enviado a un peer, sin importar el tipo del peer ni del mensaje. Luego, el mensaje pasa por dos puntos de decisión adicionales: el nivel de súper-peer y el nivel de peer regular. El paso por estos puntos, dependerá de las funciones que desempeñe el peer en cuestión.

Para expresar las políticas de control de acceso en la **Tabla 5-6**, primero se expresa la directiva que se aplica por defecto y luego las excepciones, ordenadas en el orden en que se aplican las directivas, y dentro de cada directiva el orden en que se examinan las condiciones. Por ejemplo, en el siguiente caso:

- Rechazar Todo
- Aceptar:
    - Condición A.
    - Condición B.
- Denegar:
    - Condición C.

Esta política expresa que todos los mensajes que no cumplan alguna de las condiciones A, B o C, serán rechazados. En cambio, si algún mensaje cumple con la condición A o B, será aceptado y procesado en el nodo que lo recibe. Por último, si algún mensaje cumple con la condición C, será denegado.

Tabla 5-6: Políticas de control de acceso a diferentes niveles: a) a nivel de servicio, b) a nivel de súper-peer, y c) a nivel de peer regular. La decisión de aceptar tiene como consecuencia que el mensaje sea procesado por la cadena siguiente, la decisión de denegar, tiene como efecto de ignorar el mensaje sin que se produzca notificación de error, y rechazar provocará que el mensaje sea descartado, pero además deja un mensaje de error en el sistema de trazas y produce una acción de recuperación, por ejemplo enviar un mensaje de error al emisor del mensaje.

Tipo de cabecera	**Tipo de mensaje**	**Política de Control de Acceso de servicio**	**Política de Control de Acceso de súper-peer**	**Política de Control de Acceso de peer regular**
SPEER_SUBS	SYN	Rechazar Todo Aceptar: 1) CTRL && Rol súper-peer Denegar: 1) CTRL && !Rol súper-peer	Denegar Todo Aceptar: 1) !Self	Denegar Todo
SPEER_SUBS	ACK			
RPEER_SUBS	SYN	Denegar Todo Aceptar: 1) NBHD && Rol súper-peer 2) SITE && Rol súper-peer 3) GRID && Rol súper-peer 4) NBHD && Rol peer regular Rechazar: 1) CTRL	Denegar Todo Aceptar: 1) Reenviar == 1	Aceptar Todo Denegar: 1) Self && IP && Reenviar == 1 Rechazar: 1) IP && Reenviar != 1
RPEER_SUBS	ACK		Denegar Todo	Denegar Todo Aceptar: 1) IP
RPEER_NOTIF	ACK	Denegar Todo Aceptar: 1) NBHD && Rol súper-peer 2) SITE && Rol súper-peer 3) GRID && Rol súper-peer 4) NBHD && Rol peer regular Rechazar: 1) CTRL	Aceptar Todo Denegar: 1) Self && Reenviar != 1	Aceptar Todo Denegar: 1) Self && HOST_IP == IP && Reenviar == 1 Rechazar: 1) HOST_IP == IP && Reenviar != 1 2) HOST_IP es desconocida
RPEER_INFO	SYN	Denegar Todo Aceptar: 1) NBHD && Rol súper-peer 2) SITE && Rol súper-peer 3) GRID && Rol súper-peer	Aceptar Todo	Denegar Todo Aceptar: 1) HOST_IP == IP && Reenviar != 1
RPEER_INFO	ACK		Aceptar Todo	Aceptar Todo

		4) NBHD && Rol peer regular Rechazar: 1) CTRL		Denegar: 1) Self && HOST_IP == IP && Reenviar == 1 Rechazar: 1) HOST_IP == IP && Reenviar != 1
SIGNAL	ACK	Rechazar Todo Aceptar: 1) Self	Aceptar Todo	Aceptar Todo
SIGNAL_SID	ACK	Rechazar Todo Aceptar: 1) SESSION_ID == Self SESSION_ID	Aceptar Todo	Aceptar Todo
VHOST_INFO	ACK	Denegar Todo Aceptar: 1) Rol peer regular	Denegar Todo	Rechazar Todo Aceptar: 1) VHOST_IP está registrado

Por ejemplo, si un peer recibe una solicitud de suscripción de un súper-peer, el servicio evalúa su política de acceso y solamente acepta el mensaje si este proviene del canal de control y además el peer que lo recibe tiene activado el rol de súper-peer. Entonces, el súper-peer evalúa su política de control de acceso que solamente aceptará el mensaje si éste proviene de un nodo en la red P2P cuyo identificador sea diferente de su propio identificador. Por último, si el peer receptor además cumple el rol de peer regular, se producirá una tercera comprobación para determinar si se ejecuta el código que implementa éste rol, y específicamente en este ejemplo, la política de control de acceso de peer regular descartará el mensaje porque no es de interés. En cambio, si el peer receptor no cumple el rol de peer regular, entonces no se produce la tercera comprobación.

En otro ejemplo, si un peer recibe una respuesta de una solicitud de información, entonces el servicio evalúa su política de acceso y solamente acepta el mensaje si este proviene de los canales *SITE_INPUT* o *GRID_INPUT* y el nodo que lo recibe actúa como súper-peer, o en caso de que el nodo actúe como peer regular y el mensaje provenga del canal *NBHD_CHANNEL*. En caso de tener el rol súper-peer activado, el súper-peer evalúa su política de control de acceso, que siempre aceptará el mensaje. Por último, si el peer además cumple el rol de peer regular, este rol denegará el mensaje si viene de sí mismo y tiene la etiqueta de reenvío activada (en este caso el peer ha enviado un mensaje al grupo de multidifusión *NBHD_CHANNEL*, del cual él mismo es un destino y por eso descarta cualquier mensaje que provenga de sí mismo y que se produzca dentro del vecindario. Para saber que el mensaje se produce en el ámbito del vecindario local se analiza la etiqueta de reenvío, ya que los mensajes que provienen del

exterior son repetidos por un nodo súper-peer en el vecindario, que antes de reenviar el mensaje, desactiva la etiqueta para evitar que otros súper-peers reenvíen el mismo mensaje), lo rechazará si el campo *HOST_IP* coincide con su propia dirección de Internet y la etiqueta de reenvío está desactivada (en este caso se ha producido una colisión con un peer regular externo al vecindario local, y es necesario tomar alguna acción correctiva), y lo aceptará en cualquier otro caso.

Finalmente, estas políticas establecen que las señales sólo pueden ser enviadas de un nodo a sí mismo. Este tipo de mensajes se utiliza para activar a un nodo que se ha suspendido por algún motivo. Por ejemplo, un nodo que está en proceso de suscripción se suspende hasta que llega algún mensaje de respuesta, o hasta que se agota su tiempo de espera. En caso de que no llegue ningún mensaje de respuesta, el Hilo Tiempo del Monitor Demonio se encarga de enviar un mensaje señal para activar el nodo.

5.2.9. SEGURIDAD DE LAS COMUNICACIONES EN EL SISTEMA DE MONITORIZACIÓN

Desde el punto de vista de la VO, es deseable que el estado de los recursos sea público, no sólo para los participantes de la VO, sino que, en algunos casos, también es deseable que entidades externas a la VO (como pueden ser evaluadores o posibles usuarios y colaboradores) puedan acceder a esta información. En consecuencia, no hay necesidad de asegurar la privacidad de la información de monitorización, pero es necesario garantizar, al menos, un mínimo de seguridad en cuanto a la integridad y la no repudiación de los datos. Esto último está impulsado por la necesidad de vender los recursos computacionales como servicios, y utilizar las estadísticas de uso de los recursos como una forma de compartir la carga de trabajo entre los proveedores de servicio, y también como la base sobre la cual calcular la parte de las ganancias que le corresponde a cada uno de los proveedores. De todo esto, se derivan las pautas que hemos seguido para asegurar el sistema de monitorización: A) no hay necesidad de cifrar las comunicaciones; B) es necesario proporcionar una forma de identificar, de forma inequívoca, a la fuente de los mensajes.

Todo ello nos lleva a un terreno difícil, que es la seguridad de los sistemas P2P. Normalmente, estos sistemas intentan simplificar la parte referente a la seguridad. De ahí que, en la actualidad, la mayoría de los sistemas para compartir archivos en redes P2P no soportan ni siquiera los procesos básicos de seguridad, como son la autenticación, la autorización y el control de acceso. Habitualmente, un cliente P2P comparte los archivos almacenados en una carpeta local del ordenador con todos los participantes de la red P2P y, de forma excepcional, tiene la posibilidad de prohibir el acceso a clientes específicos, identificados por su dirección de Internet.

En cambio, la implementación del SMRD se construye sobre Spread, una infraestructura de soporte para el desarrollo de aplicaciones de comunicación a grupos, sobre la cual se despliega la red de superposición que soporta a la red P2P. Spread proporciona varios esquemas de autenticación y control de acceso, que permiten garantizar la privacidad y la integridad de los datos, así como identificar a las entidades

que participan en la red y organizar el control de acceso. De todo esto, lo más importante para el SMRD es la posibilidad de autenticar las entidades.

El sobrecoste de ejecutar una comprobación de control de acceso en cada mensaje es demasiado grande para nuestros propósitos, y podría llegar a comprometer la escalabilidad del sistema de monitorización. De ahí que, la mejor opción consiste en instalar reglas en el cortafuego que aseguren que un demonio de Spread se pueda comunicar con otros demonios de Spread a través de protocolos de red. Al mismo tiempo, el Monitor Demonio se comunica con Spread a través de sockets de dominio UNIX, con lo cual se regula el acceso a los demonios de Spread a través de permisos del sistema de archivos y grupos de usuarios. De esta forma, una vez que el Monitor Demonio se conecta a Spread a través de un socket de dominio UNIX, el SMRD confía en que todo se comporte bien. En efecto, dentro del SMRD, todos los procesos Monitor Demonio conectados son considerados componentes de procesamiento de una sola aplicación en memoria distribuida. Si en algún caso de uso existiese la necesidad de separar un Monitor Demonio de otro, bastaría con desplegar dos conjuntos de demonios de Spread utilizando dos grupos o usuarios de UNIX diferentes.

Una vez establecidas las bases para el control de acceso, faltaría un mecanismo de no repudiación. Aquí es donde la autenticación juega un papel fundamental. El SMRD proporciona un contexto para la comunicación de información de monitorización, sobre un bus de comunicación a grupos construido sobre Spread. Teniendo en cuenta el volumen de datos intercambiados por el sistema de monitorización y las características generales de los sistemas de computación Grid, estaríamos hablando, como mínimo, de redes de 100Mbps y procesadores x86 de 800MHz, con tasas de transferencia de aproximadamente unos 0,15MB por minuto (más detalles en la Sección 6.5). Es probable, que en estas condiciones haya suficiente holgura para verificar el origen de cada mensaje, el reto está en cómo hacerlo afectando al mínimo las prestaciones y la escalabilidad del sistema de monitorización.

El SMRD tiene la capacidad de diferenciar dos tipos de nodos en la red P2P: los peer regulares, que actúan como sensores locales en los vecindarios, y los súper-peer, que actúan como puente entre diferentes grupos de nodos de la red. Esta diferenciación puede ser aprovechada de la siguiente forma: los sensores (peer regulares), que son mucho más numerosos que los puentes (súper-peer), heredan el nivel de seguridad de los puentes (utilizan la misma autenticación que ha sido comprobada en los puentes), y los puentes, que son mucho menos dinámicos que los sensores porque tienen que comunicarse con nodos en otros dominios administrativos, mientras que los sensores se comunican solamente con los nodos de su vecindario, pueden utilizar la autenticación de Spread.

La autenticación en Spread se produce en el momento en que dos demonios Spread se conectan, y luego, la identidad de cada demonio se mantiene durante todo el tiempo que dure la conexión. En consecuencia, cuando un demonio de Spread envía mensajes, se une a grupos, etc., no necesita información de autenticación. Este enfoque asume, explícitamente, que mientras se mantiene una conexión entre dos demonios Spread, se mantiene la identidad de la entidad que se autentica al principio de la conexión.

Adicionalmente, la autenticación en Spread se produce entre dos partes llamadas cliente y servidor. Aunque los dos demonios que participan en una conexión se autentican mutuamente, cada uno de ellos actuará como cliente cuando quiera demostrar su identidad ante el otro, que será llamado servidor, y también actuará como servidor cuando quiera autenticar la identidad del otro. Para ello, Spread proporciona funciones que permiten al demonio que actúa como servidor especificar los métodos de autenticación de los que dispone, y funciones que permiten al demonio que actúa como cliente especificar el nombre del método de autenticación que desea utilizar y la información que desea presentar para ser autenticado. Los métodos de autenticación de Spread pueden ser extendidos fácilmente, y es posible definir, en el archivo de configuración de cada demonio, los métodos de autenticación soportados.

El SMRD añade un nuevo método de autenticación a Spread, a la vez que proporciona un mecanismo de no-repudiación. Para ello, el SMRD se apoya en la infraestructura de seguridad Grid subyacente. Por lo general, la seguridad en el Grid utiliza criptografía de clave pública (también conocida como criptografía asimétrica) como la base de su funcionalidad. Tal es el caso de la GSI (Grid Security Infrastructure), la infraestructura de seguridad Grid más extendida en la actualidad. En los despliegues basados en la GSI, todos los nodos de una VO almacenan una copia local de las claves públicas de las entidades certificadoras de la VO, así como un par de claves pública y privada, firmadas por una de estas entidades. El mecanismo de autenticación del SMRD consiste en que el cliente envía su certificado, para que el servidor pueda comprobar que está firmado por una de las entidades certificadoras que le son conocidas, luego el servidor construye un mensaje cifrado con la clave pública del cliente, y por último el cliente debe descifrar el mensaje con su clave privada y enviar la respuesta al servidor, que considerará que el cliente está autenticado y completará la conexión, solamente si el contenido del mensaje descifrado por el cliente coincide con el mensaje original construido por el servidor. Este mecanismo sirve, a la vez, como mecanismo de no-repudiación porque solamente los clientes que estén en posesión de un par de claves auténticas podrán establecer conexiones en la red P2P.

Como el proceso de determinar el mensaje original puede tardar algún tiempo, la función de autenticación proporciona un mecanismo de devolución de llamada (*callback*), que hace que el proceso de autenticación no bloquee al demonio de Spread. Así, la autenticación ocurre en varios pasos, que involucran varios mensajes entre los nodos.

Esta solución, aunque mantiene la escalabilidad y las prestaciones del SMRD, tiene dos deficiencias muy importantes. La primera, y además evidente, es la ausencia de un mecanismo para descubrir y corregir a los sensores (peer regulares) que puedan estar comprometidos y que puedan estar enviando información falseada. Si bien es cierto que esta deficiencia se podría resolver utilizando una lista de certificados revocados (Certificate Revocation List o CRL), no es conveniente utilizar este tipo de mecanismos porque dificultan el despliegue del SMRD, al aumentar los requerimientos del mismo. En general, cualquier solución que dificulte el despliegue que tienen que hacer los proveedores de servicios en sus sitios, reduciría la aplicabilidad del modelo.

La forma de afrontar esta deficiencia consiste en mejorar la seguridad de los sitios Grid, prestando una atención especial a los nodos que formen parte del SMRD. En

principio, este no debe ser un problema, porque es un objetivo de todas las organizaciones que participan en la VO. La segunda deficiencia es más difícil de resolver, y consiste en que, al no estar cifradas las comunicaciones entre los participantes del SMRD, cabe la posibilidad de que se produzcan ataques del tipo man-in-the-middle (MitM), en los cuales un atacante que intercepta las comunicaciones entre dos demonios Spread, consigue autenticarse como un súper-peer válido. Si bien es cierto que la complejidad de este ataque es muy alta, en parte por la complejidad técnica que tiene y en parte porque la autenticación se produce solamente una vez (al principio de la conexión), existe la posibilidad de que aún así se produzca. Después de la autenticación es más difícil que se puedan producir ataques del tipo MitM porque el servicio de mensajería utiliza un sistema de referencias para mantener los mensajes ordenados en orden causal. La forma de afrontar esta deficiencia consiste en utilizar un sistema de expiración que fuerce la desconexión de los demonios de Spread que hayan estado conectados, de forma continua, durante un tiempo determinado. Así, los demonios que hayan sido desconectados tendrán que volver a establecer la conexión, y con ello tendrán que volver a autenticarse. Esta medida no evita que se produzcan ataques MitM, pero los dificulta considerablemente, y además disminuye los daños que puedan producir, porque cualquier atacante que consiga hacerse con la identidad de un súper-peer tendrá limitado el tiempo que podrá utilizar dicha identidad. En la práctica, el SMRD utiliza un criterio similar al que rige la utilización de los certificados proxy en los entornos de computación Grid, y considera inválida cualquier conexión de un demonio de Spread que se haya mantenido por más de 12 horas.

En cuanto a la integridad de los mensajes, el hecho de utilizar conexiones persistentes, autenticadas desde su establecimiento, dificulta la posibilidad de que un atacante pueda interceptar los mensajes y modificar su contenido sin que esto sea detectado, máxime cuando las comunicaciones entre súper-peers son redundantes. Esto último obligaría a un atacante que quisiera modificar los mensajes que envía un súper-peer a interceptar los mensajes de todos los demás súper-peers del mismo vecindario.

5.2.10. VOLCADO DE LA RHT

La forma más exacta de conocer el estado del sistema de monitorización consiste en volcar, periódicamente, la información contenida en la RHT a un archivo. Esta información es muy importante para los procesos de validación y depuración del SMRD. Sin embargo, la funcionalidad del sistema de monitorización está contenida en Monitor Demonio, una aplicación escrita en C++ que pone un énfasis especial en las prestaciones, y en evitar, en la medida de lo posible, el acceso a disco. Toda la estructura de datos que soporta a la RHT, la tabla de enrutamiento de los súper-peers y el sistema de gestión de información, se montan en la memoria del ordenador que ejecuta el Monitor Demonio, y todos los métodos de acceso a los datos, incluyendo el almacenamiento, las consultas y el ordenamiento de la información, están optimizados y estrechamente vinculados a la estructura de la red de superposición. De aquí que, volcar la información de la RHT a un archivo puede suponer un sobrecoste muy grande para el SMRD.

Sin embargo, una de las ventajas más interesantes de contar con un sistema de monitorización es la posibilidad de acceder a la información en cualquier momento del

tiempo. Para esto es necesario contar con un mecanismo que permita acceder a la información de la RHT, afectando lo menos posible a la aplicación. La solución que propone el SMRD consiste en implementar un servidor de IPC en un hilo auxiliar. Secciones anteriores explican que el Hilo IPC proporciona un servidor que hace posible la comunicación de otros procesos con el Monitor Demonio, a través de sockets IP. Este servidor permite hacer consultas a la RHT, manteniendo una cache que puede almacenar, simultáneamente, la información de varios nodos. Con todo, el servidor IPC presta sus servicios, principalmente, a las aplicaciones clientes, como aplicaciones Web para el acceso remoto a la información, y aplicaciones de gestión de recursos, como GRIDIFF. Por esta razón, aunque el Hilo IPC proporciona una forma eficiente de acceder a información de la RHT, tampoco soluciona la necesidad de volcar la RHT a disco.

Normalmente, las operaciones de E/S son más lentas, si se comparan con el procesamiento de datos. En este sentido, los sistemas operativos modernos soportan un concepto llamado entrada/salida asíncrona (Asynchronous I/O u AIO). AIO ha sido utilizado con éxito en la implementación de la versión 3 de Samba [69], mejorando las prestaciones de la transferencia de archivos de gran tamaño a través de un enlace de red. Este problema es semejante al problema de volcar la RHT a disco, quitando que en este último caso se trabaja con un sistema de archivos local, y por ello, no hay transferencia de datos por la red. Los desarrolladores de Samba utilizaron una estrategia que consiste en leer por partes la información de los archivos que van a ser transferidos por la red. El tamaño de los fragmentos es elegido de forma tal que estos puedan ser acomodados fácilmente en memoria, y a la vez, que también puedan ser transferidos a disco de forma fácil. Los resultados de un estudio llevado a cabo en el año 2005, y difundido luego en la lista de desarrollo de Samba, demostraron que, en la mayoría de los sistemas disponibles en aquel momento, un buffer de 256KB es ideal para estos propósitos. En cualquier caso, incluso cuando fuese posible disponer de un buffer mayor (por ejemplo, 512KB o 1024KB), hay que tener en cuenta que primero es necesario vaciar el buffer a disco (o llenarlo, según sea el caso) antes de disponer nuevamente de este espacio de memoria, con lo cual existe un compromiso entre la latencia asociada con el acceso a disco y la disponibilidad del buffer.

Los sistemas operativos implementan AIO por medio de llamadas a operaciones asíncronas de lectura y escritura, que permiten a las aplicaciones planificar las operaciones de acceso a disco sin bloquear el hilo hasta que se produzca la operación. Estas ideas nos permitieron implementar un componente que hace el volcado de la RHT a disco, utilizando el tiempo ocioso del procesador. El Hilo Volcado utiliza este componente para hacer el volcado, sin interferir con el hilo principal, que ejecuta Monitor Demonio.

Si consideramos que toda la información de monitorización de un nodo está contenida en una línea del archivo de volcado y que, como promedio, cada línea ocupa aproximadamente 4KB, utilizando sitios de hasta 8 vecindarios, cada uno de ellos con un máximo de 250 peer regulares, con la posibilidad de que cada nodo se ocupe de mantener la información de 128 nodos, entonces el número máximo de nodos en un sitio es de 256 mil nodos, que ocuparán aproximadamente 20MB de espacio en disco. De utilizar un buffer de 256KB, esta información podría ser volcada a disco en 80 pasos. De la misma forma, 20 sitios (con 8 vecindarios por sitio, 250 peer regulares por

vecindario y 128 nodos virtuales por peer regular) serían unos 5 millones 120 mil nodos, que ocuparían 400MB en disco.

Normalmente, en lugar de la información individual de cada nodo, tiene más interés conocer el estado de los clústeres de ordenadores, por lo cual, el ejemplo descrito en el párrafo anterior constituye un caso extremo, muy difícil de encontrar en la práctica. Los casos reales necesitan menos espacio en disco, por lo que los cálculos para implementar el Hilo Volcado fueron hechos sobre la base de 100MB máximo de espacio en disco.

El componente de volcado lee fragmentos de información de la RHT y los escribe a un archivo, utilizando varios buffer y un mecanismo de señales que le permite planificar tantas operaciones como sean necesarias, para acomodar los datos en los búferes y luego escribirlos a disco. Una de las opciones que implementa el componente de volcado consiste en obtener la información de la RHT utilizando el orden en que esta ha sido indexada. De esta forma, los búferes pueden ser llenados con la información de los nodos, siguiendo la numeración de los sitios, y luego el orden lexicográfico de las direcciones de Internet de los nodos, y por último el orden de los nodos virtuales (los campos Sitio, PeerIP y HostIP de la **Figura 4-12**). Cada vez que el sistema operativo escribe a disco el contenido de uno de los búferes, se envía una señal (a nivel del sistema operativo, no un mensaje de señal del protocolo de monitorización) que es gestionada por un manejador de señales del componente de volcado, que se ocupa de mantener un puntero de posición para la RHT, lo que le permite llenar el buffer con más información y planificar una nueva operación de escritura asíncrona. Cuando se alcanza el final de la RHT, el archivo de volcado se renombra o se mueve a su ubicación final.

Aunque el Hilo Volcado es un componente opcional, que está enfocado, principalmente, a los procesos de validación y depuración del SMRD, es posible utilizar la información de monitorización almacenada en los archivos de volcado para alimentar a una aplicación externa. Por ejemplo, es posible utilizar un intérprete para extraer la información de los archivos de volcado, en lugar de utilizar un cliente para conectar con el servidor de IPC. Algunas pruebas hechas con esta configuración demostraron una eficiencia alta, en cuanto a la utilización de los recursos, porque en promedio los perfiles de utilización de memoria y CPU son semejantes en ambos casos (teniendo en cuenta la suma de los perfiles del Monitor Demonio y de la aplicación cliente). Sin embargo, la utilización del Hilo IPC para acceder a la información de la RHT tiene la ventaja de que produce datos más inmediatos que el Hilo Volcado, lo cual puede influir positivamente en la efectividad del sistema de monitorización.

Finalmente, ambos enfoques son muy valiosos cuando el objetivo final es combinar la información de monitorización en una herramienta de decisión más sofisticada, como un gestor de carga, un planificador de trabajos o un gestor de recursos. El Hilo Volcado puede ser utilizado para almacenar la información en un formato específico, que pueda ser interpretado directamente por la herramienta de decisión. Esto podría mejorar la eficiencia del proceso de decisión, por ejemplo la meta-planificación, sobre todo si no es necesario transferir los archivos por la red (debido a los retardos), y si la decisión requiere un paso de pre-procesamiento que depende de estos datos. En general, el pre-procesamiento consiste en organizar los datos de entrada utilizando

complicados algoritmos. El hecho de disponer de estos datos, en un formato apropiado, puede mejorar las prestaciones del proceso completo.

En cambio, el Hilo IPC puede mejorar las prestaciones y las comunicaciones, cuando es necesario transferir los datos por la red. En el caso de que la decisión dependa de una parte de los datos, y no de todos los datos, la parte relevante de la información se puede transferir entre las dos aplicaciones, y además, si las dos aplicaciones se encuentran en ordenadores diferentes, se pueden utilizar técnicas de cache o de sincronización que permitan utilizar la misma información para varias aplicaciones cliente.

Como nota adicional, al leer de forma rutinaria múltiples registros consecutivos de la RHT se obtienen prestaciones muy altas. Esto sucede porque la parte más costosa de una búsqueda en una base de datos Berkeley DB está en el acceso a disco, y el SMRD monta la RHT en la memoria de los peer regulares. Incluso, en el caso de utilizar una RHT en disco, se obtienen muy buenas prestaciones, debido a que la frecuencia de las operaciones de E/S a disco depende, fundamentalmente, del tamaño de página de la base de datos y del tamaño de la cache. En el caso de la RHT, ambas características pueden ser ajustadas con mucha precisión porque el tamaño de los registros se conoce de antemano. Sin embargo, cuando se trabaja en disco con una RHT grande (de más de 20MB), el Hilo IPC consigue mejores prestaciones que el Hilo Volcado. Esto se debe a que en el volcado, además de los indicadores dinámicos, se almacena la información estática (número de procesadores, memoria y swap instalada) en el archivo de volcado. Esto hace que la lectura sea más grande, pero además, la información estática no es tan homogénea como los indicadores dinámicos, lo cual aumenta la frecuencia de acceso a disco.

En conclusión, el Hilo IPC es la forma más eficiente de acceder a la información de la RHT, sobre todo si el acceso ocurre a través de la red o si la RHT es demasiado grande para almacenarla completamente en la memoria del ordenador. El Hilo Volcado proporciona una herramienta de validación y depuración muy útil, principalmente, para obtener registros periódicos del contenido de la RHT. Esta herramienta funciona mejor cuando la RHT cabe completamente en la memoria del ordenador.

5.3. GESTIÓN DE RECURSOS EN GRIDIFF

La sección anterior describe el sistema de monitorización de GRIDIFF. Esta sección explica cómo se organiza GRIDIFF para utilizar esta información como parte de una estrategia de selección de recursos basada en indicadores de QoS a nivel de servicio.

Uno de los objetivos principales del proyecto EGEE-III es realizar la transición de EGEE al modelo EGI. El modelo EGI, desarrollado en el marco de la acción de soporte "EGI Design Study" [70], ha dado lugar a varios documentos, entre los que destaca el Blueprint [71], donde define las actividades internacionales que debería implementar la EGI.

Dentro de EGEE, el área SA1 es responsable de las operaciones de la infraestructura del proyecto, y varias de las tareas de esta área están relacionadas con la monitorización de recursos. SA1 organiza esta actividad a partir de un repositorio central de

información de monitorización, la puesta a punto de un sistema de mensajería y la monitorización, a nivel regional y de NGI, de los sistemas Grid. La primera de estas tareas está definida en el EGI Blueprint como una necesidad para verificar y ajustar el proceso de selección de recursos para las aplicaciones, y para apoyar el balance general de utilización de los recursos de EGI. El objetivo principal de estas actividades consiste en utilizar Nagios [72] para monitorizar los sitios Grid a nivel regional, y enviar la información, a través de un sistema de intercambio de mensajes entre las diferentes plataformas, regiones y proyectos, a un repositorio central en EGI.org, sede de operaciones de EGI.

Aunque GRIDIFF no está comprometido directamente con EGEE, sus objetivos están orientados en la misma dirección que las tareas de monitorización de recursos de SA1, y en este sentido apuesta por un modelo de difusión de la información más eficiente, apoyándose en redes P2P.

El proyecto EGEE ha proporcionado el marco apropiado para identificar varios requerimientos necesarios para dar soporte a las operaciones de monitorización en los entornos de computación Grid de gran dimensión. El análisis de este contexto sugiere que cualquier esfuerzo para mejorar las infraestructuras de monitorización Grid tiene que mejorar la forma de transportar la información de monitorización [73]. En este sentido, nuestro aporte ha consistido en estudiar la utilización de redes P2P basadas en súper-peers para organizar la replicación de información en el Grid, utilizando difusión a grupos de multidifusión, creados sobre la red de superposición de la red P2P. En contraste con el modelo propuesto por EGEE, que se basa principalmente en un sistema de instancias independientes con un componente central que agrega los datos y coordina el trabajo de las instancias, el modelo de GRIDIFF es completamente distribuido y descentralizado. Esta metodología nos ha permitido desarrollar un sistema distribuido para transmitir la información de monitorización en el Grid. La eficiencia de este enfoque nos permitió luego desplegar una aplicación de replicación sobre la capa de transporte, cuyo aporte principal consiste en disminuir los tiempos de latencia en los diferentes servicios de monitorización del Grid. Esta reducción en los tiempos de acceso a la información de monitorización hace que sea posible tener una visión global, con suficiente precisión para que sea factible utilizar la misma en estrategias de meta-planificación y de gestión de recursos.

Como se verá en la sección de resultados, el SMRD es capaz de recoger información de los nodos y propagarla a todo el Grid. La organización jerárquica del Grid y las diferencias entre las redes que interconectan sus componentes, han sido siempre un obstáculo para la monitorización del Grid. El SMRD tiene como objetivo hacer de estas características una ventaja, y con ello conseguir una escalabilidad muy buena, utilizando intervalos de monitorización de 10 segundos (120 veces menos que algunos de los sistemas de gestión Grid actuales).

Para demostrar los beneficios potenciales que puede aportar GRIDIFF, esta sección describe una implementación completa de la arquitectura, que incluye los componentes necesarios para asignar recursos utilizando el algoritmo propuesto en la Sección 4.3. Este componente permite clasificar a los servicios Grid en grupos, donde los servicios de cada grupo tienen prestaciones y disponibilidades similares, y luego seleccionar a los

servicios, de un grupo u otro, sobre la base de criterios económicos y de la utilización de los recursos computacionales.

Luego, el resto del capítulo presenta un estudio desarrollado en un entorno Grid real, con el objetivo de evaluar la aplicabilidad de GRIDIFF a un problema complejo: gestionar los recursos de una VO, respondiendo a los cambios que se producen en los recursos y que alteran la capacidad de los servicios de mantener un nivel de servicio apropiado, y también respondiendo a los cambios en la carga y la disponibilidad de recursos computacionales de los sistemas.

5.3.1. COMPONENTES DE GRIDIFF

GRIDIFF necesita tener una importante interacción con el middleware Grid de los recursos. Entendemos por middleware un tipo de software que tiene como principal función la de conectar a otros componentes de software y aplicaciones entre sí. Generalmente, el middleware está asociado a sistemas de computación distribuida donde participa en la interconexión de las aplicaciones con el sistema operativo de cada sitio del sistema. Para esto, el middleware aporta el conjunto de servicios comunes que posibilitan todas o la mayoría de las tareas de propósito general en un sistema distribuido, como son componer diferentes componentes o aplicaciones para dar lugar a aplicaciones más complejas, reutilizar componentes y exportar aplicaciones a otros contextos.

Para lograr todo lo anterior, el middleware se basa en una capa de comunicación que garantiza la interoperabilidad de todas sus partes y de las partes añadidas por las aplicaciones que soporta. Las entidades que se comunican a través de la capa de comunicación que aporta un middleware determinado van a establecer diferentes relaciones entre sí para desempeñar diferentes roles, como cliente-servidor o peer-to-peer, utilizando diferentes modos de interacción embebidos en diferentes patrones, como paso de mensajes asíncronos, llamadas a procedimientos remotos, mensajes síncronos, etc.

La arquitectura de GRIDIFF aporta una capa adicional de gestión de recursos sobre el middleware Grid. Esta capa está formada por un sistema de componentes que utilizan, principalmente, la capa de comunicación del middleware Grid subyacente para interactuar entre sí. Adicionalmente, GRIDIFF extiende la capa de comunicación con algunos componentes propios que posibilitan la comunicación con otros procesos que no utilizan el middleware Grid. Este es el caso del SMRD, que se apoya en una red P2P para comunicarse. GRIDIFF incluye componentes que permiten comunicarse con el SMRD.

Los servicios estructurales son fundamentales en la arquitectura de GRIDIFF, porque proporcionan los métodos que permiten crear diferentes objetos de la arquitectura e interconectarlos entre sí para dar lugar a los componentes. Dependiendo de la configuración, los servicios estructurales cargan diferentes dependencias e introducen las referencias en los objetos que van creando, dando forma al núcleo de componentes que se encarga de proporcionar las principales funcionalidades de la arquitectura, como el contenedor de IoC (para más detalles ver la Sección A.1.1.B del Anexo 1), la façade,

el planificador, etc. Una vez creado este núcleo, los servicios estructurales inician los sistemas de cada componente y ceden el control a los mismos.

Las figuras **Figura 5-19**, **Figura 5-20**, **Figura 5-21**, **Figura 5-22**, **Figura 5-23** y **Figura 5-24** representan los 6 pasos por los que transita GRIDIFF desde que un servidor de aplicaciones, o una aplicación independiente, crea cualquiera de los servicios estructurales *AdapterServiceStrut* o *PersisterServiceStrut*, hasta que GRIDIFF está completamente activo. Los servicios estructurales contienen los métodos necesarios para crear el contenedor de dependencias (*Container*), que se encarga de crear a los componentes de GRIDIFF, inyectando en cada caso las dependencias necesarias. Las figuras muestran la arquitectura de GRIDIFF organizada en cuatro capas: presentación, servicios, componentes y recursos. De esta forma, la capa lógica y la de persistencia del modelo de capas se fusionan, y a la vez se dividen en tres capas: servicios, componentes y recursos. Esta representación mejora la visualización de los componentes y sus relaciones, en las figuras.

La **Figura 5-19** presenta el primer estado: la creación del conjunto de componentes previos a la activación del Sistema de Gestión de Servicios (SGS) y del Repositorio Activo de Información de Servicios y Configuración (RAISC).

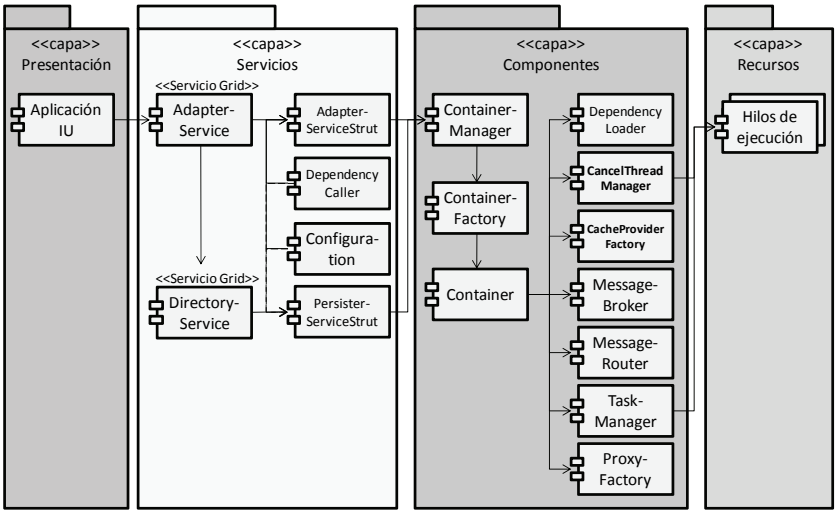


Figura 5-19: GRIDIFF: creación del conjunto de componentes previos a la activación del SGS y del RAISC.

La parte izquierda de la **Figura 5-19** muestra una aplicación que accede al Grid a través de GRIDIFF. El servicio Grid *AdapterService* publica las operaciones públicas de *AdapterServiceStrut*, un componente de la capa de servicios que proporciona todos los métodos para acceder al Grid a través de GRIDIFF. *AdapterServiceStrut*, en conjunto con *PersisterServiceStrut*, forman la API de GRIDIFF, es decir, proporcionan los métodos públicamente accesibles que pueden ser utilizados, de forma programática,

para interactuar con GRIDIFF. Por su parte, *PersisterServiceStrut* proporciona todos los métodos para acceder a la información histórica de los servicios Grid. Los servicios Grid *AdapterService* y *DirectoryService* proporcionan una implementación específica ligada a un middleware Grid.

AdapterServiceStrut y *PersisterServiceStrut* utilizan el componente *ContainerManager* para crear todos los componentes de GRIDIFF, y suscribirlos al servicio de directorio del middleware Grid subyacente. *ContainerManager* recibe las directivas de configuración en un objeto *Configuration* y, opcionalmente, puede recibir un objeto *DependencyCaller* que permite cargar dependencias en GRIDIFF, utilizando llamadas específicas al middleware Grid. *ContainerManager* utiliza *ContainerFactory*, una factoría que implementa las operaciones para crear y suscribir todos los componentes de GRIDIFF. Luego de implementados, GRIDIFF mantiene dos referencias a los componentes: una en el objeto *Container*, y la segunda en el servicio de directorio. *ContainerManager* proporciona la operación *getCurrentContainer*, que permite acceder al objeto *Container* para utilizar los componentes, y una operación *addNewOptionalComponent*, que permite adicionar nuevos componentes una vez creado el contenedor. Adicionalmente, *ContextHelper* es un componente que proporciona un conjunto de métodos que permiten buscar y recuperar componentes de GRIDIFF del servicio de directorio, utilizando para ello una interfaz JNDI (Java Naming and Directory Interface).

Los componentes se mantienen vinculados al objeto *Container* y al servicio de directorio por el tiempo de vida de GRIDIFF, a no ser que el *Container* sea destruido, en cuyo caso también se destruyen los componentes. Adicionalmente, un componente puede ser:

- Derogable: Un componente es derogable si puede ser destruido o privado de recursos en tiempo de ejecución, para lo cual debe proporcionar un método *depose* que pueda ser llamado desde el *Container* para liberar los recursos del componente de forma correcta. Por ejemplo, los objetos *Proxy* proporcionan una interfaz que permite trabajar, indistintamente, con componentes locales y remotos. Sin embargo, cuando se trabaja con componentes remotos, el establecimiento de la conexión incurre en varias operaciones costosas que se pueden evitar si se utilizan conexiones persistentes, que luego pueden ser reutilizadas como parte de llamadas posteriores al mismo componente. Para evitar que el control de las conexiones quede fuera del alcance del objeto *Container*, los objetos *Proxy* implementan el método *depose* que libera las conexiones, para garantizar que, por ejemplo, se pueda hacer una reconfiguración del sistema en tiempo de ejecución, o simplemente para asegurar que se puedan cerrar todas las conexiones en un momento dado, como respuesta a un evento determinado.
- No vinculante: Algunos componentes de tipo factoría, vinculan los componentes que crean con el servicio de directorio. A diferencia del *ContainerFactory*, que vincula a los componentes que crea de forma tal que los vínculos persisten durante todo el tiempo de vida del *Container*, los componentes no vinculantes crean componentes que pueden ser desvinculados en tiempo de ejecución, para lo cual deben proporcionar un

método *unlinkResources* que pueda ser llamado desde el *Container* para desvincular los recursos del componente de forma correcta. Generalmente, los componentes creados con una factoría no vinculante son derogables. Por ejemplo, *ProxyFactory* es un objeto de tipo factoría que gestiona la creación de objetos *Proxy*. Para asegurar que los mismos puedan ser reutilizados por cualquier componente de la arquitectura, *ProxyFactory* vincula los objetos que crea con el servicio de directorio, de forma tal que un componente que necesite un servicio proxy puede buscar entre los objetos *Proxy* disponibles en el directorio y recuperar una referencia a cualquiera de ellos. Para garantizar que no sólo las conexiones de los objetos *Proxy* sean destruidas, sino que los objetos mismos sean destruidos y que no quede ninguna referencia en el servicio de directorio, *ProxyFactory* implementa el método *unlinkResources* para asegurar que el objeto *Container* pueda gestionar centralmente la persistencia de los objetos *Proxy*.

Los componentes previos a la activación del SGS y del RAISC son los siguientes:

- *DependencyLoader*: Proporciona los métodos declarativos para cargar componentes y sus dependencias, similares a los proporcionados por el framework de Spring: cada componente es asociado con un conjunto de dependencias a través de un archivo de configuración, que especifica el orden en que las dependencias deben ser creadas e inyectadas al componente. Opcionalmente, puede utilizar un objeto *DependencyCaller* para cargar dependencias específicas del middleware Grid.
- *CancelableThreadManager*: Proporciona un entorno de gestión de hilos para ejecutar, de forma asíncrona, varias tareas de corta duración que pueden ser canceladas como consecuencia de un mecanismo de cancelación, con un tiempo de espera definido por el usuario. Esta clase gestiona hilos de forma explícita, y la especificación de J2EE prohíbe la gestión de hilos a nivel de componentes. En consecuencia, en entornos de computación J2EE, esta clase utiliza un componente definido por la especificación de J2EE (*Work Manager*) para realizar trabajos concurrentes con hilos múltiples. *CancelableThreadManager* es derogable. Por ejemplo, la clase *MDSClient* proporciona un cliente que permite consultar al WS-MDS de GT4. Se pueden dar varias condiciones que ralentizan la comunicación con el servicio WS-MDS, por ejemplo la sobresaturación del servicio, problemas de conexión, etc. Como regla general, la API de conectividad de GT4 no proporciona los medios para especificar el tiempo de espera después del cual se considera que un intento de establecer una conexión ha fallado. Para solucionar este problema, la clase *MDSClient* implementa las consultas al WS-MDS dentro de una *CancelableAsyncTask*, que es una tarea asíncrona que se ejecuta en un hilo diferente del hilo principal, y que es proporcionado por *CancelableThreadManager*, que además se encarga de monitorizar la ejecución de la tarea y de cancelar la misma, en caso de que se agote su plazo de ejecución.
- *CacheProviderFactory*: Proporciona una especificación y una factoría para crear

controladores de almacenamiento en cache. Los controladores de almacenamiento en cache proporcionan métodos para almacenar objetos arbitrarios en cache, y opcionalmente pueden implementar características ligadas con requerimientos específicos, como son almacenamiento persistente en disco y soporte para clústeres de cache. El prototipo descrito al final de este capítulo se basa en OSCache [74] para implementar los controladores de almacenamiento en cache.

- *MessageBroker*: Actúa como intermediario, traduciendo los mensajes del emisor al protocolo de mensajes del receptor, en una comunicación entre dos componentes de GRIDIFF. Esta clase implementa métodos para conectar componentes que se comunican mediante el intercambio de mensajes que han sido formalmente definidos en esquemas XML. Su función principal es la de mantener la compatibilidad entre versiones, así como permitir la interoperabilidad entre implementaciones diferentes. El prototipo descrito al final de este capítulo se basa en XMLBeans [75] para hacer todo el trabajo con XML, y en XStream [76] para serializar los objetos que van a ser transmitidos por la red.
- *MessageRouter*: Determina qué componente o servicio debe gestionar un mensaje (generalmente una notificación). Se utiliza en combinación con un componente *Forwarder*, principalmente, para redirigir los mensajes a un nuevo destino, en caso de ocurrir una descoordinación o un fallo de comunicación entre componentes.
- *TaskManager*: Proporciona un gestor de trabajos muy simple, que puede ser utilizado para enviar trabajos a un servicio Grid, siempre que se conozca de antemano el *endpoint* del servicio. Es una interfaz simple para el lanzamiento de trabajos, que tiene como función principal la de gestionar los trabajos en los nodos que no tienen acceso a un meta-planificador.
- *ProxyFactory*: Proporciona métodos de factoría para crear servicios proxy. Un servicio proxy interconecta a dos componentes de GRIDIFF, que pueden coexistir localmente en la memoria del mismo ordenador o que pueden estar en ordenadores remotos. Este componente es no vinculante, mientras que los objetos *Proxy* son derogables.

La **Figura 5-20** presenta el segundo estado: la creación del conjunto de componentes del Repositorio Activo de Información de Servicios y Configuración (RAISC).

Los componentes del RAISC son los siguientes:

- *DbContextFactory*: Proporciona métodos de factoría para crear objetos *DbContext*, que proporcionan contextos para el trabajo con bases de datos. Los objetos *DbContext* proporcionan un contexto para establecer conexiones a bases de datos, y ejecutar operaciones con los datos. Este contexto proporciona ventajas similares a las que se consiguen con un framework de persistencia, específicamente soporta agrupaciones de conexiones y sentencias pre-elaboradas en cache, para acceder a las bases de datos. Estas dos

características se construyen sobre la versión 3 del Java Database Connectivity (JDBC), de tal forma que, una vez ejecutada una sentencia, todas las conexiones tendrán acceso a su forma optimizada almacenada en cache. De esta forma, *DbContext* proporciona una forma simple de conectividad a bases de datos, con las principales ventajas de un framework de persistencia. *DbContextFactory* es derogable. El prototipo descrito al final de este capítulo se basa en el conector JDBC de PostgreSQL [77] para implementar los contextos para el trabajo con bases de datos.

- *Forwarder*: Reenvía los mensajes de notificación a los componentes correctos. Este componente es utilizado, principalmente, para redirigir las notificaciones por un nuevo camino después de que se produce un intento fallido de enviar la información al repositorio primario. En tal caso, esta clase almacena la notificación fallida en una cola, y planifica un nuevo intento de envío. Si el nuevo intento de enviar la notificación produce una excepción, *Forwarder* busca una ruta alternativa que le permita enviar la notificación al repositorio primario a través de otro servicio, preferiblemente ubicado en una red diferente. Si una vez agotado el tiempo de vida de la notificación, *Forwarder* no ha conseguido enviarla satisfactoriamente, entonces descartará el mensaje. Este componente utiliza a *MessageRouter* y a *MessageBroker* para encaminar los mensajes e interpretar los mensajes de respuesta, respectivamente.
- *ArRepositoryFactory*: Proporciona métodos de factoría para crear objetos *ArRepository*, que proporcionan repositorios para el RAISC. Por su parte, el *ArRepository* proporciona los métodos para acceder a los recursos del RAISC. Este componente es no vinculante, mientras que *ArRepository* es derogable.

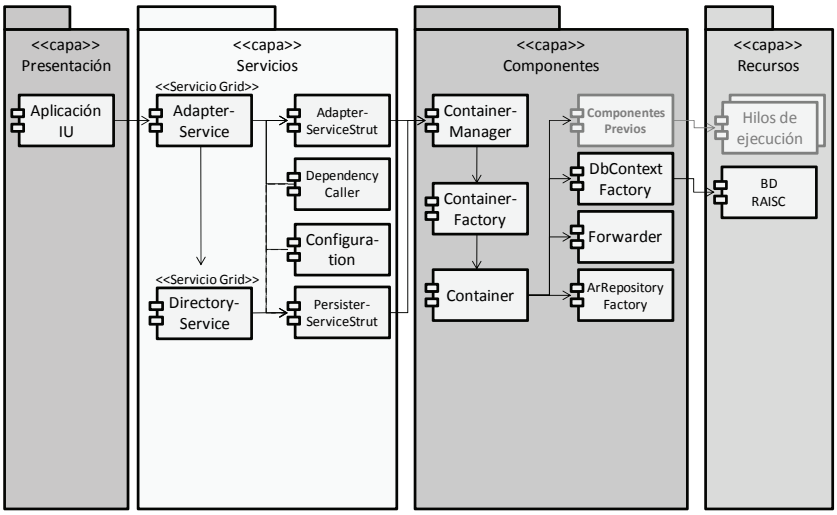


Figura 5-20: GRIDIFF: creación del conjunto de componentes del RAISC.

La **Figura 5-21** presenta el tercer estado: la creación del conjunto de componentes del Sistema de Gestión de Servicios (SGS).

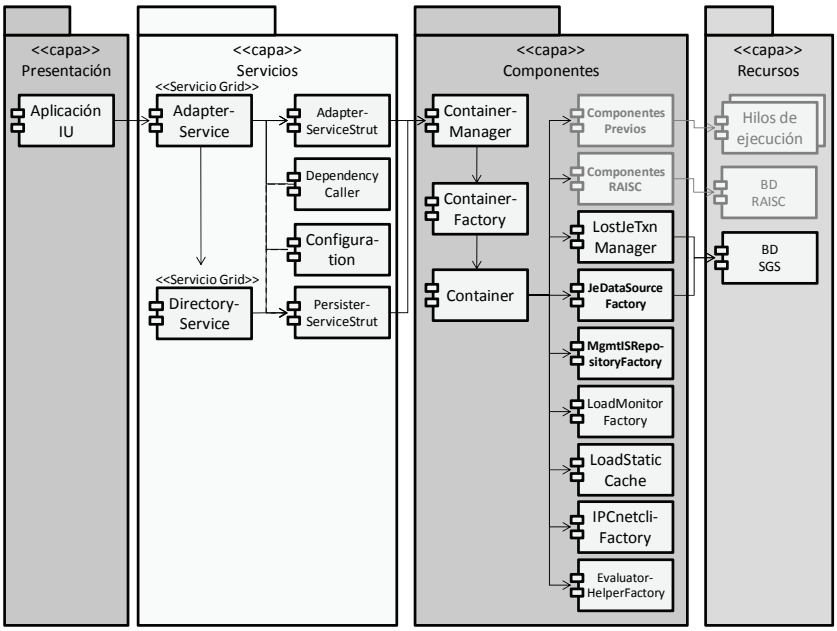


Figura 5-21: GRIDIFF: creación del conjunto de componentes del SGS.

Los componentes del SGS son los siguientes:

- *LostJeTxnManager*: Proporciona un gestor de transacciones perdidas para la base de datos Berkeley DB [78] – Edición de Java (JE). Las transacciones perdidas son operaciones transaccionales que no pudieron ser completadas en un momento dado. El gestor de transacciones perdidas almacena las transacciones perdidas en una cola y planifica una nueva ejecución en el futuro. Luego de un número de reintentos, determinado por el tipo de transacción, las transacciones perdidas pueden completarse o no. *LostJeTxnManager* no proporciona ningún mecanismo de notificación que permita avisar al cliente cuando una transacción perdida es finalmente descartada. De ahí que, cada componente es responsable de considerar si al ejecutar una transacción en la JE va a utilizar el gestor de transacciones perdidas, o no, para gestionar las transacciones fallidas. Como regla general, solamente las transacciones que contengan información temporal (por ejemplo, las notificaciones, las evaluaciones periódicas, etc.) serán tratadas como transacciones perdidas. La suscripción de servicios Grid al SGS nunca se trata como transacción perdida, para asegurar que en todo caso se pueda

dar una respuesta al servicio que solicita la suscripción. Este componente agiliza las operaciones con la JE, a la vez que mantiene el carácter transaccional de las mismas.

- *JeDataSourceFactory*: Proporciona métodos de factoría para crear objetos *JeDataSource*, que proporcionan métodos para el acceso a datos en un entorno JE. Estos objetos, de tipo *DataSource*, constituyen la representación de una fuente de datos en el lenguaje de programación Java. Este componente es derogable.
- *MgmtISRepositoryFactory*: Proporciona métodos de factoría para crear objetos *MgmtISRepository*, que proporcionan repositorios para el SGS. Por su parte, el *MgmtISRepository* proporciona los métodos para acceder a los recursos del SGS. Este componente es no vinculante, mientras que *MgmtISRepository* es derogable.
- *LoadMonitorFactory*: Proporciona métodos de factoría para crear objetos *LoadMonitor*, que proporcionan un monitor de recursos simple que permite recoger la información de carga de CPU y utilización de memoria física y swap del sistema local. Este componente proporciona un sistema de monitorización alternativo, muy inferior al que proporciona el SMRD. En primer lugar, la máquina virtual de Java aísla a las aplicaciones de la plataforma nativa, por lo que no es posible utilizar llamadas al sistema para conocer el estado de los recursos locales. En su lugar, *LoadMonitor* utiliza SNMP para comunicarse con una sonda local, con el consiguiente sobrecoste que esto lleva asociado. Además, *LoadMonitor* utiliza el sistema de información del Grid para propagar los indicadores dinámicos. De aquí que, su principal función es la de servir como respaldo en caso de que por alguna razón, el nodo local se desconecte del SMRD. En tal caso, GRIDIFF puede utilizar un objeto *LoadMonitor* para conocer el estado del nodo y proporcionar, al menos, la información del nodo local al algoritmo de selección de recursos.
- *LoadStaticCache*: Mantiene la información de los indicadores dinámicos de los recursos. Este componente funciona como un cache estático, en el sentido de que los registros son actualizados cada cierto tiempo, y no cada vez que se produce un cambio en el valor de los indicadores, como sucedería con un cache dinámico. *PeerMonitorJob* es una tarea programada, gestionada por el planificador de GRIDIFF, que interroga periódicamente al SMRD y actualiza la información almacenada por *LoadStaticCache*.
- *IPCnetcliFactory*: Proporciona métodos de factoría para crear objetos *IPCnetcli*. Los objetos *IPCnetcli* son clientes derogables que proporcionan métodos para comunicarse con el SMRD a través de un socket IP.
- *EvaluatorHelperFactory*: Proporciona métodos de factoría para crear objetos *EvaluatorHelper*, que proporcionan métodos utilizados por el Agente para la Evaluación del Nivel de Servicio (AENS) para evaluar el estado de los indicadores estáticos y dinámicos. El prototipo descrito al final de este capítulo se basa en Weka [79] para implementar los algoritmos de evaluación.

La **Figura 5-22** presenta el cuarto estado: la creación del conjunto de componentes posteriores a la activación del SGS y del RAISC.

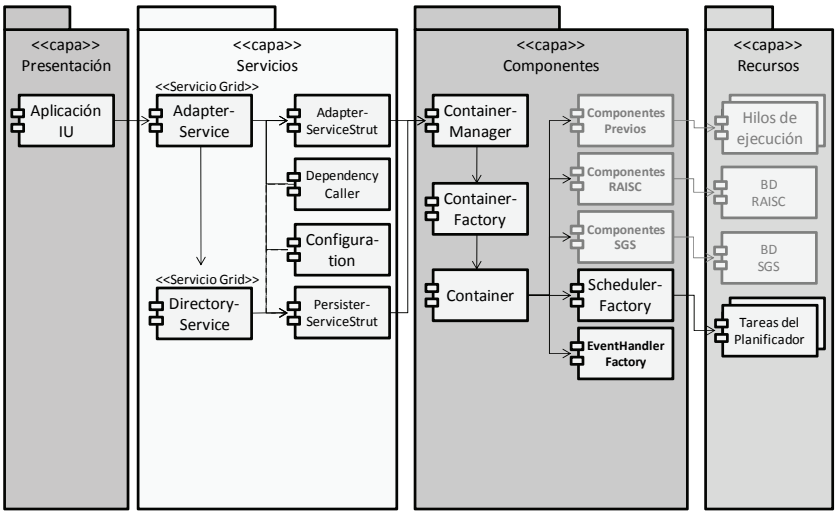


Figura 5-22: GRIDIFF: creación del conjunto de componentes posteriores a la activación del SGS y del RAISC.

Los componentes posteriores a la activación del SGS y del RAISC son los siguientes:

- *SchedulerFactory*: Proporciona métodos de factoría para crear objetos *Scheduler*, que proporcionan un planificador basado en eventos. Este planificador constituye el núcleo de todas las operaciones y actividades de GRIDIFF, ya que se encarga de gestionar las tareas relacionadas con la monitorización de los servicios Grid suscritos a GRIDIFF, así como de mantener la información del sistema, planificar las operaciones de sincronización de los sistemas SGS y RAISC, recuperar los indicadores dinámicos del SMRD, planificar el reenvío de notificaciones fallidas, planificar la ejecución de transacciones perdidas, etc. *Scheduler* recibe, en el momento de su creación, un conjunto de tareas, descritas en un archivo de configuración XML, que luego irá alternando según un cronograma que cambia continuamente, como consecuencia de los eventos que se generan por la ejecución de las tareas en el sistema. *Scheduler* utiliza un grupo de hilos de ejecución, con un número de hilos definido, que se crean con el planificador. Los hilos son reutilizados en cada ejecución, cambiando solamente la función objetivo del hilo. Adicionalmente, *Scheduler* soporta la ejecución concurrente de varias tareas del mismo tipo, para lo cual define varios contextos que se comparten entre las tareas. Cada tarea puede almacenar y recuperar información en los diferentes contextos, además del contexto privado de cada tarea, el contexto de género permite compartir

información con todas las tareas del mismo tipo, y el contexto de planificación permite compartir información con todas las tareas del planificador.

- *EventHandlerFactory*: Proporciona métodos de factoría para crear objetos *EventHandler*, que se ocupan de interpretar las notificaciones contenidas en los mensajes que llegan al nodo, y de ejecutar las acciones necesarias para responder a los eventos que son comunicados a través de estos mensajes. Los eventos locales y remotos que van llegando al nodo se almacenan en una única cola hasta que el *EventHandler* procesa los eventos locales y propaga los eventos remotos a los consumidores a los que van dirigidos.

La **Figura 5-23** presenta el quinto estado: la creación de los componentes de la "façade".

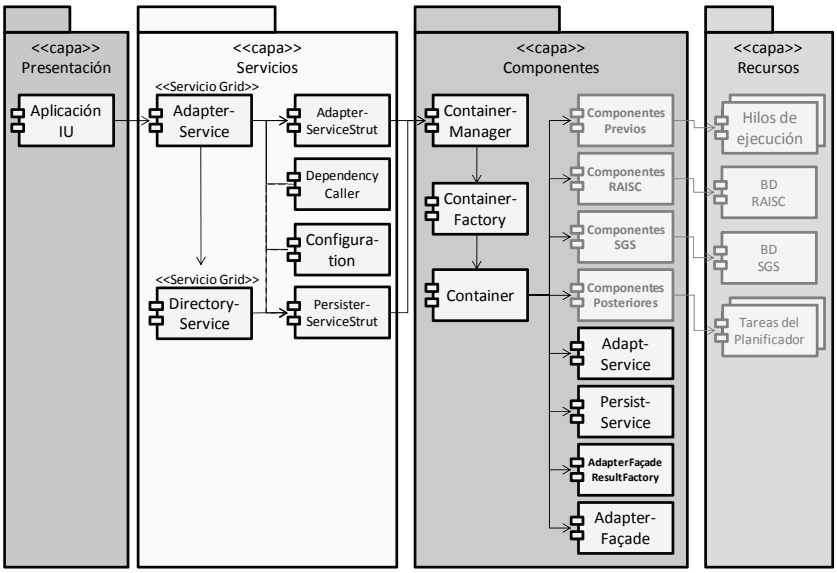


Figura 5-23: GRIDIFF: creación del conjunto de componentes de la "façade".

Los componentes de la "façade" son los siguientes:

- *AdaptService*: Proporciona los métodos para acceder a los recursos del SGS a través de la "façade".
- *PersistService*: Proporciona los métodos para acceder a los recursos del RAISC a través de la "façade".
- *AdapterFacadeResultFactory*: Proporciona métodos de factoría para crear objetos *AdapterFacadeResult*, que son objetos desconectados de la capa de persistencia.

La especificación de Java recomienda el uso de DTO (Data Transfer Objects) como modelo de transferencia entre la "façade" y el exterior. Sin embargo, al utilizar el Modelo de Dominio (para más detalles ver la Sección A.1.1.B del Anexo 1), los DTO son sustituidos por objetos del dominio, es decir, los objetos *AdapterFacadeResult* son objetos de valor, en lugar de DTO.

- *AdapterFacade*: Este componente implementa el patrón de diseño "façade", y permite acceder a todas las operaciones públicas de GRIDIFF a través de los servicios (del Modelo de Dominio) *AdaptService* y *PersistService*. También se ocupa de preparar los objetos que devuelven estos servicios. Por ejemplo, si los objetos están vinculados a una fuente de datos, *AdapterFacade* se ocupa de crear objetos del tipo *AdapterFacadeResult*, utilizando para ello *AdapterFacadeResultFactory*, y además, se ocupa de serializarlos en el caso de que vayan a ser transmitidos por la red. Por último, *AdapterFacade* se ocupa también de interpretar, preparar y transmitir la información necesaria para los objetos participen en operaciones sincronizadas. Por ejemplo, *AdapterFacade* se puede combinar con un sistema de gestión de transacciones para coordinar transacciones entre diferentes componentes de GRIDIFF.

Finalmente, la **Figura 5-24** presenta el sexto y último estado: la activación de GRIDIFF.

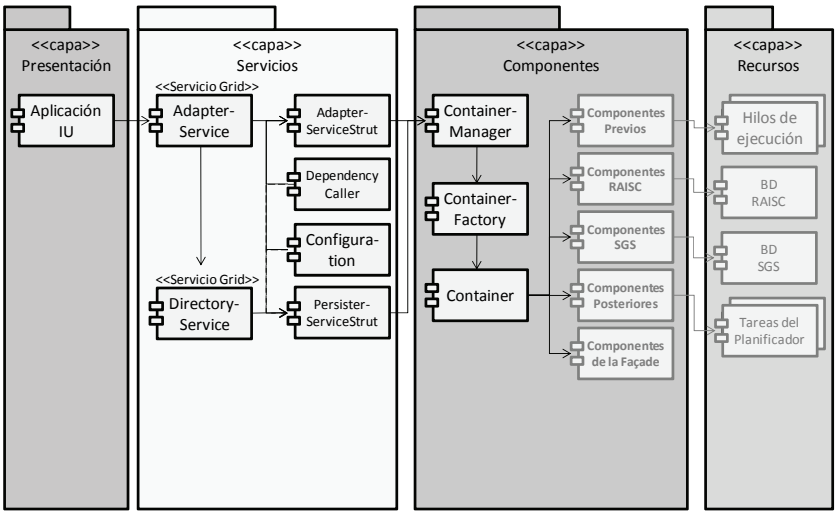


Figura 5-24: GRIDIFF: activación.

Una vez que todos los componentes han sido creados, *ContainerManager* activa el planificador y devuelve una referencia del *Container* al servicio estructural que dio inicio al proceso de activación. En este punto, los sistemas de componentes principales han sido creados y GRIDIFF se encuentra listo para gestionar peticiones de las aplicaciones

cliente. Adicionalmente, GRIDIFF se apoya en una serie de componentes auxiliares, que proporcionan otras funcionalidades y que son creados y gestionados, bajo demanda, por *ContainerManager*.

5.4. CONCLUSIONES DEL CAPÍTULO

GRIDIFF es una arquitectura de software para la asignación de recursos en el Grid, en base a requerimientos de QoS. Además de las herramientas para la asignación de recursos, proporciona un sistema de monitorización de recursos distribuidos. Toda la arquitectura ha sido diseñada para interactuar con el middleware Grid disponible, así como con los meta-planificadores actuales.

Capítulo 6

6. VALIDACIÓN Y EVALUACIÓN DE LA ARQUITECTURA GRIDIFF

Este capítulo describe los procesos de validación y evaluación de la arquitectura GRIDIFF y el resultado de los mismos, así como los detalles del despliegue en un entorno Grid real. Por último se presenta un estudio de evaluación de la experiencia de usuario con datos obtenidos con un prototipo de uso libre.

6.1. PLATAFORMA DE PRUEBAS

Las siguientes secciones describen los resultados de un estudio realizado para explorar el comportamiento de GRIDIFF en un entorno Grid real, así como los resultados de simulaciones computacionales llevadas a cabo en un entorno que simula una red IP muy grande, con múltiples dominios. Para ello, se realizaron varios experimentos de validación con un prototipo basado en Globus. El entorno de trabajo consiste en una red de máquinas virtuales creadas con VMware [59] en un clúster de servidores Linux. Los servidores Linux están interconectados entre sí a través de una red Gigabit Ethernet. Un puente de red conecta a cada máquina virtual con Internet. Cada máquina virtual ejecuta Linux en un procesador virtual Opteron™ 2.4GHz, con 512MB de RAM. Estas máquinas forman parte de un sistema de computación Grid basado en Globus Toolkit 4 (GT4). Los recursos de este sistema están comprometidos con varias VO.

En este entorno, se realizaron cuatro tipos de experimentos diferentes: E1, E2, E3 y E4. El modelo de colas presentado en la Sección 3.3 nos proporciona la base teórica sobre la cual evaluar el cumplimiento de los objetivos de esta tesis. Por ello, los experimentos están enfocados a verificar que GRIDIFF contribuye a mejorar la tasa de salidas exitosas. El experimento E1 tiene como objetivo evaluar la capacidad del sistema para monitorizar los indicadores estáticos y dinámicos. El experimento E2 compara la efectividad del proceso de selección de recursos bajo la gestión de GRIDIFF con la selección de recursos en base al mejor esfuerzo. El experimento E3 evalúa la varianza del tiempo de ejecución (CTV), y compara los tiempos de respuesta obtenidos en una serie de experimentos con GRIDIFF, con los tiempos obtenidos con un enfoque del mejor esfuerzo. Finalmente, el experimento E4 evalúa la escalabilidad del sistema de monitorización. Este experimento utiliza un número adicional de recursos ubicados fuera del entorno de trabajo. Estos recursos tienen características similares a los anteriores, con la excepción de que las máquinas virtuales fueron creadas con Xen [80] y ejecutan Linux en un procesador virtual Intel® Xeon® 2.33GHz. El primer clúster de ordenadores está ubicado físicamente en la Universidad Politécnica de Valencia, en España, mientras que el segundo está ubicado en la Rede Nacional de Ensino e Pesquisa, en Brasil.

Aunque en la práctica es imposible obtener una instantánea precisa del estado de un entorno asíncrono y distribuido, la mayoría de las veces, los sistemas Grid se organizan en sitios, y los sitios son sincronizados por un gestor de recursos que coordina las

políticas de planificación y utilización de recursos, en estrecha vinculación con los sistemas locales de gestión de recursos. En consecuencia, esta forma particular de organización puede ser utilizada para propagar, de forma eficiente, la información de monitorización. Sobre esta base, el Sistema de Información de Proveedores (SIP) fue configurado para gestionar los indicadores estáticos a través del sistema de monitorización y descubrimiento (WS-MDS), mientras que en la gestión de los indicadores dinámicos participa el Sistema de Monitorización de Recursos Distribuido (SMRD), que recibe los indicadores dinámicos directamente de los nodos o, en algunos casos, del sistema de colas. El SMRD se despliega en los dos sitios, ubicando un peer regular por cada 5 nodos y dos súper-peer por cada clúster. Los nodos que despliegan el SMRD, además del middleware Grid, despliegan un Monitor Demonio. Adicionalmente, un nodo despliega un Monitor RRDtool y un Monitor Portal. Esta configuración general del SMRD varía en algunos experimentos, en cuyo caso será indicado.

En el caso de las simulaciones computacionales, el entorno de simulación se describe en la Sección 6.5.1.

6.2. EXPERIMENTO E1: VALIDACIÓN

El experimento E1 consiste en contabilizar el número de trabajos asignados a cada clúster de servicios, y el número de trabajos completados dentro de un plazo de tiempo determinado. En este experimento, un grupo de trabajos son enviados al Grid a través de GRIDIFF, siguiendo un orden de ejecución específico. Cada trabajo del grupo utiliza un conjunto diferente de imágenes de entrada para realizar un procesamiento de imágenes digitales, utilizando algoritmos conocidos. Para esto, los nodos de trabajo del Grid despliegan una aplicación que toma como argumento el nombre del conjunto de imágenes de entrada y aplica los algoritmos de procesamiento para obtener un conjunto de imágenes de salida, que se almacenan en un directorio local del nodo que ejecuta el trabajo. Todos los nodos de trabajo del Grid se consideran equivalentes en cuanto a funcionalidades, porque todos ellos producen las mismas imágenes de salida para el mismo conjunto de imágenes de entrada. Sin embargo, los nodos se diferencian en cuanto a sus propiedades no funcionales, como la disponibilidad de recursos computacionales en un momento dado de tiempo.

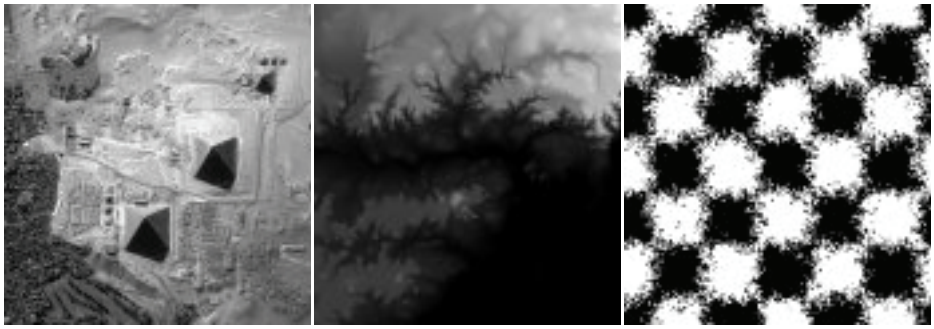
La **Figura 6-25** presenta una muestra de las imágenes que forman parte del conjunto de entrada.

Las imágenes del conjunto de entrada fueron obtenidas en Internet, y pueden ser utilizadas sin restricciones o derechos de autor. Además, han sido utilizadas, con anterioridad, para ejemplificar la utilización de algoritmos y técnicas de procesamiento de imágenes digitales [83].

Los algoritmos utilizados para procesar las imágenes son: algoritmo de crecimiento de regiones para segmentar una imagen en regiones, y algoritmo de reconocimiento de patrones con iteración aleatoria. En ambos casos se utilizaron implementaciones proporcionadas por Java Advanced Imaging (JAI) [84].

El algoritmo de segmentación utiliza una pila para almacenar la información de las regiones. En consecuencia, dependiendo de las características de la imagen, la cantidad

de memoria que utiliza este algoritmo puede ser significativamente grande. Por ejemplo, para una imagen del conjunto de entrada, que ocupa 3,2 MB en disco, el algoritmo necesita 92 MB de memoria para almacenar las regiones. Esta característica del algoritmo, hace que sus prestaciones varíen, considerablemente, con la cantidad de memoria física disponible en el ordenador. Por esta razón, este algoritmo es un candidato excelente para evaluar la eficiencia del proceso de selección de recursos, porque una selección descuidada ubicará los trabajos en nodos mal condicionados, mientras que una selección correcta utilizará solamente los nodos mejor condicionados.



Formato: JPEG	Formato: TIFF	Formato: PNG
Fuente: Archivo de imágenes del espacio [81]	Fuente: GTOPO30 [82]	Fuente: Colección de código, consejos y datos de JAI [83]
Tamaño en disco: 777,37 KB	Tamaño en disco: 1021,05 KB	Tamaño en disco: 6,99 KB
Dimensiones: 2000x2000 píxeles	Dimensiones: 439x595 píxeles	Dimensiones: 300x300 píxeles
Bits por pixel: 8	Bits por pixel: 32	Bits por pixel: 1

Figura 6-25: Ejemplos de las imágenes del conjunto de entrada.

Por otra parte, el algoritmo de reconocimiento de patrones no tiene requerimientos de memoria tan estrictos como el algoritmo de segmentación, pero hace un uso intensivo de la CPU.

El experimento E1 compara tres escenarios diferentes: el primero, donde la carga de los nodos de trabajos se mantiene, como mínimo, al 10% de su capacidad, el segundo, donde los nodos se mantienen, como mínimo, al 50%, y el tercero, donde los nodos se mantienen, como mínimo, al 80%. Utilizando stress [85], una herramienta que permite generar cargas de CPU, memoria y disco, se consigue un patrón de utilización de los recursos computacionales diferente en cada escenario. Una vez que las cargas de todos los nodos han alcanzado los valores definidos, se ejecuta un cliente que envía trabajos al Grid, siguiendo un orden determinado. Si un servicio fallase o no diese respuesta en un tiempo establecido, el cliente reenviaría el trabajo a un servicio diferente. El tiempo de reasignación también se considera parte del tiempo total de ejecución del trabajo. El

experimento E1 termina cuando todos los trabajos son completados con éxito. Se considera que el cliente tiene un presupuesto infinito, con lo cual la limitación para ubicar los trabajos está en el número de recursos disponibles.

La **Tabla 6-7** presenta una clasificación que diferencia a los servicios sobre la base de su disposición a proporcionar un cierto nivel de servicio. Esta disposición se define como una tendencia característica (determinada antes de enviar el trabajo), de un grupo de servicios Grid, a ejecutar un trabajo cumpliendo con unos niveles de disponibilidad y prestaciones determinados, y de disponer de un conjunto de recursos computacionales para ejecutar el trabajo, sin interferir con las funciones críticas del sistema, ni con otros trabajos que se pueden estar ejecutando, de forma concurrente, en el mismo nodo. Cada grupo es etiquetado con un tipo y un precio, y vinculado con un tipo de aplicación con unos requerimientos específicos. Sobre esta base, los usuarios deben crear sus solicitudes a fin de ajustar sus presupuestos con los precios ofrecidos por los proveedores de servicio.

Tabla 6-7: Una clasificación que diferencia a los servicios en grupos cualitativos.

Grupo	Precio	Descripción de los servicios	Más apropiados para
Primario	Costoso	Tienen los mejores niveles de disponibilidad y prestaciones, y por ello, la mayor probabilidad de estar ocupados con otros trabajos.	Solicitudes con plazos de tiempo precisos y otros requerimientos.
Respaldo	Asequible	Tienen niveles de disponibilidad y prestaciones convenientes, y una alta probabilidad de estar suficientemente libres de carga como para aceptar trabajos nuevos.	Solicitudes con requerimientos flexibles y bien definidos.
No garantizado	Rebajado	Es conocido que no cumplen con los requerimientos de disponibilidad y prestaciones convenidos por la VO, y por ello, tienen una gran probabilidad de estar mayormente libres de carga de trabajo.	Solicitudes sin requerimientos de tiempo.

La **Figura 6-26** muestra los resultados del experimento E1. Esta figura representa seis gráficos, uno para cada uno de los escenarios estudiados en el experimento. Cada gráfico muestra el número de trabajos asignados a cada grupo de servicios de la **Tabla 6-7**, y el número de trabajos completados con éxito.

Los resultados del experimento muestran una distribución favorable de la carga de trabajo en los diferentes grupos de servicios. En un entorno sin carga de red, cuando la carga de los servidores tiene valores bajos (10%), la mayoría de los trabajos son ubicados en los servicios de los grupos primario y de respaldo (82% del total), mientras que el 18% restante se ubica en el grupo de servicios no garantizados. En este escenario, los nodos mantienen su capacidad de procesamiento hasta que se saturan de trabajo, y entonces GRIDIFF comienza a ubicar los trabajos en los nodos que no cumplen con los requerimientos.

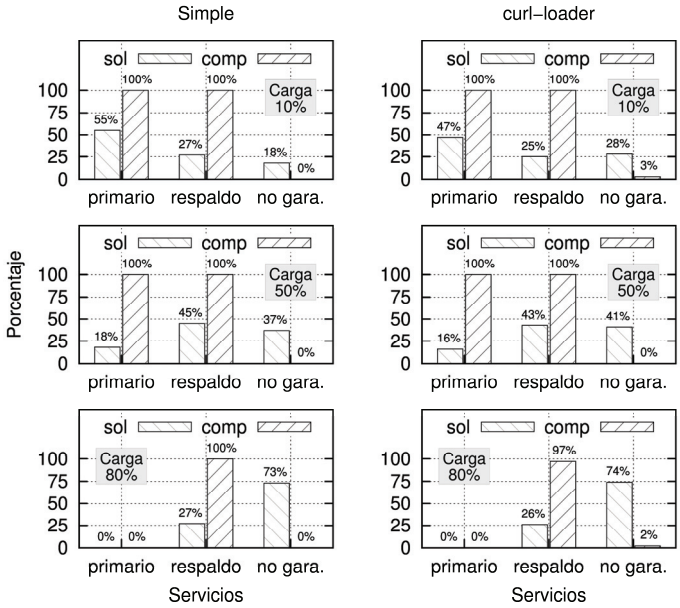


Figura 6-26: Número de solicitudes (sol) y número de trabajos completados (comp) dentro de los límites especificados de tiempo de ejecución, en el experimento E1. La nomenclatura de los grupos de servicios es la que se muestra en la **Tabla 6-7**: primario, respaldo y no garantizado (no gara). Los tres gráficos de la izquierda muestran los resultados obtenidos en un entorno simple, sin carga de red adicional, mientras que los tres gráficos de la derecha muestran los resultados obtenidos en un entorno con una carga de red muy grande.

El Anexo 4 muestra un ejemplo de los documentos XML de configuración de los servicios Grid, así como los documentos XML que acompañan una solicitud de trabajo. Este ejemplo es descrito en la Sección 6.6, pero los documentos son muy similares a los utilizados en el experimento E1. En la Sección A.4.1 de dicho anexo, se muestra un documento de requerimientos de servicios que contiene tres requerimientos. Los dos primeros comprueban la disponibilidad del servicio y el tiempo de ejecución de una prueba, y se ejecutan cada hora. El tercer requerimiento comprueba la seguridad del servicio, y se ejecuta con la frecuencia que decida el planificador de GRIDIFF. El requerimiento de disponibilidad tiene un límite inferior, que se cumple cuando el servicio tiene una disponibilidad que sobrepasa el 99%. El requerimiento de tiempo de ejecución tiene un límite superior, que se cumple cuando el servicio tarda menos de 10 segundos en completar una operación de prueba. Por último, el requerimiento de seguridad se cumple si al ejecutar una operación de prueba, ésta determina que el servicio tiene habilitado unas medidas de seguridad definidas por la VO.

Otro resultado relevante del experimento E1, que también se puede ser apreciado en la **Figura 6-26**, es que cuando aumenta la carga de los nodos a 50%, los trabajos ubicados en los servicios del grupo primario son menos que cuando la carga es del

10%. Luego, al aumentar la carga por encima del 80%, todos los trabajos se ubican en servicios del grupo de respaldo (27%) y, principalmente, del grupo de servicios no garantizados (73%). Por último, está el hecho de que todos los trabajos ubicados en servicios de los grupos primario y de respaldo terminan dentro de los límites de tiempo requeridos. En cambio, ninguno de los trabajos ubicados en los servicios del grupo de servicios no garantizados termina en tiempo. Estas observaciones nos permiten concluir que GRIDIFF se comporta de la forma esperada, diferenciando los servicios según sus capacidades, y respondiendo a los cambios dinámicos en la disponibilidad de los recursos computacionales del sistema.

Resultados similares fueron obtenidos en un experimento independiente en el que, además de simular la carga de trabajo en los nodos, se simulaba una carga muy grande de red utilizando curl-loader [86], un generador de carga que genera tráfico permanente de red, simulando miles de flujos de datos provenientes de diferentes clientes. Estos resultados, mostrados en los tres gráficos situados en la parte derecha de la **Figura 6-26**, demuestran que GRIDIFF también es capaz de gestionar la asignación de recursos en condiciones de mucha carga en la red, poniendo de manifiesto la estabilidad del sistema de monitorización, así como el hecho de que GRIDIFF no consume excesivos recursos. Sin embargo, es necesario destacar que en este caso se asignan menos trabajos a los clústeres primario y de respaldo. Por ejemplo, con la carga al 10% se asigna un 10% más de trabajos al clúster de servicios no garantizados. Además, algunos trabajos asignados a este clúster consiguen terminar en tiempo. Esto último se debe a que algunos servicios son clasificados erróneamente como no garantizados, como consecuencia de la sobrecarga de la red. No obstante, este hecho no es significativo porque el patrón de utilización de los recursos no presenta cambios significativos.

A modo de conclusión del experimento E1: en conjunto, los resultados de este experimento proporcionan una idea de los beneficios que GRIDIFF puede aportar, y demuestran la efectividad y la usabilidad del algoritmo de asignación de recursos. También demuestran que el sistema de monitorización puede trabajar, de forma concurrente, con los demás sistemas subyacentes, específicamente con el middleware Grid, sin alterar el desempeño de los servicios.

6.3. EXPERIMENTO E2: EFICIENCIA

El experimento E2 consiste en contabilizar el número de trabajos ejecutados en una franja de tiempo. A diferencia del experimento E1, donde los trabajos son enviados al Grid siguiendo un orden determinado, en este experimento los trabajos son enviados en orden aleatorio. Se utiliza una franja de tiempo lo suficientemente grande como para permitir la ejecución de todos los trabajos del conjunto de imágenes de entrada, y se repite el experimento 100 veces, registrando cada 5 minutos el número de trabajos completados. Durante el tiempo que dura el experimento se lanzan, en orden aleatorio, todos los trabajos del conjunto de entrada, y una vez que se han completado todos, se vuelven a relanzar nuevamente, repitiendo este proceso hasta que se agota el tiempo del experimento.

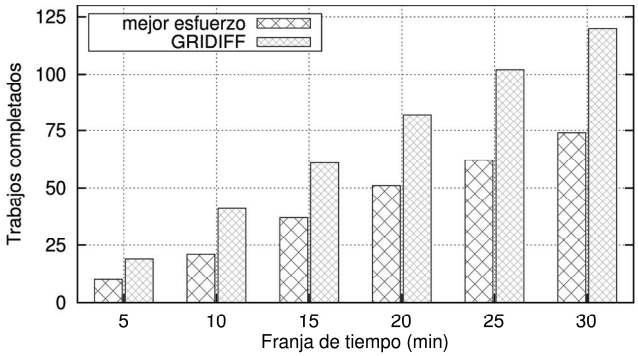


Figura 6-27: Número de trabajos completados en diferentes franjas de tiempo, en el experimento E2. Cada columna representa el número de trabajos completados con los dos enfoques, para uno de los intervalos de tiempo utilizados en E2. Por ejemplo, la primera columna muestra que en 5 minutos sólo se completaron 10 trabajos con la estrategia de mejor esfuerzo, mientras que se completaron 19 con GRIDIFF.

Este experimento compara dos enfoques diferentes para la selección de recursos: el primero basado en una estrategia de mejor esfuerzo, y el segundo que utiliza a GRIDIFF para seleccionar los recursos.

Un servicio de mejor esfuerzo no garantiza un nivel de servicio o una cierta prioridad en la ejecución de las solicitudes. De esta forma, todas las solicitudes reciben un nivel de servicio que depende, principalmente, de la carga del sistema, y por tanto, los tiempos de ejecución de las mismas son impredecibles. Un servicio de mejor esfuerzo no permite reservar recursos anticipadamente para la ejecución de una solicitud, como tampoco permite utilizar información que permita planificar globalmente la utilización de los recursos.

En consecuencia, el hecho de utilizar una estrategia de mejor esfuerzo en el experimento se refiere a que este enfoque no utiliza información de monitorización para seleccionar los recursos. En su lugar, mantiene una lista con los recursos y utiliza un algoritmo round-robin (RR) que asigna trabajos a los recursos en un orden circular, gestionando todos los trabajos con igual prioridad. Por otra parte, el enfoque asistido por GRIDIFF utiliza la información de los recursos para seleccionar los más apropiados para ejecutar un trabajo determinado en un momento dado. De esta forma, los trabajos son enviados a los servicios que mejor se ajustan a los requerimientos de los trabajos, y que tienen las mejores condiciones, en cuanto a disponibilidad de recursos computacionales se refiere, para aceptar un trabajo nuevo sin afectar al resto de los procesos que se ejecutan, de forma concurrente, en el nodo. Con el enfoque del mejor esfuerzo, estas garantías se relajan.

La **Figura 6-27** muestra un gráfico donde ha sido representado el número de trabajos ejecutados en el enfoque del mejor esfuerzo y en el enfoque asistido por GRIDIFF, para diferentes franjas de tiempo. Estos resultados muestran una mejora en la capacidad de respuesta y en el rendimiento del sistema, cuando se utiliza GRIDIFF. Comparado con el enfoque del mejor esfuerzo, GRIDIFF contribuye a mejorar la

eficiencia de utilización de los recursos Grid. Aunque el método del mejor esfuerzo no es difícil de mejorar, la utilización de este método para comparar la eficiencia de GRIDIFF tiene un propósito práctico, como se verá en la siguiente sección.

6.4. EXPERIMENTO E3: EFECTIVIDAD

El experimento E3 consiste en contabilizar el tiempo de ejecución de un conjunto de trabajos. Utilizando todas las imágenes del conjunto de entrada, se prepara una lista de ejecución con un orden estricto de los trabajos. Esta lista es utilizada para enviar los trabajos al Grid, empleando dos enfoques diferentes para seleccionar los recursos: mejor esfuerzo y GRIDIFF. Utilizando la misma lista, se reenvían 20 veces todos los trabajos. La **Figura 6-28** muestra un gráfico con los tiempos de ejecución de los dos enfoques.

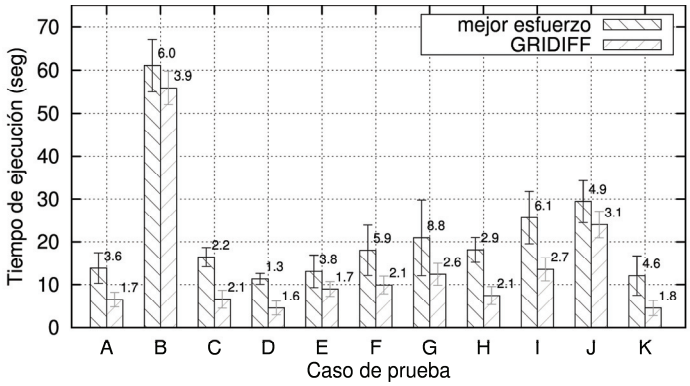


Figura 6-28: Media aritmética de los tiempos de ejecución en el experimento E3. Los valores vinculados a los puntos experimentales representan varianza del tiempo de ejecución (CTV) para las réplicas de un mismo trabajo.

Los resultados del gráfico de la **Figura 6-28** muestran una mejora en los tiempos de ejecución del enfoque asistido por GRIDIFF, en comparación con los tiempos del mejor esfuerzo. Sumando los tiempos de ejecución de cada trabajo para obtener el tiempo total de ejecución del conjunto de trabajos, se obtiene un resultado que muestra una mejora semejante.

Los trabajos que son planificados en un nodo de trabajo, pero que luego su ejecución falla son reasignados a otro nodo. Estos fallos se reflejan en la media aritmética del tiempo de ejecución y en la CTV, que son mayores cuanto más fallos se producen. Para algunos de los casos estudiados, el enfoque del mejor esfuerzo falla con frecuencia al hacer la asignación de recursos, y por ello tiene que reasignar los trabajos con más frecuencia que el enfoque asistido por GRIDIFF. Por ejemplo, el caso G utiliza un conjunto de imágenes que dificultan el proceso de crecimiento de regiones para la segmentación, porque requieren de una gran cantidad de memoria disponible en el nodo de trabajo. Con el enfoque del mejor esfuerzo es imposible seleccionar un nodo

que tenga esta cantidad de memoria disponible, y con frecuencia sucede que el trabajo es asignado a un nodo que tiene poca carga de trabajo, pero a la vez tiene poca memoria RAM disponible. Por otro lado, GRIDIFF encamina los trabajos hacia los nodos que tienen las mejores condiciones para cumplir con los requerimientos de cada trabajo, y por ello tiene menor latencia debido a la reasignación de recursos, lo que se traduce en una mejora en el tiempo de ejecución. Adicionalmente, la mejora encontrada en el tiempo de ejecución total, que es la suma del tiempo de ejecución más un sobrecoste asociado con el proceso de asignación de recursos y planificación de los trabajos, demuestra que el algoritmo de asignación de recursos no consume un tiempo excesivo.

Las varianzas representadas en la **Figura 6-28** muestran que, en la mayoría de los casos, el tiempo de ejecución del enfoque asistido por GRIDIFF está más cerca de la media aritmética que en el caso de los tiempo conseguidos con el mejor esfuerzo. Este resultado indica que GRIDIFF ofrece unos tiempos de respuesta más uniformes que el enfoque del mejor esfuerzo. Esto es particularmente importante para aquellas aplicaciones que requieren de algún nivel de predictibilidad, reproducibilidad y consistencia, como es el caso de algunos problemas de optimización, y sobre todo en problemas de simulación, además de intrínsecamente conducir a una menor tasa de fallos, evitando cancelaciones innecesarias de trabajos que se comportan sospechosamente lentos. Por ejemplo, para conseguir tiempos de ejecución semejantes, para tareas semejantes, cuando el dominio de datos de un problema se descompone en varios trabajos independientes. En este caso, si todos los trabajos son asignados a recursos con niveles de servicio semejantes, es probable que los tiempos de ejecución de todos los trabajos sean más homogéneos entre sí, que en el caso de asignarlos a recursos con niveles de servicio dispares.

Por último, utilizar un enfoque de mejor esfuerzo para comparar con GRIDIFF tiene connotaciones prácticas. En primer lugar, facilita la comparación de GRIDIFF con otros métodos descritos en la literatura. Por ejemplo, la **Figura 6-28** muestra que, en la mayoría de los casos estudiados, GRIDIFF consigue mejorar la CTV en un 20%, comparado con el enfoque del mejor esfuerzo. Este resultado puede ser utilizado para comparar el algoritmo de asignación con otros algoritmos disponibles. En segundo lugar, esta forma de sistematización de los resultados nos evita tener que desplegar otros sistemas de gestión de recursos, que a menudo es muy difícil (sino imposible) y que tiene tendencia a generar situaciones inesperadas, donde los resultados pueden ocultar errores relacionados con la configuración de los componentes. Por todas estas razones, los experimentos E2 y E3 utilizan un enfoque de mejor esfuerzo para evaluar la capacidad de GRIDIFF para gestionar recursos distribuidos en el Grid.

6.5. EXPERIMENTO E4: ESCALABILIDAD

El experimento E4 consiste en contabilizar el número de paquetes intercambiados entre dos nodos súper-peer del SMRD. Para ello se realiza un despliegue diferente al utilizado en los experimentos anteriores. Básicamente, el despliegue se diferencia en el número de nodos que utilizan el mismo súper-peer para enviar su información de monitorización en el Grid. En el experimento E4, este número varía entre 1 y 250 nodos.

En la actualidad, se conoce que las topologías de anillo tienen poca escalabilidad y robustez (en cuanto a tolerancia a fallos). Por esta razón, en el experimento E4 los súper-peer son configurados para difundir los mensajes a todos los nodos en su vecindario, aunque también sería igual (o mejor) de eficiente enviar los mensajes a grupos de multidifusión, en el caso que la red los soportara. Como explica la Sección 5.2.1, la transmisión de datos a través del anillo debe ser utilizada solamente cuando no es posible utilizar otra tecnología más eficiente para enviar, simultáneamente, el mismo mensaje a un grupo de nodos del vecindario local.

La **Figura 6-29** muestra cuatro gráficos que representan cuatro momentos diferentes en el estado de un nodo de trabajo del Grid. Los gráficos fueron trazados a partir de los datos de monitorización del SMRD, con un intervalo de monitorización de 10 segundos. De izquierda a derecha, y de arriba abajo, se puede observar el desplazamiento de dos frentes hacia la izquierda. Este desplazamiento pone en evidencia la utilización de los recursos del nodo como consecuencia de la ejecución de un trabajo.

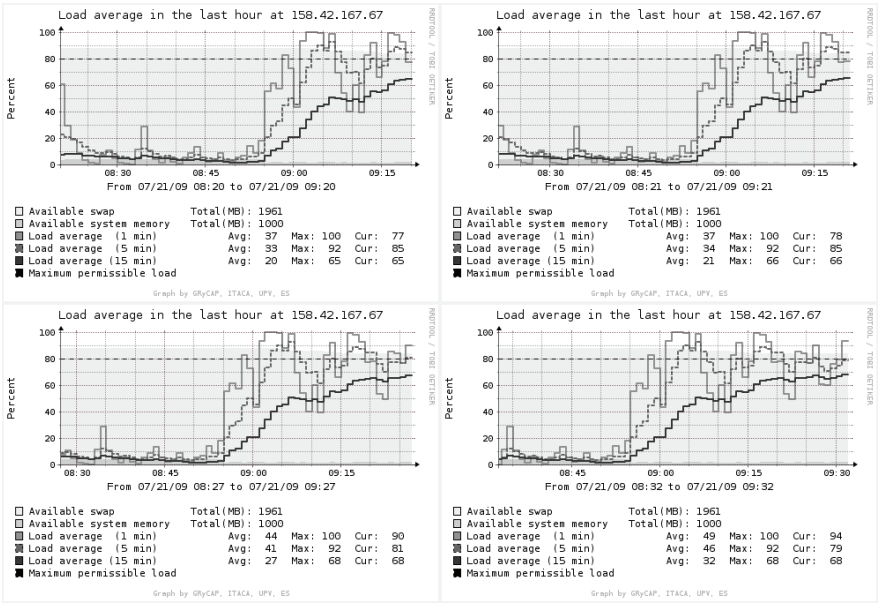


Figura 6-29: Un gráfico continuo del estado de un nodo de trabajo del Grid. De izquierda a derecha, y de arriba abajo, la figura muestra cuatro momentos diferentes en la monitorización de un nodo de trabajo del Grid. Cada uno de los gráficos representa una acumulación histórica de valores de los indicadores dinámicos, desde 1 hora antes del momento en que fue tomada la instantánea del sistema. Los gráficos fueron trazados a partir de los datos de monitorización del SMRD, utilizando la función *graph* de RRDtool [67]. En el momento de tomar las instantáneas, el nodo se encontraba ejecutando un trabajo.

La suavidad de las curvas del gráfico que representan la carga y la disponibilidad de memoria (física y swap) en el sistema, demuestran que un intervalo de monitorización de 10 segundos permite detectar, con suficiente claridad, los cambios en los indicadores dinámicos del sistema. Además, el retraso entre los gráficos trazados utilizando los datos de monitorización tomados, al mismo tiempo, en nuestro sitio en España y en el sitio remoto en Brasil, es de unos pocos segundos (en condiciones normales, entre 0 y 10 segundos). Esto demuestra que la visión global del estado del sistema es prácticamente la misma en todo el Grid, independientemente de la distancia que exista entre el nodo que toma los datos y el nodo que los interpreta, lo cual da una idea, bastante precisa, de inmediatez.

La **Figura 6-30** muestra un gráfico de la utilización de la CPU y la memoria física en un nodo de monitorización. Ni la utilización de la CPU, ni la disponibilidad de memoria (siempre que el tamaño de los datos no exceda las decenas de MB, en cuyo caso el SMRD utilizaría el disco rígido para almacenar la RHT) es un problema para la escalabilidad del sistema de monitorización. La mayor parte del tiempo, la utilización de la CPU está al 0%, con algunos picos aislados que no sobrepasan el 8%. La utilización de la memoria es muy regular, sin grandes cambios en el tiempo. Las tres aplicaciones (Monitor Demonio, Monitor RRDtool y Monitor Cliente) tienen perfiles de utilización de memoria física muy bajos, y en ningún momento llegan a utilizar memoria swap. Monitor Demonio, consume considerablemente más memoria que el resto de las aplicaciones, en total el consumo se mantiene entre 8 y 11 MB, lo cual es aceptable para los ordenadores modernos.

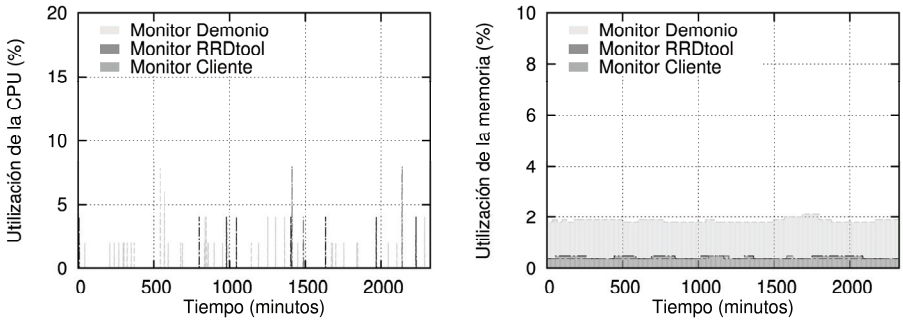


Figura 6-30: Utilización de la CPU y la memoria física. Los datos fueron tomados en un nodo que participa solamente en labores de monitorización (no ejecuta trabajos del Grid). El nodo tiene instaladas 512MB de memoria RAM, y despliega un Monitor Demonio con funciones de súper-peer y de peer regular, además despliega un Monitor RRDtool y un Monitor Cliente. El sistema consta de 251 nodos, y fue monitorizado continuamente durante más de 38 horas. Las anotaciones de la utilización de la CPU y la memoria física en el nodo de monitorización, fueron tomadas a intervalos de 1 minuto utilizando un script independiente del SMRD.

Durante el experimento E4, la utilización de la CPU del Monitor Demonio fue muy baja, entre 0 y 7,90 %, y el consumo de memoria se mantuvo en el rango de 7,1 a 11 MB, sin que existieran diferencias considerables entre las diferentes configuraciones

utilizadas en el experimento. Por otra parte, la utilización de la red varió en un intervalo considerable entre una configuración y otra.

La **Figura 6-31** muestra dos gráficos, el primero de ellos representa el número de paquetes intercambiados entre dos súper-peer distantes, y el segundo representa la suma total de bytes transmitidos por la red. Uno de los súper-peer está ubicado en nuestro sitio local, en España, mientras que el otro está ubicado en el sitio remoto, en Brasil. Cada sitio fue configurado con un único vecindario, y en cada vecindario un peer regular y un súper-peer se encargan de monitorizar y difundir la información de un grupo de nodos. El sitio en Brasil fue configurado con un único nodo, mientras que el sitio en España fue configurado con 1, 16, 32, 64, 128 y 250 nodos (considerando el caso más general donde hay un máximo de 250 nodos por subred, y por ello un máximo de 250 nodos por dominio de difusión). Los gráficos fueron trazados a partir de los datos obtenidos en nuestro sitio local, en España. Cada punto del gráfico representa la suma de los paquetes (y la suma de los tamaños de los paquetes, en el segundo gráfico) enviados desde el súper-peer local al súper-peer remoto y recibidos en el súper-peer local desde el súper-peer remoto, durante un intervalo de 10 minutos. En el momento de obtener los datos, el sistema se encontraba activo (los peer regulares y los súper-peer habían terminado los procesos de suscripción y se encontraban funcionales). El intervalo de monitorización es de 10 segundos.

Los datos fueron obtenidos con tcpdump [87], un analizador de paquetes que permite filtrar el tráfico entrante y saliente por varios criterios al mismo tiempo. Para colectar los datos del experimento E4, se utilizó un filtro que registra todo el tráfico TCP/IP y UDP/IP entre el súper-peer local y el remoto, en los puertos de red que utiliza el SMRD.

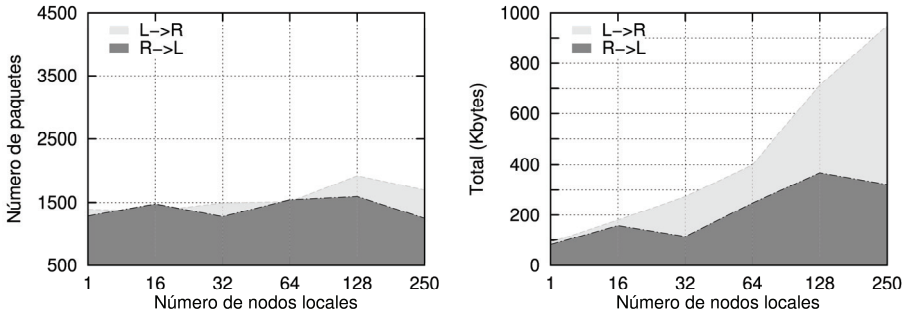


Figura 6-31: Número de paquetes intercambiados entre dos súper-peer distantes y la suma total de bytes transmitidos por la red. Cada punto del gráfico representa la suma de los paquetes (y la suma de los tamaños de los paquetes, en el segundo gráfico) enviados desde el súper-peer local al súper-peer remoto y recibidos en el súper-peer local desde el súper-peer remoto, durante un intervalo de 10 minutos. La dirección de los paquetes es: de local a remoto (L->R) y de remoto a local (R->L).

Los gráficos de la **Figura 6-31** muestran que la utilización de la red se incrementa con el aumento de los nodos. El total de datos transmitidos (en ambos sentidos) por la red, en un intervalo de 10 minutos, no supera 1,5MB. Todo el tráfico necesario para

conectar dos sitios remotos, con 251 nodos en total, está contenido en esta cantidad: la información de monitorización, los mensajes de pertenencia (o membrecía) de la red P2P, etc. Este resultado indica que el SMRD escala bien para un intervalo de monitorización de 10 segundos en una red de 251 nodos. Este resultado es perfectamente aplicable a un caso en el cual se quiera monitorizar una VO real, con muchos clústeres (del orden de hasta 200 clústeres) distribuidos en varios sitios. En tal caso, se podría utilizar una distribución de no más de 251 peer para difundir un estado general de cada clúster, que sería utilizado por el planificador para asignar un clúster determinado, en uno de los sitios. Luego, el sistema local de colas en el clúster se encargaría de asignar los trabajos a los nodos de trabajo del clúster.

Los gráficos también muestran que el número de paquetes transmitido entre los dos sitios es casi simétrico, aún cuando el número de nodos de los sitios es diferente. Es posible que este comportamiento desaparezca en redes de mayor tamaño, donde el número y el tamaño de los paquetes de monitorización son mucho mayores que el número y el tamaño de los paquetes transferidos en el experimento E4.

6.5.1. ESTUDIO DE LA ESCALABILIDAD EN UN ENTORNO DE SIMULACIÓN GRANDE

En la Sección 1.1, quedó definido que un algoritmo escala si éste puede ser aplicado de forma conveniente en problemas de gran dimensión. Teniendo esto en cuenta, creamos un entorno de simulación suficientemente grande para estudiar la escalabilidad del SMRD. Con este objetivo, utilizamos OMNet++ [88], una herramienta que permite simular redes de comunicación. Entre sus usos más frecuentes se encuentra la simulación de redes IP, apoyándose para ello en los modelos de simulación proporcionados por INET Framework [89].

Además de la herramienta de simulación y los modelos de red, es necesario contar con un escenario realista que simule una red con topología y características similares a Internet. Para ello utilizamos ReaSE [90], una herramienta que permite crear redes de topología definida, y añadirles servidores, clientes, tráfico, etc. Con la ayuda de esta herramienta creamos un modelo realístico de una red IP con múltiples dominios. A este modelo le añadimos diferentes canales de comunicación entre los diferentes Sistemas Autónomos o AS (acrónimo del inglés Autonomous System) que forman la red, para simular el efecto de la ubicación geográfica sobre la comunicación entre nodos situados en diferentes dominios. La **Figura 6-32** muestra la forma en que están interconectados los AS en la simulación. Esta forma de interconectarse los AS se conoce como topología a nivel de AS. Por otra parte, cada AS está formado por un conjunto de enrutadores que forman una topología interna, a la cual están conectados los servidores y los clientes.

Los AS que forman la red pueden ser de tránsito o TAS (acrónimo del inglés transit Autonomous System) y de frontera o SAS (acrónimo del inglés stub Autonomous System). Se consideran TAS todos aquellos AS que proporciona conexión a otras redes a través de sí mismos. Por ejemplo, si una red A puede utilizar una red B para conectarse a una red C, entonces B es un TAS. En Internet, cualquier proveedor de servicios de Internet o ISP (acrónimo del inglés Internet service provider) es un TAS.

Por otra parte, se consideran SAS todos aquellos AS que están conectados solamente con un único AS. En Internet, este tipo de AS se utiliza para desplegar redes privadas.

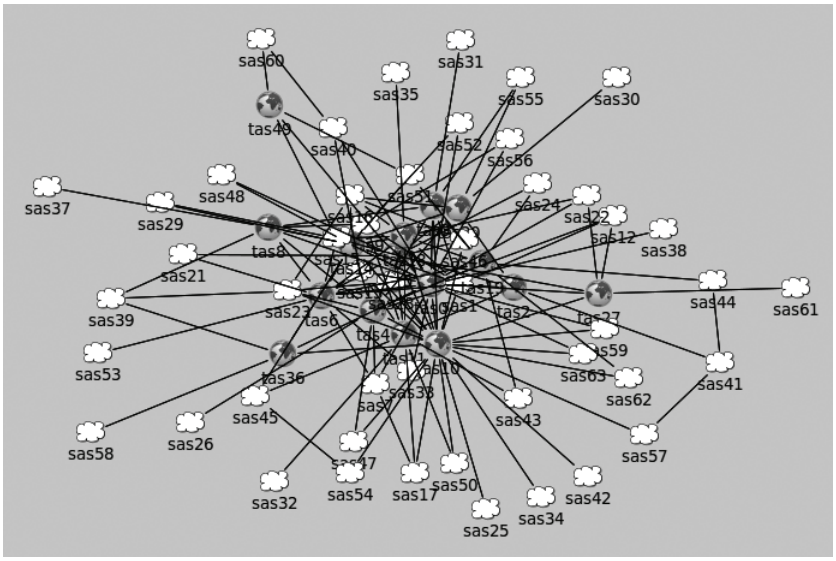


Figura 6-32: Topología a nivel de Sistemas Autónomos o AS (acrónimo del inglés Autonomous System) de la red utilizada en la simulación.

El primer objetivo del estudio consiste en obtener un modelo del SMRD. Para esto, hemos desarrollado el módulo *DRMSHubNode*, tomando como base el módulo *StandardHost* de INET Framework. Este módulo simula un nodo súper-peer del SMRD, y para ello utiliza la aplicación *UDPMonitorApp*, que simula al Monitor Demonio. La aplicación genera paquetes UDP, con un patrón de tráfico similar al obtenido en mediciones reales en el estudio anterior.

El segundo paso consiste en sustituir algunos nodos del modelo construido con ReaSE por nodos *DRMSHubNode*. Para ello, fuimos sustituyendo, de forma progresiva, los nodos cliente (nodos que simulan un tráfico de red correspondiente a aplicaciones clientes, como pueden ser un navegador Web, un cliente de correo electrónico, etc.) por nodos que simulan nodos súper-peer del SMRD. Inicialmente partimos de un entorno formado por 4081 enrutadores, 7918 servidores de diferentes tipos y 70817 nodos clientes, y a partir de éste, obtuvimos 4 entornos donde el 0.4%, 2%, 10% y 50%, de los nodos clientes fueron sustituidos por nodos súper-peer del SMRD. Los nodos súper-peer se activan siguiendo una distribución de probabilidad uniforme, y una vez activados envían periódicamente, cada 10 segundos, un flujo de datos de entre 9 y 14 paquetes, cuyos tamaños van desde 147 hasta 1472 bytes.

El estudio consistió en simular 70 segundos de funcionamiento de la red e ir aumentando el número de nodos que participan en el sistema de monitorización,

comprobando en cada caso los indicadores de funcionamiento del SMRD que se describen más adelante en esta sección.

A continuación se describen las características del generador de tráfico de la aplicación *UDPMonitorApp*, y otras características de la simulación. Por último, se presentan los resultados del estudio.

6.5.2. ESTUDIO DEL TRÁFICO REAL

Para simular el tráfico de los nodos *DRMSHubNode* se utilizó una distribución de probabilidad basada en mediciones reales obtenidas en el estudio anterior. Como parte de este estudio, se pudo determinar que un súper-peer que monitoriza un vecindario de 250 hosts, envía cada 10 segundos un flujo de datos de aproximadamente 15 Kbytes, compuesto por entre 9 y 14 paquetes, distribuidos de la siguiente forma:

Tabla 6-8: Distribución de probabilidad de los paquetes del flujo de datos correspondiente a un súper-peer que envía, a través de la red, la información de monitorización de un vecindario de 250 hosts.

Paquete	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Tamaño 0	1472	1472	1305	1305	1472	1472	1305	1472	1472	1305	1305	323	147	147
Tamaño 1	-	-	1306	1306	-	-	1306	-	-	1306	1306	327	176	176
Probabilidad	1	1	1	0.98	1	1	1	1	1	1	0.97	0.87	0.33	0.33

El 14.3% de los paquetes del flujo (14 posibles paquetes) son paquetes de 147 o 176 bytes, y otro 7.1% son paquetes de 323 o 237 bytes. Estos paquetes se encuentran al final del flujo de datos, y contienen información complementaria (por ejemplo, del estado de la red de superposición), ya que el grueso de la información de monitorización es enviada en los paquetes de mayor tamaño.

La mayor parte de los paquetes son paquetes de 1,305 o 1,306 bytes (35.7%) y paquetes de 1,472 bytes (42.8%). Estos tamaños tienen que ver con la utilización de protocolo UDP para transportar los paquetes por la red. Aunque el tamaño máximo teórico para los paquetes UDP, en redes que operan con IPv4, es de 65,507 bytes, en la práctica, la mayoría de las implementaciones de los protocolos de red a nivel de núcleo del SO o de aplicación, imponen un límite menor, generalmente entre 4 y 8KB. Adicionalmente, los dispositivos de red rechazan aquellos datagramas IP que tienen un tamaño mayor que su unidad máxima de transferencia o MTU (acrónimo del inglés maximum transmission unit) y que no tienen el bit de fragmentación activado. Por esta razón, 1,472 bytes es un tamaño máximo apropiado, que asegura que los paquetes de monitorización sean transmitidos incluso en redes con MTU tan bajas como 1,500 bytes.

6.5.3. CONFIGURACIÓN DEL GENERADOR DE TRÁFICO

Al inicio de la simulación, el generador de tráfico de la aplicación *UDPMonitorApp* utiliza un archivo de configuración para obtener el patrón que debe utilizar para

generar el tráfico del nodo en el que está desplegado. Este archivo tiene formato XML y describe el flujo de datos, especificando el número de paquetes del flujo, el tamaño de cada paquete y la probabilidad de encontrar cada paquete en un flujo de datos que contiene la información de monitorización de un súper-peer. Además, en caso de que un paquete pueda tener dos tamaños diferentes, el archivo de configuración permite definir cuál es la probabilidad de que el paquete tenga el primero de los tamaños posibles. En un mismo archivo se pueden definir varias configuraciones diferentes, que pueden ser elegidas al inicio de la simulación.

En este estudio, todos los nodos súper-peer del SMRD comparten la misma configuración, que se muestra en la **Figura 6-33**.

```
<?xml version="1.0" encoding="UTF-8"?>
<configurations>
    <config id="0"
        streamSize="14"
        packet0ByteLen="1472"       packet0Prob="1"
        packet1ByteLen="1472"       packet1Prob="1"
        packet2ByteLen="1305/1306"  packet2Prob="1"     packet2InP="0.12"
        packet3ByteLen="1305/1306"  packet3Prob="0.98"  packet3InP="0.12"
        packet4ByteLen="1472"       packet4Prob="1"
        packet5ByteLen="1472"       packet5Prob="1"
        packet6ByteLen="1305|1306"  packet6Prob="1"     packet6InP="0.12"
        packet7ByteLen="1472"       packet7Prob="1"
        packet8ByteLen="1472"       packet8Prob="1"
        packet9ByteLen="1305|1306"  packet9Prob="1"     packet9InP="0.12"
        packet10ByteLen="1305|1306" packet10Prob="0.97" packet10InP="0.12"
        packet11ByteLen="323|327"   packet11Prob="0.87" packet11InP="0.96"
        packet12ByteLen="147|176"   packet12Prob="0.33" packet12InP="0.15"
        packet13ByteLen="147|176"   packet13Prob="0.33" packet13InP="0.85"
    />
</configurations>
```

Figura 6-33: Configuración del generador de tráfico de la aplicación *UDPMonitorApp*.

La configuración *config id="0"* de la **Figura 6-33** define un patrón de tráfico que consiste en un flujo de hasta 14 paquetes (atributo *streamSize*). De cada paquete se conoce el tamaño en bytes (atributos *packet0ByteLen*, *packet1ByteLen*, etc.), la probabilidad (atributos *packet0Prob*, *packet1Prob*, etc.) y en caso de que un paquete pueda tener dos tamaños posibles (expresado por dos valores separados por una barra vertical en los atributos *packet2ByteLen*, *packet3ByteLen*, etc.), también define la probabilidad de encontrar el primero de los dos tamaños (atributos *packet2InP*, *packet3InP*, etc.). Por ejemplo, si se dispone de 60 mediciones reales y en ellas aparece el paquete 11 en 52 ocasiones, entonces la probabilidad de encontrar este paquete en el flujo es de 52/60 ~ 0.87. De la misma forma, si de estas 52 veces que aparece el paquete 11, en 50

ocasiones su tamaño es de 323 bytes y en las 2 restantes es de 327 bytes, entonces la probabilidad de que el tamaño de paquete sea de 323 bytes es de 50/52 ~ 0.96.

6.5.4. GENERADOR DE EVENTOS ALEATORIOS

Durante la simulación, el generador de tráfico de la aplicación *UDPMonitorApp* utiliza un generador de números aleatorios con una distribución de probabilidad uniforme para obtener un número aleatorio (*k*) entre 1 y el número de eventos de monitorización, que se calcula al inicio de la simulación con la siguiente ecuación:

$$\text{noEvents} = \frac{\text{simTimeLimit} - \text{startTime}}{\text{messageFrecuency}} \qquad \textbf{Ecuación 6-18}$$

Donde *simTimeLimit* es el tiempo total de la simulación, *startTime* es el instante de la simulación en que se inicia el módulo que simula al nodo súper-peer del SMRD (*DRMSHubNode*), y *messageFrequency* es la frecuencia con la que la aplicación que implementa al Monitor Demonio (*UDPMonitorApp*) envía mensajes. Todas estas variables se expresan en segundos, por lo que el número de eventos no tiene unidades.

Luego, si este número aleatorio *k* es menor o igual que:

$$x = \text{packetProbability} * \text{noEvents} \qquad \textbf{Ecuación 6-19}$$

Donde *packetProbability* es la probabilidad de encontrar un paquete determinado en el flujo de datos que se crea durante el evento de monitorización, entonces se crea el paquete como parte del flujo. En cambio, si el número es mayor, el paquete no se incluye. De forma similar, si el paquete tiene dos tamaños posibles, el tamaño de paquete a generar se obtiene de generar un número aleatorio (*k'*) utilizando una distribución de probabilidad uniforme, pero en este caso se utiliza un generador de números aleatorios diferente (en general se utiliza un generador diferente para cada distribución de probabilidad, y se utiliza exclusivamente para ello, mientras que para generar el resto de los eventos aleatorios de los otros módulos del entorno de simulación se utilizan generadores diferentes). Si el número aleatorio *k'* es menor o igual que:

$$x = \text{packetInnerProb} * \text{packetProbability} * \text{noEvents} \qquad \textbf{Ecuación 6-20}$$

Donde *packetInnerProb* es la probabilidad de que el tamaño de paquete sea el primero representado en el archivo de configuración para un paquete determinado, entonces el tamaño de paquete que se utiliza es el primero de los dos tamaños posibles.

A modo de ejemplo analicemos cómo el generador de tráfico genera el paquete 11 de la configuración mostrada en la **Figura 6-33**. En caso de una simulación de 610 segundos en la que el módulo *DRMSHubNode*, que implementa la aplicación *UDPMonitorApp*, se inicia en el instante de tiempo 7.10 segundos, y envía mensajes de monitorización cada 10 segundos de simulación, aplicando la **Ecuación 6-18** el número de eventos para *DRMSHubNode* es: (610 – 7.10) / 10 ~ 60.

Luego, en cada evento el módulo *DRMSHubNode* obtiene un número aleatorio k entre 1 y 60, que sigue una distribución uniforme en el tiempo. Suponiendo que en el evento E el valor de k=23, entonces, el generador de tráfico utiliza la **Ecuación 6-19** para determinar si debe incluir el paquete dentro del flujo de datos que está creando como parte de este evento. Sustituyendo en la ecuación: x=0.87*60=52.2, por lo cual el paquete se incluye en el flujo. Ahora, como el paquete 11 puede tener dos tamaños posibles (323 y 327 bytes), para decidir cuál de los dos tamaños tendrá en el flujo que se está generando como parte del evento, el generador de tráfico utiliza la **Ecuación 6-20**: x=0.96*0.87*60=50.1, de aquí que, si el valor del número k' generado por el generador de números aleatorios es 53, entonces el tamaño del paquete 11 será de 327 bytes.

6.5.5. RESULTADOS DE LA SIMULACIÓN

Como se utilizaron distribuciones de probabilidad uniformes, al final de la simulación el tráfico de los nodos súper-peer del SMRD tiene un patrón similar al descrito en la **Tabla 6-8**.

La **Figura 6-34** muestra un gráfico que representa el retardo de extremo a extremo o EED (acrónimo del inglés End-to-end delay) [91] y la tasa binaria del flujo de paquetes o PSB (acrónimo del inglés packet stream bit-rate) [91], medidos en dos nodos súper-peer durante todo el tiempo que dura la simulación. El nodo T está situado en un dominio de tránsito, mientras que el nodo F está situado en un dominio frontera. Esta figura muestra que el EED aumenta moderadamente al aumentar el número de nodos súper-peer en la red.

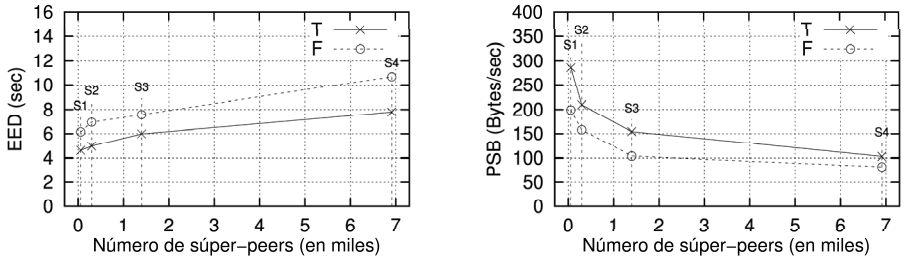


Figura 6-34: Retardo de extremo a extremo (EED) y tasa binaria del flujo de paquetes (PSB), medidos en los nodos súper-peer T (tránsito) y F (frontera) para los entornos de simulación S1, S2, S3 y S4, donde el 0.4%, 2%, 10% y 50% de los nodos clientes fueron sustituidos por nodos súper-peer, respectivamente.

Por el contrario, el gráfico de la **Figura 6-34** muestra un decrecimiento moderado del PSB al aumentar el número de nodos súper-peer. En conjunto, estas dos observaciones indican que el SMRD escala al aumentar el número de nodos que participan en el mismo.

Además, es importante notar que los valores de EED medidos en el súper-peer T son menores que los medidos en el súper-peer F, y por el contrario, que los valores de PSB medidos en el súper-peer T son mayores que los medidos en el súper-peer F. Esto es una consecuencia de la ubicación de los nodos en la red. El súper-peer T está en un dominio de tránsito, que está mejor conectado al resto de los AS en la red de lo que puede estar un dominio frontera, mientras que el súper-peer F está en un dominio frontera, y en consecuencia las rutas hacia el súper-peer F son considerablemente más largas que las rutas hacia el súper-peer T.

6.6. PROTOTIPO

El primer prototipo de GRIDIFF fue publicado en la Web en junio de 2009. Este prototipo puede ser utilizado libremente, a través de una interfaz Web (http://trencadisv03.itaca.upv.es:8000/adgrid) que permite lanzar trabajos de prueba en una infraestructura Grid real. El sistema está compuesto por 5 nodos en España y un nodo en Brasil, que son monitorizados en tiempo real y pueden ser utilizados para ejecutar trabajos.

Los usuarios necesitan adquirir créditos antes de lanzar los trabajos en el Grid. Con cada dirección IP diferente es posible adquirir 50 créditos gratuitos, que valen por 1 hora. Durante este tiempo, los usuarios pueden utilizar los créditos para adquirir recursos computacionales del Grid para ejecutar trabajos. Cada trabajo consume un número diferente de créditos, en dependencia del tipo de servicio que se utilice para ejecutar el trabajo. Mientras más fiable el servicio, más créditos costará la ejecución de trabajos en ese recurso en particular.

Los casos de prueba consisten en ejecutar BLAST [92] con diferentes secuencias de entrada, descritas en la **Tabla 6-9**, y diferentes requerimientos, utilizando una base de datos de secuencias de proteínas extraída de la división de patentes de GenBank, que contiene 737'604 secuencias, con 141'135'747 aminoácidos en total. La versión de BLAST tiene soporte para múltiples hilos, y utiliza una cantidad considerable de CPU y de espacio en memoria para ejecutar búsquedas en las bases de datos de secuencias de proteínas y nucleótidos. Esta base de datos está disponible, localmente, en cada uno de los nodos del Grid. De la misma forma, las secuencias problema también están almacenadas localmente en los nodos. En consecuencia, no hay degradación de las prestaciones debido a las operaciones adicionales para copiar los datos localmente, y el tiempo de ejecución de los trabajos se emplea, casi de forma exclusiva, en el procesamiento necesario para obtener la solución del problema.

El proceso de asignación de recursos transcurre a través de GRIDIFF. Los documentos de requerimientos de QoS de los servicios Grid, así como los requerimientos a nivel de usuario que acompañan a las solicitudes de trabajo, son presentados en el Anexo 4. El algoritmo de asignación impone penalizaciones sobre la selección de los servicios que están ejecutando trabajos adicionales, haciendo especial

hincapié en penalizar los nodos que tienen cargas de trabajo muy altas, frente a los nodos que tienen otros patrones de utilización de recursos, como puede ser una baja disponibilidad de memoria, etc. Estas penalizaciones se reflejan en los índices de idoneidad de los servicios. Además, el sistema está configurado para evitar que los nodos que están ejecutando dos procesos concurrentes, o más, ejecuten trabajos de prueba del prototipo.

Tabla 6-9: Secuencias de proteínas utilizadas en el prototipo.

Identificador	Número de secuencias	Longitud (aa)	Versión	Descripción
Q6L6Q3	1	273	Jun2009	Fragmento del antígeno del MHC de clase I humano.
Q8WZ42	1	34'350	Jun2009	Titina humana, también conocida por Conectina. Es la proteína, de una sola cadena, más grande del organismo.
alu.a	1962	159'704	Jun2009	Traducción de todas las secuencias Alu conocidas.
mito.aa	2222	625'816	Jun2009	Traducción de todas las regiones codificantes del genoma mitocondrial.

La **Tabla 6-10** muestra los trabajos enviados al Grid a través de la Web del prototipo, entre junio y julio de 2009.

Tabla 6-10: Trabajos enviados entre junio y julio de 2009.

Servicios	Q8WZ42	mito.aa	alu.a	Q8WZ42-2	mito.aa-2	alu.a-2	Total
Primarios	21	12	8	19	2	4	*66*
Respaldo	6	1	1	23	5	9	*45*
No garantizados	19	7	5	3	2	2	*38*
Total	*46*	*20*	*14*	*45*	*9*	*15*	*149*

La **Figura 6-35** muestra un gráfico que representa la eficiencia de los servicios, agrupados por disponibilidad y prestaciones, para ejecutar cada trabajo de prueba. Se considera eficiencia a la tasa de trabajos completados dentro de los plazos definidos por trabajos enviados. De forma similar, la **Figura 6-36** muestra un gráfico que representa las causas de fallo para cada grupo de servicios.

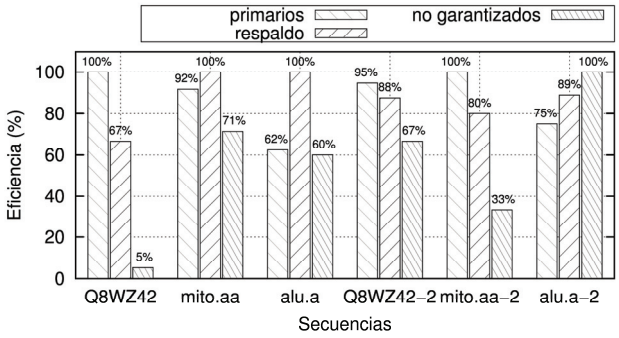


Figura 6-35: Eficiencia de los servicios, agrupados por disponibilidad y prestaciones, para ejecutar cada trabajo de prueba. Los tres primeros resultados corresponden a las secuencias problema Q8WZ42, mito.aa y alu.a, respectivamente, mientras que los tres últimos resultados (representados en el gráfico por Q8WZ42-2, mito.aa-2 y alu.a-2) corresponden a las mismas secuencias, pero utilizando dos hilos de ejecución por trabajo. Los datos del gráfico fueron tomados de la Web del prototipo, entre junio y julio de 2009.

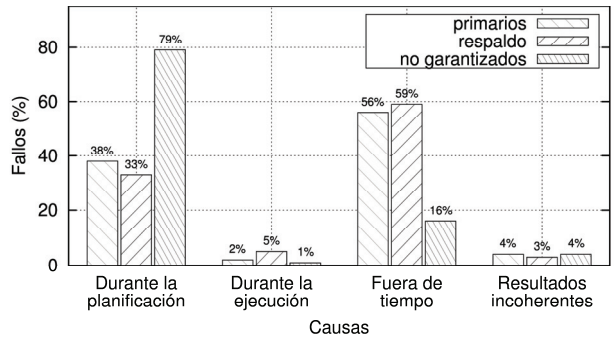


Figura 6-36: Causas de fallo para cada grupo de servicios. Los datos del gráfico fueron tomados de la Web del prototipo, entre junio y julio de 2009.

Los servicios son clasificados en tres grupos: primarios, de respaldo y no garantizados. De ellos, solamente los dos primeros garantizan que un trabajo cumpla con los requerimientos de QoS especificados en la Sección A.4.2 del Anexo 4. Este documento plantea un umbral de disponibilidad por encima del 90% y un tiempo de respuesta para la operación *testOperation* menor de 30 segundos. Esta operación está definida en la Sección A.4.1 del Anexo 4 como una búsqueda por similitud, con tolerancia 0,1, de la secuencia con identificador Q6L6Q3, utilizando un único núcleo. GRIDIFF tiene en cuenta estos requerimientos para clasificar a los servicios: los servicios que tengan indicadores estáticos que cumplan estos dos requerimientos forman parte de los grupos primario y de respaldo, el resto forman parte del grupo de servicios no garantizados.

Adicionalmente, la Sección A.4.3 del Anexo 4 especifica los límites de los indicadores dinámicos, en este caso el documento plantea que las cargas no pueden sobrepasar el 40, 60 y 80% para 1, 5 y 15 minutos, respectivamente. Además, plantea que los nodos deben disponer de, al menos, 64MB libres de memoria física y 256MB libres de swap. Por último, este documento plantea que la selección de recursos debe hacerse sobre la base de que la aplicación hace un uso intensivo de la CPU.

El gráfico de la **Figura 6-35** muestra que cada grupo de servicios tiene una eficiencia diferente para ejecutar los trabajos del grupo de pruebas. De los trabajos enviados, solamente una parte son completados dentro de los plazos de tiempo definidos. Estos plazos fueron establecidos sobre la base de los tiempos de ejecución de los trabajos en los nodos, utilizando directamente el programa BLAST (sin servicios Grid) con un hilo de ejecución por trabajo. Los plazos se calculan, añadiendo un 2% del tiempo, en cada caso.

Aunque algunos resultados se apartan de los valores ideales, en general, el gráfico se ajusta al comportamiento esperado, que consiste en que los servicios de los grupos primario y de respaldo deben tener eficiencias mayores que los servicios no garantizados. Las mayores irregularidades corresponden a los trabajos que tienen tiempos de ejecución muy largos, lo cual explica, en parte, porqué se producen las irregularidades. Mientras más tarde la ejecución de un trabajo, más probabilidades hay de que el nodo se vea involucrado en otro proceso que afecte sus prestaciones.

El gráfico de la **Figura 6-36** muestra que la mayor parte de los trabajos enviados al grupo de servicios no garantizados, que no consiguen completarse con éxito, fallan durante la planificación. Este resultado indica que la clasificación que hace GRIDIFF realmente ayuda a distinguir entre los nodos bien condicionados y los que no lo están. La fracción de trabajos que fallan en los servicios de los grupos primarios y de respaldo, ya sea durante la ejecución o debido a que producen resultados incoherentes, representa una pequeña parte del total (6% y 8% de las causas de fallo para los servicios de los grupos primario y de respaldo, respectivamente). Sin embargo, más de la mitad de los fallos que se producen en estos grupos se deben a trabajos completados después de tiempo (56% y 59%, respectivamente), lo cual puede estar siendo ocasionado por factores externos a la arquitectura, como por ejemplo, una sobrecarga de la plataforma de pruebas. Además, los plazos para la ejecución de los trabajos son muy estrictos. Por ejemplo, el 2% de 45 minutos es menos de 1 minuto, lo que significa que un trabajo que tarde 45 minutos en ejecutarse tendrá un margen de hasta 46 minutos. En aplicaciones reales el margen de tolerancia suele ser mayor, en dependencia del tipo de aplicación.

Por otra parte, el hecho de que un número considerable de trabajos falle durante la ejecución o produzca resultados incoherentes se debe, principalmente, a que los nodos de trabajo han sido configurados para utilizar un único usuario local para ejecutar todos los trabajos del Grid. Los administradores de recursos recurren con frecuencia a este tipo de configuraciones, donde se utiliza un número reducido de cuentas locales para ejecutar un número grande de trabajos, porque las mismas simplifican las operaciones de mantenimiento. Sin embargo, la desventaja que tienen es que los sistemas Linux utilizan un mecanismo para ajustar el rendimiento, que consiste en establecer límites para la utilización de recursos del sistema. Entre estos, el tamaño máximo de la pila, el tamaño máximo de memoria asignada y el número máximo de

páginas asignadas a marcos físicos de memoria, son fundamentales para la ejecución correcta del programa BLAST. Si estos límites no se ajustan bien, el sistema operativo terminará los procesos que corresponden a los trabajos del BLAST para evitar degradaciones en las prestaciones del sistema. En este caso, si el trabajo ha conseguido alcanzar un punto de la ejecución donde haya podido obtener resultados intermedios y volcarlos a disco, entonces al terminar se considera que el trabajo ha fallado porque ha producido resultados incoherentes. En cambio, si el trabajo no produce ningún resultado, se considera que el mismo ha fallado durante la ejecución.

Por último, aumentar la tolerancia de fallo del plazo de ejecución del 2 al 5% contribuiría a mejorar la tasa de trabajos completados después de tiempo, pero no mejoraría el resto de los fallos. Esta modificación tiene sentido para trabajos cortos (de hasta 12 horas de duración), como los que se utilizan en el prototipo, pero puede ser menos efectiva para trabajos con un tiempo de ejecución mayor, por ejemplo del orden de semanas. Sin embargo, en el caso concreto de utilizar aplicaciones Grid para conseguir un alto rendimiento en la ejecución de trabajos cortos, el análisis de los resultados obtenidos con el prototipo sugiere que el número de trabajos completados después de tiempo tiene un peso muy importante en la evaluación de la tasa total de fallos. Por eso una mayor tolerancia puede contribuir positivamente en la evaluación de los sistemas de asignación de recursos para la ejecución de trabajos en el Grid, principalmente porque daría más peso a otros tipos de fallos que se producen y que pasan por alto porque su incidencia es menos frecuente.

A modo de conclusiones de esta sección, los resultados del prototipo son positivos, y han contribuido a la comprensión de los retos a los que nos enfrentamos y a trazar las líneas de trabajo que debemos seguir para continuar avanzando en nuestros objetivos.

6.7. CONCLUSIONES DEL CAPÍTULO

Este capítulo aporta resultados obtenidos en simulaciones computacionales y en entornos de computación Grid reales, que evidencian la validez y la aplicabilidad de la arquitectura GRIDIFF para dar soporte de QoS a aplicaciones Grid. De especial interés son los resultados obtenidos en las pruebas de escalabilidad, utilizando comunicación a grupos de multidifusión sobre una red P2P superpuesta sobre la red física, que tiene en cuenta las características de conectividad del Grid.

Capítulo 7

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. CONCLUSIONES

El trabajo desarrollado en la presente tesis cubre todas las etapas necesarias para la asignación de recursos para la ejecución de trabajos en el Grid. Se ha estudiado la utilización de indicadores de QoS para describir los recursos del Grid y expresar los requerimientos de las solicitudes de trabajo enviadas al mismo, diferenciando entre indicadores de nivel de servicio, que deben ser evaluados mediante casos de prueba representativos que son ejecutados en los recursos, e indicadores de carga, que deben ser monitorizados continuamente. Se ha propuesto un algoritmo de asignación de recursos que permite optimizar la selección de los mismos a nivel global, en la infraestructura Grid. Además, se ha presentado un sistema de monitorización que permite recoger los indicadores de carga de los recursos, y propagarlos a un conjunto de nodos distribuidos por todo el Grid.

Asimismo, se presentan resultados de todo lo anterior. Una parte de los mismos fueron obtenidos en simulaciones en entornos controlados, y otra parte en entornos de computación Grid reales, utilizando middleware Grid actual. De la misma forma, se estudia la aplicabilidad de los resultados a casos de estudio y aplicaciones reales.

Específicamente, en la presente tesis se han abordado soluciones eficientes al problema de la asignación de recursos en el Grid, en base a requerimientos de QoS a nivel de servicio. En este sentido, se implementó un prototipo de asignación de recursos que combina técnicas de monitorización y análisis de clústeres para diferenciar a los servicios Grid, en base a requerimientos de QoS y costes económicos. Este enfoque se sustenta en los resultados obtenidos en una serie de experimentos realizados en un sistema de computación Grid. Los mismos demuestran la efectividad del modelo de asignación de recursos propuesto en esta tesis, para diferenciar los servicios y mejorar la utilización de los recursos. Asimismo, la uniformidad de los tiempos de respuesta pone de manifiesto el potencial del modelo para mejorar la eficiencia de las aplicaciones que requieren de algún nivel de predictibilidad y consistencia.

Consideramos que la principal aportación de este trabajo es proporcionar un nuevo sistema de monitorización que permite examinar periódicamente un sistema distribuido para conocer el estado de sus participantes. Este sistema hizo posible la utilización de tiempos de refresco tan breves como 10 segundos, sin perder escalabilidad en un sistema formado por 251 participantes. Para esto, el sistema se basa en una red de súper-peers que ha sido modificada para almacenar una estructura de datos replicada sobre los participantes de la red. Este enfoque nos permitió utilizar algoritmos de análisis de clústeres, sin modificar, para clasificar los servicios, lo cual constituye un gran reto, no sólo para el Grid, sino para cualquier entorno distribuido. A su vez, el análisis de clústeres nos permitió abordar la solución de problemas complejos, como la diferenciación de servicios y la minimización del coste económico de la ejecución de trabajos en el Grid.

Además, se hizo un estudio profundo y detallado de la asignación de recursos en el Grid, y se realizó una revisión del estado del arte de los principales métodos de solución planteados en la literatura. De forma análoga, se realizó un estudio del estado del arte del middleware Grid y de las infraestructuras de computación Grid actuales. Sobre esta base, se diseñó un modelo para la asignación de recursos en el Grid, y se propuso una arquitectura de software que da soporte para la diferenciación de servicios Grid en base a requerimientos de QoS a nivel de servicio, aporta programas y bibliotecas para la monitorización de recursos en el Grid, y proporciona una estructura y una lógica de trabajo que puede ser extendida por las aplicaciones Grid para prestar diferentes niveles de servicio a diferentes perfiles de usuario. Finalmente, esta arquitectura fue implementada en un prototipo, con el cual se realizaron diferentes pruebas de validación y caracterización.

La arquitectura proporcionada se basa en un diseño estandarizado, que utiliza varios modelos y patrones de diseño de software conocidos, como el modelo de dominio. Por su parte, el prototipo se caracteriza por la utilización de técnicas de programación avanzadas para conseguir un acceso rápido a los datos. Los elementos principales de esta implementación son: la utilización de bases de datos en memoria embebidas en las aplicaciones, la separación de datos en dependencia de las características de acceso para favorecer las operaciones de búsqueda y entrada/salida, la utilización intensiva de sistemas de cache, la utilización de protocolos de comunicación a grupos con garantías de ordenamiento de mensajes, y el desacoplamiento de las operaciones de soporte y mantenimiento, como el volcado de datos a disco. Todo ello favorece la eficiencia del prototipo, principalmente porque ayuda a mantener perfiles de utilización de CPU y de red bajos. En consecuencia, la utilización de memoria es el elemento principal a tener en cuenta, porque puede decidir en qué casos es aplicable el modelo de monitorización. En general, para sistemas medianos (de decenas de miles de nodos) la información de monitorización ocupa decenas de megas en memoria, lo cual no significa un problema para los ordenadores modernos.

Hemos utilizado la infraestructura de seguridad Grid (GSI), no sólo para los servicios Grid, sino también asegurar el sistema de monitorización. En consecuencia, de utilizarse el prototipo fuera de un contexto Grid, el sistema de monitorización no garantizaría la protección de las comunicaciones.

Los resultados de la simulación computacional indican que el sistema de monitorización escala en entornos grandes. Sin embargo, la misma utiliza el mecanismo más eficiente para la comunicación a grupos, que es la difusión a grupos de multidifusión o *multicast*. En consecuencia, para obtener las prestaciones que se espera del sistema de monitorización, sería necesario configurar las redes LAN que utilizan los proveedores, para soportar este mecanismo, y luego extenderlo, donde sea posible, a las redes que sirven de enlace entre estos. La principal dificultad que prevemos para la aplicabilidad del prototipo consiste en que sería necesario contar con las rutas de multidifusión para que puedan ser utilizadas durante el despliegue del sistema de monitorización.

7.2. PROPUESTAS PARA EL TRABAJO FUTURO

El trabajo realizado en esta tesis constituye un punto de partida para abrir nuevas líneas de investigación, que extiendan y perfeccionen los resultados obtenidos hasta el momento. Estas son las que se proponen:

- Estudiar otros métodos de clasificación que puedan ser aplicados a la diferenciación de servicios, haciendo énfasis en otros métodos de análisis de clústeres.
- Estudiar la posibilidad de aplicar métodos probabilísticos, como el algoritmo de esperanza-maximización (EM), para completar la información de monitorización. La utilización de este tipo de técnicas podría complementar a los métodos de monitorización, disminuyendo el número de comunicaciones necesarias, aunque aumentaría la utilización de la CPU. Este enfoque es coherente con el hecho de que la utilización de la CPU que hace el SMRD es muy baja, y bien se podría aumentar, en una cantidad limitada, si con ello se consigue disminuir aún más el tamaño de los datos transmitidos por la red.
- Acoplar el modelo de asignación de recursos con los planificadores Grid. Es de especial interés conseguir una vinculación con el sistema de gestión de carga (WMS) de gLite, para estudiar el efecto de la asignación de recursos sobre BLAST in Grids (BiG) [93], una aplicación desarrollada en el contexto del proyecto EGEE, que permite ejecutar BLAST en entornos de computación Grid.
- Estudiar, mediante simulaciones, el comportamiento de la arquitectura en sistemas distribuidos de gran dimensión, explorando nuevas configuraciones y nuevos esquemas para el sistema de monitorización.
- Aplicar el modelo de asignación de recursos en otros problemas, además de la minimización del coste económico por la ejecución de trabajos en el Grid, como la minimización del tiempo de espera de los trabajos en las colas.

7.3. SOPORTE DE LA TESIS

El trabajo realizado en la presente tesis doctoral ha dado lugar a las siguientes publicaciones y ponencias en congresos:

1. Ignacio Blanquer, Vicente Hernández, Damià Segrelles, **Erik Torres**, "*Workload balancing model of tasks with deadlines and other QoS requirements, in service-oriented Grid computing environments*", Proceedings of the 2nd Iberian Grid Infrastructure Conference, IBERGRID'2008, Ed. NETBIBLO, S.L, ISBN 978-84-9745-288-5
2. Ignacio Blanquer, Vicente Hernández, Damià Segrelles, **Erik Torres**, "*A supporting infrastructure for evaluating QoS-specific activities in SOA-based Grids*", Distributed and Parallel Systems. In Focus: Desktop Grid Computing, Springer, 2008, ISBN 978-0-387-79447-1

3. Ignacio Blanquer Espert, Vicente Hernández García, Damià Segrelles Quilis, **Erik Torres Serrano**, "*A QoS-Aware Tool for Meta-Scheduling Services in High-Throughput Grids*", Proceedings of the First EELA-2 Conference, Bogotá, Colombia, February 25-27, 2009. Publisher: Editorial CIEMAT
4. Ignacio Blanquer Espert, Vicente Hernández García, Damià Segrelles Quilis, **Erik Torres Serrano**, "*Service Monitoring and Differentiation Techniques for Resource Allocation in the Grid, on the basis of the Level of Service*", Enviada a: Future Generation Computer Systems, en marzo de 2010

El trabajo ha recibido soporte financiero de los siguientes proyectos e instituciones:

- Proyecto: Componentes de Nueva Generación para la Explotación Eficiente de Infraestructuras Grid en e-Ciencia. Referencia: TIN2006-12890. Entidad Financiadora: Ministerio de Ciencia.
- Proyecto: Extending EGEE in Latin America (EELA), Phase II. Entidad Financiadora: Comisión Europea.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ABREVIATURAS

ACL	Access Control List
AIO	Asynchronous input/output
API	Application Programming Interface
AS	Autonomous System
CLI	Command Line Interface
CTV	Completion-Time Variance
DHT	Distributed Hash Table
DRMAA	Distributed Resource Management Application API
DRMS	Distributed Resource Management Systems
DTO	Data Transfer Objects
EED	End-to-end Delay
EELA	E-Infrastructure shared between Europe and Latin America
EGEE	Enabling Grids for E-science
EGI	European Grid Initiative
E/S	Entrada/Salida
FIFO	First Input First Output
GSI	Grid Security Infrastructure
GT4	Globus Toolkit 4
HPC	High-Performance Computing
HTC	High-Throughput Computing
IPC	Inter-Process Communication
J2EE	Java 2 Platform, Enterprise Edition
JAI	Java Advanced Imaging
JDBC	Java Database Connectivity
JNDI	Java Naming and Directory Interface
JSDL	Job Submission Description Language
JSP	JavaServer Pages

LoS	Level of Service
LB	Logging and Bookkeeping Subsystem
LMT	Last Modification Time
MDS	Monitoring and Discovery System
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NGI	National Grid Initiative
OGF	Open Grid Forum
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
P2P	Peer-to-peer
PKI	Public Key Infrastructure
PSB	Packet Stream Bit-rate
QoS	Quality of Service
RHT	Replicated Hash Table
SAML	Security Assertion Markup Language
SAS	Stub Autonomous System
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SO	Sistema Operativo
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
TAS	Transit Autonomous System
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
VO	Virtual Organization
WAN	Wide Area Network
WSDL	Web Service Description Language
WMS	Workload Management System
WSDD	Web Service Deployment Descriptor

WSRF Web Service Resource Framework

WTV Waiting-Time Variance

BIBLIOGRAFÍA

[1] National Grid Initiatives, http://web.eu-egi.eu/partners/ngi/

[2] BalticGrid, http://www.balticgrid.org/

[3] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J.L. Nielsen, M. Niinimäki, O. Smirnova, A. Wäänänen, "*Advanced Resource Connector middleware for lightweight computational Grids*", Future Generation Comp. Syst. 23(5): 219-240 (2007)

[4] NAREGI: the National Research Grid Initiative, http://www.naregi.org/index_e.html

[5] TeraGrid, http://www.teragrid.org/

[6] EGEE: Enabling Grids for E-sciencE, http://www.eu-egee.org/

[7] EELA-2: E-science Grid Facility for Europe and Latin America, http://www.eu-eela.eu/

[8] EGEE in numbers, http://project.eu-egee.org/index.php?id=417

[9] H.C. Crawford, D.M. Dias, A.K. Iyengar, M. Novaes, L. Zhang: "*Commercial Applications of Grid Computing*", In: V. Getov, M. Gerndt, A. Hoisie, A. Malony, B. Miller (eds.) Performance Analysis and Grid Computing, pp. 211-229. Springer, Heidelberg (2006)

[10] D. Fernandez-Baca: "*Allocating modules to processors in a distributed system*", IEEE Trans. Software Eng. 15(11): 1427-1436 (1989)

[11] C. Richardson: "*POJOS in Action. Developing Enterprise Applications with Lightweight Frameworks*", Manning Publications Co, ISBN: 1932394583, January 2006

[12] E. Gamma, R. Helm, R. Johnson, J. Vlissides: "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley, ISBN 0-201-63361-2, October 1994

[13] Spring Framework, an open source application framework for the Java platform and .NET Framework (Spring.NET). http://www.springsource.org/

[14] Hibernate, an object-relational mapping (ORM) library for the Java language. https://www.hibernate.org/

[15] Java Data Objects (JDO), a specification of Java object persistence. http://java.sun.com/jdo/

[16] L. Lamport: "*Time, clocks, and the ordering of events in a distributed system*", Communications of the ACM, 21(7): 558-565 (1978)

[17] I. Foster, C. Kesselman: "*Globus: a toolkit-based grid architecture*", The grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, Los Altos, CA, pp. 259-278, (1999)

[18] J. Almond, D. Snelling: "*UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce*", Future Generation Comp. Syst. 15(5-6): 539-548 (1999)

[19] gLite, the EGEE middleware for Grid computing. http://glite.web.cern.ch/glite/

[20] Y. Tanimura, T. Hiroyasu, M. Miki, K. Aoi: "*The system for evolutionary computing on the computational grid*", Proc. of Parallel and Distributed Computing and Systems, ISBN: 0-88986-366-0. ACTA Press Inc., Calgary (2002)

[21] E. Huedo, R.S. Montero, I.M. Llorente: "*The GridWay Framework for Adaptive Scheduling and Execution on Grids*", Scalable Computing - Practice and Experience 6(3): 1-8 (2005)

[22] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke: "*Condor-G: A computation management agent for multi-institutional grids*", Cluster Computing 5(3):237-246 (2002)

[23] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, A. Roy: "*A distributed resource management architecture that supports advance reservations and co-allocation*", Proc. of 7th Int. Workshop on QoS (IWQoS'99), (1999)

[24] K. Czajkowski, I. Foster, C. Kesselman: "*Agreement-based resource management*", Proc. of the IEEE, 93(3): 631-643 (2005)

[25] G. Singh, C. Kesselman, E. Deelman: "*A provisioning model and its comparison with best-effort for performance-cost optimization in grids*", Proc. of the 16th Int. Symposium on High Performance Distributed Computing (HPDC '07), pp. 117–126, ACM Press, (2007)

[26] I. Foster, M. Fidler, A. Roy, V. Sander, L. Winkler: "*End-to-end quality of service for high-end applications*", Computer Communications, 27(14): 1375-1388 (2004)

[27] R.J. Al-Ali, O.F. Rana, D.W. Walker, S. Jha, S. Sohail: "*G-QoSM: Grid service discovery using QoS properties*", Comput. Inform. J. (Special Issue on. Grid Computing), 21(4): 363-382 (2002)

[28] R. Al-Ali, A. Hafid, O. Rana, D. Walker: "*An Approach for QoS Adaptation in Service-Oriented Grids*", Concurrency Comput. Pract. Exper. 16(5): 401-412 (2004)

[29] N. Doulamis, A. Doulamis, A. Litke, A. Panagakis, T. Varvarigou, E. Varvarigos: "*Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing*", Computer Communications, 30(3): 499-515 (2007)

[30] K. Lu , R. Subrata, A.Y. Zomaya: "*On the performance-driven load distribution for heterogeneous computational grids*", J. Comp. System Sci. 73(8): 1191-1206 (2007)

[31] R. Subrata, A.Y. Zomaya, B. Landfeldt: "*A Cooperative Game Framework for QoS Guided Job Allocation Schemes in Grids*", IEEE Transactions of Computers, 57(10): 1413-1422 (2008)

[32] S.E. Middleton, M. Surridge, S. Benkner, G. Engelbrecht: "*Quality of Service Negotiation for Commercial Medical Grid Services*", J. Grid Comput. 5(4): 429-447 (2007)

[33] Mark Hirschey: "*Fundamentals of Managerial Economics*", South-Western Pub., ISBN: 0324288891, June 2005

[34] Q. Zheng, C.-K. Tham, B. Veeravalli: "*Dynamic Load Balancing and Pricing in Grid Computing with Communication Delay*", J. Grid Comput., 6(3): 239-253 (2008)

[35] G. Stuer, K. Vanmechelen, J. Broeckhove: "*A commodity market algorithm for pricing substitutable Grid resources*", Future Generation Comp. Syst. 23(5): 688-701 (2007)

[36] C. Hughes, J. Hillman: "*QoS Explorer: A Tool for Exploring QoS in Composed Services*", Proc. of the IEEE Int. Conf. on Web Services (ICWS), pp. 797-806,

(2006)
[37] D.P. Anderson: "*BOINC: A System for Public-Resource Computing and Storage*", Proc. of the 5th IEEE/ACM Int. Workshop on Grid Computing, (2004)
[38] GRIDCC: Grid Enabled Remote Instrumentation with Distributed Control and Computation. http://www.gridcc.org/cms/
[39] BEinGRID: Business Experiments in Grid. http://www.beingrid.eu/
[40] SmartLM: Grid-friendly software licensing for location independent application execution. http://www.smartlm.eu/
[41] GridEcon: Grid Economics and Business Models. http://www.gridecon.eu/
[42] SORMA: Self-Organizing ICT Resource Management: http://sorma-project.org/
[43] BREIN: Business objective driven reliable and intelligent Grids for real business. http://www.eu-brein.com/
[44] SLA@SOI: Empowering the service industry with SLA-aware infrastructures. http://sla-at-soi.eu/
[45] AssessGrid: Advanced Risk Assessment & Management for Trustable Grids. http://www.assessgrid.eu/
[46] RESERVOIR: Resources and Services Virtualization without Barriers. http://www.reservoir-fp7.eu/
[47] Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.107.pdf
[48] Open Grid Services Architecture (OGSA) Glossary of Terms Version 1.6. http://www.ogf.org/documents/GFD.120.pdf
[49] The Open Grid Services Architecture (OGSA), Version 1.5. http://www.ogf.org/documents/GFD.80.pdf
[50] Grid Economy Use Cases. http://www.ogf.org/documents/GFD.60.pdf
[51] S. Rajasekaran, J. Reif (eds.): "*Handbook of Parallel Computing: Models, Algorithms and Applications*", Chapman & Hall/CRC Computer & Information Science Series, ISBN: 9781584886235, December 2007
[52] G.R. Dattatreya: "*Performance Analysis of Queuing and Computer Networks*", Chapman & Hall/CRC Computer & Information Science Series, ISBN: 9781584889861, June 2008
[53] D. Thain, T. Tannenbaum, M. Livny: "*Distributed Computing in Practice: The Condor Experience*", Concurrency Comput. Pract. Exper., 17(2-4): 323-356 (2005)
[54] Sun Grid Engine: a distributed resource manager. http://www.sun.com/software/sge/
[55] Platform LSF. http://www.platform.com/Products/platform-lsf
[56] R. L. Henderson: "*Job Scheduling under the Portable Batch System*", Proc. of the Workshop on Job Scheduling Strategies for Parallel Processing table of contents, ISBN: 3-540-60153-8. Springer-Verlag, London (1995)
[57] G. Staples: "*TORQUE resource manager*", Proc. of the 2006 ACM/IEEE conference on Supercomputing, ISBN: 0-7695-2700-0. ACM, New York (2006)
[58] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, D. Walker, M. Hategan, N. Zaluzec: "*Analysis and Provision of QoS for Distributed Grid Applications*", J. Grid Comput. 2(2): 163-182 (2004)

[59] VMware Infraestructure. http://www.vmware.com/
[60] J.B. MacQueen: "*Some Methods for classification and Analysis of Multivariate Observations*", Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1: 281-297 (1967)
[61] D. Krivitski, A. Schuster, R. Wolff: "*A Local Facility Location Algorithm for Large-scale Distributed Systems*", J. Grid Comput. 5(4): 361-378 (2007)
[62] S. Hochbaum: "*A best possible heuristic for the k-center problem*", Math. Oper. Res. 10(2): 180-184 (1985)
[63] J. Nocedal: "*Updating quasi-Newton matrices with limited storage*", Math Comp., 35(151): 773-782 (1980)
[64] C. Mastroianni, D. Talia, O. Verta: "*A super-peer model for resource discovery services in large-scale Grids*", Futur. Gener. Comp. Syst. 21(8): 1235-1248 (2005)
[65] B. Yang, H. Garcia-Molina: "*Designing a super-peer network*", Proc. of the 19th Int. Conf. on Data Engineering (ICDE'03), pp. 49-60 (2003)
[66] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, J. Stanton: "*The Spread Toolkit: Architecture and Performance*", Johns Hopkins University, Technical Report CNDS-2004-1 (2004)
[67] T. Oetiker: Round-robin database tool. http://oss.oetiker.ch/rrdtool/
[68] D. Stenberg: cURL and libcurl. http://curl.haxx.se/
[69] Samba: A free software re-implementation of SMB/CIFS networking protocol. http://www.samba.org/
[70] European Grid Initiative. http://web.eu-egi.eu/
[71] The European Grid Initiative (EGI) Blueprint. http://web.eu-egi.eu/blueprint.pdf
[72] Nagios: The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org/
[73] EGEE-III Operations Automation Strategy. EU DELIVERABLE: MSA1.1 (2008)
[74] OSCache: A high performance J2EE caching framework. http://www.opensymphony.com/oscache/
[75] XMLBeans: The Apache Java-to-XML binding framework. http://xmlbeans.apache.org/
[76] XStream: A simple library to serialize objects to XML and back again. http://xstream.codehaus.org/
[77] PostgreSQL: An object-relational database management system (ORDBMS). http://www.postgresql.org/
[78] Oracle Berkeley DB: The family of open source, embeddable databases. http://www.oracle.com/database/berkeley-db/index.html
[79] Weka: A collection of machine learning algorithms for data mining tasks. http://www.cs.waikato.ac.nz/ml/weka/
[80] Xen hypervisor, the open source industry standard for virtualization. http://www.xen.org/
[81] Space imaging archives. http://www.spaceimaging.com/
[82] GTOPO30. http://eros.usgs.gov/products/elevation/gtopo30/gtopo30.html
[83] Java Advanced Imaging (JAI) Stuff. https://jaistuff.dev.java.net/
[84] Java Advanced Imaging (JAI). https://jai.dev.java.net/

[85] stress, a simple workload generator for POSIX systems. http://weather.ou.edu/~apw/projects/stress/
[86] curl-loader, an open-source software performance testing tool. http://curl-loader.sourceforge.net/
[87] tcpdump, a command line packet analyzer. http://www.tcpdump.org/
[88] OMNET++ Simulation Environment. http://www.omnetpp.org/
[89] INET Framework. http://inet.omnetpp.org/
[90] T .Gamer and M. Scharf: "*Realistic simulation environments for IP-based networks*", Proc. of the 1st Int. Conf. on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (SIMUTools 2008), pp. 83-89 (2008)
[91] Recommended Video over IP Metrics. Video Services Forum, Inc. (VSF). Report of the Test and Measurements Activity Group (2006)
[92] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman: "*Basic local alignment search tool*", J. Mol Biol. 215: 403-10 (1990)
[93] G. Aparicio, I. Blanquer, V. Hernandez, D. Segrelles: "*Blast in Grid (BiG): A Grid-Enabled Software Architecture and Implementation of Parallel and Sequential BLAST*", Proc. of the Spanish Conference on e-Science Grid Computing, ISBN: 978-84-7834-544-1, (2007)
[94] G.R. Liu: "*Mesh Free Methods: moving beyond the finite element method*", CRC Press, ISBN: 0-8493-1238-8, July 2002

ANEXO 1

En este anexo se presentan, introducen y fundamentan las herramientas de diseño y desarrollo de software utilizadas en la presente tesis. Primeramente se describen los patrones de diseño de software empleados en la modelación del problema y de su solución, y posteriormente se detallan las características de las herramientas software, las estructuras de datos y los protocolos de comunicación empleados.

A.1.1 MODELO DE CAPAS

Una de las metodologías más usadas para el diseño de aplicaciones de red consiste en dividir el diseño en capas, con el objetivo de simplificar el análisis de los componentes. Cada una de las capas responde a una necesidad de diseño:

- Capa de presentación: ¿Cómo se encapsula la lógica de la aplicación para interactuar con entidades externas?
- Capa lógica: ¿Cómo se organiza e implementa la lógica de la aplicación en un sistema de componentes de software?
- Capa de persistencia: ¿Cómo se garantiza la persistencia de las operaciones y los datos de la aplicación?

Aunque este modelo está estrechamente vinculado a la arquitectura cliente-servidor, donde los componentes están fuertemente acoplados entre sí, también tiene un gran valor para analizar las relaciones que se establecen entre los componentes de otras arquitecturas no tan acopladas. Principalmente, la visión que proporciona el modelo de capas es muy útil para comprender y modelar las diferentes responsabilidades de los componentes de una arquitectura.

A.1.1.A CAPA DE PRESENTACIÓN

La capa de presentación constituye el nivel más alto de una aplicación, y tiene como objetivo traducir las tareas y presentar resultados de forma tal que el usuario pueda interactuar con la aplicación utilizando conceptos del dominio de usuario. En otras palabras, la capa de presentación representa la interfaz de usuario en el modelo de capas. Actualmente, una buena parte de las aplicaciones de red utilizan tecnologías basadas en la Web, como Java Servlets, JavaServer Pages (JSP), y más recientemente Asynchronous JavaScript and XML (AJAX), para su interfaz de usuario. Sin embargo, una interfaz de usuario no es necesariamente una interfaz visual, sino una Interfaz de Programación de Aplicaciones (Application Programming Interface o API) para la comunicación, desde el exterior, con la aplicación.

A.1.1.B CAPA LÓGICA

En la capa lógica suceden todos los procesos que implementan la lógica de una aplicación. Para modelar esta capa es necesario contar con una herramienta de diseño que traslade los conceptos y las relaciones que ocurren en el entorno del usuario al contexto de la lógica de programación que se quiera emplear en el desarrollo de la aplicación, por ejemplo a un contexto de Programación Orientada a Objetos (Object-oriented programming u OOP).

El Modelo de Dominio es una herramienta que consigue muy bien este objetivo. Para ello define una red de clases, relativamente pequeñas, que se corresponden directamente con conceptos del dominio de la aplicación. Siguiendo esta metodología es posible diferenciar las clases según los roles que juegan dentro del dominio. Cada grupo diferente, o categoría, implica ciertas responsabilidades y relaciones con el resto de las clases del dominio, facilitando el desarrollo de los componentes de aplicación. El autor de [11] aplica esta sistematización para definir el siguiente conjunto de roles:

- Entidades: Objetos con una identidad propia que no está definida por el valor de sus atributos.
- Objetos de valor: Objetos sin identidad propia, que son definidos por el valor de sus atributos.
- Factorías: Objetos que definen métodos para crear entidades. Proporcionan métodos para crear conjuntos de objetos y establecer relaciones entre estos, dando lugar a objetos más complejos.
- Repositorios: Objetos que gestionan colecciones de objetos de valor y encapsulan los procedimientos para acceder a las mismas, definiendo los métodos de búsqueda y actualización sobre las colecciones que gestionan.
- Servicios: Implementan el flujo de trabajo de la aplicación. Generalmente, incluyen métodos que actúan sobre múltiples objetos y que no pueden ser asignados a una entidad simple.

A modo de ejemplo, consideremos un sistema de gestión de trabajos en el que un servicio recibe solicitudes de trabajo de un grupo de clientes, y asigna los recursos para la ejecución de los mismos. En el Modelo de Dominios, el servicio de asignación es un servicio que accede a los datos de los diferentes recursos, a través de un repositorio que recupera esta información en forma de objetos de valor, que representan, cada uno, un recurso para la ejecución de trabajos. El servicio utiliza también factorías que le permiten crear entidades que lo asisten en sus funciones, como pueden ser un planificador y un monitor de trabajos, vinculados además al repositorio de recursos.

En la actualidad existen muchas formas de implementar estos roles en una aplicación. Una de las más extendidas consiste en encapsular toda la lógica de la aplicación en un patrón de diseño "façade". Este patrón de diseño se popularizó en 1994, en gran parte debido a la publicación de [12]. El patrón de diseño "façade" proporciona una interfaz unificada al conjunto de interfaces de un sistema, definiendo una interfaz de alto nivel que facilita la utilización del sistema en cuestión. El éxito de este patrón radica en su capacidad para ocultar la complejidad, resolviendo varios problemas de diseño que se

presentan de cara al cliente. Por ejemplo, una "façade" puede definir una interfaz que sirva para acceder solamente a cierta funcionalidad de un sistema más complejo, así como eliminar la dependencia entre el código del cliente y la parte interna de un sistema.

Algunos defectos de diseño muy concretos de la tecnología Enterprise JavaBeans (EJB) propiciaron la popularización en los últimos años de un patrón de diseño "façade" basado específicamente en objetos Plain Old Java Object (POJO) [11]. La distinción de que el patrón se basa en objetos del tipo POJO tiene como objetivo llamar la atención sobre el hecho de que en la implementación se utilizan clases de Java que no son de algún tipo especial (EJBs, JavaBeans, etc.) y que no cumplen ningún otro rol, ni implementan alguna interfaz especial. Esto supone un beneficio directo para el desarrollo de aplicaciones Java porque simplifica el proceso de diseño, facilitando y acelerando el diseño de aplicaciones, y en consecuencia, la implementación.

Usualmente, este patrón se utiliza en combinación con un mecanismo de Programación Orientada a Aspectos (Aspect-oriented programming o AOP), como el framework de Spring [13], para gestionar las transacciones, conexiones a bases de datos y seguridad, en aplicaciones empresariales de Java. La AOP es una técnica de programación que complementa a la OOP, al proporcionar herramientas que permiten a los programadores organizar, de forma modular, los conceptos que atraviesan múltiples tipos y objetos, como por ejemplo las transacciones y las trazas. En un desarrollo orientado a objetos, el enfoque tradicional para la gestión de trazas consiste en incluir las declaraciones necesarias para la gestión de trazas en todas las clases. En cambio, el enfoque AOP consiste en programar la gestión de trazas en un módulo (aspecto) que se puede aplicar, de forma declarativa, a las clases que requieran de esta funcionalidad.

Adicionalmente, el framework de Spring utiliza un principio de OOP conocido como Inversión de Control (Inversion of control o IoC), que reduce el acoplamiento entre los componentes de software, facilitando el diseño de aplicaciones modulares. La IoC es un patrón de diseño que se fundamenta en una metodología de diseño de software conocida como el principio de Hollywood, que toma su nombre del "cliché" que reciben como respuesta los actores amateurs que se presentan a audiciones en Hollywood: "No nos llames, nosotros te llamaremos". La ventaja principal de aplicar este patrón al diseño de aplicaciones es que permite desarrollar código con alta cohesión (la información que almacena una clase es coherente y está, mayormente, relacionada con la clase) y bajo acoplamiento (las clases están poco ligadas entre sí), que además de potenciar la reutilización de código y disminuir la dependencia entre las clases, simplifica los ciclos de depuración, prueba y mantenimiento.

La forma específica de implementar la IoC es la Inyección de Dependencias (Dependency Injection o DI). La DI describe un patrón de diseño que consiste en inyectar un proveedor de una capacidad, o un recurso, al objeto que lo necesita para implementar una funcionalidad, en lugar de programar internamente en la clase los métodos para crear este proveedor o recurso. Es decir, la DI se refiere al proceso de suministrar una dependencia externa (en tiempo de ejecución, una referencia externa) a un componente de software. Este patrón de diseño se ha generalizado rápidamente, y actualmente existen varios frameworks que lo soportan y que permiten desarrollar

componentes intercambiables, que pueden ser sustituidos dinámicamente de forma programática o declarativa. La **Tabla A.1-11** muestra algunos ejemplos de frameworks de DI que existen en la actualidad.

Tabla A.1-11: Ejemplos de frameworks de Inyección de Dependencias (Dependency Injection o DI).

Framework de DI	**Lenguaje/Plataforma**	**Modelo de Distribución**	**Descripción**
C++ Builder	C++	Propietario	Un entorno integrado de desarrollo que permite desarrollar aplicaciones en C++, y proporciona soporte para DI.
Google Guice	Java	Código Abierto	Un framework para el desarrollo de aplicaciones que proporciona soporte para DI a través de anotaciones.
Spring Framework	Java	Código Abierto	Un framework para el desarrollo de aplicaciones que proporciona soporte para DI.
JBoss Microcontainer	Java	Código Abierto	Una refactorización del microkernel JMX de JBoss que permite utilizar objetos POJO fuera del servidor de aplicaciones de JBoss, proporcionando soporte para DI.
Apache Tapestry	Java	Código Abierto	Un framework para el desarrollo de aplicaciones que proporciona soporte para DI.
Spring.NET	.NET	Código Abierto	Un framework para el desarrollo de aplicaciones que proporciona soporte para DI.
Symfony Dependency Injection	PHP	Código Abierto	Una biblioteca software de PHP que proporciona un contenedor para DI.
The IOC Module	Perl	Código Abierto	Un módulo de Perl que proporciona un framework de DI.
Spring Python	Python	Código Abierto	Un framework para el desarrollo de aplicaciones que proporciona soporte para DI.

Aunque el desarrollo de la DI está fuertemente vinculado a Java, al framework de Spring y a otros frameworks, es posible utilizar un patrón de diseño "façade" con DI para implementar la lógica de una aplicación sin necesidad de utilizar un framework externo a la aplicación. La forma habitual de implementar la DI es mediante un

contenedor de dependencias que inyecta a cada objeto los objetos necesarios según las relaciones definidas en un fichero de configuración. También existe la Inyección de Dependencias en tiempo de ejecución (Run-Time Dependency Injection o RTDI), que se utiliza cuando hay información de los componentes que sólo se conoce en tiempo de ejecución. La RTDI es menos utilizada que la DI, en parte porque la mayoría de los casos de uso descritos hasta la fecha utilizan información estática.

De no utilizar un framework con soporte para DI, la aplicación tendría que implementar su propio contenedor de dependencias y proporcionar una forma de especificar las relaciones entre los objetos, siendo además necesario, en algunos casos, un cargador de clases. Luego, con cada ejecución de la aplicación, la "façade" que se crea en la capa lógica obtiene los componentes de la aplicación del contenedor de dependencias. A partir de ese punto, los clientes pueden utilizar la "façade" para invocar a los métodos de los servicios del Modelo de Dominio. Los servicios, a su vez, recuperan objetos de valor de los repositorios y delegan en alguna entidad la responsabilidad de ejecutar cierta funcionalidad que implementa una parte de la lógica de la aplicación.

Generalmente, los métodos de los servicios se corresponden con algún paso de un caso de uso. La "façade" proporciona los mismos métodos que los servicios, pero sus funcionalidades son diferentes. Por ejemplo, si un servicio tiene un método llamado *X*, la "façade" tendrá un método equivalente que llamará al método *X* del servicio, pero además se ocupará de tareas adicionales, en su mayoría relacionadas con los objetos que devuelve el método *X* del servicio. Por ejemplo, la "façade" se ocupa de desacoplar los objetos de la capa de persistencia, y además se ocupa de serializarlos si estos van a ser trasmitidos por la red y de prepararlos si estos van a ser utilizados en la capa de presentación. La "façade" se ocupa también de preparar la información necesaria para los objetos participen en operaciones sincronizadas, por ejemplo en transacciones entre diferentes componentes. La ventaja principal de mantener a los servicios separados de la "façade" radica en el hecho de que los servicios se ocupan exclusivamente de la lógica de la aplicación, y la "façade" de los detalles de interoperabilidad entre componentes, separando así el Modelo de Dominio de los detalles de estructura y organización de los componentes.

Por todo lo anterior, se considera que los servicios constituyen un buen punto de partida para comenzar la implementación del Modelo de Dominio. Partiendo de esta base, nos propusimos diseñar una arquitectura muy desacoplada, pero a la vez interoperable, que consiste en un núcleo computacional que incluye las operaciones principales, y un sistema de componentes, cargados de forma dinámica en la arquitectura, que permite interactuar con componentes middleware y de sistema (la Sección 5.3.1 proporciona más detalles de esta arquitectura). Para ello, utilizamos una "façade" basada en POJO con DI, con el objetivo de beneficiarnos de las posibilidades que brinda este diseño para facilitar el manejo de diferentes componentes y ocultar la heterogeneidad, tanto del middleware, como de los sistemas. De esta forma, se simplifica la portabilidad de la arquitectura a los diferentes sitios del Grid.

A.1.1.C CAPA DE PERSISTENCIA

Una parte considerable de las aplicaciones actuales utiliza datos almacenados en sistemas externos, como por ejemplo bases de datos. Las características del acceso a estas fuentes de datos definen detalles del diseño e implementación de las aplicaciones, como la gestión de seguridad en el acceso a los datos, el acceso a datos compartidos, el acceso a datos distribuidos, etc.

Muchas veces, el acceso a las fuentes de datos se lleva a cabo a través de consultas paginadas dinámicamente. En algunos casos, el acceso se hace directamente a través de conectores, por ejemplo JDBC, pero en otros se utiliza un framework de persistencia como Hibernate [14] y Java Data Objects (JDO) [15]. La ventaja que tiene la utilización de un framework de persistencia con respecto a un conector de datos radica en que los frameworks de persistencia permiten mantener objetos dinámicamente enlazados a los datos, con la posibilidad de reflejar los cambios en ambos sentidos. Es decir, que mediante el acceso con un conector se consigue una copia local e independiente de los datos, mientras que con un framework de persistencia se obtiene una referencia, permanentemente actualizada, de los datos.

Además de ello, los frameworks de persistencia permiten optimizar consultas y manejar grandes volúmenes de datos. Para esto, se precisa de la programación reflexiva, que se combina con el uso de agrupaciones de conexiones y de sentencias pre-elaboradas en cache para acceder a las bases de datos.

La programación reflexiva es un paradigma de programación motivado por la reflexión, siendo la reflexión un proceso que permite la adaptación de una aplicación mientras se ejecuta. Una aplicación utiliza la reflexión para observar su comportamiento y modificar su estructura y su proceder, en respuesta a condiciones abstractas expresadas en su código. En particular, un framework de persistencia utiliza la reflexión para enlazar objetos con estructuras de datos. El uso combinado de la reflexión con la programación declarativa, entendida como la utilización de estructuras de programación que describen a los objetos en lugar de especificar cómo se crean, permite inyectar algoritmos y otros detalles de la implementación en tiempo de ejecución.

El segundo elemento de importancia es la utilización de agrupaciones de conexiones y sentencias pre-elaboradas en cache. El establecimiento de nuevas conexiones de red con fuentes de datos externas es un procedimiento costoso en términos de tiempo y de recursos del sistema. La utilización de agrupaciones de conexiones disminuye esta sobrecarga.

Una agrupación de conexiones es un conjunto preestablecido de conexiones que pueden ser reutilizadas por una aplicación, en lugar abrirlas y cerrarlas repetidamente. Se considera que el tamaño óptimo de una agrupación de conexiones es aquel que permite responder a todas las solicitudes sin que ninguno de los componentes de aplicación tenga que esperar por una conexión, aunque en la práctica el número de conexiones concurrentes está limitado por otros factores relacionados con la disponibilidad de recursos en el sistema.

El registro de sentencias en cache consiste en almacenar consultas (por ejemplo, sentencias SQL) que han sido optimizadas y ejecutadas con anterioridad, para evitar que sean procesadas nuevamente (evitando las validaciones sintácticas y de direcciones,

y la optimización de planes de ejecución y de rutas de acceso), cuando se repite la ejecución. Esta práctica suele mejorar considerablemente las prestaciones de las operaciones de acceso a los datos.

A.1.2 HERRAMIENTAS SOFTWARE

Este trabajo utiliza varias aplicaciones y bibliotecas software para el desarrollo y validación de la arquitectura software que se propone como parte del mismo. En esta sección describimos brevemente las más relevantes para el diseño e implementación de las aplicaciones finales. El resto de las aplicaciones y bibliotecas se explican en las secciones donde se utilizan.

A.1.2.A SISTEMAS DE GESTIÓN DE BASES DE DATOS

La arquitectura accede a dos fuentes externas de datos. La primera de ellas es una base de datos de PostgreSQL [77]. Este sistema de gestión de bases de datos objeto-relacional (ORDBMS) se distribuye libremente bajo una licencia del estilo BSD. PostgreSQL tiene la mayoría de las características presentes en los grandes sistemas de gestión de bases de datos comerciales, como transacciones, consultas anidadas, disparadores (triggers), vistas, integridad referencial de claves externas, y sofisticados mecanismos de bloqueo (locking); pero además tiene otras características que no están presentes en estos productos, como los tipos de datos definidos por el usuario, la herencia, las reglas, y el control de concurrencia de múltiples versiones para reducir el problema de la contención de bloqueos (lock contention). En cuanto a prestaciones, PostgreSQL es comparable con otras bases de datos comerciales y libres. PostgreSQL está avalado por más de 20 años de historia, en la que destacan la estabilidad y la fiabilidad de sus versiones.

La segunda fuente de datos externa es un conjunto de bases de datos de Oracle Berkeley DB [78]. Aunque no es un sistema de gestión en sí, esta familia de bases de datos embebidas permite incorporar un motor transaccional de bases de datos en las aplicaciones. La principal ventaja de utilizar este sistema radica en su velocidad y escalabilidad. Como Berkeley DB elimina el sobrecoste del SQL y de la intercomunicación entre procesos es posible conseguir unas prestaciones muy altas en el acceso a los datos. Sin embargo, la utilización de Berkeley DB se limita al desarrollo de aplicaciones que no necesiten una interfaz para la administración de bases de datos. En consecuencia, los costes de instalación y operación son muy bajos porque además de no tener costes de administración, utiliza menos recursos que un sistema de gestión, y se distribuye libremente bajo una licencia del estilo BSD. Por último, Berkeley DB aporta mucha flexibilidad a los desarrolladores porque permite controlar detalles de configuración de muy bajo nivel que permiten desarrollar aplicaciones muy optimizadas para un fin específico.

A.1.2.B SISTEMAS DE CACHE

Una de las técnicas más utilizadas para conseguir buenas prestaciones en los sistemas distribuidos consiste en almacenar los resultados de las operaciones más frecuentes (o

más costosas) en un sistema de cache, para reutilizar los mismos resultados con varios clientes. OSCache [74] es un sistema de cache que se distribuye libremente bajo una licencia del estilo Apache. Fue diseñado originalmente para resolver las limitaciones de los sistemas tradicionales de cache que almacenan solamente contenido estático y objetos binarios. De esta forma, OSCache almacena adicionalmente, secciones dinámicas de páginas JSP (JavaServer Pages). Para conseguir el dinamismo necesario para lograr este objetivo, OScache utiliza un sistema de cache en memoria basado en claves definidas programáticamente, lo cual le confiere unas características muy buenas en cuanto a prestaciones. Adicionalmente, OSCache puede ser utilizado en un entorno Java para cachear objetos arbitrarios. Este ha sido el uso que se le ha dado dentro de la arquitectura, principalmente para almacenar colecciones de objetos de valor que son reutilizados para disminuir la carga computacional.

A.1.2.C HERRAMIENTAS DE ANÁLISIS DE DATOS

Uno de los pilares fundamentales de este trabajo es la utilización de algoritmos de análisis de clústeres para diferenciar recursos distribuidos. Para ello hemos utilizado implementaciones de estos algoritmos proporcionadas por Weka [79], la suite para aprendizaje automático y minería de datos desarrollada por la Universidad de Waikato, y que es ampliamente utilizada por la comunidad. Weka está escrita en Java y se distribuye libremente bajo la licencia GPL. La suite contiene una colección de herramientas de visualización y de algoritmos para análisis de datos y modelado predictivo. Además soporta varias tareas estándar de minería de datos, especialmente, pre-procesamiento de datos, clasificación, regresión, análisis de clústeres, reglas de asociación y visualización.

A.1.2.D PROCESAMIENTO DE XML

El metalenguaje XML constituye una de las bases actuales de Internet. La SOA y las especificaciones de servicios Web, constituyen ejemplos claros de la importancia del XML. Sin embargo, la naturaleza misma de este metalenguaje dificulta el procesamiento de los documentos XML. El problema principal es el sobrecoste computacional que lleva asociado, sobre todo teniendo en cuenta que cada vez hay más documentos XML que procesar, porque cada vez los protocolos de comunicación son más complicados (cada vez hay más especificaciones de servicios Web, y cada vez son más complejas), y cada vez hay más componentes software definidos en XML (configuraciones, aspectos, etc.).

Este uso generalizado del XML ha propiciado la aparición de un gran número de frameworks, con características muy variadas, para el procesamiento de XML. XMLBeans [75] es un framework de XML que forma parte del Proyecto XML de la Apache Software Foundation (ASF), y es distribuido libremente bajo la licencia Apache 2.0. XMLBeans proporciona una tecnología para vincular objetos de Java con documentos XML, y de esta forma permite acceder a documentos XML como objetos en memoria, desde el código de una aplicación. Además, incluye soporte para esquemas XML y proporciona analizadores que permiten generar, de forma automática, el conjunto de clases de Java que representan a un tipo de documento

XML. A su vez, estas clases proporcionan constructores que implementan las reglas y funcionalidades expresadas en el esquema XML, por lo cual permiten crear nuevos documentos XML que cumplen con el esquema. Igualmente, XMLBeans mantiene a la vez dos copias sincronizadas de cualquier documento XML convertido en objetos, una como información XML y la otra como objetos de Java, y permite trabajar con cualquiera de ellas, según la necesidad de cada aplicación. En cuanto a prestaciones, XMLBeans es comparable con otros frameworks de XML.

La arquitectura utiliza XMLBeans para implementar todas las operaciones que requieren del manejo de documentos XML, como por ejemplo las operaciones de configuración y el volcado de objetos de valor a formatos intercambiables de datos. En estos casos la arquitectura se apoya en esquemas XML que definen el formato de los documentos XML que se utilizan en cada caso. Sin embargo, en algunas ocasiones es necesario obtener una copia rápida de un objeto Java, por ejemplo para trasmitirlo por la red. En este caso, no es necesario disponer de un esquema XML, y tampoco es necesario utilizar un framework XML con todas las operaciones de validación que aseguran que la estructura de un documento XML tiene una coincidencia estricta con un conjunto de esquemas. Por el contrario, sí es necesario asegurar que la operación ocurre en el menor tiempo posible y utilizando la menor cantidad de recursos de cómputo.

XStream [76] es una biblioteca de software que permite serializar objetos de Java a XML y recuperar objetos de Java serializados como XML. XStream se distribuye libremente bajo una licencia del estilo BSD. Entre sus principales características destaca la necesidad de muy pocos recursos. Para utilizarla no es necesario modificar los objetos y genera documentos XML muy sencillos en los que serializa, además de los atributos públicos y protegidos, los atributos privados y finales. Además, XStream incluye soporte para clases no públicas e internas. Estas excelentes prestaciones hacen que se utilice frecuentemente como transporte. Este ha sido el uso que se le ha dado dentro de la arquitectura, para serializar objetos que se transmiten por la red entre componentes remotos de la arquitectura.

A.1.2.E HERRAMIENTAS DE COMUNICACIÓN A GRUPOS

Un protocolo de comunicación es un conjunto de reglas estándares para la representación de los datos, que permite organizar los procesos que ocurren durante el intercambio de información a través de un canal de comunicación. Son varios los protocolos de comunicación utilizados en este trabajo, pero la mayoría de ellos son conocidos estándares. En esta sección describiremos brevemente Spread Toolkit [66], una herramienta que proporciona varios protocolos de comunicación a grupos, que son menos conocidos, y que han sido importantes para el diseño e implementación del sistema de monitorización que forma parte de la arquitectura presentada en este trabajo.

Spread Toolkit es un paquete de software que proporciona un sistema de comunicación a grupos que se caracteriza por sus altas prestaciones y su tolerancia a fallos, tanto en redes de área local (LAN) como en redes de área amplia (WAN). Spread es distribuido libremente bajo una licencia del estilo BSD. Este sistema

proporciona una interfaz común para el paso de mensajes entre aplicaciones distribuidas, dando soporte a un amplio número de servicios que pueden ser utilizados como parte de varios patrones de comunicación como son multidifusión a nivel de aplicación, comunicación a grupos y comunicación punto a punto. Algunos de los servicios proporcionados por Spread son: mensajes sin orden, mensajería fiable y mensajes totalmente ordenados con garantía de entrega. El paquete de software consiste en un servidor de mensajería y bibliotecas clientes para varios tipos de entornos de programación, como por ejemplo C/C++ (con y sin soporte para hilos), Java, Perl, Python y Ruby.

Además de enviar mensajes a grupos de destinatarios y de recibir información sobre la disponibilidad y la accesibilidad de los servicios, Spread proporciona garantías de ordenamiento y fiabilidad en la entrega de mensajes. En cuanto al ordenamiento, los servicios de mensajería proporcionados por Spread pueden enviar mensajes sin orden, mensajes ordenados en colas FIFO, mensajes ordenados mediante el orden causal de Lamport [16] y mensajes en orden total (todos los mensajes son enviados en el mismo orden a todos los receptores). Adicionalmente, los mensajes sin orden y los mensajes ordenados en colas FIFO y en el orden causal de Lamport, pueden ser entregados con garantías de fiabilidad. Para esto, Spread proporciona métodos de recuperación en caso de producirse pérdidas de mensajes. De estas garantías, el ordenamiento causal es especialmente importante para nuestros propósitos, porque asegura que todos los receptores reciban los mensajes en el mismo orden en que estos fueron enviados por el remitente. Asimismo, la entrega fiable puede servir para garantizar que los destinatarios reciban todos los mensajes que les son enviados. En este sentido, una opción a tener en cuenta es la utilización de canales redundantes, sobre todo teniendo en cuenta que muchas de las comunicaciones ocurrirán en redes no fiables, de las cuales se espera obtener una escalabilidad alta. En este caso en particular, mantener canales redundantes es una opción preferible a tener entrega fiable, porque puede conseguir mejores prestaciones con menos sobrecarga de la red. Por todo ello, la arquitectura utiliza comunicación a grupos con ordenamiento causal, sin garantías de fiabilidad.

A.1.2.F HERRAMIENTAS DE VISUALIZACIÓN DE DATOS

Una vez que la información de los recursos distribuidos ha sido recuperada, surge la necesidad de visualizar los datos, utilizando para ello algún tipo de gráfico que permita representar la ocupación de los recursos en el tiempo. RRDtool [67] es una herramienta que permite gestionar series cronológicas de datos, como la utilización del ancho de banda, la variación de la temperatura y la utilización de la CPU en el tiempo. RRDtool está escrita en "C" y es distribuida libremente bajo la licencia GPL. Esta herramienta permite almacenar los datos en una base de datos round-robin, de forma tal que el espacio utilizado se mantiene constante en el tiempo. Además, RRDtool permite visualizar los datos en varios formatos gráficos que son altamente configurables. La arquitectura proporciona una herramienta, basada en RRDtool, que permite almacenar y visualizar los datos de carga de la CPU y de ocupación de la memoria en el tiempo para cada recurso monitorizado.

A.1.2.G HERRAMIENTAS DE PROCESAMIENTO DE IMÁGENES

Como parte del proceso de validación de la arquitectura, se implementaron varios prototipos de aplicación. Uno de ellos consiste en un servicio Grid para el procesamiento de imágenes digitales. Las funcionalidades de este servicio se basan en la API de Java Advanced Imaging (JAI) [84]. Esta API proporciona un conjunto de interfaces orientadas a objeto que permiten manipular imágenes. Su desarrollo está dividido en varios sub-proyectos que son distribuidos bajo licencias diferentes. Los dos sub-proyectos más importantes: el núcleo computacional y el conjunto de demostraciones, se distribuyen bajo licencias Java Research License (JRL) y BSD, respectivamente. La licencia JRL es una licencia que impone algunas restricciones sobre el uso comercial de las aplicaciones, pero permite la distribución libre para aplicaciones no comerciales. El núcleo de JAI soporta las operaciones básicas para el procesamiento de imágenes, como son la adquisición y visualización de imágenes, y la manipulación básica con algoritmos para mejorar imágenes, manipulación geométrica y análisis básico. Además, soporta algunas operaciones avanzadas, como son la compresión y descompresión de imágenes, y algunos algoritmos de procesamiento avanzado, como la segmentación y el análisis de patrones.

A.1.2.H HERRAMIENTAS DE ANÁLISIS DE REDES Y SISTEMAS

Como parte del proceso de validación y caracterización de la arquitectura, se utilizaron varias herramientas para establecer las condiciones del entorno de prueba, y para monitorizar y analizar los cambios producidos en el sistema. Una de las herramientas utilizadas permite generar cargas de trabajo para sistemas POSIX, imponiendo un determinado estrés de CPU, memoria y E/S en el sistema. El nombre de esta herramienta es "stress" [85], está escrita en "C" y es distribuida libremente bajo la licencia GPL. Usualmente es utilizada para evaluar la escalabilidad de los sistemas, y para exponer errores que se manifiestan solamente, o con mayor frecuencia, cuando el sistema está sometido a una carga muy grande.

Otra herramienta permite ejecutar pruebas que sirven para evaluar el rendimiento de las aplicaciones de red. El nombre de esta herramienta es "curl-loader" [86], está escrita en "C" y es distribuida libremente bajo la licencia GPL. Puede simular el comportamiento de miles de clientes HTTP/HTTPS y FTP/FTPS, cada uno con su propia dirección IP. Utilizando implementaciones reales de estos protocolos, consigue simular el flujo de paquetes que se produciría en una situación real, posibilitando la evaluación del comportamiento de las aplicaciones de red para diferentes cargas de trabajo y de red.

Por último, "tcpdump" [87] es un analizador de paquetes de red, distribuido libremente bajo la licencia BSD con casi todos los sistemas tipo UNIX. Permite interceptar y visualizar los paquetes de varios protocolos, incluyendo TCP/IP y UDP/IP, que están siendo enviados y recibidos en la red donde se encuentra el ordenador que ejecuta el "tcpdump". Una de sus características principales es que permite definir filtros que luego son utilizados para filtrar los paquetes capturados. Estos filtros sirven para

separar los paquetes de interés del resto del tráfico de red, sobre la base de las direcciones IP de los remitentes y destinatarios de los paquetes y del tipo de protocolos, entre otras características.

A.1.3 ESTRUCTURAS DE DATOS

Una estructura de datos es una forma de organizar un conjunto de datos, que define la forma de operar con los mismos. En consecuencia, al almacenar un conjunto de datos en una estructura de datos determinada, se produce una subordinación de los datos a una lógica de trabajo que define cómo se adicionan nuevos valores a la estructura, cómo se borran valores existentes, cómo se organizan los valores, y cómo se producen las búsquedas, entre otras operaciones. Por otra parte, cada estructura ofrece ventajas en relación a la eficiencia de cada una de estas operaciones. Dichas ventajas están relacionadas con la forma y la frecuencia con que se produce el acceso a los datos. En esta sección describiremos brevemente las estructuras de datos que han sido más importantes para el diseño e implementación de las aplicaciones finales.

A.1.3.A SISTEMAS DE BASES DE DATOS EN MEMORIA

Una "Tabla hash lineal extendida" es una estructura de datos de Oracle Berkeley DB que proporciona una forma de almacenamiento de datos en una tabla hash lineal. Una tabla hash es una estructura de datos que asocia claves con valores, y proporciona una operación muy eficiente de búsqueda por claves. Se dice que una tabla hash es lineal cuando utiliza una función de hash lineal, lo que le permite aumentar gradualmente su capacidad. Una tabla hash lineal extendida permite utilizar tipos de datos de cualquier complejidad, tanto para las claves, como para los datos de los registros. Por ejemplo, estos pueden contener valores simples como números enteros y cadenas de caracteres, o tipos complejos como estructuras y clases. De forma opcional, una tabla hash lineal extendida puede almacenar registros cuyas claves sean iguales, en cuyo caso se considerarán duplicados. Esta estructura de datos mantiene muy poca información interna, por lo que ocupa muy poca memoria con información adicional. Esta característica es una ventaja para un tipo específico de aplicaciones de base de datos que utilizan, fundamentalmente, la memoria principal para el almacenamiento de datos. Por otra parte, el hecho de almacenar poca información que relacione a los datos obliga a que el tipo de búsquedas que se pueden hacer con esta estructura de datos sea muy simple. Esto último no es un problema si todas las búsquedas pueden ser organizadas por claves. Para ello, Oracle Berkeley DB proporciona un tipo de estructura de datos que sirve para almacenar, de forma eficiente, un conjunto alternativo de claves para acceder a los datos: la base de datos secundaria. Esta estructura de datos es un índice que contiene referencias que vinculan a las claves con posiciones de memoria (o disco, en el caso de bases de datos persistentes) que corresponden a campos independientes dentro de los datos. De esta forma, se pueden establecer nexos adicionales, utilizando muy poca cantidad de memoria.

ANEXO 2

Las funciones de amortiguamiento juegan un papel muy importante en el algoritmo de asignación de recursos de la Sección 4.3, porque modulan la puntuación de los clústeres. Esta sección presenta dos funciones de amortiguamiento, seleccionadas de un estudio de funciones utilizadas en la simulación numérica, que fueron utilizadas en la implementación del algoritmo.

La Sección A.2.1 presenta la función de amortiguamiento spline de cuarto grado, que fue utilizada para reforzar la aportación de los indicadores estáticos más relevantes a la puntuación de los clústeres, mientras que la Sección A.2.2 presenta la función de amortiguamiento exponencial, que fue utilizada para atenuar la aportación del resto de los indicadores estáticos. Por último, la Sección A.2.3 presenta un análisis gráfico de estas funciones y de sus derivadas primera y segunda.

A.2.1 FUNCIÓN DE AMORTIGUAMIENTO SPLINE DE CUARTO GRADO

La función de amortiguamiento spline de cuarto grado [94] está definida por la ecuación:

$$QS(x) = f(x) = \begin{cases} 1 - 6x^2 - 8x^3 - 3x^4 & x \leq 1 \\ 0 & x > 1 \end{cases}$$ **Ecuación A.2-21**

A.2.2 FUNCIÓN DE AMORTIGUAMIENTO EXPONENCIAL

La función de amortiguamiento exponencial [94] está definida por la ecuación:

$$EX(x) = f(x) = \begin{cases} e^{-(x/\alpha)^2} & x \leq 1 \\ 0 & x > 1 \end{cases}$$ **Ecuación A.2-22**

Donde α es una constante que toma un valor conveniente, según el caso.

A.2.3 ANÁLISIS GRÁFICO DE LAS FUNCIONES DE AMORTIGUAMIENTO Y SUS DERIVADAS PRIMERA Y SEGUNDA

Tanto la función de amortiguamiento spline de cuarto grado, como la función de amortiguamiento exponencial tienen forma de campana, y son continuas en el intervalo $\{x \mid 0 \leq x \leq 1\}$. Además, las derivadas primera y segunda de ambas funciones están definidas en el intervalo, y son continuas y suaves. Las figuras **Figura A.2-37**, **Figura A.2-38** y **Figura A.2-39** representan las funciones de amortiguamiento spline de cuarto grado y exponencial, desplazadas en *x-1*, y sus derivadas primera y segunda, respectivamente, también desplazadas. Esta transformación de coordenadas permite comparar, visualmente, las funciones de amortiguamiento con la función *y=x*. La **Figura A.2-37** muestra el efecto conseguido al aplicar las funciones de amortiguamiento para transformar una coordenada *x* en su imagen en *y*. Por ejemplo, un valor de *x=0,5* se transformaría en aproximadamente *0,3* con la función de amortiguamiento spline de cuarto grado, y en aproximadamente *0,1* con la función de amortiguamiento exponencial. Sin embargo, para valores de *x* por encima de *0,8*, la función de amortiguamiento spline de cuarto grado considera que se ha alcanzado el máximo valor posible para esta coordenada (en el gráfico, la curva se va por encima de *y=x*). Lo mismo sucede para la función de amortiguamiento exponencial, para valores de *x* por encima de *0,9*.

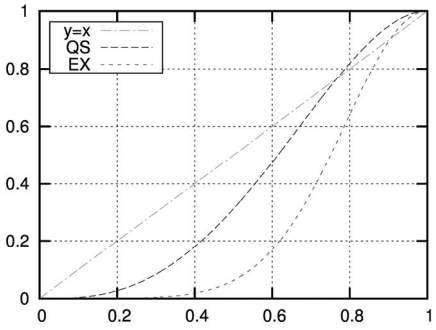


Figura A.2-37: Funciones de amortiguamiento comparadas con la función *y=x*. QS: spline de cuarto grado, EX: exponencial (*a=3*). Las funciones han sido desplazadas en *x-1*.

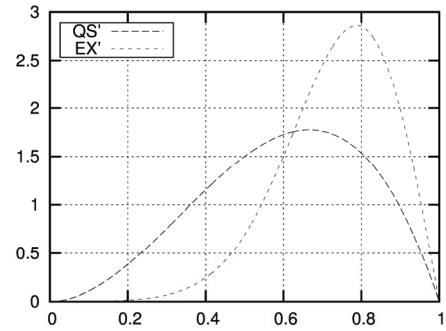


Figura A.2-38: Primera derivada de las funciones de amortiguamiento. QS: spline de cuarto grado, EX: exponencial (*a=3*). Las funciones han sido desplazadas en *x-1*.

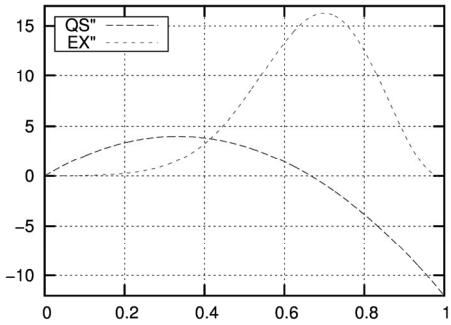


Figura A.2-39: Segunda derivada de las funciones de amortiguamiento. QS: spline de cuarto grado, EX: exponencial (*a=3*). Las funciones han sido desplazadas en *x-1*.

ANEXO 3

Archivos de configuración del Sistema de Monitorización de Recursos Distribuido. Configuraciones global y local de los nodos del sistema de monitorización.

A.3.1 CONFIGURACIÓN GLOBAL

```
################################################################################
#
# File: adgrid.conf
#
# Purpose: This file defines how the application-level network is organized in
#          the Grid. It is consulted by adgrid when start.
#
################################################################################

[general]
### Name of the Grid. It will be used as the prefix of all the groups derived
### from this Grid. The name cannot contain the character '@'.
grid.name = ES_UPV_ITACA_GRyCAP

### Is the site scope enabled?
site.scope.enabled = yes

### Renew the status of the regular peers every these seconds. Both super and
### regular peers use this field to manage registrations. Regular peers need
### to schedule the renew of their status in accordance with this value.
refresh.interval.secs = 10

[application-level.network]
### Maximum number of Grid sites.
n.sites.max = 20

### Maximum number of neighborhoods per Grid site.
n.neighborhoods.max = 8

### Super-peer redundancy.
super-peer.redundancy = 2

[site.mapping]
### Enumerate the sites starting by 0. The number of sites mapped must be less
### than or equal to the maximum number of Grid sites defined by
### 'n.sites.max'. A site out of the order of increasing index will be
### ignored, as well as the sites after the site.
```

```
site.0.name   = Idun
site.1.name   = GRyCAP
site.2.name   = RNP
#site.3.name  = UNUSED
#site.4.name  = UNUSED
#site.5.name  = UNUSED
#site.6.name  = UNUSED
#site.7.name  = UNUSED
#site.8.name  = UNUSED
#site.9.name  = UNUSED
#site.10.name = UNUSED
#site.11.name = UNUSED
#site.12.name = UNUSED
#site.13.name = UNUSED
#site.14.name = UNUSED
#site.15.name = UNUSED
#site.16.name = UNUSED
#site.17.name = UNUSED
#site.18.name = UNUSED
#site.19.name = UNUSED
```

A.3.2 CONFIGURACIÓN LOCAL

```
##############################################################################
#
# File: adgrid-local.conf
#
# Purpose: This file defines how the application-level network is locally
#          organized. It is consulted by adgrid when start.
#
##############################################################################

[agent.speer]
### Should this local agent act in the ARSIC (super-peer) role? Possible
### values are: yes and no. Default is no.
arsic.role = yes

### Use this parameter to specify how this agent behaves when it is in a
### local neighborhood and redundancy is enabled. Note that only super-peers
### (ARSIC services) are replicated. Possible values are: ROOT, APPEND and
### APPEND_ONLY. ROOT and APPEND_ONLY are exclusive policies because they
### obligates an agent to be the primary replica and a secondary replica in
### the neighborhood, respectively. APPEND is a flexible policy that enables
### the agent to start a new replication schema in its neighborhood, if there
### is no previous primary replica declared for the neighborhood, or to join
### a replication schema if a primary replica was previously declared. APPEND
```

```
### is also the default behavior when no replication policy is defined.
replication.policy = APPEND

[agent.rpeer]
### Should this local agent act in the SMS (regular peer) role? Possible
### values are: yes and no. Default is yes.
sms.role = yes

### Nodes acting in the SMS role need to provide a valid IP that will be used
### in notifications. If you leave this field commented out, the system will
### try to discover the IP address of the host. In most of cases, this will
### work. Nevertheless, if you have a multi-addressed server (a host more than
### one IP in public domains, not internal IP addresses but valid Internet
### addresses) the system will fail to discover the node IP. Please, also note
### that at this moment we only support for Internet Protocol version 4 (IPv4)
### addresses.
# node.notification.ip = 127.0.0.1

### Should the local node be monitored? Possible values are: yes and no.
### Default is yes.
node.include.local = yes

### Number of physical processors in this host. It will be used in SMS nodes
### as part of the notifications. If you leave this field in blank, the system
### will assume 1 physical processor.
# node.num.physical.proc = 1

### Number of logical processors per physical processor in this host (due to
### hyper-threading, multiple cores, etc.). It will be used in SMS nodes as
### part of the notifications. If you leave this field in blank, the system
### will assume 2 logical processors per physical processor.
# node.num.logical.proc = 2

### Enumerate the virtual nodes starting by 0. The number of virtual nodes
### must be less than or equal to 250. Virtual nodes (or hosts) are those hosts
### that use this node to make available their status. A virtual node out of
### the order of increasing index will be ignored, as well as the virtual
### nodes after the node.
virtual.node.0.ip = 158.42.167.5
virtual.node.1.ip = 158.42.167.18
virtual.node.2.ip = 158.42.167.12
virtual.node.3.ip = 158.42.167.52
virtual.node.4.ip = 158.42.167.85

### Use this parameter to specify how and when the data in the Replicated
### Hash Table (RHT) should be dumped to the file system. Possible values
### are: NEVER_DUMP and ASYNC_DUMP. ASYNC_DUMP schedule the data to be dumped
### to a single file during the periods of system inactivity. Use this policy
```

```
### to create and maintain an snapshot of the RHT in a file. This policy is
### especially useful when troubleshooting a problem. NEVER_DUMP is the policy
### that should be used when it is not desirable to store data to disk.
### NEVER_DUMP is also the default behavior when no dump policy is defined.
dump.policy = ASYNC_DUMP

### Should enable the client support? If you activate this option, a file will
### be create in the var/run directory with the private name of the daemon and
### the host name and port where Spread is running. This information can be
### used by client applications to send information to this daemon. Possible
### values are: yes and no. Default is no.
### Activate this option if you has plans to use the Web interface to AdGrid.
client.support.enabled = yes

[membership]
## Site name. It should be mapped to a valid site numeric identifier (NID) in
### the global configuration file (adgrid.conf).
site.name = GRyCAP

### Neighborhood NID, starting by 0. It should be less than the maximum number
### of neighborhoods per Grid site defined by 'n.neighborhoods.max' in the
### global configuration file (adgrid.conf).
neighborhood.nid = 1
```

ANEXO 4

Requerimientos de QoS, a nivel de servicio, de los servicios Grid, así como los requerimientos a nivel de usuario que acompañan a las solicitudes de trabajo.

La Sección A.4.1 muestra un ejemplo de un documento de requerimientos de servicio, estos son los indicadores de disponibilidad y prestaciones (indicadores estáticos) que serán monitorizados para un determinado servicio Grid. La Sección A.4.2 muestra un ejemplo de un documento de condiciones de solicitud de servicio, este documento acompaña la solicitud del usuario y contiene los requerimientos a nivel de usuario. Por último, la Sección A.4.3 muestra un ejemplo de un documento de condiciones de disponibilidad de recursos, que también acompaña la solicitud de usuario, pero en este caso contiene los límites (mínimo y máximo) de la disponibilidad de recursos computacionales (indicadores dinámicos) de los nodos de trabajo.

A.4.1 REQUERIMIENTOS DE SERVICIO

```
<?xml version="1.0" encoding="UTF-8"?>
<qos:requirements version="1.5"
    xmlns:qos="http://upv.org/adgrid/domain/qos_requirements "
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://upv.org/adgrid/domain/qos_requirements
../schemas/qos_requirements.xsd ">
    <qos:refresh-interval>
        <!-- renew this registration every 10 minutes -->
        <qos:subscription-renew>600</qos:subscription-renew>
        <qos:state-renew xsi:nil="true" />
    </qos:refresh-interval>
    <qos:preferred-record-keeper xsi:nil="true" />
    <qos:constraint>
        <qos:measured-item xsi:type="qos:availability">
            <qos:name>Availability</qos:name>
            <qos:description>
                Availability of the service end point.
            </qos:description>
            <qos:target>
                <qos:type>Endpoint</qos:type>
                <qos:id />
            </qos:target>
            <qos:sli-percent>
                <qos:units>%</qos:units>
                <qos:threshold superior="false">99</qos:threshold>
            </qos:sli-percent>
            <qos:collectible>average</qos:collectible>
```

```
            <qos:contrastable>true</qos:contrastable>
            <qos:time-slide>1</qos:time-slide>
            <qos:configurations />
            <qos:test>
                <qos:test-operation>testOperation</qos:test-operation>
                <qos:message-in>
                    <![CDATA[
                    <input-arguments>
                        <measurable>Availability</measurable>
                        <size>1000</size>
                    </input-arguments>
                    ]]>
                </qos:message-in>
                <qos:message-out>
                    <qos:expected-result xsi:nil="true" />
                    <qos:expected-status-code>
                        ALL_OK
                    </qos:expected-status-code>
                </qos:message-out>
                <qos:result-on-error>0.0</qos:result-on-error>
                <!-- Execute this test every 1 hours -->
                <qos:execIntervalSecs>3600</qos:execIntervalSecs>
                <qos:preferred-tester xsi:nil="true" />
            </qos:test>
        </qos:measured-item>
    </qos:constraint>
    <!--
        Notice that this response time constraint uses a test operation
        instead of an notifiable operation as the other response time
        constraints do.
    -->
    <qos:constraint>
        <qos:measured-item xsi:type="qos:response-time">
            <qos:name>ResponseTime</qos:name>
            <qos:description>
                Response Time for the Test operation.
            </qos:description>
            <qos:target>
                <qos:type>Operation</qos:type>
                <qos:id>Test</qos:id>
            </qos:target>
            <qos:sli-time>
                <qos:units>seconds</qos:units>
                <qos:threshold superior="true">10</qos:threshold>
            </qos:sli-time>
            <qos:time-slide>1</qos:time-slide>
            <qos:test>
                <qos:test-operation>testOperation</qos:test-operation>
```

```
                <qos:message-in>
                    <![CDATA[
                    <input-arguments>
                        <measurable>ResponseTime</measurable>
                        <size>0</size>
                        <test>ADGRID_TEST|3|pataa|Q6L6Q3|1|0.1</test>
                    </input-arguments>
                    ]]>
                </qos:message-in>
                <qos:message-out>
                    <qos:expected-result xsi:nil="true" />
                    <qos:expected-status-code>
                        ALL_OK
                    </qos:expected-status-code>
                </qos:message-out>
                <qos:result-on-error>Double.MAX_VALUE</qos:result-on-error>
                <!-- Execute this test every 1 hours -->
                <qos:execIntervalSecs>3600</qos:execIntervalSecs>
                <qos:preferred-tester xsi:nil="true" />
            </qos:test>
        </qos:measured-item>
    </qos:constraint>
    <qos:constraint>
        <qos:measured-item xsi:type="qos:security">
            <qos:name>Security</qos:name>
            <qos:description>
                Security for Test operation.
            </qos:description>
            <qos:target>
                <qos:type>Operation</qos:type>
                <qos:id>Test</qos:id>
            </qos:target>
            <qos:sli-boolean>
                <qos:units>boolean</qos:units>
                <qos:threshold superior="false">1</qos:threshold>
            </qos:sli-boolean>
            <qos:time-slide>1</qos:time-slide>
        </qos:measured-item>
    </qos:constraint>
</qos:requirements>
```

A.4.2 CONDICIONES DE SOLICITUD DE SERVICIO

```
<?xml version="1.0" encoding="UTF-8"?>
<qrr:request-requirements version="1.1"
    xmlns:qrr="http://upv.org/adgrid/domain/qos_requirements-request"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://upv.org/adgrid/domain/qos_requirements-request
../schemas/qos_requirements-request.xsd ">
    <qrr:constraint>
        <qrr:measured-item>
            <qrr:name>Availability</qrr:name>
            <qrr:measurable>sli-percent</qrr:measurable>
            <qrr:units>%</qrr:units>
            <qrr:threshold superior="false">90</qrr:threshold>
            <qrr:opossite_limit_considered xsi:nil="true" />
        </qrr:measured-item>
    </qrr:constraint>
    <qrr:constraint>
        <qrr:measured-item>
            <qrr:name>ResponseTime</qrr:name>
            <qrr:measurable>sli-time</qrr:measurable>
            <qrr:units>seconds</qrr:units>
            <qrr:threshold superior="true">30</qrr:threshold>
            <qrr:opossite_limit_considered xsi:nil="true" />
        </qrr:measured-item>
    </qrr:constraint>
</qrr:request-requirements>
```

A.4.3 CONDICIONES DE DISPONIBILIDAD DE RECURSOS

```
<?xml version="1.0" encoding="UTF-8"?>
<rcd:resource-conditions version="1.1"
    xmlns:rcd="http://upv.org/adgrid/domain/resource-conditions"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://upv.org/adgrid/domain/resource-conditions
../schemas/resource-conditions.xsd ">
    <rcd:primary-purpose>CPU-INTENSIVE_APPS</rcd:primary-purpose>
    <rcd:system-architecture>
        <rcd:os-name>Linux</rcd:os-name>
        <rcd:os-arch>i386</rcd:os-arch>
        <rcd:os-version>2.6.24</rcd:os-version>
        <rcd:cpu-bitness>32</rcd:cpu-bitness>
    </rcd:system-architecture>
    <rcd:system-load>
        <rcd:load-1>
            <rcd:min xsi:nil="true" />
            <rcd:max>800</rcd:max>
        </rcd:load-1>
        <rcd:load-5>
            <rcd:min xsi:nil="true" />
```

```
            <rcd:max>600</rcd:max>
        </rcd:load-5>
        <rcd:load-15>
            <rcd:min xsi:nil="true" />
            <rcd:max>400</rcd:max>
        </rcd:load-15>
    </rcd:system-load>
    <rcd:system-memory>
        <rcd:physical-memory>
            <!-- 64 mb -->
            <rcd:min>65536</rcd:min>
            <rcd:max xsi:nil="true" />
        </rcd:physical-memory>
        <rcd:swap>
            <!-- 256 mb -->
            <rcd:min>262144</rcd:min>
            <rcd:max xsi:nil="true" />
        </rcd:swap>
    </rcd:system-memory>
</rcd:resource-conditions>
```