Universidad de Lleida
Escuela Politécnica Superior
Máster en Ingeniería de Software Libre

Trabajo Final de Máster

CoDiP2P: Sistema de Computación Distribuida Peer to Peer

Autores: Ignasi Barri Vilardell
Josep Maria Rius Torrentó

Directores: Fernando Guirado Fernández
Francesc Solsona Tehàs

Marzo de 2008

Prólogo

Los chips actuales se están aproximando a los límites teóricos de capacidad de cómputo máxima, es por eso, que el mundo de la computación tiende a distribuir los sistemas para permitir seguir creciendo. Éste es uno de los puntos más populares dentro del mundo de la investigación en ciencias de la computación, ya sea aumentando el número de chips por sistema o agrupando distintos sistemas para trabajar de forma conjunta. En la actualidad, existen dos grandes grupos: memoria distribuida y memoria compartida.

Las dos alternativas implican el nacimiento de nuevos paradigmas de programación respecto al modelo secuencial tradicional que permiten la división de grandes problemas en pequeñas partes, para que puedan ser tratados individualmente para cada nodo del sistema (algoritmos paralelos distribuidos).

Éste trabajo final de máster se centra en la agregación de distintos recursos para formar un sistema colaborativo capaz de resolver problemas computacionalmente intratables para sistemas tradicionales. La potencia del sistema reside en el número de nodos capaces de trabajar unidos, por éste motivo la heterogeneidad de los recursos tiene un peso muy importante. Además, la colaboración entre distintas organizaciones y usuarios en distintas localizaciones da un valor añadido al proyecto.

Sin embargo, otros proyectos son extremadamente complejos debido a la complicidad de unir distintas organizaciones y los problemas de seguridad implicados.

El proyecto se basa en la creación de una red global de recursos donde todos los nodos son tratados por igual siguiendo la filosofía emergente de las arquitecturas peer-to-peer, que permite la comunicación entre los recursos utilizando distintas plataformas, etc. Dejando de banda las arquitecturas más tradicionales, donde cada nodo es tratado de forma muy distinta, incluso donde determinados roles solo eran aptos para que según que tipo de hardware, estos tipos de arquitecturas permiten una organización más dinámica y efectiva, ampliando posibilidades de éste sistema de cómputo distribuido.

Siguiendo la filosofía de otros sistemas peer-to-peer como los de compartición de ficheros, el sistema trabaja en segundo plano, de éste modo, se consigue que el usuario no sea consciente de el trabajo que realiza la aplicación mientras ejecuta tareas aprovechando los tiempos de inactividad de la máquina.

A diferencia de otros sistemas de computación distribuida entre iguales ya desarrolladas como *Boinc* o *SETI@home*, los cuales solo permiten ofrecer recursos para aplicaciones especificadas por el proveedor, éste sistema ofrece al usuario la posibilidad de ejecutar tareas propias, convirtiendo la alternativa que se presenta como una plataforma de cómputo distribuido totalmente peer-to-peer.

Para optimizar los costes de mantenimiento de la red, los nodos de auto-organizan siguiendo distintas políticas. En éste trabajo se muestran distintas políticas para mantener la integridad y conectividad del sistema, el mantenimiento de la red y finalmente distintas políticas que permiten ejecutar las tareas deseadas.

En definitiva, CoDiP2P plantea como una alternativa real a las aplicaciones tradicionales como *Grid* o *SETI@home*, dando al usuario doméstico la potencia de cálculo de grandes data-centers.

Íñigo Goiri Presa

Agradecimientos

Todo trabajo es complejo de realizar, pero en este caso, el apoyo recibido ha sido abundante. La ayuda que se nos ha prestado, ha permitido realizar el trabajo que ahora mismo están leyendo. Por tanto, agradecer a aquellas personas que han ayudado a que la realización de este trabajo haya sido lo más placentera posible.

Recordar a Íñigo Goiri Presa, que conjuntamente con Josep Rius Torrentó, crearon el primer prototipo CompP2P, padre de la actual aplicación, y por tanto, una fuente de inspiración importante.

Sin la colaboración de nuestros colegas Miquel Orobitg Cortada y Albert Guim Mas el diseño, implementación y especificación de la plataforma no hubiera sido tan rápida y eficaz.

La inestimable ayuda de Damià Castellà Martínez en el campo de la simulación de la aplicación, nos ha ayudado enormemente a la hora de tomar decisiones respeto a la implementación.

Tener en cuenta la dirección magistral del proyecto por parte de de los directores del proyecto Dr. Fernando Guirado Fernández y Dr. Francesc Solsona Tehàs.

Hacer mención especial al apoyo recibido en materia técnica y moral al conjunto de profesores de el Grupo de Computación Distribuida de la Escuela Politécnica Superior de la Universidad de Lleida: Dr. Francesc Giné de Sola, Dr. Fernando Cores Pardo, Dra. Concepció Roig Mateu y el profesor Josep Lluís Lérida Monsó.

La colaboración de Héctor Blanco de Frutos, que ha apuntado consejos importantes a la implementación del proyecto.

Hacer referencia a la ayuda prestada por la consultora externa Indra, que mediante Oriol Arbonés Liñán ha aportado un punto de vista importante para la realización del proyecto.

Y como no, a nuestras familias. Nuestros padres, hermanos y compañeras, porque a pesar de nuestra importante dedicación y esfuerzo en estos últimos meses al proyecto, siempre han podido percibir que ellos, para nosotros, son lo más importante.

Índice general

Índice de figuras

Capítulo 1

Introducción

Actualmente, las necesidades de cálculo crecen día tras día. Distintos ámbitos de la ciencia (climatología, medicina, física...) plantean problemas que requieren gran capacidad de cómputo que gestionan cantidades enormes de datos gracias a puntos de intersección o unión de varios elementos que confluyen en el mismo lugar, también llamados nodos. En determinadas aplicaciones, la potencia y los recursos necesarios para solucionar estos problemas resultan intratables desde un único nodo de procesamiento.

La creación de supercomputadores tienen sus limitaciones, entre ellas, destacar sus precios elevados y su poca flexibilidad. Para solucionar estas limitaciones han surgido varias tendencias dentro de la computación distribuida; estas son:

1. **Clúster:** emular un supercomputador a partir de un conjunto heterogéneo de computadores conectados entre ellos mediante una LAN[1].

2. **Grid:** utilizar muchos computadores domésticos que realizarán tareas de cómputo que serán servidas a través de Internet a un servidor. De ésta forma se consigue simular un gran supercomputador.

3. **Peer-to-Peer:** es una red informática constituida entre iguales (en inglés peer-to-peer -que se traduciría de par a par- o de punto a punto, y más conocida como P2P) se refiere a una red que no tiene clientes ni servidores fijos, sino una serie de nodos que se comportan simultáneamente como clientes y como servidores de los demás nodos de la red.

El sistema que se plantea en este trabajo está basado en la tercera opción. Se trata de una opción que nos permite lograr una mayor potencia de cálculo y disponer de un amplio abanico de recursos más amplios y masivos.

Para ser consciente de las posibilidades que ofrecen estos tipos de cómputo distribuido, solo es necesario reflexionar con un pequeño ejemplo: muchos de los computadores personales tienen una capacidad de cómputo o cálculo superiores a los requerimientos computacionales de sus usuarios. Actualmente, un equipo de usuario estándar dispone de nos núcleos de 2 GHz de procesador y 2 GB de memoria RAM. Durante la mayoría de tiempo, solo se utiliza la mitad de esta capacidad.

[1] Local Area network.

Por lo tanto, si se pudieran aprovechar los recursos de 100 máquinas como las mencionadas, se dispondría de un total de 200 GHz de procesamiento y hasta 100 GB de almacenamiento RAM. Esto pone de manifiesto la potencia de la cual se podría sacar provecho si se utilizasen todos estos recursos ociosos de los computadores. Es por este motivo, que este TFM (Trabajo Final de Máster) intenta modelizar un mecanismo que intente sacar provecho a la baja utilización de los recursos de cómputo.

En este capítulo se comentan todos los conceptos necesarios para poder entender sin problemas el resto del trabajo. Entre muchos conceptos, se definirá la computación distribuida (por ejemplo, tipos y características). De forma adicional, se introduce el funcionamiento de las aplicaciones que funcionan bajo la filosofía *peer-to-peer*, por último, se da una visión más detallada de los sistemas de cómputo distribuido, especialmente los que están basados en el framework de *JXTA*.

1.1. Contenido

En este apartado se realiza una breve descripción del resto de capítulos de la memoria. La relación de capítulos y secciones de este trabajo es la siguiente:

- **Capítulo 1 Introducción**: En éste capítulo, se realiza una breve introducción a conceptos y estado actual que constituye el ámbito del trabajo, así como la tecnología utilizada. También se definen los objetivos del trabajo y la metodología seguida para el cumplimiento de los objetivos marcados.

- **Capítulo 2: Estado del arte**. En este capítulo, se comenta el objetivo y la necesidad de estudiar las plataformas existentes, sus puntos a favor y en contra, para definir claramente cuales son los competidores de nuestra idea (si existen), para facilitar de este modo, la posterior comparación de resultados de rendimiento.

- **Capítulo 3: Punto de partida**. En este capítulo, se especifica el primer prototipo que si hizo en el año 2006 llamado *CompP2P*. En esta parte del trabajo se realiza un breve resumen de las partes más significativas del modelo anterior.

- **Capítulo 4: Modelo del Sistema**. Es el cuerpo del trabajo y uno de los puntos más importantes del mismo, ya que se explica detalladamente las decisiones tomadas en el diseño del modelo y el porqué de dichas decisiones, así como los algoritmos que nacen de estas ideas. Dentro de este capítulo, destacar las secciones donde se comentan las decisiones de diseño tomadas a lo largo del desarrollo y también la sección dedicada a los distintos algoritmos que se proponen en el sistema. Se repasarán cada uno de ellos mostrando distintas formas de modelizarlo y el porqué de su existencia.

- **Capítulo 5: Simulación**. Capítulo destinado a detallar la simulación del modelo y a interpretar los resultados obtenidos.

- **Capítulo 6: Trabajo Futuro**. Se reflexiona sobre las posibles mejoras que se pueden introducir a el trabajo propuesto.

- **Capítulo 7: Conclusiones**. Se exponen las conclusiones extraídas de la realización del trabajo.

1.2. Contexto actual

Actualmente hay distintos entornos de computación distribuida que utilizan Internet, como puede ser Grid. Este tipo de entorno presenta ciertos inconvenientes; por ejemplo la dificultad de instalación y una jerarquía demasiado estricta.

Por este motivo nace la necesidad de desarrollar un modelo de cómputo distribuido, fácil de instalar, sin que incorpore jerarquías excesivas en la red. Esta filosofía de trabajo es la que presentan las aplicaciones que siguen una arquitectura peer-to-peer (entre iguales).

Actualmente las aplicaciones basadas en arquitecturas entre iguales se usan en un alto porcentaje, para la compartición de datos y no tanto para el cómputo. Las alternativas en este campo son escasas. De hecho, las más destacadas están destinadas a tareas concretas, perdiendo de este modo, el sentido de la computación entre iguales.

Otros sistemas de computación actuales basados en el cálculo ejecutado por peers, no son muy usables para los usuarios, ya que solo permite la cesión de los recursos y no su aprovechamiento. Para intentar resolver estos puntos, se ha realizado una arquitectura peer-to-peer utilizando la plataforma JXTA, pensada para diseñar e implementar, este tipo de entorno de trabajo.

Este trabajo pretende dar una alternativa distinta a la computación distribuida que se utiliza comúnmente, modelando un sistema fácil de ejecutar para cualquier máquina del mundo y sin necesidad de amplios conocimientos en el campo de la computación para ejecutar con suma facilidad el aplicativo. De esta forma, se consigue un sistema entre iguales y usable por todo el mundo.

1.3. JXTA

JXTA[3], proviene de la palabra Juxtapose, es una plataforma peer-to-peer de código abierto creada por Sun Mycrosystems en febrero de 2001 con la colaboración de investigadores de varias instituciones académicas. Esta plataforma define un conjunto de protocolos basados en *XML* que permite a cualquier tipo de dispositivo conectado a la red peer-to-peer, intercambiar mensajes y colaborar con independencia de la topología de la red. JXTA es el *framework* para redes de peers más maduro que existe en la actualidad. Fue diseñado para dar soporte a un amplio abanico de dispositivos -PCs, servidores, teléfonos móviles, PDAs- permitiendo una comunicación entre ellos de forma descentralizada.

JXTA está basado en un conjunto de protocolos abiertos, por lo tanto, permite que sea portado por cualquier lenguaje de programación moderno. La versión implementada con el lenguaje Java es la más estable de todas, aunque existen versiones con otros lenguajes como C/C++.

Los peers de JXTA crean un entorno de red virtual que permite a un peer interactuar con el resto de peers directamente incluso cuando estos se encuentran detrás de firewalls y enrutadores o utilizan distintos protocolos de transporte. Además, cada recurso se identifica por un único identificador o ID: 160 bits SHA-1 URN. Esto implica que un peer puede cambiar su ubicación y mantener su identificador.

Además de los identificadores, JXTA tiene otras funcionalidades básicas como el descubrimiento de peers, los grupos de peers, y los pipes.

Remarcar que el hecho que sea un framework implementado con código abierto, permite que los desarrolladores de aplicaciones peer-to-peer pueden utilizar todos los servicios de forma gratuita y por tanto, también es posible contribuir con el proyecto en tareas como la detección de errores, la mejora de los protocolos JXTA o la traducción de manuales.

1.3.1. Conceptos de JXTA

En este apartado se exponen los conceptos principales que se utilizan en JXTA para crear una red de peers.

1.3.1.1. Endpoints

Un *endpoint* se puede definir como el emisor o receptor de un conjunto de datos que se transmiten sobre la red de peers. Un *endpoint* proporciona abstracción sobre las interfaces de red usadas para enviar y recibir datos olvidándose de detalles como protocolos inferiores o arquitectura del sistema.

1.3.1.2. Advertisement

Un *advertisement* (o anuncio), es una manera de presentar al resto de la red los recursos disponibles y su ubicación. Los anuncios permiten simplificar la tarea de organizar las redes de peers.

Formalmente, un anuncio se define como una representación estructurada de una entidad, servicio o recurso disponible por parte de un peer o un grupo de peers como parte de la red peer-to-peer.

Todos los recursos utilizados por JXTA pueden ser representados como un anuncio, incluyendo peers, grupos de peers, pipes, endpoints, servicios, etc...

1.3.1.3. Pipe

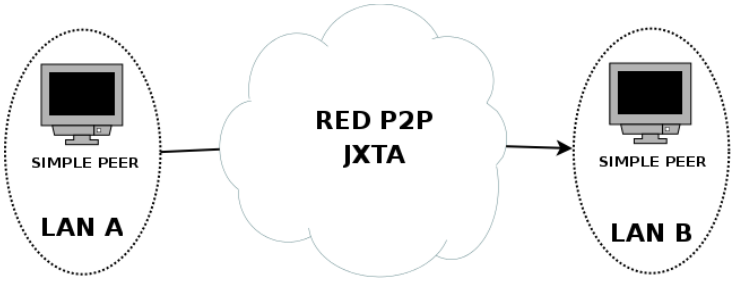

Figura 1.1: Funcionamiento de un *pipe* en JXTA.

Un *pipe* se define como un objeto de un sistema operativo utilizado para conectar la salida de un proceso con la entrada de otro. En JXTA, un *pipe* es un canal virtual de comunicaciones unidireccional y asíncrono que conecta dos o más *endpoints*. Los *pipes* pueden disponer de otros mecanismos adicionales como el cifrado.

Los *pipes* constituyen la herramienta principal para la comunicación de una red peer-to-peer implementada con JXTA.

En la figura 1.1 se muestra las posibilidades que ofrece JXTA para establecer comunicaciones. Gracias a JXTA el programador puede olvidarse de identificar un peer o tener en cuenta la red en la que se encuentra, estableciendo únicamente los *endpoints* (los peers que quieren establecer la comunicación). El sistema propagará los mensajes teniendo en cuenta los routers, firewalls y otros obstáculos existentes en la comunicación.

1.3.1.4. Peers

Un *peer* es un nodo dentro de una red peer-to-peer que forma la unidad de procesamiento fundamental en cualquier solución peer-to-peer. Hasta ahora, se tenía que definir un peer como una aplicación funcionando en solo un computador conectado a una red como Internet, pero ésta definición limitada, no incluye la verdadera función que tiene un peer, ya que descarta la posibilidad de que un peer pueda ser una aplicación distribuida entre distintas máquinas o que pueda ser un pequeño dispositivo, como una PDA, la cual se conecta a una red indirectamente. Desde este punto de vista, una sola máquina puede soportar múltiplos peers al mismo tiempo.

Para sintetizar todas estas definiciones, en este trabajo, se va a redefinir el concepto de peer como:

"Cualquier entidad capaz de realizar *trabajo útil* y de comunicar los resultados, directa o indirectamente, a otra entidad sobre una red."

La definición de *trabajo útil* depende del tipo de peer que sea. Existen tres tipos de peers en una red peer-to-peer de JXTA:

1. **Simple Peers**: Un simple peer está diseñado para servir a un solo usuario, habilitando la posibilidad de proveer servicios de su dispositivo y consumir los que otros peers de la red le provean. Como norma general, un simple peer está situado detrás de un cortafuegos, separado de la red. Los peers situados delante del cortafuegos probablemente no serán capaces de comunicarse directamente con un simple peer situados detrás del mismo. Debido a su accesibilidad limitada en la red, los single peers tienen la responsabilidad más baja en las redes peer-to-peer JXTA.

2. **Rendezvous Peers**: En general, un rendezvous es una puerta o punto de encuentro; en peer-to-peerP2P, un peer rendezvous ofrece a otros peers con una localización de la red usada para descubrir otros peers y sus recursos. El rendezvous peer almacena la información de otros peers y las publica para poner a disposición dicha información al resto de peers de la red.

   Un peer rendezvous puede aumentar su capacidad, almacenando información sobre los peers para que pueda hacer de la misma un uso futuro. Este tipo de peers, tienen el potencial de

mejorar el cortafuegos, reducir el tráfico de la red y de proporcionar un mejor servicio a los simple peers.

Tal y como se aprecia en la figura 1.2, un peer rendezvous se encarga de propagar los mensajes de la red local a otros equipos del exterior. Esto permite que con un único peer con salida al exterior da acceso al resto de peers, ya que actúa como intermediario

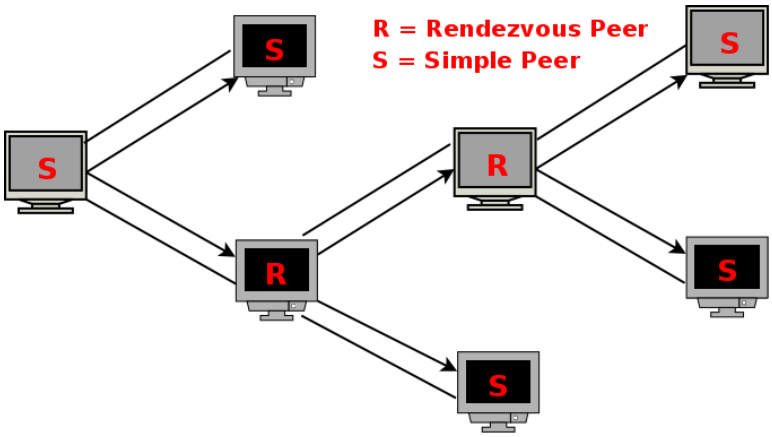

Figura 1.2: Rendezvous Peer propagando mensajes.

Un peer rendezvous existirá generalmente, delante de un cortafuegos en la red. En algunos casos también es posible que exista detrás, pero tendría que ser capaz de atravesarlo usando un protocolo autorizado por el cortafuegos o un router peer de fuera.

3. **Router Peers**: Un router peer (o relay peer) proporciona un mecanismo al resto de peers de la red para que se puedan comunicar con el resto de peers que se hallan fuera de la red, debido a una separación a causa de la existencia de un cortafuegos o de algún tipo de mecanismo *NAT*[2].

   Para enviar un mensaje a un peer mediante un router, el emisor debe determinar qué router usará para comunicarse con el peer destino. Esta información de enrutamiento proporciona un mecanismo al sistema peer-to-peer para substituir el *DNS*[3] tradicional, de tal forma, que habilita las conexiones de dispositivos que tienen una dirección *IP* dinámica, ya que posibilita encontrar de forma dinámica estos elementos dentro de la red.

   En la figura 1.3 se muestra cómo mediante un router peer (o relay peer) se puede lograr que toda una red se pueda comunicar con el exterior.

[2] *Network Address Translation.*
[3] *Domain Name Server.*

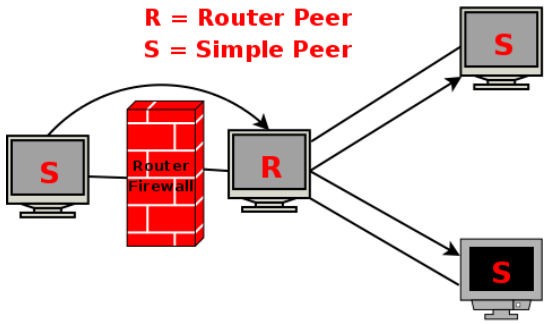

Figura 1.3: Router Peer enviando mensajes a través de un firewall.

1.3.1.5. Peer Groups

Antes de la aparición de JXTA, la naturaleza de las soluciones peer-to-peer y de sus protocolos asociados consistía en dividir el espacio de la red según sus finalidades o usos. Si se quería realizar algún tipo de compartición de archivos, se usaba el protocolo de *Gnutella* que simplemente permitía la comunicación con otros peers que usasen el mismo protocolo. De forma similar, si el que se pretendía era realizar mensajería instantánea, se usaba *ICQ*, que solo funcionaba entre peers del mismo protocolo.

Las incompatibilidades de los protocolos reducen la eficacia de uso de la red por parte de los peers. Si se considera un sistema peer-to-peer en el cual los clientes entienden el mismo sistema de protocolos, tal y como sucede en JXTA, el concepto de peer group es necesario para subdividir el espacio de la red. Por tanto, se define un peer group como:

"Un sistema de peers que tienen un interés común o una meta dictada por los peer implicados. Los peer groups pueden proporcionar servicios a sus peers inaccesibles por otros peers en la red peer-to-peer."

Los distintos motivos por los cuales la plataforma JXTA divide la red de peers en grupos son:

- **La aplicación por la que tienen que colaborar.** Un peer group se crea para dar un servicio de intercambio a los miembros que no deseen tener disponibilidad para toda la red peer-to-peer. Una razón para llevar a cabo esta separación en grupos, puede ser en casos en que la privacidad de los datos usados por la aplicación distribuida es alta.
- **Los requisitos de seguridad de los peers implicados.** Un peer group puede usar servicios de autenticación para restringir el acceso al mismo.
- **La necesidad de la información del estado sobre los miembros del grupo.** Los miembros de un peer group pueden supervisar a otros miembros. La información de estado ayuda a mantener unos niveles mínimos de servicio para la aplicación del peer group en cuestión.

Los miembros del peer group pueden proporcionar acceso redundante a un servicio, asegurando de esta manera que siempre esté disponible el servicio.

1.4. Objetivos del trabajo

El objetivo del trabajo es modelizar un sistema de computación distribuida peer-to-peer, bien documentado, flexible, con una base modularizada apta para su ampliación, robusto, eficaz, eficiente y multiplataforma.

Por tanto, el reto principal recae en el diseño y modelización de una arquitectura peer-to-peer de cómputo distribuido. Con este tipo de arquitectura se puede llegar a obtener un supercomputador de coste nulo, ya que aprovecha recursos disponibles.

Uno de los puntos fuertes de un sistema peer-to-peer tiene que ser la popularidad del mismo, es decir que tenga una aceptación grande de los usuarios para tener el máximo alcance posible. Es por esto que es interesante desarrollar la arquitectura usando el lenguaje Java para poder realizar un sistema basado en peers lo más grande y heterogéneo posible. En este punto también entran en juego las facilidades que se brindan al usuario, ya que un sistema que precise demasiadas tareas técnicas (compilación, gestión de demasiados ficheros...) no sería utilizado por su manejabilidad compleja, es por eso, que ha de ser fácil de instalar, manejar y utilizar.

Para lograr un buen sistema de cómputo, éste ha de ser tolerante a los fallos; esto significa que si cualquier peer del sistema se desconecta de forma esperada o inesperada, el sistema tiene que repararse en caso de existencia de daño y conservar la información delicada para asegurar el correcto funcionamiento global.

A continuación se detallan los objetivos específicos que el diseño y modelación de la aplicación han de cumplir.

1. Los proyectos que se han realizado en el campo de la computación distribuida entre iguales, no están demasiado documentados y solamente hay latentes pequeñas pinceladas de su funcionamiento. Por eso, nace la necesidad de documentar de forma detallada este proyecto.

2. Teniendo en cuenta que las intenciones puestas en este proyecto es de una continuidad y innovación permanente, se tiene que proponer un diseño modular, para facilitar la modificación y ampliación del sistema con suma facilidad.

3. La capacidad de cálculo del sistema tiende a ser directamente proporcional al número de usuarios del mismo, por eso, cuantos más usuarios haya, se tiende a una capacidad mayor de cómputo. Por este motivo surge la necesidad de llegar al mayor número de usuarios posibles. Esto se puede lograr llevando a cabo una aplicación que se puede ejecutar en arquitecturas o entornos de trabajo dispares, se logrará si se modela una aplicación multiplataforma.

4. Para lograr la independencia del sistema comentada en el punto anterior, se ha escogido el lenguaje de programación Java. Este lenguaje tiene el inconveniente (cada vez menor) de ser menos eficaz respecto a otros lenguajes no compilados sobre máquina virtual. Por tanto, el sistema modelizado tendrá que ser lo más eficiente posible para poder vencer el hándicap comentado.

5. El sistema tiene que ser escalable, si se logra un sistema integrado por usuarios de dispares entornos de hardware y software, puede repercutir en el número de usuarios finales del aplicativo.

Una vez desarrollada toda la aplicación, se tiene que comprobar el correcto funcionamiento del modelo en un entorno de simulación, que será introducido en el capitulo 5.

Todos estos puntos tienen el objetivo final de crear un entorno de trabajo no realizado hasta el momento y que puede suponer una fuerte innovación en el campo de la computación distribuida, abriendo una nueva área de investigación con grandes posibilidades de futuro.

1.5. Enfoque y metodología

Las pruebas o experimentación efectuadas con el simulador, se han efectuado con un software de simulación de plataformas Grid llamado *SimGrid*. Mediante esta simulación, se ha podido extraer valoraciones que han sido fundamentales para la implementación del modelo.

Con esta metodología se logra optimizar las tareas de implementación, ya que se implementa en función de los resultados obtenidos de la simulación, hecho que asegura la fiabilidad y correcto funcionamiento de la implementación.

Para cumplir con el objetivo de desarrollar un diseño estructurado, se ha decidido por un diseño de módulos independientes. Con esta decisión, es posible implementar un módulo por completo, sin necesidad de tener implementado ningún otro módulo. La metodología que se ha seguido en todas las fases de desarrollo de la aplicación ha sido totalmente orientada a objetos, para lograr un sistema poco acoplado y fuertemente cohesionado.

Las herramientas de trabajo han sido múltiples, a continuación vamos a detallar una lista de las más significativas:

- Sistema Operativo: K-Ubuntu 7.10.
- Compilador: Java versión 6.0.
- Editor de código Java: NetBeans 6.0.
- Gestor de versiones: Subversion 1.4.3.
- Gestor del proyecto vía web: Trac 0.10.3.
- Servidor web: Apache 2.2.4.
- Lenguaje de la documentación del código: JavaDoc.
- Lenguaje de la memoria: LaTeX.
- Editor del lenguaje de la memoria: Kile.
- Editor de imágenes de la memoria: DIA.

Capítulo 2

Estado del Arte

Estado del Arte es un capítulo dedicado a repasar el estado actual de los sistemas distribuidos peer-to-peer, así como el tipo de redes existentes en este entorno.

Debido a la carencia de sistemas de cómputo distribuido peer-to-peer, nos hemos visto obligados a comentar, principalmente, sistemas basados en tablas de hash distribuidas que se centran en la compartición de archivos en redes peer-to-peer, tanto en el repaso a los propios sistemas peer-to-peer, como en el tipo de redes de esta topología. No obstante, vamos a clasificar de forma general, las distintas disciplinas que se pueden hallar dentro de los sistemas peer-to-peer. Éstas disciplinas son:

- Sistemas peer-to-peer de compartición de ficheros: *Napster*, *Gnutella*, *KaZaA*, *aMule*, etc...
- Sistemas peer-to-peer basados en DHTs: *Chord*, *Pastry*, *CAN*, *Tapestry*, etc...
- Sistemas peer-to-peer de comunicación: mensajería instantánea (*ICQ*) o voz sobre IP (*Skype*), etc...
- Sistemas peer-to-peer de computación: *seti@home*, *folding@home* (ver figura 2.1), *JNGI*, *JxtaCAT*, etc...
- Sistemas peer-to-peer variados: *PlanetLab*, *Omemo*, etc...

2.1. Sistemas peer-to-peer

Sistemas peer-to-peer, hoy en día, son de los más usados por los usuarios, existen multitud de aplicaciones de uso diario que siguen una filosofía p2p. La mayoría de aplicaciones que siguen esta filosofía, son de compartición de archivos. La popularidad de estos sistemas, está generando en la actualidad debates morales y legislativos en muchos países acerca de la legalidad o no de su uso, debido a una supuesta violación de los derechos de autor. Sin embargo, y a pesar de las controversias legales y morales, parece ser, que el mercado poco a poco se va decantando por éste tipo de sistemas, y algunas empresas ya están desarrollando aplicaciones de visionado de contenidos bajo demanda, telefonía y otro tipo de servicios fomentados en redes peer-to-peer.

Figura 2.1: Captura de la aplicación *Folding@Home* en una *PlayStation 3* de *Sony*.

Por esta razón, es sumamente importante, conocer los sistemas peer-to-peer que existen en el mercado, para poder tomar nota de los puntos positivos y negativos de ellos y perfilar nuestro sistema teniendo en cuenta el trabajo ya realizado en éste campo.

2.1.1. Sistemas DHT

Para evitar hacer "flooding"[1] por la red con mensages de peticiones, se han propuesto varios métodos de indexación de contenidos, entre los cuales cabe destacar los sistemas basados en tablas de hash distribuidas (*DHT*). En este tipo de sistemas, cada punto de información es asociado a una punto obtenido después de aplicar una función de hash sobre un atributo de dicho objecto, como por ejemplo su nombre. Una vez realizadas estas asociaciones, cada nodo de la red se hace responsable de mantener información sobre un conjunto de puntos con sus correspondientes objectos asociados, así como una lista de todos sus nodos vecinos conocidos. En este tipo de sistemas, una petición se entiende como una búsqueda de una clave dentro de la DHT. Concretamente, cuando un peer recibe una petición, si no tiene almacenado lo que se busca, automáticamente se reenvía la petición a un nodo vecino conocido cuyas claves sean lo más próximas posibles a la que se está buscando.

Con el objetivo de mejorar el rendimiento de la búsqueda, algunos protocolos basados en *DHT* organizan los nodos del sistema de forma estructurada. Por ejemplo, *Chord* organiza sus nodos en forma de circulo virtual, *CAN* por su parte, lo hace dentro de un espacio Cartesiano d-dimensional dividido en zonas del mismo tamaño y cada nodo es responsable tan solo de una de ellas. Otro ejemplo relevante de este tipo de sistemas es *Pastry* y *Tapestry*, que aunque muestran un buen rendimiento y una buena escalabilidad, solo son capaces de gestionar consultas exactas, como por ejemplo peticiones de ficheros asociados a un punto. Además, los mecanismos de hash funcionan bién con identificadores de objectos estáticos como nombres de archivos, pero no son adecuados para la manipulación de objectos con contenidos dinámicos.

[1] *Flood* es un término en inglés que significa literalmente inundación. Se usa en la jerga informática para designar un comportamiento abusivo de la red de comunicaciones, normalmente por la repetición desmesurada de algún mensaje en un corto espacio de tiempo.

A continuación se presentan distintos sistemas peer-to-peer basados en la compartición de contenidos (ficheros de todo tipo).

2.1.1.1. Chord

Chord es un protocolo sencillo y escalable de búsqueda distribuida en redes peer-to-peer que relaciona claves (keys) con nodos. Está diseñado para funcionar en redes descentralizadas (es decir, sin nodos privilegiados), y su funcionamiento permite concluir con un resultado exhaustivo de la búsqueda.

Funcionamiento general

Se considera una red formada por 2^n nodos, que pueden estar activos o ausentes de la red. Dado un conjunto de claves en esa red, el protocolo Chord se encarga de asignar las claves existentes a los nodos activos y mantener esas asignaciones dinámicamente, es decir, a medida que los nodos van entrando y saliendo de la red. El resto de tareas vinculadas a la red -autenticación, almacenamiento de datos, interfaz, etc...- son responsabilidad de los niveles superiores de la arquitectura.

La asignación de identificadores a nodos y claves se realiza mediante una función de hash consistente. En efecto, cuando un nodo o una clave entran en la red, reciben una identificación (id) que se obtiene calculando el hash (SHA-1) de la IP del nodo o del valor de la clave. Con este hashing se aleatorizan los accesos al sistema, de forma que para un atacante resulte difícil conocer nodos o claves consecutivas o de posición estratégica en la red.

Una clave de id "k" se asigna al nodo de id "k" si éste está activo en la red. Si "k" no está, se busca el primer nodo posterior a "k" que esté activo y se le asigna a la clave: este nodo sustituivo se denomina sucesor de "k" (successor("k")).

Cuando "k" se conecte a la red, su nodo sucesor le transferirá las claves que estuvieran destinadas a él. Cuando un nodo abandona la red, transfiere las claves de las que se hacía cargo a su sucesor, es decir, al nodo con id siguiente a la suya. Con estos mecanismos, se garantiza la autorregulación de la red y el mantenimiento de las claves aún en un contexto de movilidad de los nodos participantes.

Chord está diseñado para ser altamente escalable, es decir, para que los cambios en las dimensiones de la red no afecten significativamente a su rendimiento. En particular, si "N" es el número de nodos de la red, su coste es proporcional a $\theta log(N)$.

Puntos débiles

Un problema importante, se localiza en el punto más importante del sistema: la búsqueda de recursos. En un sistema de cómputo distribuido, la búsqueda de recursos está basada en el correcto mantenimiento de los nodos sucesores, ya que esto, garantiza que las búsquedas se procesan de forma exhaustiva. Sin embargo, cada nodo almacena una cierta información adicional sobre la red que permite acelerar las búsquedas.

Esta información adicional se reduce a unos pocos nodos activos de la red, y se refleja en una tabla de rutas (finger table) interna. En la i-ésima fila de esta tabla consta el sucesor del nodo situado a distancia $2^{(i-1)}$ del nodo correspondiente (en dirección negativa en el círculo). De esta forma, cada nodo tiene un conocimiento más exhaustivo de sus homólogos más cercanos (aproximadamente, si N es el número de nodos totales, almacena información sobre $\theta(log(N))$).

Cuando un nodo solicita una clave "j", examina su propia tabla de rutas; si encuentra el nodo responsable de "j", envía la petición directamente al nodo afectado. En caso contrario, pregunta al nodo cuya id sea más cercana (menor) a "j", que devolverá la id del nodo más cercano a "j" (menor) que encuentre en su tabla de rutas. De esta manera, el nodo remitente obtiene en cada nueva iteración un nodo más cercano a "k", hasta llegar a "k" o a su sucesor. Todo este mantenimiento requiere el desarrollo de una señalización (mensajes entre nodos para mantener actualizadas las tablas de rutas) del orden de $\theta(log(N))^2$.

Este proceso de búsqueda, es eficiente cuando se realiza una consulta exacta de algún recurso en la tabla de hash, no así, cuando se intenta obtener recursos con referencias parciales (*Substring*), ya que en este caso, resultan bastante ineficientes. Por la tanto, una limitación importante, se da en los casos en que no tenemos una idea exacta de qué contenidos o recursos debemos buscar.

Una posible solución, radica en realizar las búsquedas de recursos o contenidos sin exactitud mediante la parametrización de parte del contenido o recursos, es decir, dividir las referencias o parámetros en n partes, por ejemplo:

```
Contenido a buscar: "Beethovens 9th"
Posibles cadenas de búsqueda: "Bee", "eet", "tho", "hov",
"ven", "ens", "ns%", "s%9",
"&9t", "9th"
```

En definitiva diversificar la búsqueda. De ésta forma permite hacer búsquedas con subcadenas y de contenidos parecidos en la tabla de hash. Por otro lado, de ésta solución surgen otros problemas a esta solución:

- La porcionalización de los contenidos provoca la entrada de un mayor número de entradas en el DHT.
- La porcionalización de las búsquedas, pueden dar falsos positivos.
- Al diversificar las búsquedas, también se incrementa la capacidad computacional necesaria para realizarlas.
- Al incrementar las posibles búsquedas, también se incrementa la sobrecarga de las mismas.

Otro problemas en la búsqueda de recursos en éste tipo de sistemas, se debe a que éstas búsquedas son asimétricas, la cual cosa provoca que una búsqueda desde un nodo "n" a un nodo "p" (y viceversa) no siempre realiza el mismo número de saltos. Problema reflejado en la figura 2.2.

El tener un gran anillo de peers, también dificulta las búsquedas de recursos, pues se realizan mediante saltos en el sentido de las agujas del reloj, la cual cosa resulta ineficiente en aquellos casos en que se debe llegar a un nodo próximo antecesor. Problema reflejado en la figura 2.3.

Otro punto negro en estos sistemas, es la seguridad, ya que existen múltiples ataques que se pueden ejecutar en este tipo de arquitecturas si no se incluyen en los mismos mecanismos de seguridad avanzados.

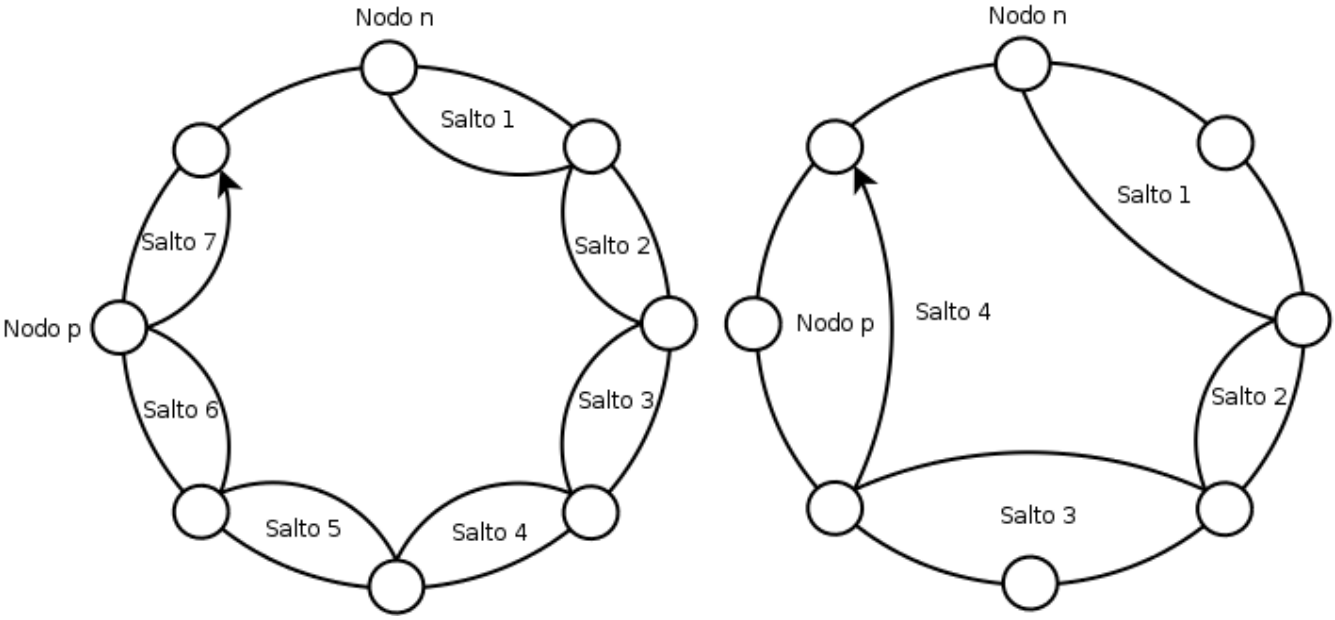

Figura 2.2: Problema de saltos en una búsqueda con Chord.

2.1.1.2. CompuP2P

CompuP2P es una aplicación ligera creada para formar una arquitectura peer-to-peer. Los nodos del sistema se unían a la red creada para realizar cómputo de forma distribuida.

Resaltar que a diferencia de *Chord*, *CompuP2P* es un aplicativo claramente enfocado al cómputo y no a el intercambio de ficheros. Por tanto, podemos efectuar comparaciones más cercanas entre el modelo que se presenta en éste TFM y el propio *CompuP2P*.

Funcionamiento general

Siguiendo en la misma linia de las aplicacions basadas en *DTH*s encontramos *CompuP2P*, cuyo modelo de organización de sus nodos en la red está basado en *Chord*. Esto significa que cada nodo está representado con un identificador único dentro del sistema (*ChordID* y está basado en una estructura organizada. Igual que en *Chord*, *CompuP2P* trata los *IDs* como un espacio de identifiación circular donde cada nodo mantiene actualizada una lista de identificadores con sus respectivas direcciones IP de sus "r" sucesores inmediatos dentro del anillo. Con ésta organización se consigue reducir el número de mensajes necessarios para relizar busquedas dentro del sistemas a un coste $(\theta \log N)$.

Por otro lado, *CompuP2P* establece un sistema de mercados para regular la cesión/petición de recursos. Esto significa que trata a los nodos que ceden sus recursos al sistema como "vendedores" y a los que necessitan de los recursos de otros pera ejecutar ciertos cálculos como "compradores". Mediante el uso de ciertas teorias económicas, el sistema consigue auto-regularse para controlar un intercambio "justo" de recuros.

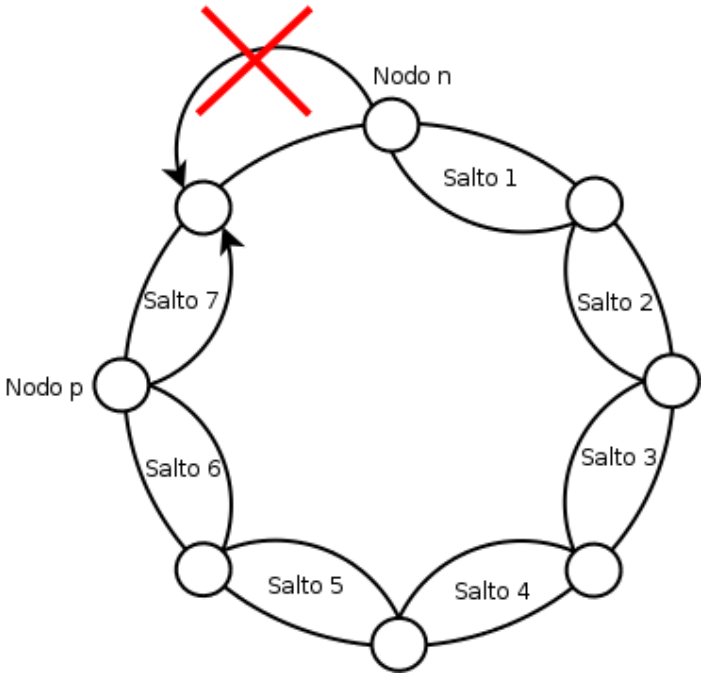

Figura 2.3: Problema de recorridos largos en búsquedas de nodos próximos con Chord.

Puntos débiles

El principal problema, es que al configurar éste mercado sobre su estructura de red basada en *Chord* es muy costoso mantener la información del sistema actualizada en todo momento, ya que esto implicaría que cada vez que los recursos de un nodo se ven alterados hay que notificar este cambio a todo el sistema, con el gran coste que esto implica.

Para solucionarlo, se propone que cada nodo, en el mismo momento de unirse al sistema, realice una estimación de la capacidad de cómputo que va a tener ociosa durante un cierto periodo de tiempo estipulado por el sistema. De esta forma, se consigue tratar lo que en principio sería un recurso dinámico como si fuera un recurso cien por cien estático. Por otra parte, esta solución asume lo sumamente complicado que es el hecho de realizar dicha estimación a la práctica, ya que se supone que los usuarios de estos sitemas peer-to-peer no tienen el uso limitado de su propio ordenador una vez hayan entrado al sistema.

2.1.2. El proyecto JNGI

Actualmente no hay una gran variedad de plataformas peer-to-peer encaradas a la computación distribuida. A continuación se describe un proyecto desarrollado siguiendo esta filosofía, *JNGI*[4].

2.1.2.1. Funcionamiento general

JNGI es una plataforma de computación distribuida creada utilizando las librerías peer-to-peer JXTA. Estas extensiones facilitan a otros desarrolladores la posibilidad de implementar aplicaciones peer-to-peer complejas de forma eficiente. Este proyecto se propuso en el año 2002 con el título

"*Framework for Peer-to-Peer Distribution Computing in a Heterogeneous Decentralized Environment*", es decir, un marco de trabajo para la computación distribuida peer-to-peer para entornos descentralizados y heterogéneos. La meta de JNGI es permitir a un desarrollador con pocos conocimientos de JXTA, elaborar una aplicación con todo el potencial que ofrece JXTA que permite realizar cómputo distribuido con grandes requerimientos de cómputo.

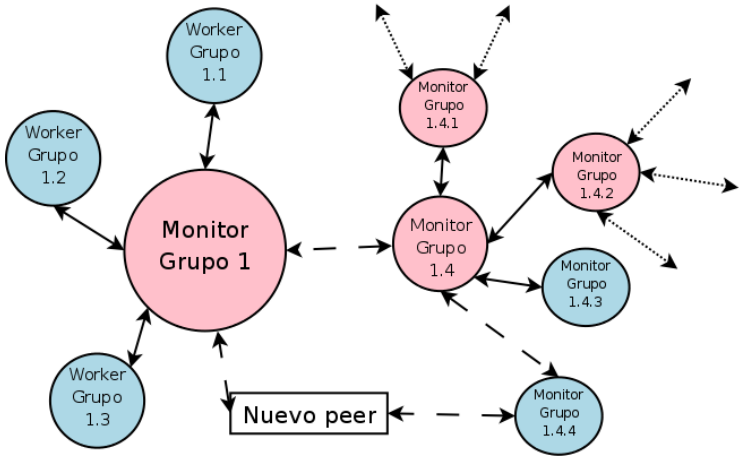

Figura 2.4: Distribución de los peers en JNGI.

JNGI define un *Monitor*, un *Worker* y un *Task Dispatcher peer group*[2]. Los miembros de este grupo se usan para definir los roles de otros peers participantes. Un peer puede ser miembro de una o más instancias en cada grupo de peers en una red JNGI. El *Peer Group Monitor* decide que peticiones de entrada al sistema se aceptan o no. El grupo monitor también determina qué grupos y roles se tienen que asignar a cada peer.

El *Peer Group Monitor* tiene de un miembro subministrador de tareas a la red, que resulta ser como cualquier otro peer del sistema. Una vez determinado el miembro subministrador de tareas, éste empieza a subministrar nuevas tareas formadas por datos y la propia aplicación. El programa adopta la forma de una clase codificada con lenguaje Java (que implementa las interfaces *java.lang.Runnable* y *java.lang.Serializable*) compilada con el código byte[3] estándar. Todos los peers *Worker* involucrados en la misma tarea tienen que ejecutar la misma clase Java. La única diferencia son los datos de entrada. Éstos datos adoptan la forma de instancias serializadas de clases Java. JNGI se refiere a éstas instancias como tareas. Las tareas son instanciadas con datos distintos antes de ser serializadas y distribuidas.

El grupo de peers de distribución de tareas reciben el código byte (o código serializado) y las tareas del subministrador de trabajo. Los peers *Worker* solicitan trabajo constantemente al distribuidor de tareas desde que entran a formar parte de la red JNGI. Una vez el distribuidor tiene un juego de tareas para distribuir, responde los *Workers* con el código serializado, si es necesario, y la tarea. El *Worker* utiliza el código byte para deserializar la tarea y poder tratar el

[2]Grupo de peers de distribución de tareas.

[3]El bytecode es un código intermedio más abstracto que el código máquina.

objeto como un *Thread* normal de Java. El objeto completa su faena y almacena el resultado en una variable. Posteriormente se retorna asíncronamente al grupo distribuidor de tareas. Por otro lado, el *Task Dispatcher group* tiene la capacidad de enviar la misma tarea a múltiples *Workers*. Casos en los que, el *Task Dispatcher group* utilizaría el primer resultado recibido y descartaría el resto.

En el momento que se le asigna un trabajo, el otorgado solicita constantemente al distribuidor tareas finalizadas. Una vez éste último recibe la respuesta de cada tarea, las retorna al sometedor de trabajo con todos los objetos serializados que le hayan retornado los *Workers*. Remarcar también que el que asigna trabajo es responsable de cualquier proceso que pueda ser necesario.

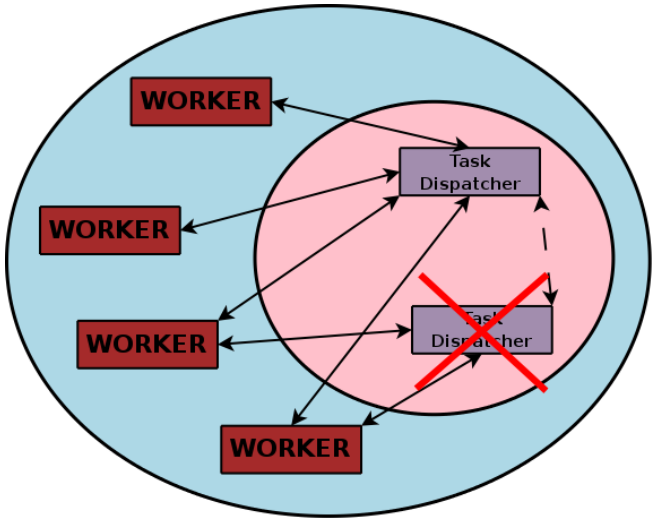

Figura 2.5: Distribución de los grupos en JNGI.

2.1.2.2. Puntos débiles

Es conocido que los sistemas *peer-to-peer* estructurados són más escalables que los no estructurados, en terminos de balanceo de tráfico, pero tienen como inconveniente que necessitan una fuerte capacidad de auto-organización para ser capaces de mantener su rigida estructura. Los sitemas estructurados son más propensos a sufrir fallos en sus nodos, asi como tener que asumir desconexiones inesperadas de algunos otros, pero a pesar de todo esto, proporcionan otras ventajas como mantener la información actualizada del sistema, aplicar mejores políticas de planificación de tareas gracias a esa información, implementar medidas de control sobre los nodos del sistema, etc...

Como ya se ha comentado, *JNGI* hace uso de la plataforma *JXTA* para conectar, comunicar y organizar sus nodos dentro de su red virtual, cosa que permite obtener un control dinamico, aunque ligeramente centralizado, de todos los nodos conectados. Además, el sistema mantiene un "*repositorio*" de tareas controlado donde no todos los usuarios del sistema disponen de un libre accesso para realizar sus peticiones. Cosa que atenta contra el principio fundamental de los

sistemas "*peer-to-peer*" (entre iguales) de que todos los usuarios del sistema dispongan de los mimos "derechos" y "privilegios" dentro del sistema.

Por todo esto, se puede afirmarar que uno de los principales puntos débiles del proyecto *JNGI* es su alta regidez y robustez de su estructura de la red que caracteriza el sistema.

2.2. Aplicaciones peer-to-peer: estudio y análisis

En esta sección se llevará a cabo un pequeño repaso a las aplicaciones de tipos peer-to-peer, comentando cuales fueron sus orígenes, la clasificación de redes peer-to-peer, sus ventajas, el tipo de redes peer-to-peer que existen, aplicaciones de estas aplicaciones en cómputo distribuido, y casos conocidos que nos servirán para comparar el modelo que se propone con otros ya creados.

2.2.1. Inicios de las aplicaciones peer-to-peer

La primera aplicación peer-to-peer, fue *Napster* en 1999. La transferencia de archivos se determina entre dos máquinas. *Napster* utilizaba servidores centrales para almacenar la lista de equipos y los archivos que proporcionaba cada uno de ellos, por tanto no era una aplicació puramente *p2p*. No obstante, por aquellos tiempos, ya existían aplicaciones que permitían el intercambio entre usuarios, como *IRC* y *Usenet*.

Debido a que la mayoría de ordenadores domésticos no tienen una dirección *IP* fija, si no que esta se asigna por un proveedor (*ISP*[4]) en el momento de conectarse a Internet, no es posible la conexión entre usuarios de forma directa, ya que los equipos no conocen las direcciones de los peers remotos que han de usar con antelación. Por esta razón *Napster* usaba unos servidores centrales (con dirección *IP* fija), a los cuales los usuarios se conectaban para recibir la relación de direcciones *IP* de los clientes de la res, del resto de servidores de la red y de otro tipo de información relacionada con la cantidad de archivos de los que los clientes disponen. Después de esta fase de conexión con los servidores centrales, los clientes ya disponen de información suficiente de la red como para poder intercambiar archivos entre ellos sin la necesidad de ningún tipo de intervención por parte de los servidores.

Poner fin a las redes centralizadas por parte de organismos privados o públicos (discográficas, etc...) era relativamente sencillo. Solo era necesario cerrar los servidores que almacenaban las listas de usuarios y archivos compartidos. Después del cierre de aplicaciones de este estilo, surgieron nuevas aplicaciones más modernas, se concibieron las redes descentralizadas, las cuales no tenían servidores centrales y por tanto, na había constancia de los archivos que se almacenaban ni se intercambiaban. Estas nuevas tipologías provocaron un avance tecnológico muy importante y significativo.

[4]Internet Service Provider o proveedor de servicios de internet, es una empresa que se dedica a conectar a Internet los usuarios de distintas redes que tengan y dar al mismo tiempo servicio técnico a los mismos.

2.2.2. Clasificación de las redes peer-to-peer

Las redes peer-to-peer se pueden clasificar según su grado de centralización de la siguiente manera:

- **Redes peer-to-peer centralizadas:** Este tipo de red peer-to-peer se basa en una arquitectura monolítica donde todas las transacciones se realizan a través de un único servidor, que actúa como punto de enlace entre dos nodos. Este servidor almacena y distribuye los distintos nodos que almacenan los contenidos. Poseen una administración muy dinámica y una disposición más permanente del contenido. Sus puntos débiles están relacionados con problemas relacionados con la protección de la privacidad de los usuarios, la escalabilidad de un único servidor, los enormes gastos en mantenimiento y de consumo de ancho de banda.

  Las redes de este tipo se caracterizan por:

  - Existe un punto de enlace o servidor entre nodos que gestiona todos los contenidos y peticiones de los nodos.
  - Todas las comunicaciones (peticiones y encaminamientos entre nodos) dependen exclusivamente de la existencia del servidor; si este falla, el sistema es inoperable.

  Algunos ejemplos de redes peer-to-peer centralizadas son: *Napster* y *Audiogalaxy*.

  En la figura 2.6 se puede ver un esquema de una red peer-to-peer de tipo centralizado.

- **Redes peer-to-peer "puras" o totalmente descentralizadas:** Este tipo de red peer-to-peer son las más comunes. Se trata de un tipo de red peer-to-peer muy versátil, ya que no requiere de un punto de gestión central de ningún tipo, permitiendo la eliminación de la idea de poseer un nodo central, ya que son los propios nodos o usuarios del sistema los que gestionan las conexiones y almacenan la información necesaria. Por tanto, las comunicaciones se realizan directamente de usuario a usuario con la ayuda de un nodo (que se trata de otro usuario) que actúa como enlace para comunicar los dos primeros.

  Las redes de este tipo se caracterizan por:

  - Los nodos actúan como cliente y servidor.
  - No existe un servidor central que gestione las conexiones de la red.
  - No hay un enrutador central que administre las direcciones útiles del sistema.

  Algunos ejemplos de redes peer-to-peer descentralizadas "puras" son: *Ares Galaxy*, *Gnutella*, *Freenet* y *Gnutella2*.

  En la figura 2.7 se puede ver un esquema de una red peer-to-peer de tipo descentralizada.

- **Redes peer-to-peer híbridas, semi-centralizadas o mixtas:** En este caso, el servidor de la red peer-to-peer no conoce la identidad de ningún nodo de la red y tampoco almacena información alguna. Por tanto, el servidor no comparte archivos con los nodos del sistema. Tiene la pecularidad de funcionar en algunos casos (como puede ser *Torrent*) de ambas

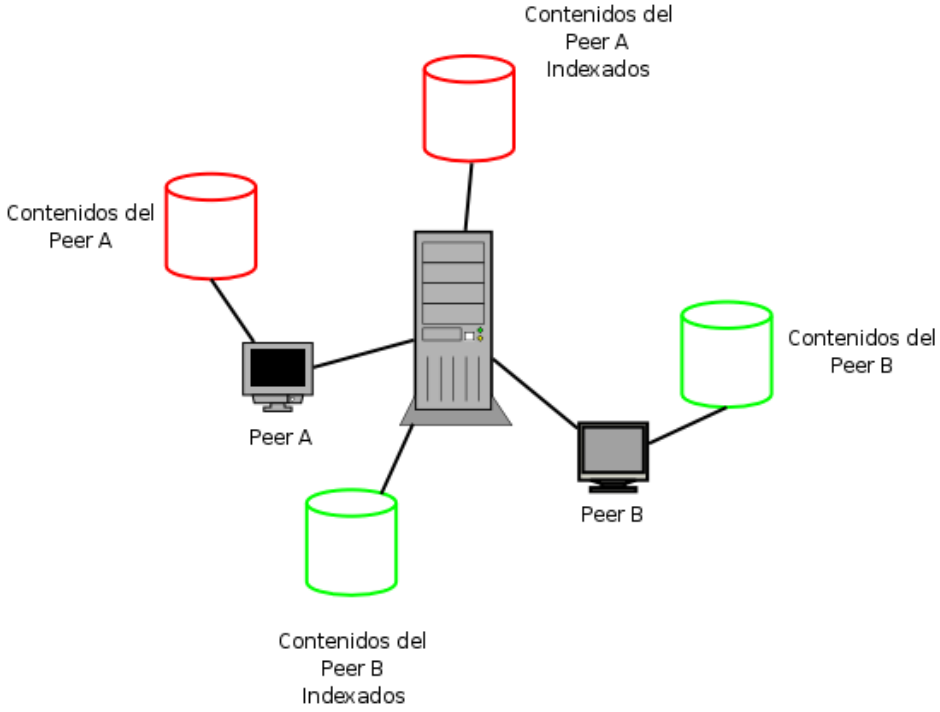

Figura 2.6: Esquema de una red peer-to-peer de tipo centralizado.

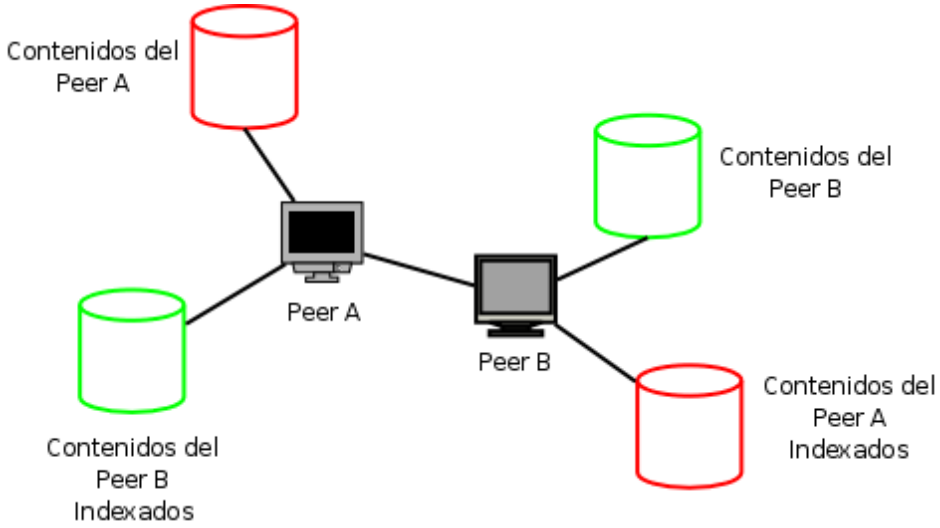

Figura 2.7: Esquema de una red peer-to-peer de tipo descentralizada.

formas, es decir, puede incorporar mas de un servidor que gestiona los recursos compartidos, pero también en el caso que uno o más servidores del sistema fallen, el grupo de nodos sigue en contacto a través de una conexión directa entre ellos mismos, con lo cual, se habilita la comunicación, compartición y descarga de información en ausencia de los servidores.

Las redes de este tipo de caracterizan por:

- Tienen un servidor central que almacena información y permanece en espera para responder las peticiones que se le realizan acerca de la información que dispone.
- Los nodos son responsables de almacenar la información (recordamos que el servidor central no realiza esta tarea), que permite al servidor central reconocer los recursos que los nodos desean compartir, para facilitar la descarga de recursos a los peers que solicitan dicha información.
- Las terminales de enrutamiento son direcciones usadas por el servidor, administradas por un sistema de índices que permiten obtener una dirección absoluta.

Algunos ejemplos de una red peer-to-peer híbrida son: *BitTorrent*, *eDonkey2000* y *Direct Connect*.

En la figura 2.8 se puede ver un esquema de una red peer-to-peer de tipo híbrida o mixta.

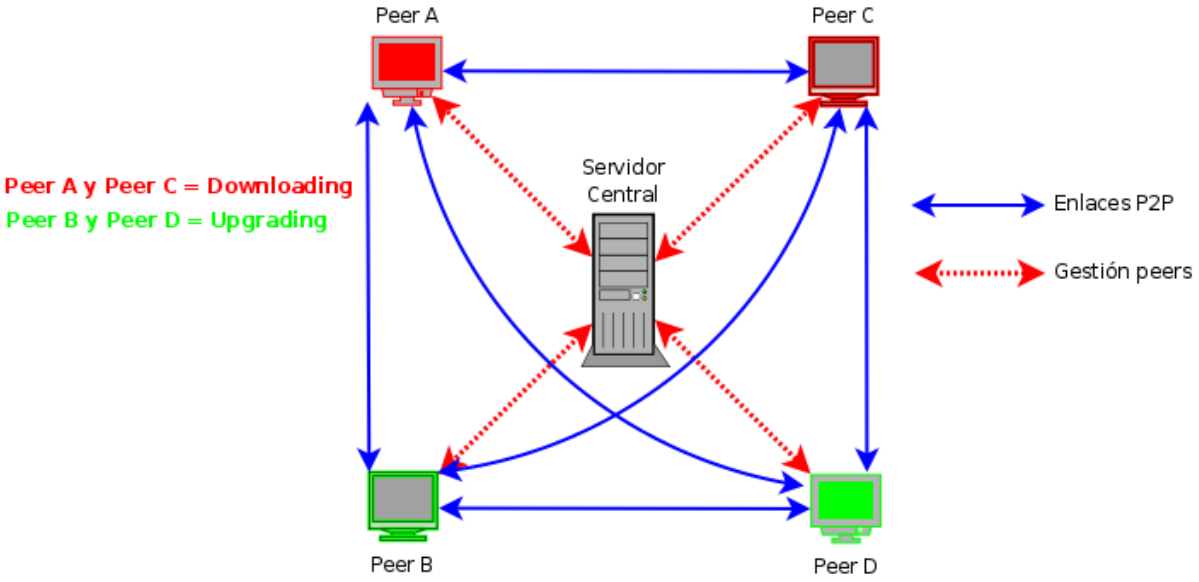

Figura 2.8: Esquema de una red peer-to-peer de tipo híbrida o mixta.

2.2.3. Ventajas de las redes peer-to-peer

Un punto importante en las redes peer-to-peer, es que todos los clientes que se conecten a ella, son proveedores de recursos (incluido ancha de banda, espacio de almacenaje y procesamiento de

cómputo). De esta manera, a medida que se van conectando los nodos, van solicitando y prestando recursos. Esta demanda y cesión de recursos en el sistema se va incrementando, hecho que provoca que los recursos totales del mismo vayan incrementando progresivamente. Esta paradoja no sucede de misma la manera en una arquitectura servidor-cliente, ya que se constituyen de piezas fijas como servidores, a los cuales los clientes se van conectando a ellos, hecho que a medida que se incrementa el nombre de clientes por servidor, provoca una calidad de servicio peor debido a la saturación de conexiones al servidor central.

La naturaleza distribuida de las redes peer-to-peer también permite incrementar la robusteza en casos en los que aparezcan errores debido a la réplica excesiva de datos en múltiples destinaciones (en sistemas peer-to-peer puros). Estos permiten a los peers encontrar la información sin necesidad de efectuar peticiones a ningún servidor centralizado indexado.

2.2.4. Red de sobrecapa en los sistemas peer-to-peer

La red de sobrecapa de los sistemas peer-to-peer está formada por todos los peers que participan como nodos de la misma red. Hay enlaces entre dos nodos cualesquiera que se conocen, es decir, si un peer participante conoce la localización de otro peer que se encuentra en la red peer-to-peer, entonces, existe un "camino" entre el primer nodo y el último dentro de la red. Según se enlacen los nodos entre ellos, se pueden clasificar las redes peer-to-peer como no estructuradas y estructuradas.

2.2.5. Redes peer-to-peer sin estructura vs. redes peer-to-peer estructuradas

A continuación, teniendo en cuenta el concepto de red de sobrecapa explicado en 2.2.4, vamos a comentar brevemente las diferencias entre las redes peer-to-peer estructuradas y las no estructuradas.

- **No estructurada:** se forma cuando los enlaces de la sobrecapa se establecen de forma arbitraria. Estas redes se construyen o crean fácilmente. El proceso consta de los siguientes pasos:

  1. Se inicia cuando un peer cualquiera quiere unirse a la red.
  2. El nuevo peer copia los enlaces de un peer para poder acceder dentro de la estructura del sistema.
  3. Al cabo de un período corto de tiempo, este peer procederá a la creación de sus propios enlaces.

  En este tipo de redes, si un peer desea encontrar un recurso en el sistema peer-to-peer, la petición tiene que recorrer todo el sistema (red) con la finalidad de encontrar tantos peers como sea posible que compartan este recurso. La desventaja principal de este tipo de redes, consiste en que las peticiones no siempre pueden ser resueltas. Un contenido popular es muy probable que esté disponible en varios peers del sistema, por tanto, cualquier búsqueda de este tipo de contenidos, resultará exitosa. Por otro lado, si un peer realiza una búsqueda de

algún recurso o dato poco popular, es decir, que pocos peers alberguen dichos contenidos, es muy probable que la búsqueda no tenga éxito. Esta desventaja se debe al hecho que peers y contenidos compartidos no tengan ninguna correlación, luego no existe garantía que las peticiones resulten exitosas. El primer punto débil comentado, deriva en un segundo punto débil: el *flooding*. Este fenómeno, genera una gran cantidad de tráfico en la red, hecho que repercute en la eficacia y eficiencia de estas redes, que acostumbran a ser muy bajas.

Algunos ejemplos de redes peer-to-peer no estructurada son: *Napster*, *Gnutella* y *KaZaA*.[23]

En la figura 2.9 se puede ver un esquema de una red peer-to-peer de tipo no estructurada.

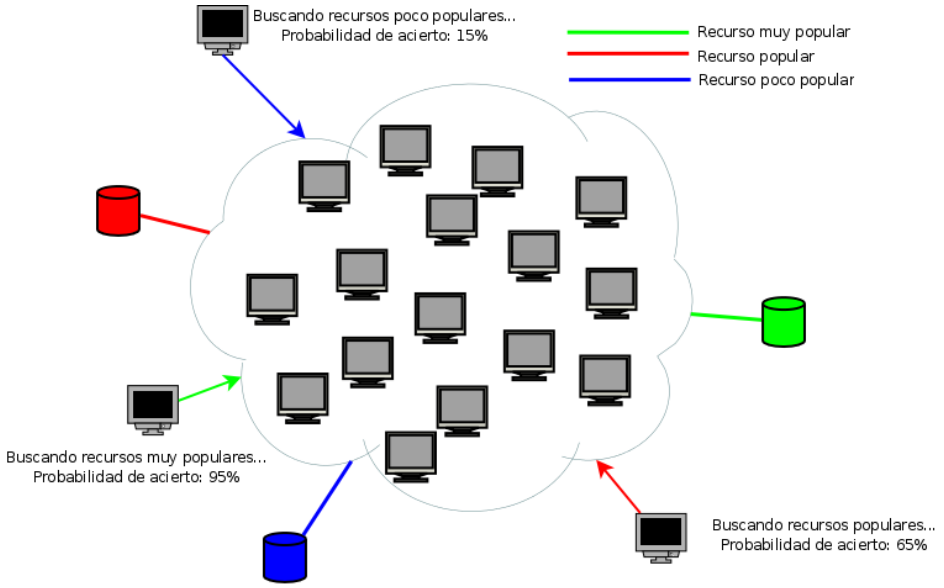

Figura 2.9: Esquema de una red peer-to-peer de tipo no estructurada.

- **Estructurada:** superan las limitaciones de las redes no estructuradas manteniendo una tabla de hash distribuida (*DHT*[5]) y permitiendo que cada peer sea responsable de una parte específica del contenido de la red. Estas redes usan funciones de hash distribuidas y asignan valores a cada contenido y a cada peer de la red. Después proceden al seguimiento de un protocolo global en el que se determina "qué" peer es responsable de "qué" contenido. De esta forma, siempre que un peer desee buscar ciertos contenidos, utiliza el protocolo global para determinar el/los responsable(s) de los datos y después dirige la búsqueda hacia el/los peer(s) responsable(s).

  Algunos ejemplos de protocolos usados en redes peer-to-peer estructurados son: *Chord*, *Pastry P2P Network*, *Tapestry P2P Network*, *Tapestry P2P Network*, *Content Addressable Network*

[5]Las Tablas de Hash Distribuido (Distributed Hash Tables, DHT) son una clase de sistemas distribuidos descentralizados que reparten la propiedad de un conjunto de claves (keys) entre los nodos que participan en una red, y son capaces de encaminar eficientemente mensajes al dueño de una clave determinada.

y *Tulip Overlay*.[23]

En la figura 2.10 se puede ver un esquema de una red peer-to-peer de tipo estructurada.

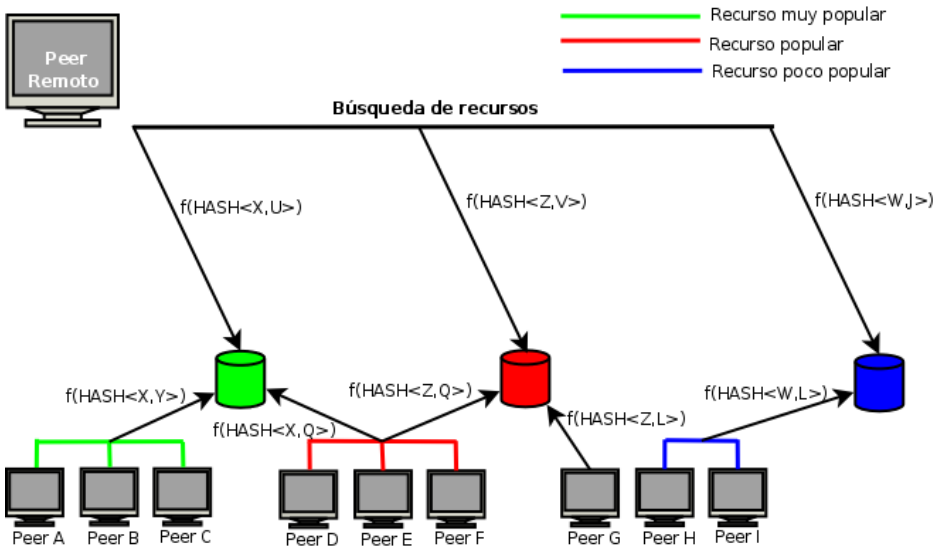

Figura 2.10: Esquema de una red peer-to-peer de tipo estructurada.

2.2.6. Usos del peer-to-peer

Actualmente, las redes peer-to-peer se usan en muchos ámbitos. Debido a esta creciente extensión de sistemas peer-to-peer, se ha considerado oportuno explicar de forma resumida dos ejemplos de uso de este tipo de redes:

1. **Uso del peer-to-peer por parte de la indústria:** a partir del año 2006, compañías americanas y europeas han hecho uso de redes del tipo peer-to-peer. Compañías como *Warner Bros* o *BBC*, han empezado a ver este tipo de redes como una alternativa a la distribución convencional de películas y programas de televisión, ofreciendo parte de sus contenidos a través de tecnologías como *BitTorrent*.

2. **Uso del peer-to-peer por parte de los técnicos:** las redes peer-to-peer han empezado a atraer la atención de científicos de muchas disciplinas de la ciencia; especialmente aquellos que gestionan y procesan enormes bases de datos, como los bioinformáticos.

   Las redes peer-to-peer pueden ser también usadas para funcionar con sistemas de software diseñados para realizar pruebas que identifiquen la presencia de posibles drogas. En el año 2001 se diseñó el primer sistema con este propósito, en el Centro Computacional por el Descubrimiento de Drogas (*Center for Computational Drug Discovery*) en la prestigiosa universidad de Oxford con la cooperación de la Fundación Nacional para la Investigación del Cáncer (*National Foundation for Cancer Research*) de los Estados Unidos de America.

Actualmente, existen varios sistemas de software similares que se desarrollan en una escala más pequeña, como sistemas de administración autónoma para biólogos computacionales. Un ejemplo es el *Chinook*, que se usa para ejecutar y comparar datos de temática bioinformática con más de los 25 servicios de análisis distintos que ofrece. Uno de sus propósitos consiste en facilitar el intercambio de técnicas de análisis dentro de una comunidad local.

Las instituciones académicas también han iniciado la experimentación con redes peer-to-peer para compartir archivos, como es el caso de *LionShare*. Se trata de un proyecto cuyo esfuerzo se focaliza en facilitar la compartición legítima de archivos digitales entre individuos e instituciones educativas del mundo dentro de un marco legal.

2.2.7. Retos del peer-to-peer

El diseño de una arquitectura peer-to-peer de Cómputo Distribuido y la gestión que se realiza en la plataforma peer-to-peer para la distribución de cómputo tiene que hacer frente a los retos siguientes:

- **Gestión distribuida**. Para la compartición de recursos para aprovechar la capacidad computacional de los nodos que integran el sistema, es necesario diseñar políticas que basan su decisión en información no completa del sistema así como sus recursos[12, 13]. Por otro lado, teniendo en cuenta que la escala del sistema puede ser grande, será necesario disponer de políticas que actúan en distintos niveles de gestión: planificador interno del nodo, planificación de área local y planificación en el área global.

- **Tolerancia a fallos**. El gran dinamismo que caracteriza un sistema peer-to-peer (conexiones y desconexiones de nodos de forma continua sin previo aviso) y el gran número de componentes que pueden formar parte de estos, hace que la posibilidad de que se produzca un fallo sea elevada. La ejecución distribuida de las aplicaciones se ha de llevar a cabo de forma transparente a los errores que surjan, de tal modo, que el sistema garantice el resultado final de forma independiente de esta casuística. Será necesario definir de mecanismos de reconfiguración de la aplicación para lanzar de nuevo el cómputo perdido y mecanismos de reconfiguración de la arquitectura para mantener el estado del sistema desde un punto de vista de los recursos y tareas[14, 10].

- **Auto-organización**. En los sistemas peer-to-peer los roles que cada uno de los nodos tienen que "jugar" no están definidos a priori. Los roles como proveedores, gestores o planificadores de recursos, se asignan de forma dinámica en función de las necesidades del sistema. En el momento que un nodo que ejerce un rol con responsabilidad de gestión se desconecta del sistema de forma inesperada, el sistema tiene que garantizar que otro nodo asumirá la tarea o rol del peer caído.

- **Gestión de recursos heterogéneos**. La gestión de recursos en una plataforma peer-to-peer introduce dos retos importantes: los recursos deben ser heterogéneos y debe existir la gestión de la compartición de estos recursos. La heterogeneidad de los recursos puede afectar a la ejecución de las aplicaciones y las prestaciones ofertadas por la arquitectura. Por tanto,

resulta crítico el diseño de políticas distribuidas y dinámicas de planificación y balanceo de carga que permitan enmascarar esta heterogeneidad, para proporcionar una gestión eficiente de los recursos[15, 16, 17]. Por otro lado, el principio básico sobre los que se fomentan los sistemas peer-to-peer, es la compartición de recursos. Para que esta gestión se lleve a cabo de forma justa y proporcional a los recursos ofrecidos al sistema se necesita contabilizar la utilización/cesión de recursos por parte de los usuarios del sistema y fomentarla vía políticas para incentivar la compartición[18].

- **Seguridad**. Los nodos de una red peer-to-peer tienen que interactuar con otros nodos desconocidos, por tanto, es necesario gestionar el riesgo derivado de estas interacciones (transacciones), sin contar con la presencia de una tercera entidad de confianza. Para superar estos problemas, se necesita usar técnicas de encriptación, autenticación y ejecución de código en entornos seguros[19, 20, 21].

Al mismo tiempo, se ha de hacer frente a los retos que plantea el diseño de una arquitectura peer-to-peer dentro del marco de las aplicaciones actuales, ya que es muy importante, garantizar un tiempo de ejecución y unos costes acotados. Por eso, se tiene que diseñar un sistema donde el usuario de las aplicaciones puedan especificar los siguientes requisitos de calidad de servicio respeto a su ejecución: tiempo de respuesta y entorno de la aplicación, coste de ejecución y disponibilidad del servicio.

Capítulo 3

Punto de partida

3.1. Introducción

En esta sección se comenta de forma muy resumida el proyecto CompP2P. Esta fue la primera aplicación de cómputo distribuido peer-to-peer que surgió en la Universidad de Lleida[1].

La experiencia adquirida durante el período de desarrollo de este primer prototipo ha sido un gran punto de apoyo para la realización de este segundo proyecto sobre una aplicación de cómputo distribuido peer-to-peer. Aunque la estrategia que se ha adoptado en este proyecto ha sido empezar prácticamente desde cero en la mayoría de los aspectos, es evidente que *CompP2P* ha estado siempre en la mente de todos los que hemos participado en la realización del nuevo *CoDiP2P*, ayudando, entre otras cosas, a tomar decisiones relativamente complejas con mejor conocimiento de causa y con cierta agilidad.

3.2. Modelo inicial del sistema

La arquitectura del sistema trabaja sobre la máquina virtual de Java. Este factor facilita que la multiplataformidad de la aplicación sea casi inmediata, pudiendo trabajar en entornos con distintas arquitecturas (i386, ppc, x86_64...) y de su sistema operativo (Windows, Linux, MacOS...).

En la arquitectura podemos diferenciar dos partes claras que se comunican entre ellas mediante *sockets TCP*. Ambas partes están representadas en la figura 3.1

Por un lado, CompP2P tiene una parte central que se encarga de gestionar la parte de los peers trabajadores (workers peers), de los gestores (mánagers peers) y de la planificación de todos los eventos. Esta capa se encarga al mismo tiempo de establecer una comunicación con el usuario mediante el uso de *sockets TCP*.

En segundo lugar, también se diseñaron mecanismos de comunicación con el sistema para el usuario, que ofrece una interfaz de comunicación simple pero ampliable, que facilita su uso, modificación y/o ampliación de CompP2P. Para brindar al usuario mecanismos concretos de uso, también se desarrollaron utilidades de alto nivel.

[1]Proyecto Final de Ingeniería Técnica de Informática de Sistemas de Íñigo Goiri Presa y Josep Rius Torrentó; ambos, estudiantes de la Escuela Politécnica Superior de la Universidad de Lleida.

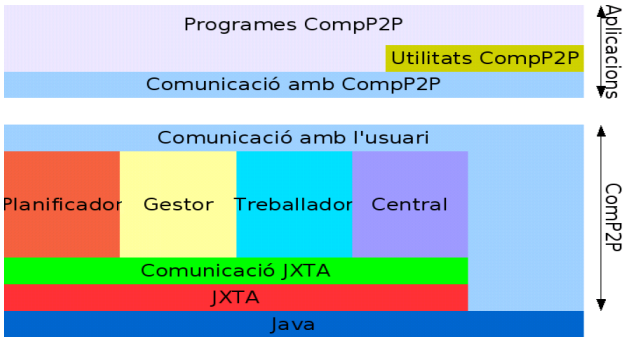

Figura 3.1: Esquema de la arquitectura de CompP2P.

Siguiendo la pauta de obtener un diseño estructurado, se eligieron los módulos independientes como parte fundamental del aplicativo (representados en la figura 3.2). De esta manera se asegura la implementación completamente aislada de todos los módulos. La metodología que se siguió en todas las fases de desarrollo del aplicativo, fueron las que basas sus principios en la orientación a objetos, intentando lograr un sistema poco acoplado y fuertemente cohesionado.

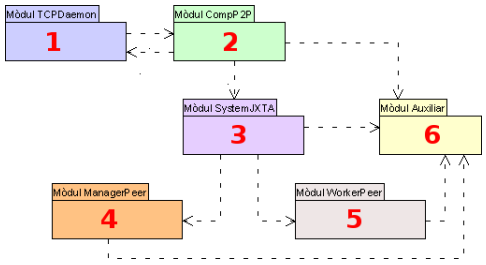

Figura 3.2: Diagrama de módulos de CompP2P.

1. **TCPDaemon:** es el módulo de comunicación con el usuario. Se encarga de recibir los objetos de las aplicaciones, serializarlos, enviarlos a un peer remoto, esperar recibir ejecutados remotamente, deserializarlos y finalmente retornarlos al peer que generó la petición.

2. **CompP2P:** es el módulo central de la aplicación. Se encarga de poner en marcha todo el sistema.

3. **SystemJXTA:** este módulo se puede definir como el gestor JXTA de CompP2P. El resto de la aplicación intenta no tener contacto con esta plataforma de desarrollo peer-to-peer para liberar al programador de la carga que supone tener que conocer todos los detalles de el framework.

4. **ManagerPeer:** su principal tarea es la de operar como un gestor o mánager peer de un grupo de peers.

5. **WorkerPeer:** este módulo tiene la responsabilidad de gestionar los trabajos que se quieren ejecutar desde este nodo y distribuirlos por el sistema.

6. **Auxiliar:** el módulo contiene todas las estructuras de datos que ayudan y facilitan la implementación del resto del sistema.

3.3. Resultados previos obtenidos con el prototipo

Antes de mostrar los resultados que se obtuvieron, se efectuará una pequeña introducción con la finalidad de describir el entorno de trabajo utilizado en la experimentación, así como la aplicación usada en ella.

3.3.1. Características del entorno de trabajo

Para poner de manifiesto la multiplataformidad de la aplicación, las pruebas se realizaron en un entorno de trabajo heterogéneo. Se usaron 30 Pentium 4 con 512 MB de memoria RAM cada uno de ellos, 12 de los cuales incorporaban un procesador de 2'8 GHz y el resto de 2'4 GHz. 30 peers, de los cuales 2 usaban el sistema operativo Debian GNU/Linux y 28 Microsoft Windows 2000. Cada máquina tenía correctamente configurado el intérprete y compilador de Java, concretamente en su versión 5.

3.3.2. Aplicación a ejecutar: la valla óptima

La aplicación calculará cual es la combinación óptima de árboles que se tienen que cortar para realizar una valla alrededor del resto de árboles. para definir un árbol, se utiliza su ubicación, la longitud de la valla que se puede realizar a partir del árbol y de su valor económico.

La resolución de este problema es de costo exponencial, cada árbol que se añade al cálculo implica duplicar los gastos ya que ha de calcular las combinaciones que había más las que resultan de la suma de un nuevo árbol. Se trata de un problema fácilmente distribuible. A cada peer o nodo del sistema se le asignará unas combinaciones para calcular y retornará la combinación óptima. Por último, es necesario evaluar todos los resultados parciales y retornar el mejor de todos ellos.

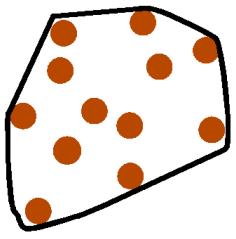

Figura 3.3: Ejemplo de valla óptima.

La figura 3.3 muestra un ejemplo de cálculo de la valla óptima con 12 árboles. Como característica principal, se puede apreciar que todos los árboles que quedan, se encuentran dentro del área delimitada por la valla.

3.3.3. Evaluación de los resultados (26 árboles y 30 particiones)

En el momento de obtener los resultados se tiene que tener en cuenta la plataforma de ejecución. Las ventajas de Java son conocidas, pero también es conocido la penalización en tiempo de ejecución debido al uso de la máquina virtual. Esta sobrecarga tiene que ser medida para obtener los resultados reales.

Primero se comparó una versión serie de la aplicación de pruebas en el lenguaje Java y C++. Esto supone una pérdida de un 45 %, por tanto, se puede confirmar y cuantificar la bajada de eficiencia introducida por la propia máquina virtual y la interpretación del código.

Programa interpretado (Java): 15' 43.345"
Programa compilado (C++): 10' 59.911"

Una vez tomada conciencia de las limitaciones del lenguaje, se midió el retardo introducido por el propio sistema de computación distribuida. Se comparó la ejecución en serie sobre la máquina virtual de Java con la ejecución en un solo peer con el sistema CompP2P (introduciendo la comunicación máster-mánager-worker).

Java ejecución en serie: 15' 43.345"

3.3.4. Escalabilidad

Partiendo de la base que Java es más ineficiente que otros lenguajes de programación, quedó evidente que el uso de CompP2P en dos máquinas reduce el tiempo de ejecución en 35 segundos, lo que significa una reducción del 5 %. La mejora no es sumamente importante, pero con un número superior de peers, esta diferencia mostrada con la versión en paralelo con C++ sería mínima.

Una vez vista la diferencia básica en relación a una versión con C++, se analizó la evolución de los tiempos de ejecución en función el número de peers. La gráfica de la figura 3.4 se observa que la evolución de los tiempos es muy próxima a una situación ideal, esta situación se daría en el caso que la tarea fuera dividida entre todos los peers de forma equitativa, partiendo como base el tiempo de ejecución dado con un único trabajador.

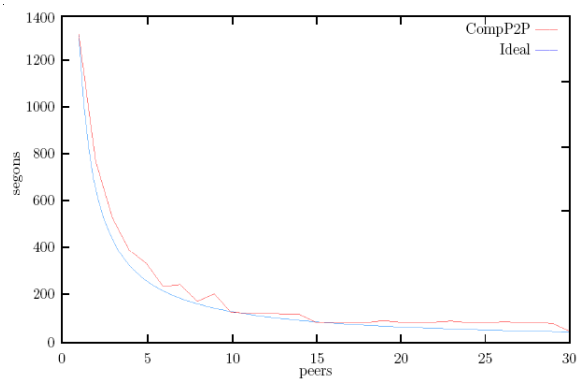

Figura 3.4: Gráfica de los resultados obtenidos.

Remarcar la importancia que la distribución en 30 particiones fijas, disminuye la eficiencia del sistema para un cierto número de peers. Teniendo en cuenta que CompP2P dispone de la posibilidad de conocer el número de trabajadores que hay disponibles, se podría aprovechar para dividir el trabajo con un número variable de tareas en función de los trabajadores existentes. Con esta medida, se eliminarían los escalones de tiempo, haciendo que fueran más cercanos al caso ideal en todo momento.

Capítulo 4

Modelo de Sistema

En el capítulo siguiente, se especificarán los puntos más importantes que forman parte de el modelo de nuestro sistema. Se van a introducir conceptos e ideas muy importantes, como qué es un peer, una área, un mánager... etc.

4.1. Conceptos Previos

Peer
Un peer es la unidad más pequeña que se puede hallar en una red *p2p*. También conocido como nodo, el *peer* se usará para aprovechar sus recursos computacionales ociosos, tales como capacidad de cómputo, de almacenamiento ... etc.

Peer es el concepto que engloba a los trabajadores (*workers*) y a los gestores (*mánagers*). Un peer puede desarrollar el rol de trabajador o de gestor dependiendo de las necesidades del sistema. La determinación de un rol a un peer no es definitiva, como hemos comentado, su rol es variante durante su permanencia dentro de el sistema.

Área o grupo
Una área (o grupo) es un espacio lógico formado por un único mánager que gestiona un conjunto limitado de workers. Una área constituye la unidad mínima de trabajo. El sistema tiene una estructura jerárquica integrada por áreas. Puede estar formado por distintos niveles de áreas, permitiendo que mánagers de una área inferior, sea a la vez worker de su área superior. Esta definición implica que haya una sobreposición entre dos áreas: la superior y la inferior a ésta. La comunicación entre distintas áreas siempre se ejecuta a través de los mánagers.

Mánager
Un mánager es uno de los dos tipos de rol que puede desarrollar un peer cualquiera dentro del sistema. Las atribuciones de un mánager son distintas a las de un worker.

Los mánagers, serán los encargados de gestionar a los peers que integran su área. Entendemos la gestión, como el proceso por el cual, el mánager se encuentra permanentemente en contacto con los peers que constituyen el área que forman, con los que establece una comunicación continuada con

la finalidad de poder ejecutar tareas distribuidamente, obtener los resultados de estas ejecuciones y mantener la estructura de su área.

La condición de mánager no es exclusiva, ya que todo mánager del sistema es a la vez un worker en su área superior.

La información que almacena un mánager es la siguiente:

- Suma de toda la capacidad computacional de los peers de su área.
- Número de workers que controla directamente.
- Número de mánagers que controla directamente.
- Número de peers (workers y mánagers) que controlan los mánagers de las áreas inferiores a la suya.

De forma simultánea, esta información se ha de replicar a unos workers de su área (elegidos bajo criterio X) denominados como *Mánager de reserva o replicado*[1], para asegurar la integridad del sistema.

Worker

Un worker es uno de los dos tipos de rol que puede desarrollar un peer cualquiera dentro del sistema.

Los workers serán los encargados de ejecutar las tareas que el mánager de su área les ha planificado y comunicarles los eventos que surjan (resultados, estadísticas, mensajes de mantenimiento...etc).

La condición de worker no es exclusiva, ya puede ser que un worker del sistema sea a la vez un mánager en su área inferior. Este caso sucede cuando una área no soporta la entrada de nuevos peers (workers en este caso) y se debe ampliar el sistema, hecho que provoca que ciertos workers peers se conviertan en mánagers peers para que al crear una nueva área, se puedan ubicar en ella nuevos peers entrantes.

Localidad de los peers

Se entiende por localidad de los peers, a la zona geográfica de donde provienen los nodos que quieren formar parte del sistema. Es decir, la localidad tiene en cuenta o es capaz de diferenciar los nodos de distintos continentes, países o ciudades, con el fin de minimizar al máximo comunicaciones entre nodos lejanos localmente hablando.

Capacidad de las áreas o grupos

Cada área o grupo del sistema tiene una capacidad limitada de aforo de peers. La obtención de la capacidad óptima se detalla en el capítulo 5 donde se ha realizado un pequeño estudio de la plataforma simulada.

[1] Peers del sistema que ejecutan un rol del tipo worker, pero que de forma simultánea almacenan información del mánager de su área para poder ocupar su lugar en caso que este se desconecte del sistema.

Rama R_i

Entendemos el concepto de rama como el conjunto de peers que un mánager M_i controla directa (peers de su propia área) e indirectamente (peers que gestionados por los peers de su área); podemos visualizar este concepto en la figura 4.1.

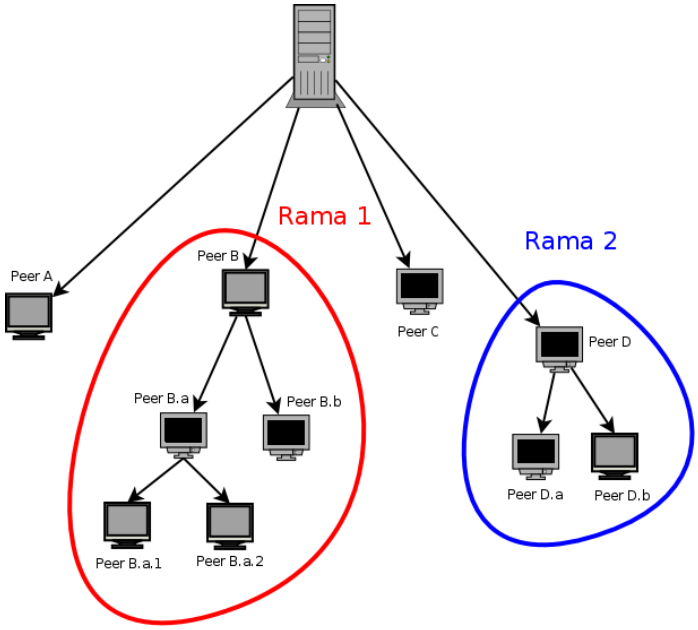

Figura 4.1: Muestra una rama dentro del sistema.

Cómo se aprecia en la figura 4.1, la estructura consta de dos ramas: Rama 1 y Rama 2. La primera de ella tiene un mánager (Peer B) que controla a dos peers en su área (Peer B.a y Peer B.b), de forma simultánea, el Peer B.a, es mánager de los peers Peer B.a.1 y Peer.a.2 respectivamente. Esta definición concuerda con la definición de rama comentada. La misma metodología aplicaríamos para comentar la Rama 2.

Nivel N_i

Se entiende nivel como la unidad de crecimiento de la estructura del sistema. Los niveles están formados por uno o más grupos o áreas. Los niveles pueden contenerse dentro de una o varias ramas. Habrá tantos niveles cómo mánagers creados verticalmente en la estructura del sistema.

En la figura 4.2 se aprecia claramente que el número de mánagers creados verticalmente son 3, por lo tanto, 3 serán en número de niveles del sistema. Se puede ver como el nivel 1 está compuesto por un mánager y dos descendientes a su vez mánagers de áreas inferiores. El nivel 2 consta de dos áreas y abarca parte de las dos ramas (A y B). Por último, el nivel 3 está constituido por un mánager y dos workers, el nivel coincide con todo el grupo o área 3.

NMajor

Un concepto fundamental sin el cual el sistema no podría iniciarse, es conocido como NMajor. NMajor es un servidor estático que tiene la funcionalidad de ser el punto de entrada del sistema.

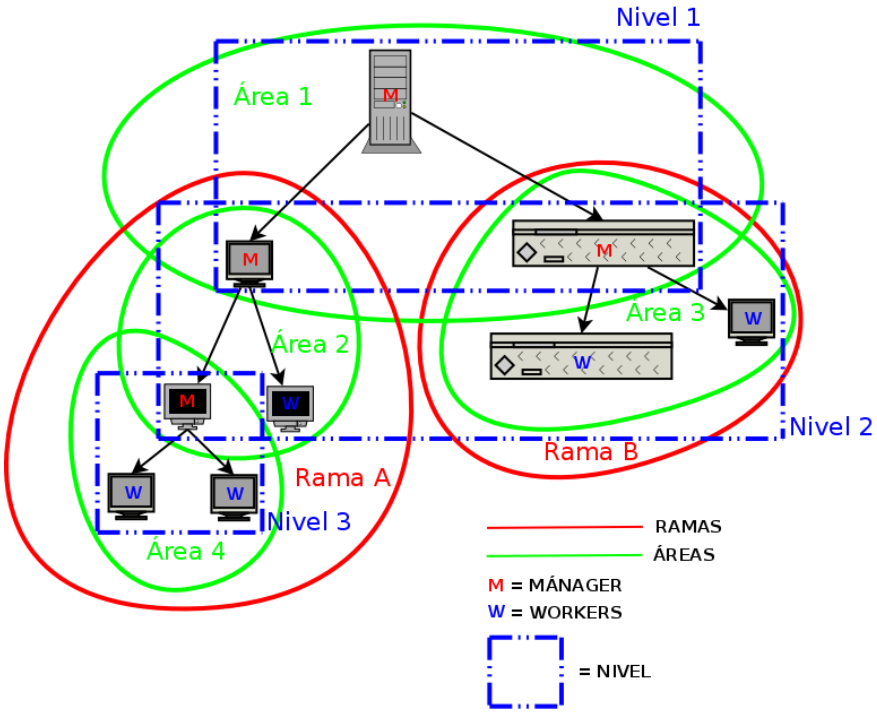

Figura 4.2: Esquema del sistema constituido por mánagers, workers, grupos o áreas, niveles y ramas.

Su tarea consiste en la búsqueda, registro y actualización de una lista de nodos para facilitar el acceso de nuevos usuarios al sistema. Podríamos generalizar diciendo que es un servidor que almacena una lista de *bootstrap nodes*[2].

Nodo Final

Entendemos como Nodo Final, aquel nodo (peer) que debido a limitaciones técnicas no permite que la estructura del sistema crezca haciendo uso de su situación en el sistema, es decir, no permite la creación de nuevos niveles y áreas a través suyo. Un ejemplo de estructura con presencia de nodos finales, es la que se muestra en la figura 4.3.

Aplicación

Una aplicación en el ámbito del trabajo, es un programa implementado con el lenguaje *Java*, que sigue las directrices de programación paralelizable, con la finalidad que sea fácilmente distribuible.

Toda aplicación está formada por un conjunto de tareas, que serán ejecutadas indivisiblemente por una única unidad de cómputo, es decir, un worker.

[2]Punto de acceso al sistema, es la terminología usada para indicar el nodo o entidad responsable de dar acceso a los nuevos nodos que se conectan al sistema.

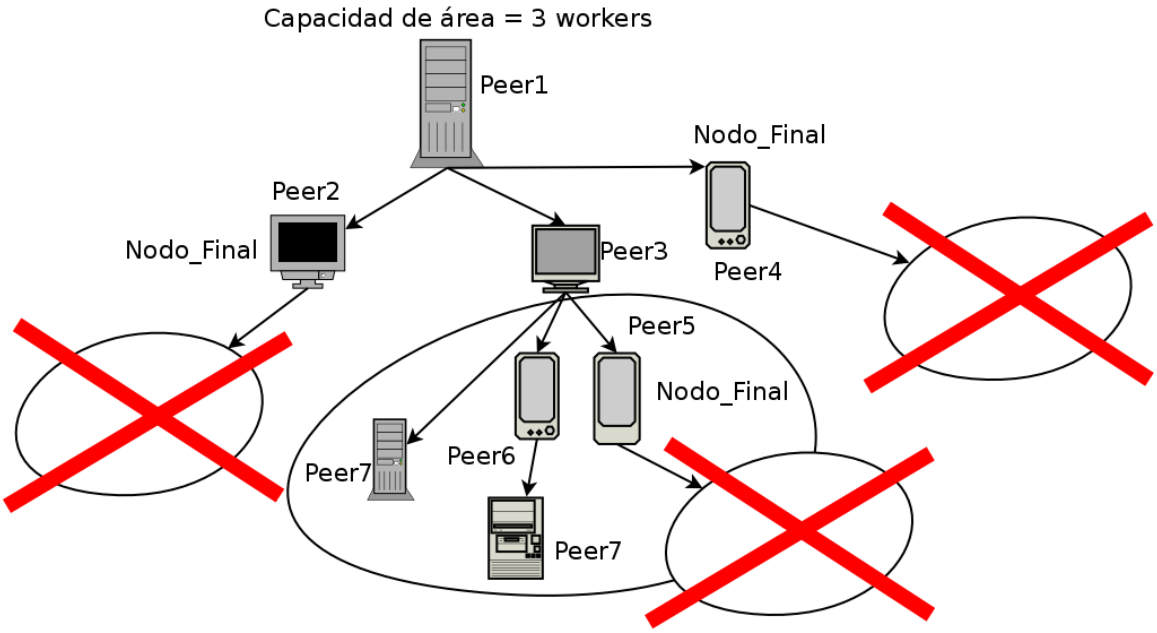

Figura 4.3: Diagrama a modo de ejemplo de grupos y niveles del sistema creados excepcionalmente con nodos finales.

Tareas
Unidad mínima e indivisible ejecutable por una única unidad de cómputo, es decir, un worker. Un conjunto de tareas constituyen una aplicación.

4.2. Decisiones acerca del sistema

Diseñar e implementar un sistema es la parte más importante en todo desarrollo de un proyecto, en el caso que nos ocupa resulta sumamente importante

4.2.1. El nodo NMajor

Aunque pueda parecer que *NMajor* es un punto de entrada permanente para todos los nodos que entran en el sistema y por tanto la presencia del mismo implique que el sistema pierde parte de la filosofía peer-to-peer, debemos corregir este pensamiento, pues no resulta acertado.

NMajor, es un punto de entrada en los procesos iniciales de creación del sistema, cuando aún no existen suficientes mánagers registrados para cederles tareas que consisten en dar entrada a nuevos nodos. Es decir, cuando el sistema se inicia por primera vez, éste solo está compuesto por *NMajor*. El siguiente paso se sucede cuando un nuevo nodo se quiere añadir al sistema, éste, consultará primero un fichero local con la finalidad de encontrar un mánager del cual pender, al ser el nodo que pretende entrar al sistema el primer peer del mismo, éste fichero resultará vacío, con lo cual se sucederá el segundo paso, que es la conexión a *NMajor*. En ésta conexión, se consulta una tabla almacenada en el propio *NMajor* con la misma finalidad que en el primer paso: encontrar

la dirección de algún mánager que le pueda ofrecer cobijo en su área. El procedimiento se repite; no hay mánager, por tanto, el nodo de entrada se convertirá como nuevo mánager y las listas de mánagers a contactar contendrán la primera y única entrada (de momento) de nodos de entrada con quién contactar.

Por tanto, cuando el sistema vaya creciendo, podemos decir que el mismo va aprendiendo direcciones de mánagers con la finalidad que el primer contacto con el sistema no sea necesariamente con *NMajor*, sino que puede ser contacto con un mánager determinado.

De ésta forma, se explica la vital importancia de la existencia de *NMajor* en los primeros compases del sistema, así como en los momentos en que haya muchas peticiones de entrada y una lista de mánagers de contaco poco abundante.

En en algoritmo 1 contenido en el apartado 4.3.1 de la sección 4.3, se especifica detalladamente el procedimiento de conexión del sistema, en el cual, tienen importancia notable las tareas que lleva a cabo el nodo *NMajor*.

4.2.2. Jerarquía de peers con dos roles diferenciados

"*En un sistema peer-to-peer todos los nodos son tratados por iguales*", la definición filosófica es lo que pretende dar a entender. En los sistemas prácticos, los sistemas peer-to-peer, tienen por necesidad nodos con distintos roles que realizan tareas distintas. Sin la "diferenciación de clases", resultaría imposible mantener el sistema, establecer criterios de prioridad de lanzamiento de tareas o planificar las tareas por la red peer-to-peer.

Es por ésta razón, que la existencia de un rol de mánager peer y otro de worker peer, resulta primordial, aún violando en cierto sentido la filosofía peer-to-peer.

Necesitamos mánagers peers que se dediquen a tareas de mantenimiento y planificación de tareas en la red peer-to-peer así como workers peers que realicen las tareas mandadas por los mánagers, en aspectos como cómputo, paso de mensajes, notificación resultados, etc...

Al terminar la lectura del documento que se presenta, se tendrán nociones generales del sistema, que ayudarán a despejar dudas en este aspecto. Seguramente, la sección 4.3 de algoritmos también le resulte de gran ayuda para acabar de digerir los conceptos.

4.2.3. Necesidad de los "mánagers replicados (o mánagers de reserva)"

La tolerancia a fallos es uno de los puntos críticos en todos los sistemas, más aún si cabe, en los que son de tipo peer-to-peer. En medio de la cesión de recursos se pueden originar errores debidos a una caída de la red de o un fallo de suministro eléctrico de un nodo, la cual cosa, puede tener resultados muy negativos en el caso que aquel nodo estuviera realizando algún tipo de tarea importante.

Más especificamente, si ejecutamos una aplicación distribuida en la red peer-to-peer, hay muchas probabilidades que los workers que realizan cómputo o los mánagers que planifican las tareas a ejecutar se desconecten de la red por motivos dispares. Las consecuencias que acarrea si no se implementa un mecanismo de tolerancia a errores son muchas y graves: perdida del cómputo realizado por uno o más workers, aislamiento de una área debido a la caída del mánager que la gestionaba, etc...

Por tanto, resulta fundamental, pensar en métodos de prevención y reparación de errores en el sistema. Uno de los mecanismos que nos permite llevar a cabo esta parte del sistema, es la inclusión de mánagers replicados o de reserva, ya que en caso que el mánager de un área se desconecte del sistema, rápidamente un worker de la propia área que resultaba ser un mánager de reserva, sucederá al anterior mánager, disponiendo de la misma información que el anterior, disminuyendo considerablemente el daño causado.

Los algoritmos 2, 3 y 4 se encuentran dentro de la sección de algoritmos en 4.3, más concretamente en la subsección 4.3.2, ayudan a entender el por qué de la existencia de los mánagers replicados o de reserva.

4.2.4. Estructura basada en distintos grupos y niveles

La presencia de peers con roles diferenciados, la implementación de mecanismos que minimicen el impacto negativo de la aparición de errores en el sistema y la utilización del framework *JXTA*, conduce inevitablemente a la utilización de grupos en el sistema.

Un ejemplo de la estructura de grupos y niveles del sistema se referencia en la figura 4.4. En el ejemplo se ha considerado que cada área o grupo del sistema tiene una capacidad máxima de 2 workers (y e mánager que los controla), de tal modo, que no se crea un nuevo nivel hasta que en cada grupo o área contiene dos trabajadores.

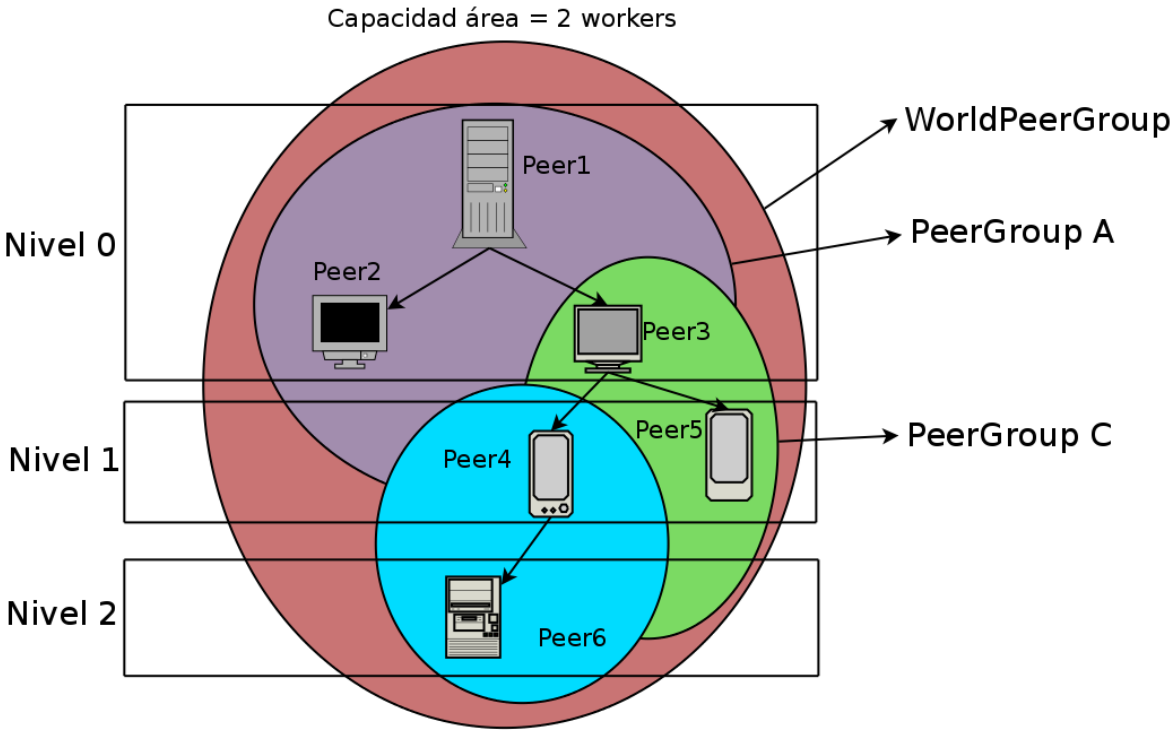

Figura 4.4: Diagrama a modo de ejemplo de grupos y niveles del sistema.

Ésta capacidad máxima de la que hablamos, se intenta agotar, para conseguir estructuras

compactadas y con el menor número de niveles en el sistema, para facilitar el control y actualización de la información del sistema.

Es por esa razón, que no se crea un nuevo nivel si el anterior no tiene todas sus áreas o grupos creados con la capacidad agotada. De tal modo, que una estructura como la que se refleja en la figura 4.5 no se daría nunca a no ser que fuera un caso excepcional que comentaremos a continuación.

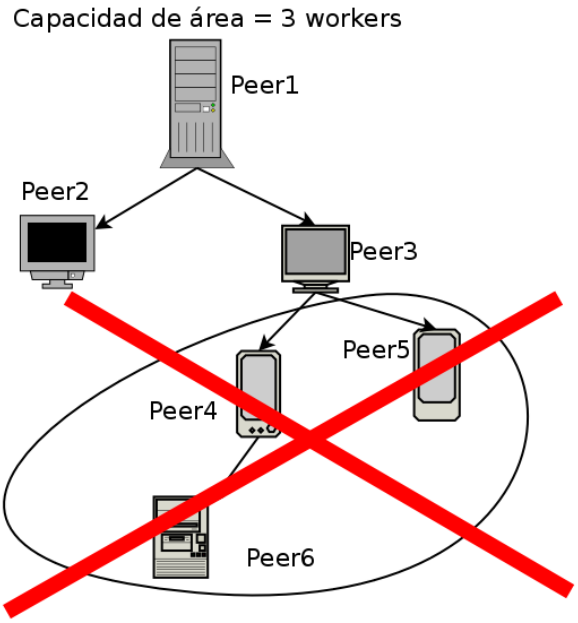

Figura 4.5: Diagrama a modo de ejemplo de grupos y niveles del sistema creados erróneamente.

La excepción que puede convertir la estructura mostrada en la que la figura 4.5 en una estructura plausible, es la que se basa en una estructura que contiene nodos finales. Siguiendo la definición comentada en este capítulo, un nodo final es aquel nodo (peer) que debido a limitaciones técnicas no permite que la estructura del sistema crezca haciendo uso de su situación en el sistema, es decir, no permite la creación de nuevos niveles (ni áreas) a través suyo. Por tanto, si en un grupo o área hay un nodo final, a través de éste, la estructura no podrá crear un nuevo nivel, sino que tendrá que hacerlo por el resto de nodos que existan en aquel área. En la figura 4.3 se ilustra una estructura excepcional con presencia de nodos finales en la misma.

4.2.5. Crecimiento del sistema de niveles superiores hacia niveles inferiores

Una estructura peer-to-peer puede crecer de distintas formas, las dos que se estudiaron fueron:

1. Crecimiento de la estructura de los primeros nodos entrantes del sistema hacia los últimos nodos entrantes al sistema, sería un crecimiento “de arriba hacia abajo”. Tipo de crecimiento ilustrado en la figura 4.6.

2. Crecimiento de la estructura de los últimos nodos entrantes del sistema hacia los primeros nodos entrantes al sistema, sería un crecimiento "de abajo hacia arriba". Tipo de crecimiento ilustrado en la figura 4.7.

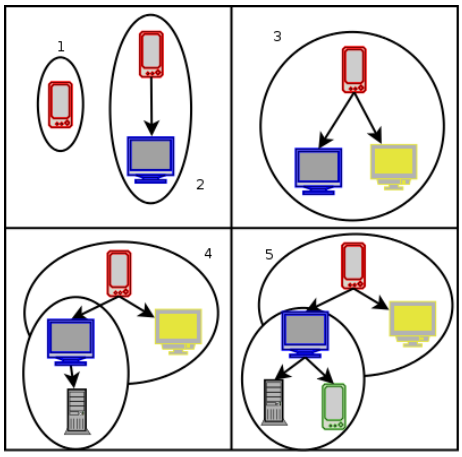

Figura 4.6: Diagrama a modo de ejemplo de la creación del sistema "creciendo hacia abajo".

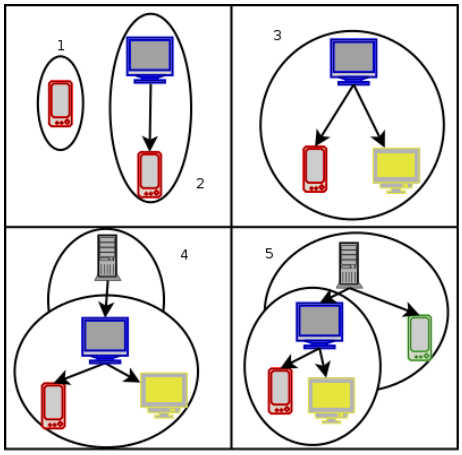

Figura 4.7: Diagrama a modo de ejemplo de la creación del sistema "creciendo hacia arriba".

Tal y como se aprecia en las figura 4.6, el crecimiento "de arriba hacia abajo", es el crecimiento natural, por tanto, el que se ha implementado en el sistema. El modo de crecer de "hacia arriba" (ver figura 4.7) es antinatural además de muy difícil de implementar, ya que precisa de la reestructuración de la estructura cada vez que se crea un nuevo nivel.

Crecer "de arriba hacia abajo" potencia una estructura estable del sistema, ya que los nodos "antiguos" son los que permanecen en los niveles inferiores, dando paso a nuevos niveles. De éste modo, se consigue una estabilidad mediante el propio crecimiento. Creciendo de forma natural permite añadir nodos estériles al sistema de forma trivial, no en el caso en que se crezca hacia

arriba, pues de un nodo final no puede depender nada, hecho que choca frontalmente en este tipo de crecimiento, por tanto, para añadir éste tipo de nodos, se debería rebalancear la estructura y volver a generarla, hecho que supone un coste inadmisible.

4.3. Algoritmos

En esta sección vamos a introducir los distintos mecanismos o algoritmos que van a ser implementados en el sistema de cómputo. Dividiremos la temática de éstos mecanismos según:

- Creación del sistema: formado por el algoritmo de conexión. Es el punto donde se crea el sistema: la entrada de nuevos peers, la creación de áreas, la gestión de los roles de los peers, etc...
- Mantenimiento del sistema: especificado en un algoritmo de reasignación y de actualización. Cuando el sistema ya está creado y en funcionamiento, se debe de mantener, la organización de la red peer-to-peer tiene que ser actualizada, modificada y reparada en caso de errores como desconexiones, para ello precisaremos de ambos algoritmos.
- Ejecución de tareas al sistema: dentro de este subgrupo se halla el algoritmo de envío o lanzamiento de aplicaciones y el de planificación de de tareas. La finalidad del sistema, es realizar cómputo de forma distribuida, por lo tanto, se deben de implementar algoritmos que permitan lanzar una aplicación de cómputo distribuido peer-to-peer que contenga tareas que serán planificados a lo largo de la red, para poder aprovechar al máximo los beneficios que nos brinda el sistema peer-to-peer.

4.3.1. Creación del sistema

Entendemos como creación del sistema, al conjunto de procedimientos que permite crear la red de cómputo distribuido utilizando una filosofía *peer-to-peer*.

Dicha creación tiene un punto fundamental: la conexión.

4.3.1.1. Algoritmo de conexión

La conexión permite que nuevos nodos sean añadidos al sistema, la cual cosa induce a la creación de más áreas que provocan un crecimiento de la propia red. Éste crecimiento, a su vez, crea una tendencia de incremento de la capacidad total de cómputo del sistema hecho que permitirá afrontar retos computacionales cada vez más complejos.

Consideraciones

- $Peer$: nodo que solicita entrar en el sistema.
- N_{Major}: punto de entrada del sistema; además registra los mánagers que se crean en el sistema, para distribuir de esta manera, el punto de entrada de nuevos nodos en sucesivas etapas. Resumiendo: $N_{Major} = \{M_{A_i} | 1 \leq i \leq n \wedge M_i \, is \, an \, entry \, point\}$.

- Una área (o grupo) A_i es el conjunto formado por el mánager del área M_{A_i} y un grupo de peers (mánagers o workers) controlados directamente por ese mánager, es decir, $A_i = \{M_{A_i}, P_j \mid 1 \leq j \leq Capacidad_{A_i} - 1 \wedge P_j \epsilon M_i\}$.
  - $Capacidad_{Área}$: número de peers máximo que caben en esa área.
- *Mánager*: es el mánager con el que negocia, en cada momento, el *Peer* para entrar en el sistema. Un mánager M_{A_i} de una área A_i es el conjunto de peers (mánagers o workers) de esa área que controla directamente, es decir, $M_{A_i} = \{P_i \mid 1 \leq i \leq Capacity_{A_i} \wedge P_i \epsilon A_i\}$.
- La función $PeerControl(M_{A_i})$ devuelve el número total de peers que controla, tanto directamente como indirectamente, el mánager M_{A_i}.

El algoritmo de conexión al sistema se especifica en en 1.

Algorithm 1: Algoritmo de conexión del sistema.

Require: ($Peer$, N_{Major}) : Parámetros de entrada
  Peer consulta a N_{Major}
  if $\exists M_{A_i} \epsilon N_{Major}$ **then**
    $Mánager := M_{A_i}$
    $Área := A_i$
    if $|Área| < Capacidad_{Área}$ **then**
      $W_i := Peer$
      $Mánager := Mánager \cup \{W_i\}$
      $Área := Área \cup \{W_i\}$
    else
      if $\exists M_{A_j} \epsilon Mánager \,||A_j| < Capacidad_{A_j}$ **then**
        $Mánager := M_{A_j}$
        $Área := A_j$
        Ir al paso 6
      else
        if $\exists W_j \epsilon Área$ **then**
          $Mánager := W_j$
          $Área := \{Mánager\}$
          Ir al paso 6
        else
          $Mánager := M_{A_k} | M_{A_k} \epsilon Área \wedge \forall M_{A_f} \epsilon Área \; PeerControl(M_{A_k}) < PeerControl(M_{A_f})$
          $Área := A_k$
          Ir al paso 6
        end if
      end if
    end if
  else
    $Mánager := Peer$
    $Área := \{Mánager\}$
    $N_{Major} := N_{Major} \cup \{Mánager\}$
  end if

Diagramas

En la figura 4.8 se muestran los pasos que se suceden en el algoritmo de conexión al sistema.

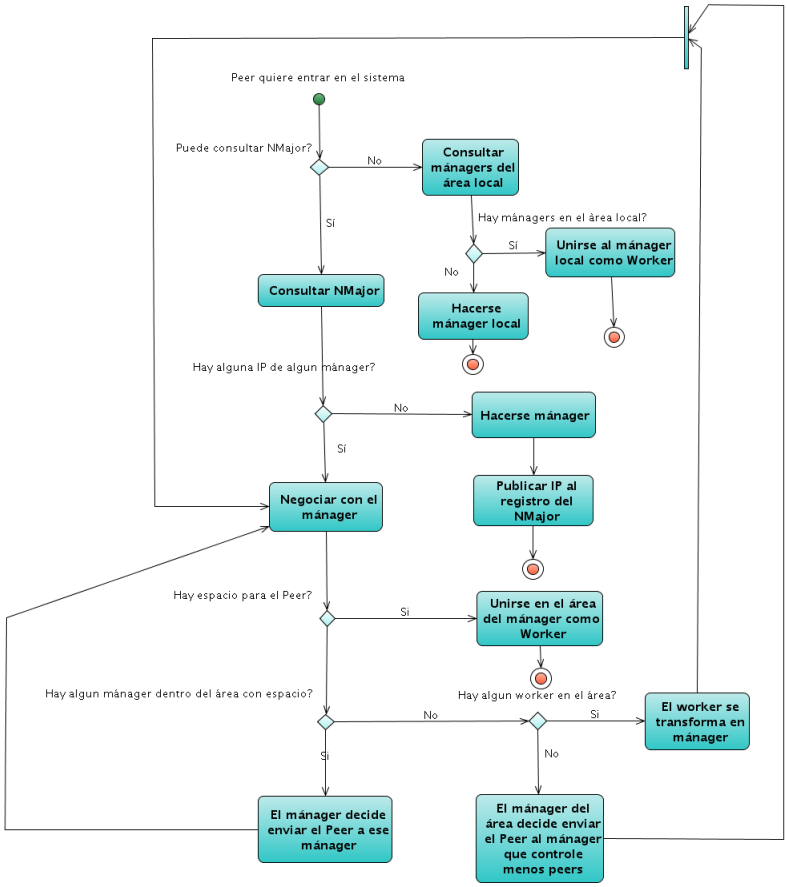

Figura 4.8: Diagrama de secuencia del algoritmo de conexión.

Coste del algoritmo

El coste del algoritmo de inserción viene determinado por el número de consultas que se realizan a los mánagers para conocer si el nuevo peer entrante tiene cabida o no en el área de éstos.

El pero de los casos sucede cuando la altura del árbol es considerable, es decir, cuando hay gran cantidad de niveles en la estructura, debido a que hay gran cantidad de áreas o grupos llenos. Entonces se sucede una ristra de comprobaciones que puede retardar el acceso del nuevo peer al sistema, ya que tal y como se ha visto en el algoritmo, las comprobaciones (reiteramos en el peor de los casos) se irán sucediendo recursivamente de los nieveles superiores a los inferiores, donde finalmente, encontrará áreas con espacios suficientes para albergar la entrada de nuevos peers.

El coste del algoritmo, se basa en el número de veces que un peer envía una petición de entrada a un mánager que tiene coste $\theta(h)$, donde h es el número de niveles del árbol. Si se especifica el coste dado en función de los peers, se obtiene un coste equivalente a $\theta(log_{CAPACIDAD}(N) - 1)$,

donde N es el número total de peers y $CAPACIDAD$ es el número máximo de peers que puede albergar una área.

4.3.2. Mantenimiento del sistema

Entendemos como mantenimiento del sistema, al conjunto de procedimientos que permiten detectar y reparar la caída de peers de la red de cómputo distribuido, indistintamente del rol que desarrollaba el peer caído.

Dicho mantenimiento tiene dos puntos fundamentales: la reasignación y la actualización. La reasignación nos permite compactar la estructura de la red peer-to-peer siempre que haya uno o varios nodos que se desconecten del sistema (voluntaria o involuntariamente).

En el algoritmo de reasignación, se usa con asiduidad el concepto de mánager de reserva o replicado especificado.

Para conocer con un mayor detalle ambos mecanismos o algoritmos que permiten mantener el sistema, vamos a mostrar las iteraciones y pasos que tienen, así como un diagrama de flujo que ayudarán a entender la idea que queremos plasmar en el sistema.

4.3.2.1. Algoritmo de actualización

En un sistema de cómputo distribuido, es esencial controlar los recursos computacionales disponibles existentes para así, facilitar el lanzamiento y distribución de aplicaciones con requerimientos de recursos de cómputo. Un conjunto de intercambio de mensajes, constituyen el mecanismo de actualización esbozado anteriormente.

Consideraciones

1. Cada X segundos, los Mánagers envían un mensaje del tipo *Mánager Alive* a cada uno de los peers que controla. En función de si el peer es o no un mánager de reserva[3] se enviará la siguiente información:

   a) Caso negativo: solo se envía el mensaje *Mánager Alive*.

   b) Caso afirmativo: a parte del mensaje común, se le enviará toda la información que contiene el Mánager.

2. Cada X segundos los peers que controla un mánager le envían al mismo según el rol que desarrollan dentro del área lo siguiente:

   a) Si el peer es a la vez mánager de otra área:

[3]Cada mánager escoge sus peers de reserva en función del orden de entrada dentro del área. Cada uno de los elegidos tendrá una prioridad determinada por el mismo orden de entrada. Por lo tanto, serán peers de reserva los N primeros en unirse al área. Si un mánager de reserva se desconecta, el mánager escogerá al candidato que haya permanecido más tiempo en el área usando el siguiente mensaje de *Mánager Alive*.

1) Un entero que contendrá la suma del total capacidad de cómputo de los peers que controla.
2) Un entero que contendrá el número de workers que controla directamente.
3) Un entero que contendrá el número de mánagers que controla directamente.
4) Un entero que contendrá el número de peers (workers y mánagers) que controlan los distintos mánagers (se entiende de subáreas) que existen en aquella área.

b) Si el peer es un simple worker peers:

1) Un entero que contendrá la capacidad de cómputo disponible.
2) **NOTA:** también se enviarán los campos vistos en el punto 2.*a* con valores establecidos a *NULL* para aprovechar la implementación de los mensajes.

Mánager

El algoritmo de actualización del mánager, consiste en que, cada X segundos, el *Mánager* envía a todos los peers $P_i \epsilon Mánager$ que no son de reserva un mensaje de *ManagerAlive*; y a todos los peers $P_i \epsilon Mánager$ de reserva les envía el *ManagerAlive* más la información replicada del mánager. Vamos a analizar primero el algoritmo de actualización del sistema desde la vertiente del mánager en 2.

Consideraciones

El tipo de información del mensaje de tipo *ManagerAlive* es solamente un mensaje informativo que permite informar a los workers que controla que sigue ejerciendo las tareas de mánager del área o grupo. Por tanto, simplemente contendrá una cadena de texto que contendrá el mismo nombre del mensaje.

El algoritmo de actualización del sistema desde la vertiente del peer mánager se especifica en en 2.

Algorithm 2: Algoritmo de actualización del sistema (Mánager).

Require: (*Mánager*) : Parámetros de entrada
  loop
    for all ($P_i \epsilon Mánager$) **do**
      if P_i es un *PeerReserva* **then**
        Mánager envía $MánagerAlive + InformaciónReplicadaa P_i$
      else
        Mánager envía *MánagerAlive* a P_i
      end if
    end for
    sleep(x segundos)
  end loop

Worker

El algoritmo del actualización de un worker, consiste en que cada X segundos el peer $P_i \epsilon Mánager$

envía un mensaje del tipo *PeerInfo* a *Mánager*. Si el peer es un worker solo envía el mensaje *PeerInfo* indicando que sigue conectado al sistema; en cambio, si el peer es mánager de una subárea se envía el *PeerInfo* más la información de este.

Consideraciones

El tipo de información del mensaje de tipo *PeerInfo* es la siguiente:

- Suma de todas las CPUs de los peers que controla.
- Número de workers que gestiona directamente.
- Número de mánagers que gestiona directamente.
- Número total de peers (workers + mánagers) de su rama, es decir, el número de peers que controla tanto directamente como indirectamente.
- CPU disponible, de todos los workers que controla tanto directamente como indirectamente.
- Número de workers disponibles, que gestiona tanto directamente como indirectamente.

El algoritmo de actualización del sistema desde la vertiente del peer worker se especifica en en 3.

Algorithm 3: Algoritmo de actualización del sistema (Peer).

Require: *Mánager*: Parámetros de entrada.
  loop
    for all ($P_i \epsilon Mánager$) **do**
      if P_i es un Worker **then**
        P_i envía $PeerInfo$ un *Mánager*
      else if P_i es una Mánager **then**
        P_i envía $PeerInfo + InformacióndelMánagerM$ a *Mánager*
      end if
    end for
    sleep(x segundos)
  end loop

Diagramas

En el dibujo de la figura 4.9 se puede diferenciar el tipo de peer que controla el mánager y el tipo de mensaje que se intercambian entre ellos.

En la figura 4.10 se muestran los pasos que se suceden en el algoritmo de actualización del sistema.

Coste del algoritmo

El coste del envío de mensajes de actualización de mánager a workers y viceversa será de $\theta(2 \cdot CAPACIDAD)$, donde $CAPACIDAD$ es el número máximo de peers que puede albergar una área. Se puede determinar que $\theta(2 \cdot N) \simeq \theta(0)$.

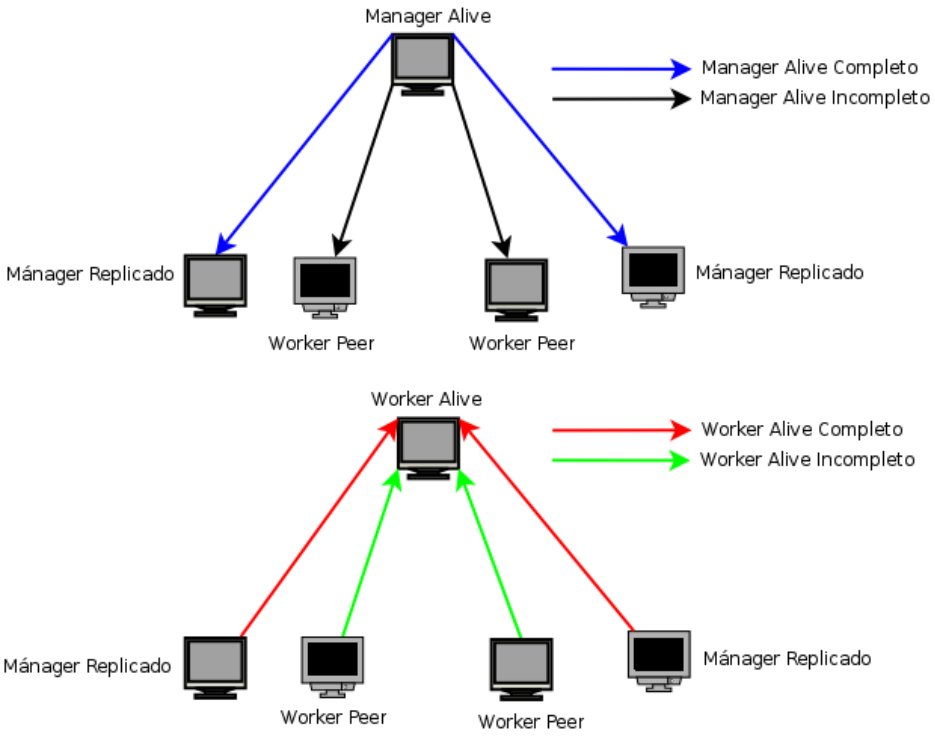

Figura 4.9: Esquema del tipo de peers y mensajes existentes en el sistema.

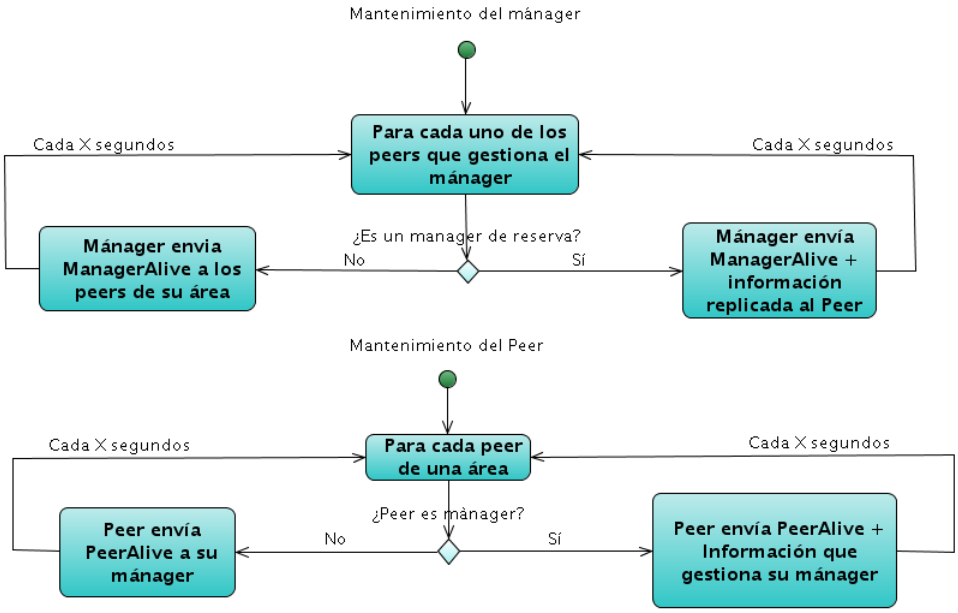

Figura 4.10: Diagrama de secuencia del algoritmo de actualización.

4.3.2.2. Algoritmo de reasignación

El algoritmo de reasignación, analizado en 4 se ejecuta cada vez que se detecta que un mánager de una área se ha desconectado de forma inesperada y se tiene que reestructurar el sistema.

Consideraciones

- Una área A_i es el conjunto formado por el mánager del área M_{A_i} y un grupo de peers (mánagers o workers) controlados directamente por ese mánager, es decir, $A_i = \{M_{A_i}, P_j \mid 1 \leq j \leq Capacidad_{A_i} - 1 \wedge P_j \epsilon M_{A_i}\}$.
- $Mánager$: es el mánager que se desconecta inesperadamente.
- $Peer_{Candidate}$ es el peer de reserva candidato a sustituir al mánager caído.
- $ReservePeer(P_i)$ es una función que devuelve si el peer P_i es un peer de reserva o no.
- $Time(P_i)$ es una función que devuelve el tiempo que el peer P_i lleva conectado al sistema.

El algoritmo de reasignación del sistema se especifica en en 4.

Algorithm 4: Algoritmo de reasignación del sistema.

Require: $Máanager \epsilon Área$ se desconecta inesperadamente
  while ($Área$ no tiene mánager) **do**
    $P_i \epsilon Área$ detecta que $Área$ no tiene mánager
    $Área := Área - \{Mánager\}$
    if $ReservePeer(P_i) = true$ **then**
      $Peer_{Candidato} := P_i$
      if $Peer_{Candidato}$ es un mánager de A_j **then**
        for all ($P_j \epsilon A_j$) **do**
          $P_{Candidato}$ notifica P_j que se convertirá en el mánager del $Área$
        end for
        $Área := Área - \{Peer_{Candidato}\}$
        $Mánager := Peer_{Candidate}$
        $Área := Área \cup \{Mánager\}$
        $Peer_{Candidato} := P_k | P_k \epsilon A_j \, ReservePeer(P_k) = true \wedge (\forall P_f \epsilon A_j | ReservePeer(P_f) = true \, Time(P_k) < Time(P_f))$
        $Área := A_j$
        Ir al paso 6
      else
        $Área := Área - \{Peer_{Candidato}\}$
        $Mánager := Peer_{Candidato}$
        $Área := Área \cup \{Mánager\}$
      end if
    end if
  end while

Diagramas

En el dibujo de la figura 4.11 se puede ver el algoritmo de reasignación esquematizado en un diagrama de flujo.

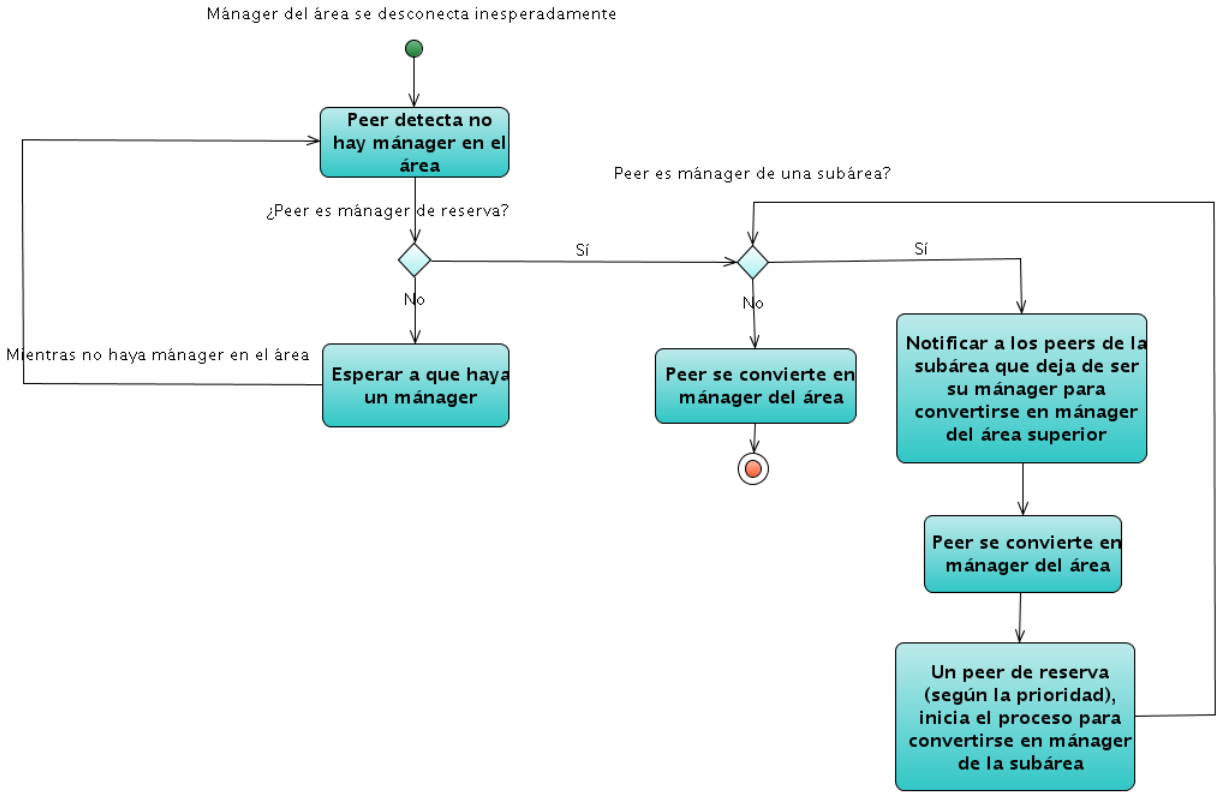

Figura 4.11: Diagrama de secuencia del algoritmo de reasignación.

Coste del algoritmo

El coste del algoritmo está determinado por el número de peers que resultan afectados por la caída de un peer. En el peor de los casos cuando un peer cae del sistema, el algoritmo comprueba si el peer tiene el rol de mánager, en caso afirmativo tiene que escoger un peer que se convertirá en mánager de reserva. Si el peer nombrado mánager de reserva, es a la vez mánager de una subárea y éste cae, el algoritmo procederá a realizar el mismo ejercicio, con lo cual, se tendrá que realizar el procedimiento de forma recursiva hasta llegar a un peer de reserva de niveles inferiores que no resulta ser mánager de ningún subárea. La tendencia descrita solamente sucederá cuando el sistema esté soportando un gran número de peers y de niveles.

El coste teórico del algoritmo, entonces, resultará ser el mismo que el coste del algoritmo de conexión, es decir $\theta(log_{CAPACIDAD}(N)-1)$, donde N es el número total de peers y $CAPACIDAD$ es el número máximo de peers que puede albergar una área.

4.3.3. Ejecución de aplicaciones

Entendemos como lanzamiento de aplicaciones al sistema el conjunto de procedimientos que permiten planificar y ejecutar aplicaciones de cómputo de forma distribuida a lo largo de toda la red p2p.

La ejecución de aplicaciones está fundamentada en dos puntos: el lanzamiento de aplicaciones y planificación de tareas. El lanzamiento de aplicaciones permite a cualquier peer del sistema sea cual sea su rol, hacer una petición global de ejecución de una aplicación con requerimientos de cómputo. La planificación de las tareas, es el proceso por el cual se seleccionan los workers del sistema para que ejecuten las tareas que forman la aplicación global lanzada.

4.3.3.1. Algoritmo de Lanzamiento

El algoritmo de lanzamiento de aplicaciones se analiza en 5. Antes, se deben tener en cuenta las siguientes consideraciones:

Consideraciones

- P_{Job} es el peer que lanza la aplicación.
- $M\acute{a}nager$ es el mánager con el que P_{Job} negocia para poder lanzar la aplicación.
- $Queue_{M\acute{a}nager}$ es la cola del mánager representada por un conjunto de trabajos.
- Una rama B_i son todos los peers que gestiona un mánager M_{B_i}, tanto directamente como indirectamente, incluido éste, es decir, $B_i = \{M_{B_i}, P_1, \ldots, P_n | 1 \leq n < \alpha)$.
  - $Rama$: Es la rama de mánager actual.
  - $NWorkersDisp_{B_i}$: Es el número de workers disponibles que hay en la rama B_i.
  - $NWorkersTotals_{B_i}$: Es el número total de workers (disponibles o no) que hay en la rama B_i.
- Job es la aplicación a lanzar y es un conjunto de tareas, es decir, $Job = \{T_1, T_2, \ldots, T_n | 1 \leq n < \alpha)$.
- Se dice que un worker está disponible cuando no ejecuta ninguna otra tarea lanzada por un peer de la plataforma CoDiP2P.
- $W_{choosen}$ es el conjunto de workers disponibles escogidos por el mánager encargado de realizar la planificación.
- $RequestJob$ es el mensaje de negociación entre P_{Job} y $M\acute{a}nager$. Contiene:
  - $NWorkers$: Número de workers necesarios para ejecutar la tarea.
  - QoS: Número máximo de veces que se quiere que se ejecute la misma tarea.

- *Agree* es el mensaje que envía *Mánager* a P_{job} para indicarle que se puede ejecutar el trabajo.
- *RequestTask* es el mensaje que envía un worker $W_i \epsilon W_{choosen}$ a P_{Job} para pedirle a P_{Job} que le envíe su tarea $T_i \epsilon Job$ que le corresponde.
- *SendTask* es el mensaje que contiene una tarea $T_i \epsilon Job$.
- *FinishedTask* es el mensaje que indica que se ha terminado de ejecutar una tarea y contiene los resultados de la tarea.
- *FinishedJob* es el mensaje que indica al *Mánager* que la ejecución de un trabajo ha finalizado correctamente.
- *WaitRequest* es el mensaje enviado por el *Mánager* a T_{Job} preguntándole si este quiere esperar a que se pueda ejecutar su trabajo en su rama B_i, o quiere negociar con un mánager del nivel superior. Este contiene:
  - Número de trabajos que se deben ejecutar antes de poderse ejecutar.
  - CPU disponible B_i.
  - CPU total B_i.
- *Work* es el mensaje que envía P_{Job} al *Mánager* para indicarle a éste que quiere ejecutar en este nivel.
- *NoWait* es el mensaje que envía P_{Job} al *Mánager* para indicarle a éste que quiere ejecutar la tarea en un nivel superior.

El algoritmo de lanzamiento de aplicaciones al sistema se especifica en en 5.

Por lo tanto, el usuario que envía petición de ejecución de la aplicación siempre tiene las opciones de esperar (WaitResponse) para ejecutar en caso que haya capacidad computacional en la rama actual pero esta, no esté ociosa y por otro lado, también tiene la opción de no esperar (*NoWait*) y para que dicha petición se envíe en un nivel superior de la red.

Diagramas

En la figura 4.12 se muestran los pasos que se suceden en el algoritmo de lanzamiento de aplicaciones al sistema.

Gracias a la figura 4.13 y a su explicación, se puede entender mejor el algoritmo de envío o *dispatch* del sistema.

Tal y como muestra la figura 4.13 hay un peer que quiere que se ejecute una aplicación el la red p2p. Este peer se trata del worker *Peer B.a.1* y la aplicación tiene el nombre "X" que se encuentra dividida en 5 tareas. Consideraremos que todos los workers de la figura 4.13 están libres para ejecutar tareas. Los pasos que se irán sucediendo serán:

1. *Peer B.a.1* envía un mensaje del tipo *RequestJob* a su mánager *Peer B.a.*

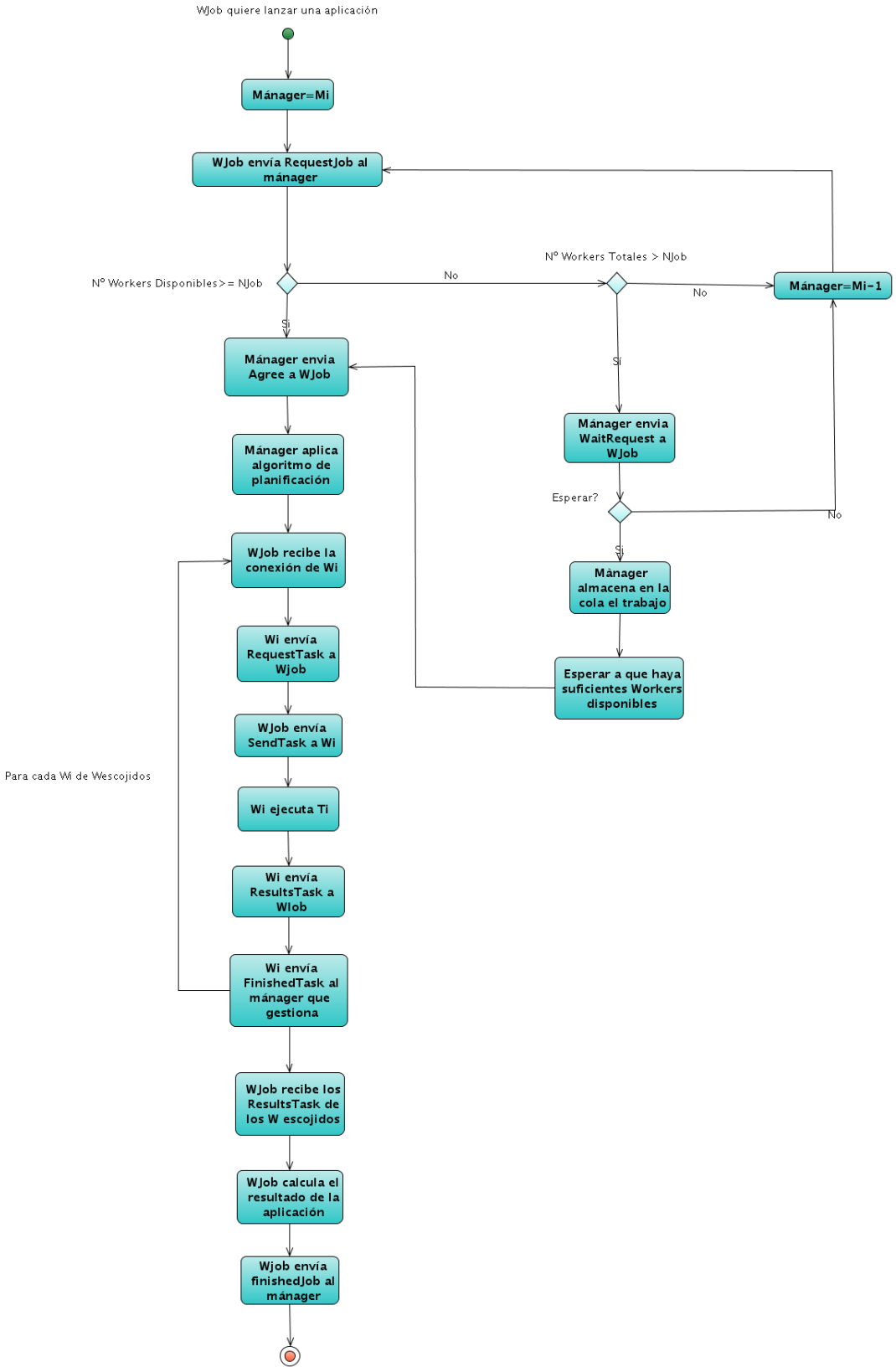

Figura 4.12: Diagrama de secuencia del algoritmo de lanzamiento de aplicaciones.

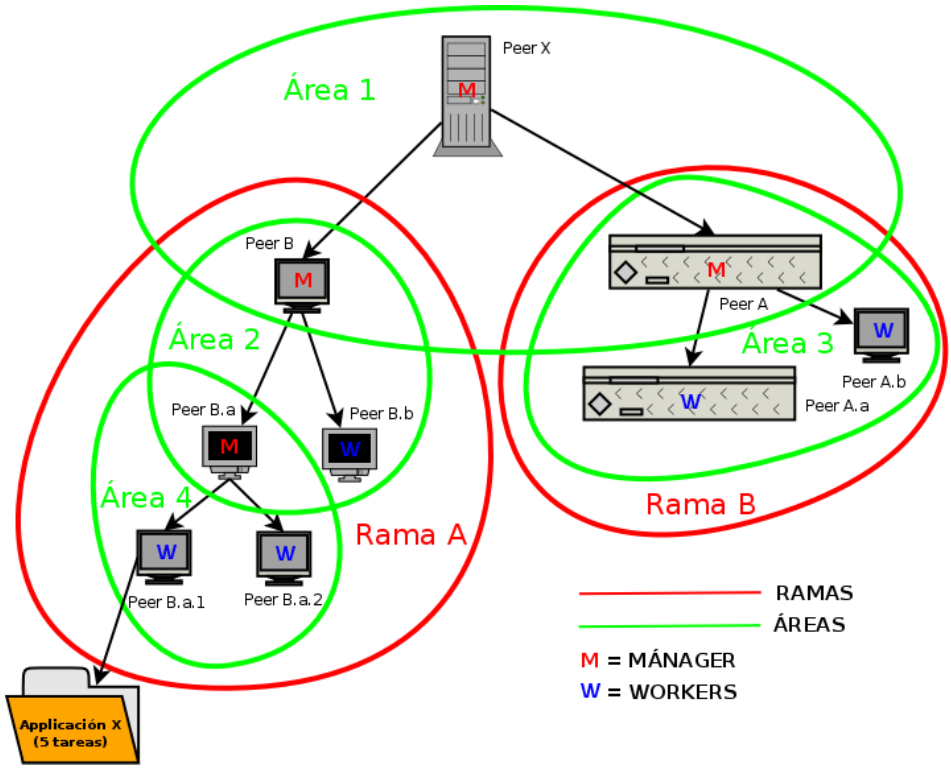

Figura 4.13: Captura (figurada) de un envío de una aplicación en el sistema.

2. El mánager *Peer B.a* comprueba si la rama que controla (compuesta por *Peer B.a.1* y *Peer B.a.2*) tiene suficientes workers para ejecutar la aplicación. Tiene dos workers y la aplicación requiere 5 workers.

3. El mánager *Peer B.a* reenvía la petición *RequestJob* a su mánager, es decir, *Peer B*.

4. El mánager *Peer B* comprueba si su rama (Rama A) tiene suficientes workers para ejecutar la aplicación. La Rama A tiene de 3 workers, por lo tanto es una capacidad de cómputo insuficiente.

5. El mánager *Peer B* reenvía la petición *RequestJob* a su mánager, es decir, *Peer X*.

6. El mánager *Peer X* comprueba si sus ramas (Rama A y Rama B) tienen entre las dos suficientes workers para ejecutar la aplicación. La Rama A consta de 3 workers, la Rama B tiene 2 workers por lo tanto es una capacidad de cómputo total de 5 workers, suficiente, para ejecutar la aplicación X que requiere 5 trabajadores.

7. Una vez se notifica (siguiendo el camino inverso) al *Peer B.a.1* que se puede lanzar la aplicación, este último da su visto bueno enviando un mensaje del tipo *SendJob*.

8. El mánager *Peer X* informa a sus trabajadores inmediatos de las tareas que tienen que planificar. Al *Peer B* le indica que deberá ejecutar 3 tareas de la aplicación X, ya que son 3 workers los que tiene a cargo. Por otro lado, indica al *Peer A* que tiene que planificar 2 tareas de la aplicación X, ya que son 2 workers los que tiene a cargo.

9. El mánager *Peer B* a su vez informa al *Peer B.a* que deberá ejecutar 2 tareas de la aplicación X, ya que son 2 workers los que tiene a cargo. Al mismo tiempo, notificará al peer *Peer B.b* que tiene que ejecutar una tarea, para ello le indica que si precisa datos o librerías para ejecutar la tarea deberá ponerse en contacto con el peer que ha lanzado la tarea: *Peer B.a.1*.

10. El mánager *Peer B.a* informará a sus workers *Peer B.a.1* y *Peer B.a.2* que tienen que ejecutar una tarea cada uno. De forma paralela el mánager notifica a *Peer B.a.2* que deberá comunicarse con *Peer B.a.1* si precisa datos o librerías para ejecutar la tarea.

11. El mánager *Peer A* informará a sus workers *Peer A.a* y *Peer A.b* que tienen que ejecutar una tarea cada uno. De forma paralela el mánager notifica ambos workers que deberán comunicarse con *Peer B.a.1* si precisan datos o librerías para ejecutar las tareas.

12. Los distintos workers se comunicarán con el peer *Peer B.a.2* para solicitarle archivos o datos para ejecutar las tareas mediante un mensaje del tipo *RequestJob*.

13. Una vez finalizadas las tareas, los distintos workers enviarán al peer *Peer B.a.2* el mensaje *ResultsTasks* con los resultados de la tarea.

14. Por último el peer *Peer B.a.2* notifica a su mánager *Peer B.a* que ha finalizado la aplicación mediante un mensaje del tipo *FinishedJob*.

Coste del algoritmo
El coste de éste algoritmo viene determinado por el número de consultas que se han de realizar a los mánagers del sistema para averiguar el número de workers disponibles que permitirá decidir si se puede lanzar una tarea o no. Por tanto, en el peor de los casos en que el peer que lanza la tarea se encuentra en el nivel inferior de la estructura y los workers necesarios para ejecutar una tarea son todos los del sistema, la petición de lanzamiento circulará del mánager inferior a los mánagers de niveles superiores hasta llegar al mánager del nivel superior. El coste de esta operación es $\theta(log_{CAPACIDAD}(N) - 1)$, donde N es el número total de peers y $CAPACIDAD$ es el número máximo de peers que puede albergar una área.

4.3.3.2. Algoritmo de planificación

Entendemos como planificación de tareas dentro del sistema, al conjunto de procedimientos que permiten distribuir las tareas a los workers una vez se ha lanzado una aplicación de cómputo de forma distribuida a lo largo de la red peer-to-peer.

La planificación de las tareas una vez lanzada una aplicación, consiste en asignar tareas a los workers, eso si, teniendo en cuenta del número de tareas a ejecutar, el número de áreas, mánagers y workers ociosos que hay en el sistema.

Se deben tener en cuenta las siguientes consideraciones:

Consideraciones

- P_{Job} es el peer que lanza la tarea.
- *Mánager* es el mánager encargado de planificar las tareas entre los distintos workers disponibles que tiene conectados.
- Una rama B_i son todos los peers que gestiona un mánager M_{B_i}, tanto directamente como indirectamente, incluido este, es decir $B_i = \{M_{B_i}, P_1, ..., P_n | 1 \leq n < \alpha)$.
  - *Rama* : es la rama gestionada por el mánager actual.
  - $NWorkersDisp_{B_i}$: es el número de workers disponibles que hay en la rama B_i.
- *Job* es la aplicación a lanzar y es un conjunto de tareas, es decir, $Job = \{T_1, T_2, ..., T_n | 1 \leq n < \alpha)$.
- *JobAddress* es el mensaje que contiene la dirección de P_{Job}.
- La instrucción $fork$ significa que simultáneamente a la ejecución principal del algoritmo inicia otro proceso de la ejecución de éste por el punto indicado.

El algoritmo de planificación de tareas se analiza en 6.

Diagramas
En la figura 4.14 se muestran los pasos que se suceden en el algoritmo de planificación de tareas dentro del sistema.

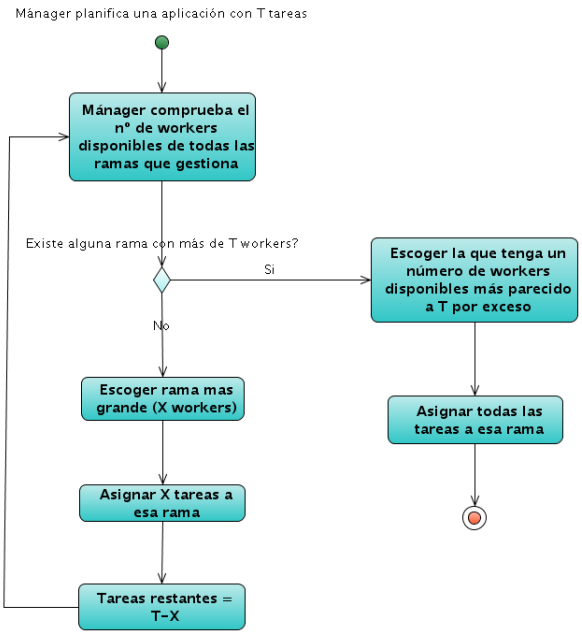

Figura 4.14: Diagrama de secuencia del algoritmo de planificación de tareas.

Coste del algoritmo

El coste de éste algoritmo viene determinado por el número de consultas que se han de realizar a los mánagers del sistema para averiguar el número de workers disponibles para poder planificar a posteriori las tareas que forman la aplicación. Por tanto, en el peor de los casos en que el peer que lanza la tarea se encuentra en el nivel inferior de la estructura y los workers necesarios para ejecutar una tarea son todos los del sistema, la petición de lanzamiento circulará del mánager inferior a los mánagers de niveles superiores hasta llegar al mánager del nivel superior. Una vez llegado al punto destino, será preciso planificar las tareas. Al ser necesarios todos los peers del sistema, se volverá a recorrer el árbol, en éste caso de los niveles superiores a niveles inferiores, dando un coste igual a $\theta(log_{CAPACIDAD}(N) - 1)$, donde N es el número total de peers y $CAPACIDAD$ es el número máximo de peers que puede albergar una área.

Sumando el coste de lanzamiento de una aplicación y el coste de planificar las tareas que forman la aplicación obtenemos $\theta(2 \cdot log_{CAPACIDAD}(N) - 1)$, donde N es el número total de peers y $CAPACIDAD$ es el número máximo de peers que puede albergar una área.

Algorithm 5: Algoritmo de lanzamiento de tareas al sistema.

Require: P_{Job}, Job: Parámetros de entrada
  $P_{Job} \epsilon M_i$ envía $RequestJob$ a M_i
  $Mánager := M_{B_i}$
  $Rama := B_i$
  if $NWorkersDisp_{Rama} \geq |Job|$ **then**
    $Mánager$ envía $Agree$ a P_{Job}
    $Mánager$ ejecuta el algoritmo de **Planificación**
    P_{Job} recibe conexiones de $W_{escogido}$
    for all $(W_i \epsilon W_{escogido})$ **do**
      W_i envía $RequestTask$ a P_{Job}
      P_{Job} envía $SendTask$ a W_i
      W_i ejecuta $T_i \epsilon Job$
      W_i envía $FinishedTask$ a P_{Job}
      W_i envía $FinishedTask$ a $M_{A_j} | W_i \epsilon M_{A_j}$
    end for
    P_{Job} calculate the result of Job
    P_{Job} envía $FinishedJob$ a Mánager
  else if $NWorkersTotals_{B_i} \geq |Job|$ **then**
    $Mánager$ envía $WaitRequest$ a P_{Job}
    P_{Job} responde $Mánager$ con un $message$
    if $message = Work$ **then**
      $Queue_{Mánager} = Queue_{Mánager} \cup \{Job\}$
      while (No sea su turno) **do**
        No hacer nada
      end while
      Ir al paso 5
    else if $message = NoWait$ **then**
      if $\exists M_{B_{i-1}}$ **then**
        $Mánager := M_{B_{i-1}}$
        $Rama := B_{i-1}$
        Ir al paso 5
      else
        $Mánager$ envía NotAgree a P_{Job}
      end if
    end if
  else
    Ir al paso 28
  end if

Algorithm 6: Algoritmo de planificación de tareas al sistema.

Require: M_{A_i}, B_i, Job: Parámetros de entrada
  while $(|Job| \neq 0)$ **do**
    $Mánager := M_{A_i}$
    $Rama := B_i$
    if $\exists B_j \subseteq Rama | NWorkersDips_{B_j} = |Job|$ **then**
      $Mánager$ envía $JobAddress$ to M_{B_j}
      $Mánager := M_{B_j}$
      $Rama := B_j$
      while $(\exists W_j \epsilon Mánager \wedge W_j \, is \, available \wedge |Job| \neq 0)$ **do**
        $Mánager$ envía $JobAddress$ a W_j
        W_j conecta con P_{Job}
        $Job := Job - \{T_i\}$
      end while
      while $(\exists M_{B_k} \epsilon Mánager \wedge |Job| \neq 0)$ **do**
        $Job_{copia} := Job$
        $Job := SubJob_i | SubJob_i \subseteq Job \wedge |SubJob_i| = NWorkersDisp_{B_k}$
        $Mánager$ envía $ScheduleJob$ a M_{B_k}
        $Rama_{copia} := Rama$
        $Mánager_{copia} := Mánager$
        $Mánager := M_{B_k}$
        $Rama := B_{B_k}$
        Fork hacia 8
        $Job := Job - Job_{copia}$
        $Rama := Rama_{copia}$
        $Mánager := Mánager_{copia}$
      end while
    else if $\exists B_j \subseteq Rama | \; NWorkersDisp_{B_j} > |Job|$ **then**
      $Rama := B_j | \forall B_k \subseteq Rama \, NWorkersDisp_{B_j} < NWorkersDisp_{B_k}$
      $Mánager := M_{B_j} | M_{B_j} \epsilon B_j$
      Ir al paso 8
    else
      $Rama_{copia} := Rama$
      $Mánager_{copia} := Mánager$
      $Rama := B_j | B_j \subseteq Rama \wedge \forall B_k \subseteq Rama \, NWorkersDisp_{B_j} > NWorkersDisp_{B_k}$
      $Mánager := M_{B_j} | M_{B_j} \epsilon B_j$
      $Job_{copia} := Job$
      $Job := SubJob_i | SubJob_i \subseteq Job \wedge |SubJob_i| = NWorkersDisp_{B_j}$
      Fork hacia 8
      $Job := Job - Job_{copia}$
      $Rama := Rama_{copia}$
      $Mánager := Mánager_{copia}$
    end if
  end while

Capítulo 5

Simulación

En el capítulo siguiente, se especificarán los puntos más importantes que forman parte de la simulación realizada en éste proyecto para tener un punto de referencia al simular el sistema.

Para llevar a cabo proyectos de investigación hay dos opciones marcadas.

Una de las soluciones se basa en el hecho de llevar a cabo gran cantidad de ensayos o experimentos con plataformas reales que ayuden a consolidar experiencia para ir customizando los sistemas hasta que se logra un óptimo, esta solución, resulta intratable debido a la gran cantidad de tiempo y recursos requeridos para implementar distintos sistemas destinados a la toma de decisiones.

Otra solución es la que aporta la simulación.

Un modelo es una imagen o una representación de un sistema, generalmente simplificada o incompleta. Llamaremos simulación a la experimentación con un modelo para extraer conclusiones o realizar predicciones.

La simulación ayuda en el desarrollo de proyectos complejos, ya que permite imitar ciertos aspectos de la realidad (modelo) y establecer en ese ambiente situaciones similares en las que los desarrolladores se pueden encontrar durante la evolución del proyecto, de manera que se puede "experimentar" sin riesgo y extraer conclusiones.

La simulación parece ser la única forma factible de analizar algoritmos a gran escala de sistemas distribuidos con recursos heterogéneos. A diferencia de los sistemas reales, la simulación permite analizar puntos concretos de un modelo, sin la necesidad de considerar o analizar partes complejas que pueden resultar innecesarias, evitando así, la sobrecarga que supone armonizar o coordinar un abanico amplio de recursos reales.

La simulación es también eficaz cuando se trabaja con problemas grandes en los que se requiere un número elevado de usuarios y recursos compartidos, tarea nada sencilla, ya que requiere una capacidad de concentración y organización a gran escala, solamente alcanzables para equipos de investigación con grandes infraestructuras.

A continuación se presentará la herramienta de simulación elegida, las adaptaciones que se han realizado para portar ésta herramienta en un entorno peer-to-peer y por último, los resultados obtenidos después de haber realizado un conjunto de simulaciones sobre el modelo propuesto.

5.1. GridSIM

El objetivo principal del proyecto *GridSIM* es investigar las técnicas eficaces de asignación de recursos basado en la economía a través de la simulación computacional. Es necesaria la simulación cuando se tiene que analizar un modelo que constan de millones de recursos y millares de usuarios, la escalabilidad de un modelo, los algoritmos que implementa, la eficiencia de políticas de asignación de recursos así como la satisfacción de los usuarios entre otros. *GridSIM* también hace incapié en la exploración de la economía local y el posicionamiento global -por ejemplo, la zona horaria- de un recurso en particular que juega un rol en la seguridad de trabajos bajo distintos situaciones de demanda.

GridSIM permite modelar y simular entidades en paralelo distribución de aplicaciones y planificación de tareas, etc ... Además, ofrece una documentación que facilita la comprensión de la herramienta, con el fin de hacer de una forma sencilla la creación de distintas clases de recursos heterogéneos que pueden ser agregados utilizando agentes de recursos para resolver aplicaciones de cómputo intensivo. Un recursos puede ser un o múltiples procesadores con memoria compartida y distribuida, gestionada por los planificadores de tiempo y espacio compartido. Los nodos de proceso pueden ser -dentro de un recurso- heterogéneos en temas de capacidad de cómputo, configuración y disponibilidad. Los agentes de recursos utilizan unos algoritmos de planificación o políticas para la asignación de tareas a recursos para optimizar el sistema dependiendo de sus objetivos.

Las características más importantes de *GridSIM* son las siguientes:

- Incorpora un mecanismo de errores de los recursos durante el tiempo de ejecución.
- Incluye mecanismos o algoritmos de asignación de recursos.
- Su infraestructura soporta la reserva anticipada de un sistema Grid.
- Incorpora la funcionalidad de leer trazas de trabajo obtenidas de supercomputadores para simular un entorno lo más real posible.
- Incorpora un modelo de subastas, una extensión de *datagrids*[1] y una extensión de red donde los recursos y otras entidades pueden ser enlazadas con una topología de red.
- Incluye un entorno en modo *background* de funcionalidades de tránsito de red basado en una distribución probabilística que resulta útil para simular una red pública congestionada.

5.1.1. Adaptando GridSIM para un entorno Peer-To-Peer

GridSIM es un simulador basado en entornos Grids. Para sacar provecho a las funcionalidades de este simulador en nuestro sistema peer-to-peer se ha tenido que adaptar. Para adaptar el

[1]Son tablas de datos distribuidos pertenecientes a la información de una red. Los datagrids se usan para controlar y gestionar gran cantidad de datos distribuidos. Se usan de forma común (aunque no de forma exclusiva) con sistemas de computación en grid.

simulador al sistema peer-to-peer se ha definido cada peer con una entidad *GridUser* que es a un hilo de ejecución del simulador que se comunica con otros hilos a través del envío y recepción de mensajes o eventos. Cada peer tiene asociado un recurso Grid con una sola máquina y un valor determinado de potencia de CPU. Cada CPU de una máquina está formada por un solo elemento de procesador. El enlace de red entre el recurso Grid y el peer tiene un ancho de banda muy grande y una latencia muy baja para simular que peer y recurso Grid son una sola entidad, la misma.

Cada peer se comunica con los peers usando los pedidos de envío de mensajes de este simulador. El envío y recepción de mensajes normalmente se realiza de un modo asíncrono, es decir, cuando un peer envía un mensaje no sabe la respuesta del mismo de forma inmediata. También se ha añadido en ciertas partes del modela, la posibilidad de recibir mensajes de forma síncrona usando puertos de comunicación según la información que se debe enviar para evitar el recibimiento de información errónea de un mensaje fuera de contexto.

Además cada peer contiene todas las funciones de los algoritmos de inserción, actualización, lanzamiento de tareas y planificación. Todos los peers están enlazados con un router con sus respectivos anchos de banda y latencias que son parametrizadas.

El primer paso cuando se inicia el simulador, consiste en crear los peers, tarea realizada por una entidad que se encarga de leer un fichero de entrada pasado por parámetros con la declaración de los peers. Después los inicia y éstos se comunican con un nodo estático para saber quién será el mánager principal. El algoritmo de inserción decide donde se ubicará el peer que entra, con la finalidad de mantener el árbol balanceado.

A continuación los peers están conectados a la red peer-to-peer simulada y éstos se comunican con sus mánagers respectivos para hacer el intercambio de información de forma periódica que permite a éste almacenar información de estado de los workers para poder usarla en el algoritmo de lanzamiento y planificación. Este procedimiento constituirá el segundo paso.

El tercer paso consiste en el lanzamiento de tareas virtuales desde un peer determinado (especificado previamente en otro fichero de texto). Entonces, para cada trabajo, se aplica el algoritmo de lanzamiento de tareas en un momento especificado. Cuando el algoritmo de lanzamiento ha encontrado suficientes peers para ejecutar la tarea entonces se inicia la ejecución del algoritmo de planificación. Al final cada tarea queda asignada a un worker, que a continuación se comunica con su máquina Grid asociada para empezar a ejecutar la tarea.

El cuarto y último paso es la finalización de la simulación del sistema que se indica previamente a través de un valor de tiempo, momento en el que todos los peers finalizan de forma automática.

5.2. Resultados de la simulación

En esta sección se presentan los resultados obtenidos durante el proceso de simulación del modelo del sistema. Se han realizado multitud de pruebas con la obtención de un gran número de resultados. Solo los resultados que tienen cierta significación, serán presentados a continuación.

5.2.1. Capacidad de las áreas

Somos plenamente conscientes que el buen funcionamiento del sistema que se presenta en el TFM va a depender muy directamente del número de peers que las áreas del sistema sean capaces de albergar. Ya se han comentado anteriormente las grandes ventajas que tiene el hecho de que haya pocos niveles en el sistema y es evidente que un factor fundamental para lograr ese objetivo es que las áreas sean los más grandes posibles, es decir, que un mánager pueda controlar el máximo número de workers.

Por todo esto, lo primero que se ha querido averiguar mediante el simulador es el número de peers que un mánager es capaz de controlar, y por lo tanto, la capacidad de una área. Para ello, hay que tener en cuenta multitud de factores que se pueden simplificar en dos grandes grupos:

- **Características del propio sistema:** básicamente, forman parte de este grupo los diferentes mensajes que intercambían los peers entre ellos para realizar el mantenimiento de la estructura de la red.
- **Características de la conexión de los nodos:** ancho de banda, latencia, uso del ancho de banda por parte de otras aplicaciones que el usuario pueda estar ejecutando y que son ajenas a nuestro sistema, etc...

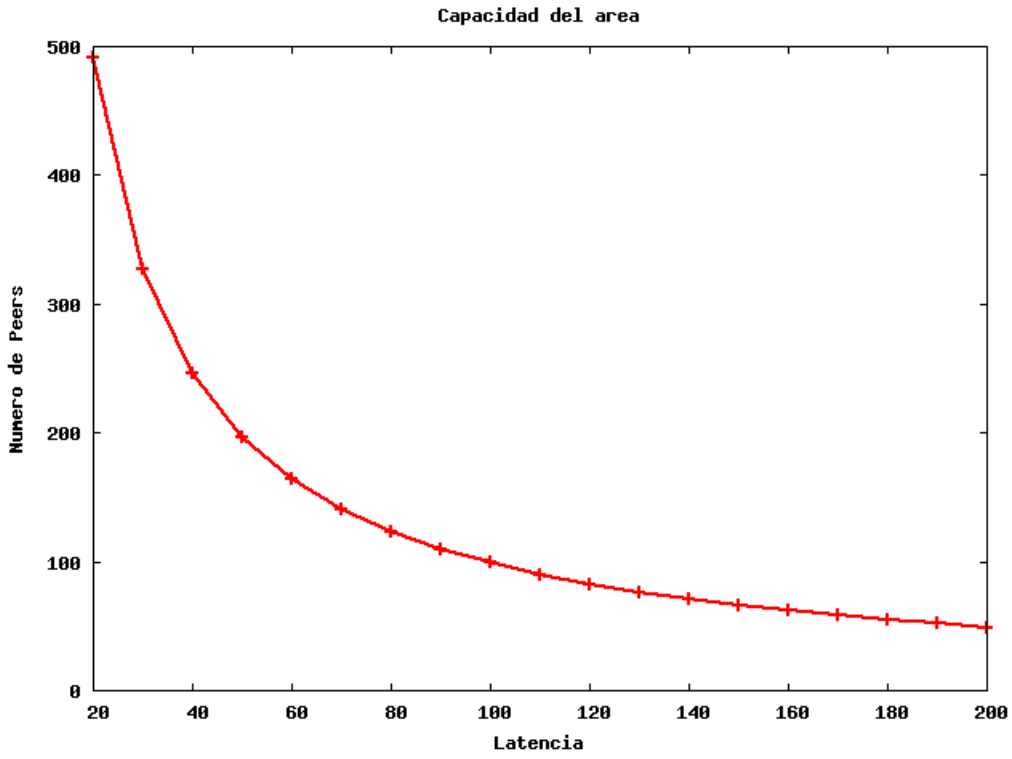

Figura 5.1: Capacidad de las áreas en función de la latencia.

De todos estos factores hay algunos sobre los que podemos influir como desarrolladores del sistema (básicamente los del primer grupo), sin embargo, hay otros que escapan completamente de

nuestro control (los del segundo grupo). Por otro lado, hemos observado que la latencia es el factor que más influye por encima del ancho de banda debido a que los mensajes de mantenimiento que fluyen por el sistema son abundantes pero cortos y por tanto el tiempo en modularlos por la red es más pequeño que la latencia.

En la gráfica 5.1 se muestra el número de peers que soporta el sistema en función de la latencia de la conexión.

5.2.2. Inserción de nodos al sistema

La gráfica 5.2 muestra el tiempo en segundos que tardan los peers en unirse al sistema en función de tres latencias de la conexión diferentes (10, 50, y 100 milisegundos). Todas las pruebas se han realizado con una conexión de 1Mbps y con tan solo 5 peers de capacidad máxima de las áreas. El motivo de escoger una capacidad tan baja se ha hecho para que se pueda apreciar mejor la penalización de tiempo que incorpora cada nivel extra de peers.

Otro punto a tener en cuenta es que, en estas pruebas, todos los nodos tienen el mismo punto de entrada al sistema, pero como ya se ha comentado en secciones anteriores, se han pensado en un mecanismo para que a la práctica (a medida que crece el número de usuarios del sistema) más de un peer actúe como punto de entrada, reduciendo así, el coste en tiempo de entrada de nuevos peers.

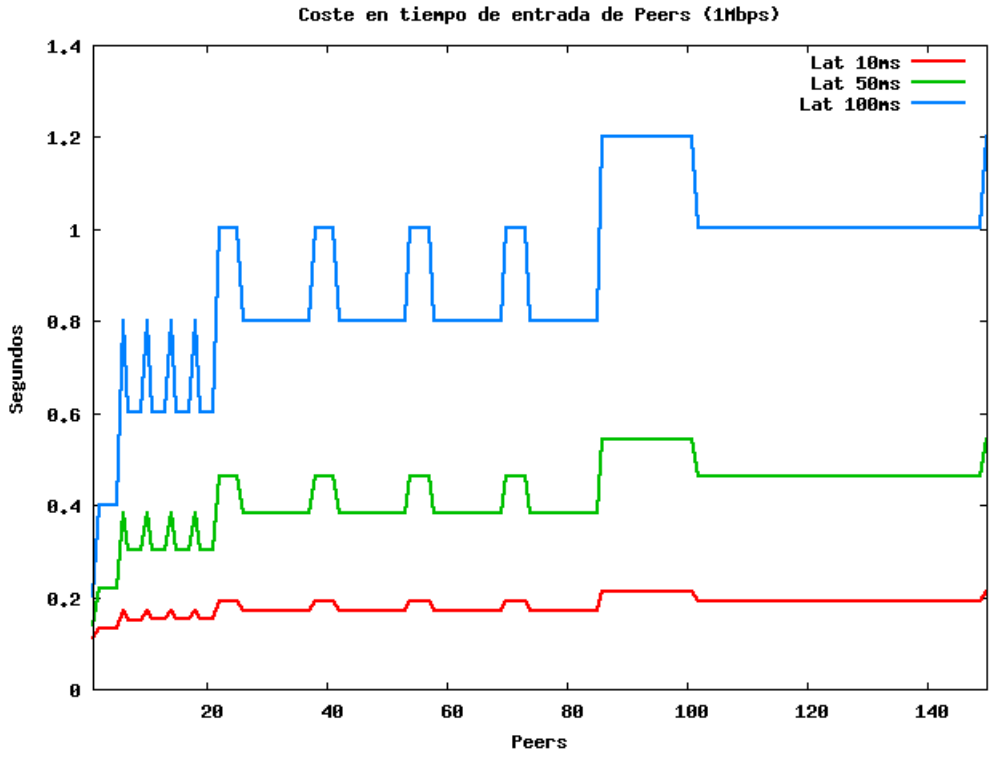

Figura 5.2: Tiempo que tardan los peers en entrar al sistema.

En la gráfica 5.2 se puede apreciar perfectamente el incremento de tiempo que supone convertir un worker a mánager (los diferentes picos) así como el coste en tiempo que supone incrementar un nivel, ya que si los peers continúan entrando por el mismo punto tendrán que negociar con más mánagers conforme más niveles haya. Por último también se observa que el comportamiento que tienen los peers con las diferentes latencias es practicamente igual, salvo por la correspondiente penalización de tiempo que incorpora cada una de ellas

5.2.3. SpeedUp

El *SpeedUp* se obtiene de la división del tiempo de ejecución en serie por el tiempo de ejecución en paralelo. Este último se calcula a partir de la suma de los tiempos de lanzamiento, planificación, ejecución y retorno de los resultados. Por otro lado, el tiempo en serie se obtiene a partir del tiempo de ejecución del nodo con CPU más potente, entre todos los peers utilizados para la ejecución en paralelo.

En la 5.3 se puede apreciar que, conforme incrementa la latencia de la conexión, disminuye el *SpeedUp* generado por el sistema. Se observa que con una latencia de 10ms se obtiene un resultado de una ejecución (trabajo de 90.000.000 Mis dividido entre 81 tareas) en paralelo unas 22 veces mejor que el tiempo de ejecución en serie.

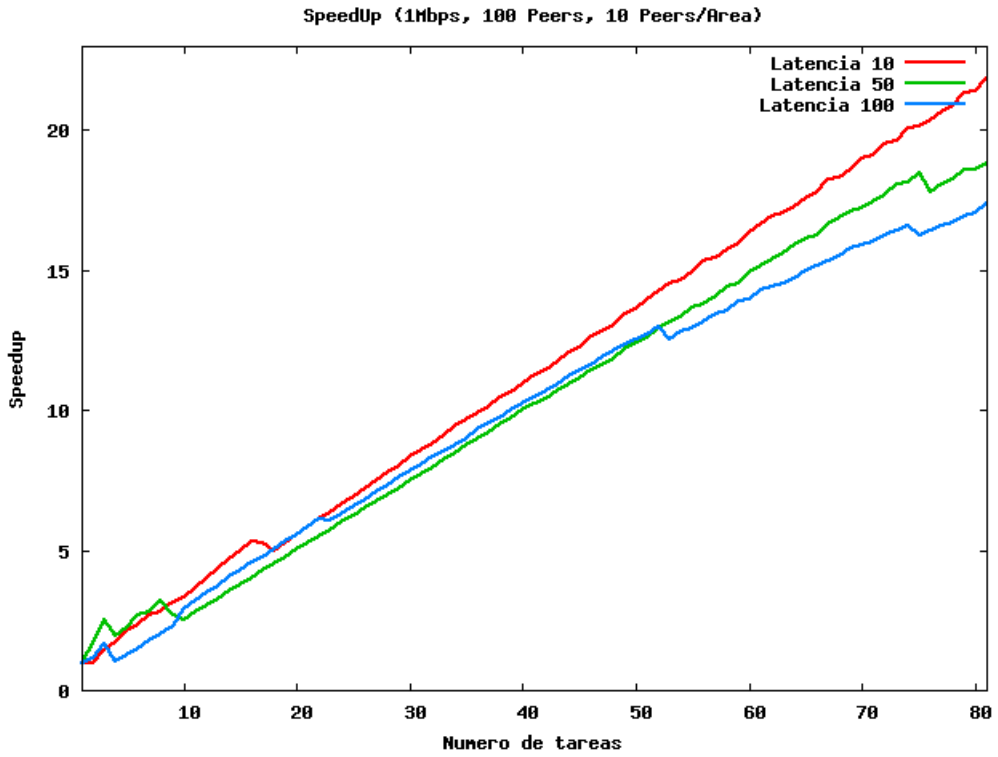

Figura 5.3: Speed Up de un trabajo de 90.000.000 Mis con 100 Peers.

5.2.4. Planificación de tareas

Una primera consideración a tener en cuenta es que el tiempo de planificación no es solo el tiempo que tarda el sistema en decidir que nodos y que ramas serán la elegidas para realizar las ejecuciones de cada tarea, sino que además incluye el tiempo de lanzamiento de todas ellas.

En la gráfica 5.4 podemos observar la gran penalización que incorpora la latencia en el tiempo total de planificación de un trabajo de 90.000.000 dividido hasta un máximo de 81 tareas. Esto se debe al incremento de mensajes que supone el hecho de incorporar más nodos en el proceso de ejecución.

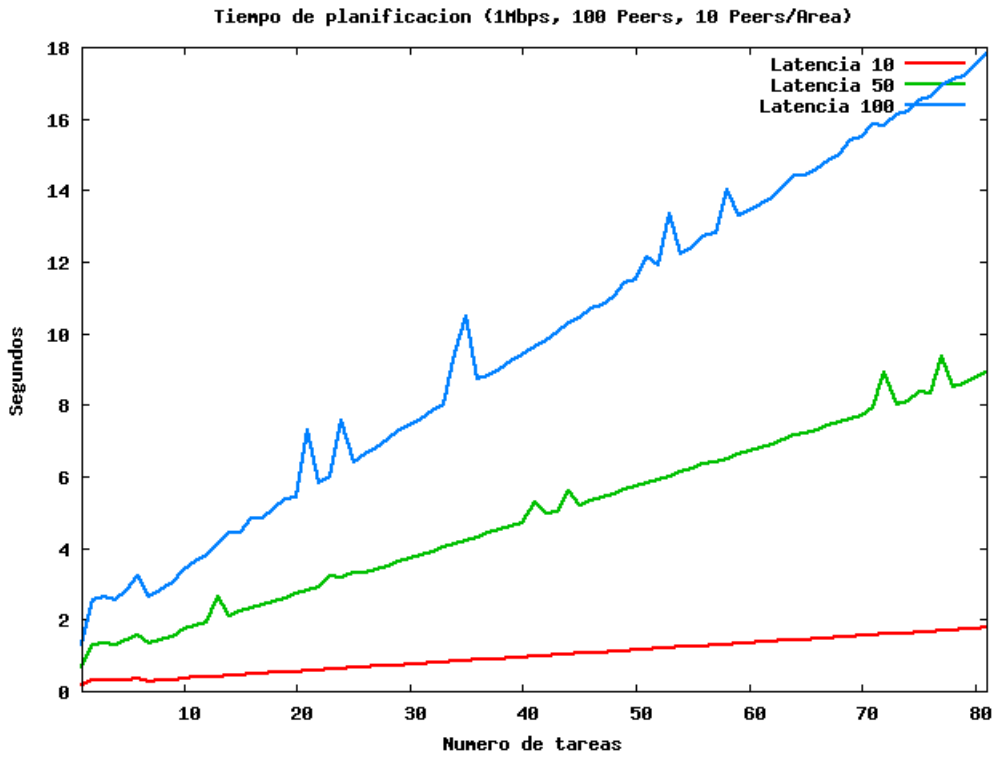

Figura 5.4: Planificación de un trabajo de 90.000.000 Mis con 100 Peers.

Capítulo 6

Trabajo futuro

Dejar abiertas muchas lineas del trabajo estaba implícito en la concepción del propio trabajo, por ello, creemos que el trabajo realizable es extremadamente extensible. Evidentemente también se pueden mejorar los contenidos presentados.

Por tanto, a continuación se realizará una breve enumeración de los contenidos en los que se puede hacer incapié con la finalidad de mejorar el sistema presentado.

Localidad de los peers
En éste primer prototipo de la aplicación no se han tenido en cuenta la localidad de los peers del sistema. Por esta razón, es una tarea interesante que realizar en posteriores prototipos.

Tratar la localidad de los peers, puede beneficiar a sistemas de monitorización de la aplicación, ya que sería posible mostrar información de los grupos o comunidades internacionales que más usan la aplicación, los que lanzan más tareas, los que más recursos ceden, los que poseen máquinas computacionalmente más potentes, etc ... En definitiva, dar un plus de información a la aplicación.

La localidad del sistema, también sería importante tratarla para facilitar la adhesión de los peers en áreas o grupos localmente cercanas al mismo, para minimizar el impacto negativo de comunicaciones de larga distancia, para en definitiva, optimizar el rendimiento de la aplicación.

Nodos estériles o finales
En éste primer prototipo de la aplicación no se ha implementado mecanismos de detección o reposicionamiento de los nodos finales, sino que son tratados como un worker permanente, sin aspiraciones a convertirse en mánager del algún área o grupo del sistema. Se delegan las tareas para tratar estos nodos peculiares en los siguientes prototipos que se realizarán.

Una línea de investigación podría dedicarse a inventar o encontrar un mecanismo por el cual los nodos finales no tuvieran limitada su "capacidad de descendencia", permitiendo así, la eliminación de los nodos estériles como tal, para dar paso a un sistema más amplio y tolerante, que en definitiva, ayudaría a obtener cada vez una capacidad computacional global más amplia.

Sistema de méritos de los peers
En éste primer prototipo de la aplicación no se ha implementado mecanismos de atribución y retribución de créditos o méritos a los peers del sistema como medida para decidir qué nodos

tienen privilegios de ejecución y cuales son solamente simples workers. Por tanto, implementar teorías de mercado entre los nodos del sistema, es una tarea muy interesante que incluso puede abarcar toda una tesis doctoral, sin embargo, esbozos simples se tendrían que implementar en próximas revisiones del prototipo.

Sistema de predicción de resultados

Un tema muy importante, hoy en día, es saber para cuándo se tendrá terminada una tarea, sea cual sea. En el mundo de la computación, someter tareas a una unidad de cómputo fija permite conocer cuando se obtendrán los resultados de forma exacta, ya que se cuenta con recursos fijos y el cálculo del tiempo se puede hacer de forma precisa. La complicación surge cuando la unidad de cómputo no es fija, sino variable, como puede ser una unidad de cómputo basada en peer-to-peer, los recursos no son estables, ya que fluyen libremente por la red de peers, hecho que dificulta enormemente, la tarea de predecir con exactitud cuándo se terminará de realizar una tarea determinada.

Es por esa razón que creemos muy importante desarrollar mecanismos que permitan al sistema calcular de una forma acertada el tiempo necesario para que el sistema sea capaz de entregar los resultados de una aplicación que se ha lanzado a la red de peers.

En definitiva, las opciones presentadas y muchos temas más pueden ser motivo de trabajo futuro en un aplicación como la que se presenta en éste trabajo.

Capítulo 7

Conclusiones

Todos los objetivos planteados inicialmente, se han cumplido exitosamente. A continuación, se realizará un breve repaso por cada uno de ellos.

La documentación del proyecto es más que aceptable. Todas las ideas del modelo así como sus algoritmos se encuentran extensamene especificados y comentados.

El diseño que se ha llevado a cabo, ha cumplido los objetivos iniciales de modularidad y de fácil modificación y ampliación.

La multiplataformidad de la aplicación ha sido casi inmediata desde el momento que se escogió el lenguaje *Java* como lenguaje principal de programación de la aplicación.

La aplicación permite el acceso a la red peer-to-peer a todo tipo de dispositivos, permitiendo de ésta forma una heterogeneidad total dentro de la red de cómputo distribuido. Por tanto, se puede afirmar que el objetivo de escalabilidad marcado a inicio del trabajo se ha logrado satisfactoriamente.

Los algoritmos que constituyen el funcionamiento del sistema, forman parte de módulos separados convenientemente dentro del aplicativo, hecho que permite la fácil modificación de ellos, así como la inclusión de nuevas metodologías o algoritmos.

La instalación del sistema es trivial, ya que un simple script permite la ejecución del aplicativo, que lanza una simple interfaz gráfica de usuario, hecho que permitirá su manejabilidad con suma facilidad.

Analizados los objetivos, es necesario analizar el propio aplicativo.

La primera característica y la más importante, era la de realizar un sistema completamente peer-to-peer. Se puede afirmar que la aplicación es totalmente "entre pares".

El segundo objetivo, era implementar una aplicación que se desmarcara de un ámbito local, permitiendo así, la creación de una red *peer-to-peer* de alcance mundial, de tal modo que la conectividad vía internet fuera un estándar defacto del aplicativo, Estamos orgullosos de comentar que éste reto complicado se ha solucionado y por tanto, tenemos entre las manos una aplicación de alcance mundial.

Remarcar, que el desarrollo de la arquitectura entre iguales usando el framework de *JXTA* ha sido ciertamente compleja, ya que la plataforma trabajo a muy bajo nivel en temas de comunicaciones: creación de grupos, establecimiento de pipes de comunicaciones, mantenimiento de la red, etc... Sin embargo, el handicap de la dificultad del proyecto se ha superado.

Por último, comentar que ambos autores del proyecto, están sumamente satisfechos del proyecto entregado, ya que gracias a el trabajo realizado, las opciones de realizar un doctorado están presentes. Por tanto, afirmar que en caso que tengamos opción a obtener una beca de doctorado, no dudaremos ni un minuto en seguir la investigación en el ámbito peer-to-peer, eje del proyecto.

Bibliografía

[1] JXTA 2.3.7 API `http://platform.jxta.org/nonav/java/api/index.html`

[2] JXTA Programmer's Guide `http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf`

[3] JXTA Book `http://www.brendonwilson.com/projects/jxta-book/`

[4] JNGI `http://jngi.jxta.org/`

[5] Wikipedia `http://es.wikipedia.org/`

[6] Memoria del *TFC CompP2P*, Iñigo Goiri Presa y Josep Rius Torrentó, 2006

[7] N. Drost, R.V. van Nieuwpoort, H. Bal, "Simple Locality-Aware Co-allocation in Peer-to-Peer Supercomputing", Proc. of the 6th IEEE Int. Symposium on Cluster Computing and Grid Workshops (CCGRIDW?06), 2006.

[8] A. Auvinen, M. Vapa, M. Weber, N. Kotilainen and J. Vuori, "Chedar: Peer-to-Peer Middelware", Proc. of the 20th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS'06), 2006.

[9] I. Foster and A. Iamnitchi, "On death, taxes, and the convergence of peer-to-peer and grid computing", Proc of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.

[10] M. Venkateswara Reddy, A. Vijay Srinivas, Tarun Gopinath, D. Janakiram, "Vishwa: A reconfigurable P2P middleware for Grid Computations", Proc. of the International Conference on Parallel Processing (ICPP'06) pp. 381-390, 2006.

[11] D. Talia, P. Trunfio, "Towards a synergy between P2P and Grids", IEEE Internet Comput. 7 (4), pp. 94-96, 2003.

[12] Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of ACM SIGCOMM, San Diego, CA, USA, 2001.

[13] A. Gupta, D. Agrawal, A. E. Abbadi, "Distributed Resource Discovery in Large Scale Computing," SAINTW 2005.

[14] F. Mao, H. Jin, D. Zou, B. Chen, L. Qi, "QoS Oriented Dynamic Replica Cost Model for P2P Computing", Proc. of the 25th IEEE Int. Conference on Distributed Computing Systems Workshops (ICDCSW?05), 2005.

[15] Zhiyong Xu Bhuyan, "Effective Load Balancing in P2P Systems", Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 81- 88, 2006.

[16] Y. Murata, T. Inaba, H. Takizawa, H. Kobayashi, "A distributed and cooperative load balancing mechanism for large-scale P2P systems", Proc. of the Int. Symposium on Applications and the Internet Workshops (SAINTW?06), 2006.

[17] J. Hu and R. Klesftad, "Decentralized Load Balancing on Unstructured Peer-2-Peer Computing Grids", Int. Symposium on Applications and the Internet Workshops (SAINTW?06), 2006.

[18] C. Anglano, "Interceptor: Middleware-level Application Segregation and Scheduling for P2P Systems", 20th International Parallel and Distributed Processing Symposium, 2006.

[19] N. Griffiths, K.-M. Chao, and M. Younas, "Fuzzy Trust for Peer-to-Peer systems", 26th International Conference on Distributed Computing Systems (ICDCS 2006), 2006.

[20] Wei Wu, Phu Dung Le: "An Efficient and Secure Code Sharing for Peer-to-Peer Communications", International Conference on Information Technology: New Generations, pp. 476-481, 2006.

[21] A. Mondal, M. Kitsuregawa, "Privacy, Security and Trust in P2P environments: A Perspective", 3rd Int. Workshop on P2P Data Management, Security and Trust, pp. 682-686, 2006.

[22] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-peer computing", Technical Report HPL-2002-57, HP Lab, 2002.

[23] Article sobre *P2P* `http://es.wikipedia.org/wiki/P2P`

Capítulo 8

Licencia del documento

Attribution-ShareAlike 2.5

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a) "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

b) "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

c) "Licensor" means the individual or entity that offers the Work under the terms of this License.

d) "Original Author" means the individual or entity who created the Work.

e) "Work" means the copyrightable work of authorship offered under the terms of this License.

f) "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

g) "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a) to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b) to create and reproduce Derivative Works;

c) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

d) to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

e) For the avoidance of doubt, where the work is a musical composition:

1) Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.
2) Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

f) Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a) You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(c), as requested.

b) You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with

the same License Elements as this License, or a Creative Commons iCommons license that contains the same License Elements as this License (e.g. Attribution-ShareAlike 2.5 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

c) If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE MATERIALS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a) This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b) Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a) Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b) Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c) If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d) No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e) This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at `http://creativecommons.org/`