Universidad Nacional de La Plata

Facultad de Informática

Maestría en Redes de Datos

Trabajo de tesis para acceder al grado de

Magister en Redes de Datos

"Arquitectura compañero a compañero (P2P)
para un servicio de búsqueda distribuida
en el espacio web"

Director:

Prof. Mg. Jorge R. Ardenghi

Tesista:

Gabriel Hernán Tolosa

- 2003 -

Agradecimientos

Encontrarme en esta instancia es un paso muy importante, tanto en mi formación académica como en el aspecto personal. Durante todo este tiempo colaboraron conmigo muchas personas las cuales merecen mi reconocimiento y con quienes seguramente estoy siendo injusto.

Agradezco especialmente a mi colega, compañero y amigo Fernando Bordignon por su constante apoyo, su crítica siempre constructiva y por ayudarme día a día en los quehaceres de la vida académica dentro de la Universidad Nacional de Luján.

A mi director, el profesor Jorge Ardenghi, por su calidad y calidez como docente y director; y por su guía en la ardua tarea de desarrollar una tesis de maestría, con justas e invalorables sugerencias y críticas.

Finalmente, mi más profundo agradecimiento a la Universidad Nacional de Luján, especialmente al Centro Regional Chivilcoy. Porque me permitió desarrollar toda mi carrera brindándome – de manera completamente gratuita y desinteresada – formación como profesional y como persona. Porque he vivido más de diez años bajo su techo, primero como alumno y ahora me permite continuar la vida académica como docente. Porque no es ni más ni menos que mi segunda casa. A la Universidad Nacional de Luján y a mi querido Centro Regional Chivilcoy – nuevamente – muchas gracias.

Indice

Resumen

“Arquitectura compañero a compañero (P2P) para un servicio de búsqueda distribuida en el espacio web”

Gabriel Hernán Tolosa

Maestría en Redes de Datos
Universidad Nacional de La Plata
Facultad de Informática

Director Prof. Mg. Jorge R. Ardenghi

El presente trabajo define una arquitectura para soportar un servicio de búsquedas distribuidas en el espacio web, como alternativa a los motores de búsqueda tradicionales. Dicha arquitectura se basa en el modelo de comunicaciones denominado compañero a compañero (P2P), en el cual todos los nodos participantes de una red son capaces de generar y contestar consultas de otros nodos.

En los últimos años han surgido aplicaciones en Internet basados en el modelo P2P que permiten que computadoras de usuario final se conecten directamente para formar comunidades, cuya finalidad sea el compartir recursos y servicios computacionales. Bajo este esquema, se toma ventaja de recursos existentes en los extremos de la red, tales como tiempo de CPU y espacio de almacenamiento.

Por otro lado, se han planteado algunos inconvenientes directamente relacionados con las búsquedas en el espacio web utilizando los motores de búsqueda tradicionales, basados – en general – en arquitecturas centralizadas. El tamaño actual del espacio web, su constante crecimiento y la frecuencia de actualización de contenidos generan una serie de importantes problemas, entre los cuales se encuentran el mantenimiento de los índices, su limitada cobertura y el aumento de la proporción de respuestas irrelevantes.

Se desarrolla un modelo de consulta distribuida entre sitios de una red que cooperan, tratando de potenciar sus capacidades. La arquitectura propuesta, denominada IndiSE, tratar de minimizar estos problemas a partir de cambiar la naturaleza centralizada de los motores de búsqueda

tradicionales y plantear un modelo distribuido. En este sentido, quien posee la información es capaz de contestar a una consulta determinada, es decir, los proveedores de información se convierten – además – en proveedores del servicio de búsqueda, operando sobre su propio dominio.

La contribución principal del trabajo se encuentra básicamente en los desarrollos de un middleware – denominado gnutWare – que permite a las aplicaciones acceder a una red de propagación de mensajes inespecíficos y el sistema de búsquedas distribuidas IndiSE. En el primer caso, el prototipo de middleware aporta:

- Un modelo operativo de utilización de una red basada en el protocolo Gnutella como infraestructura de intercambio de mensajes inespecíficos.

- Un modelo de comunicación entre aplicaciones de usuario final, a través de una red P2P de mensajes.

- Una arquitectura sencilla y robusta para implementar servicios distribuidos sobre redes compañero a compañero.

- Una interfaz normalizada para aplicaciones que requieren acceder a la red de transporte de mensajes, para operar en un ambiente distribuido.

En el segundo caso, la arquitectura IndiSE aporta:

- Un diseño modular de un sistema que permite soportar búsquedas distribuidas en el espacio web.

- Una interface sencilla para usuarios finales que les permite acceder a búsquedas en paralelo sobre un repositorio de información distribuido.

- Una arquitectura simple que utiliza los motores de búsqueda internos de cada proveedor de información, lo que brinda a los mismos mayor control sobre las tareas de indexación y consulta.

Estructura del Trabajo

El presente trabajo se encuentra dividido en cuatro capítulos donde se presenta el tema a tratar y la arquitectura propuesta. Por último, se enuncian las conclusiones y se plantean algunas posibles líneas de investigación y desarrollo para continuar el proyecto.

En el Capítulo I, se presenta el modelo de comunicaciones compañero a compañero, a partir del surgimiento de un conjunto de aplicaciones en Internet. Se plantean las características deseables de los nodos P2P y se definen posibles usos y aplicaciones del modelo. Se describen dos protocolos paradigmáticos en el área: Gnutella y Freenet. Finalmente, se plantean los inconvenientes y desafíos a resolver para la evolución y adopción masiva del mismo.

El Capítulo II brinda una caracterización de la World Wide Web (web) como repositorio de información y de las herramientas disponibles para facilitar búsquedas sobre este ambiente. En este sentido, se hace una descripción de las arquitecturas de motores de búsqueda, tanto centralizadas como distribuidas. Para el último caso, se presentan enfoques distribuidos basados en el modelo cliente/servidor y en el modelo P2P para resolver el problema de las búsquedas en la web.

El Capítulo III presenta una contribución original consistente en un prototipo de middleware (denominado gnutWare) que brinda acceso a las aplicaciones a una red de transporte de mensajes. Dicha red se encuentra basada en el modelo P2P y permite que las aplicaciones de usuario final utilicen servicios de comunicaciones operando en un ambiente completamente distribuido. La idea y desarrollo de este prototipo es un desarrollo conjunto con el Lic. Fernando R.A. Bordignon, en el marco de un proyecto de investigación conjunto en la Universidad Nacional de Luján.

El capítulo final (Capítulo IV) muestra el desarrollo de una arquitectura – denominada IndiSE (por INDIvidual Search Engines) – para soportar búsquedas distribuidas en el espacio web. Se trata también de un desarrollo propio, donde existe una aplicación de búsqueda distribuida que accede a una red P2P utilizando el middleware gnutWare y permite a los usuarios realizar búsquedas sobre una comunidad de participantes de dicha red.

Finalmente, se enuncian las conclusiones del trabajo general, planteando las ventajas e inconvenientes de la arquitectura propuesta y dejando posibles alternativas para la continuación de su desarrollo.

Capítulo 1

El Concepto de Compañero a Compañero

En los últimos años han surgido aplicaciones en Internet que aprovechan las capacidades de los equipos de usuario final. Cada una de estas aplicaciones implementan un servicio que opera de manera distribuida en computadoras de usuario final. Por ejemplo ICQ [ICQ03] y AIM [AIM03] brindan servicios de mensajería, Gnutella [BOR01], Freenet [CLA00] y Napster [NAP00] servicio de distribución de archivos y SETI@Home [KOR01] procesamiento distribuido.

Algunas de estas aplicaciones operan bajo el modelo de comunicaciones compañero a compañero (P2P). Esta modalidad de operación no es nueva sino que había sido relegada por el modelo cliente/servidor. Básicamente, se lo puede ver como la interacción entre dispositivos considerados pares (o iguales) utilizando un sistema de comunicación.

Para ejemplificar esta situación, se puede estudiar el origen de Internet. En los comienzos, las computadoras se comunicaban directamente, tanto para utilizar una acceso remoto como para intercambiar archivos. Si bien estas aplicaciones operan bajo el esquema cliente/servidor, el uso de los equipos donde residían tales clientes y servidores era simétrico, debido a que casi no existían computadoras dedicadas a tareas de servidor exclusivamente. El crecimiento de la red originó que se utilice el modelo cliente/servidor a los efectos de separar los roles entre los equipos y permitir manejar la escalabilidad. Complementariamente, se pueden ver varios ejemplos de sistemas que operan sobre Internet siguiendo un esquema P2P. Los más típicos son el sistema de noticias USENET y el servicio DNS [MIN01].

Los sistemas compañero a compañero permiten que computadoras de usuario final se conecten directamente para formar comunidades, cuya finalidad sea el compartir recursos y servicios computacionales [FIP00] [MIN01]. En este modelo, se toma ventaja de recursos existentes en los extremos de la red, tales como tiempo de CPU y espacio de almacenamiento. Las primeras aplicaciones emergentes se orientaban a compartir archivos y al procesamiento distribuido, a partir de reconocer y aprovechar las capacidades (muchas veces ociosas) del equipamiento de los usuarios finales.

1.1 – Hacia una definición de P2P

La primera característica que define a un nodo en un sistema P2P, es que cumple tanto el rol de cliente como de servidor (en la terminología se lo denomina *servent*, palabra que deriva de la conjunción de los términos *serv*er-*client*). Este modelo basado en interacciones entre pares, puede reemplazar al paradigma cliente/servidor, donde cada nodo es capaz de generar y contestar consultas de otros nodos. Desde el punto de vista de la comunicación, las interacciones son simétricas.

Este concepto se basa en la concepción de una red donde todos los nodos tienen capacidades y responsabilidades equivalentes y – a diferencia del modelo cliente/servidor – no hay nodos dedicados exclusivamente a prestar servicios a otros. Este escenario es común en ambientes de redes locales donde un conjunto de usuarios forman un grupo de trabajo y comparten los recursos de sus equipos, como impresoras locales ó dispositivos de almacenamiento.

En [YAN01] se define un sistema P2P como *"nodos de computación distribuida con iguales roles y capacidades de intercambio de información y servicios, directamente entre ellos*". La gran ventaja que aporta P2P es que los recursos de muchos usuarios y computadoras pueden unirse para producir un gran cúmulo de información y un significativo poder de cómputo.

Clay Shirky, del The Accelerator Group definió que *"Compañero a compañero es una clase de aplicaciones que toma ventajas de los recursos – almacenamiento, ciclos de CPU, contenido y presencia humana – disponible en los entremos de Internet*" [SHI01], y planteó un test para definir si una aplicación es P2P basado en dos preguntas:

1) ¿Permite (la aplicación) conectividad variable y direcciones de red temporarias?

2) ¿Brinda a los nodos de los extremos de la red autonomía significativa?

Según el mencionado autor, si la respuesta a ambas preguntas es afirmativa, entonces la aplicación es considerada P2P.

Se considera que tanto la definición como el test son solo aplicables a casos particulares (como la nueva generación de aplicaciones que volvieron a impulsar el concepto de P2P) y que pueden existir aplicaciones basadas en el modelo compañero a compañero que no cumplan alguno de sus requisitos. Bajo esta definición, las aplicaciones que dieron origen al

modelo P2P no calificarían como tal.

Finalmente, se plantea una definición del concepto de compañero a compañero a partir de cuestiones fundamentales, a saber:

- Los nodos participantes se comportan tanto como clientes y servidores de un determinado servicio.

- Los nodos son autónomos.

- Dichos nodos se comunican entre sí por intercambio directo.

Entonces, se define como sistema compañero a compañero a aquel donde los nodos intercambian coordinadamente recursos – ciclos de CPU, espacio de almacenamiento, archivos – en base a la comunicación directa entre ellos (los que requieren y los que poseen). Operan bajo una topología de red donde todos los participantes tienen igual acceso a todos sus pares y – eventualmente – pueden existir nodos con alguna capacidad especial.

1.2 – Características de los sistemas P2P

Las aplicaciones P2P tienen aspectos técnicos interesantes, tales como control descentralizado, auto organización, adaptación y escalabilidad. En [ROW01] se presenta una definición posible:

"Un sistema P2P puede clasificarse como un sistema distribuido en el cual todos los nodos tiene idénticas capacidades y responsabilidades, y toda la comunicación es simétrica."

En esta definición, aparecen algunas características importantes de los sistemas compañero a compañero. Estas pueden incluirse dentro de un conjunto de características básicas y que muestran aspectos referidos a su modo de operación, a saber:

- Los nodos cumplen roles tanto de cliente (cuando solicita un recurso) como de servidor (cuando provee un recurso), es decir, las interacciones son simétricas.

- No existe un nodo que cumpla un rol de manera centralizada en todo el sistema, por ejemplo, no existe ni un coordinador central ni una base de datos central.

- Los nodos participantes se comunican entre sí por intercambio directo a los efectos

de compartir recursos.

- Los nodos participantes son autónomos. Cada uno regula su grado de participación en la red, a través de definir que recursos ofrece y en que cantidad.

- Ninguno de los nodos tiene una vista global del sistema.

- El comportamiento global emerge de las interacciones individuales directas.

- Todos los datos y servicios deben ser accesibles por cualquier nodo.

- Los nodos en una red P2P pueden ingresar y salir constantemente de forma arbitraria. Algunos son nodos de usuario final, que no se encuentran permanentemente conectados a la red, es decir, que su conectividad es variable.

- La ubicación de los recursos y disponibilidad de servicios es dinámica, ya que depende del estado del sistema en un momento del tiempo.

- La red es un ambiente dinámico y heterogéneo, dado que la conforman nodos con conectividad variable y de diversas plataformas de hardware y software. Todo nodo en Internet tiene la posibilidad de que sobre éste corra un nodo P2P.

Según Intel, uno de los más grandes beneficios de P2P es el concepto de comunidad, dado que hace posible que los usuarios se organicen ellos mismos en grupos “ad hoc”, y puedan eficientemente y de forma segura llenar requerimientos, compartir recursos, colaborar y comunicarse [BAR00].

1.3 – Modelos de Operación

Las aplicaciones emergentes que utilizan nodos P2P prestan servicios bajo diferentes modelos de operación y cooperación básicos: centralizado, descentralizado y jerárquico. En los tres modelos, el concepto de P2P se aplica en mayor ó menor medida, de acuerdo a las posibilidades de operación y a los requerimientos del servicio prestado.

Si un sistema P2P está íntegramente conformado por nodos que tienen iguales roles y ninguno cumple tareas administrativas o especiales, se lo considera P2P puro o completamente descentralizado [YAN01]. En cambio si algún nodo posee alguna característica diferente que le hace cumplir con tareas centralizadas, entonces al sistema se lo considera

híbrido.

1.3.1 – Modelo Centralizado

En este modelo existe un nodo que cumple con alguna tarea administrativa especial, ó existe una autoridad central que administra el servicio. Generalmente, este nodo funciona como mediador entre el proveedor y el cliente del servicio, para permitir el establecimiento de un vínculo. Una vez establecido dicho vínculo, los nodos participantes intercambian la información de manera directa (Gráfico 1.1).

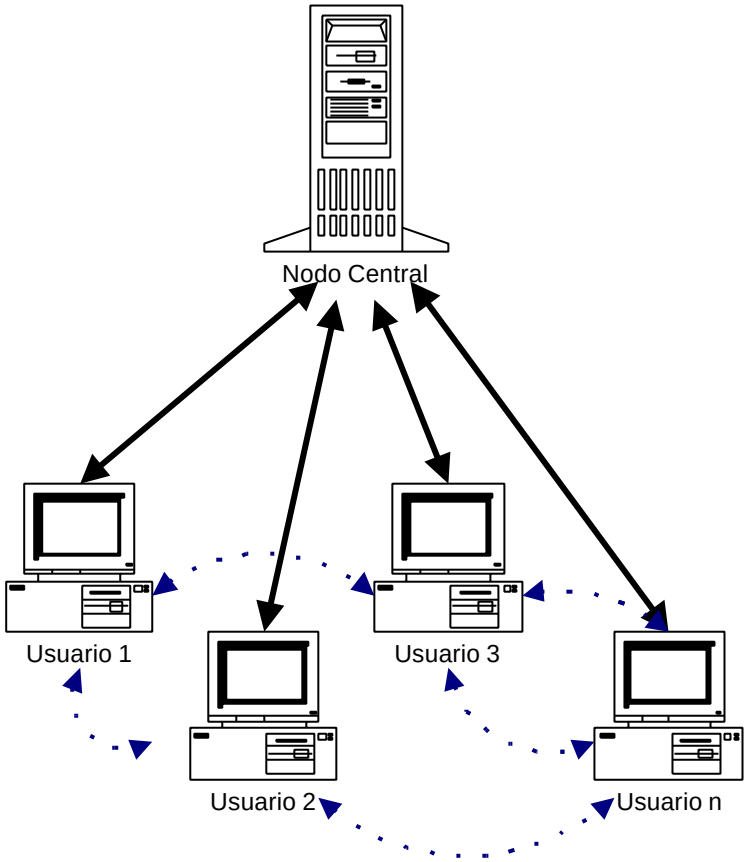


Gráfico 1.1 – Modelo de Operación Centralizado

El ejemplo típico de esta arquitectura es el servicio brindado por Napster (Gráfico 1.2) para compartir archivos de música. Los usuarios del servicio se registran y – cada vez que se conectan – envían una lista con todos los archivos que tienen para intercambiar, los cuales son registrados por dicho servidor central. Cuando un usuario realizar una búsqueda por un determinado archivo, ésta es enviada al servidor, el cual consulta su base de datos de nombres de archivos y retorna como resultado un conjunto de usuarios que poseen algún archivo que satisfaga la búsqueda. De la respuesta, el usuario puede solicitar la descarga del archivo a su

equipo local. Esta descarga se realiza directamente del usuario que posee en archivo, de manera directa (sin intervención del servidor central).

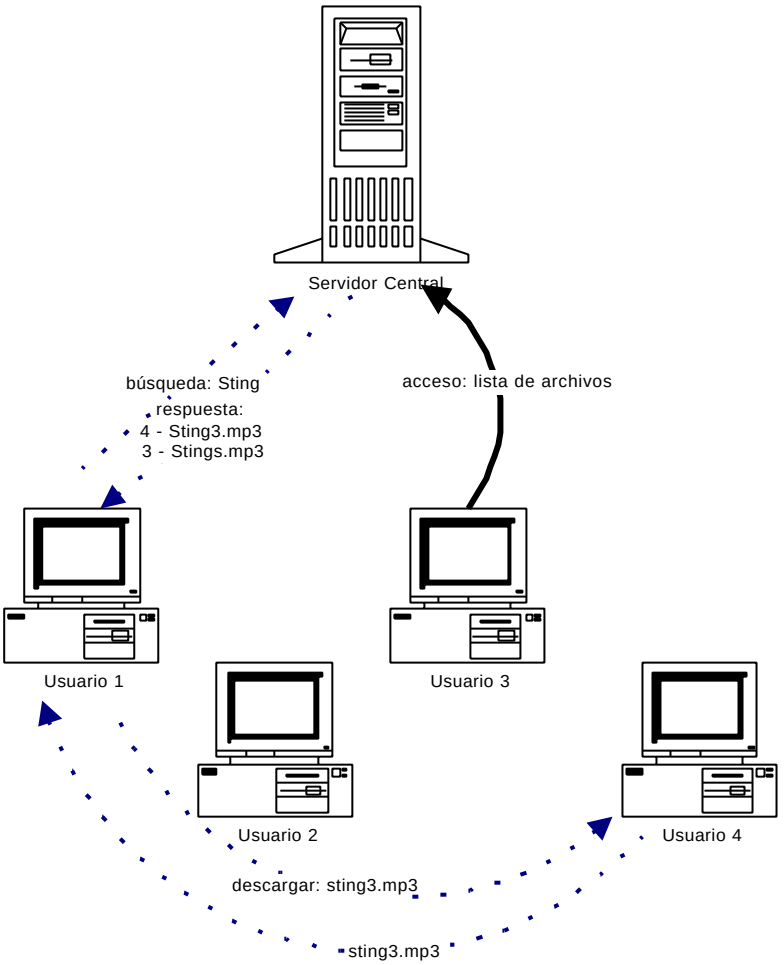


Gráfico 1.2 – Modo de operación de Napster

En este modelo centralizado el servidor actúa como autoridad central, registro de usuarios e intermediario en la prestación del servicio pero – además – es un único punto de falla y un posible cuello de botella del sistema. Desde el punto de vista del concepto de P2P, el intercambio directo de archivos entre los usuarios es el rasgo más puro dentro de las características de este sistema.

1.3.2 – Modelo Descentralizado

El modelo descentralizado, completamente distribuido ó puro se basa en que todos los nodos participantes de la red poseen las mismas capacidades y se comportan de la misma

manera. Los nodos se conectan – por uno ó más vínculos – entre “vecinos” de la red y son usuarios y proveedores (clientes y servidores) de un determinado servicio, es decir, su funcionalidad es simétrica. En este modelo, no existen coordinadores ó administradores centrales, donde la tarea de mantener el funcionamiento del sistema se realiza de manera completamente distribuida. De aquí, los nodos realizan intercambio directo de información, ya sea referente al servicio que prestan ó a funciones de control de la red (Gráfico 1.3).

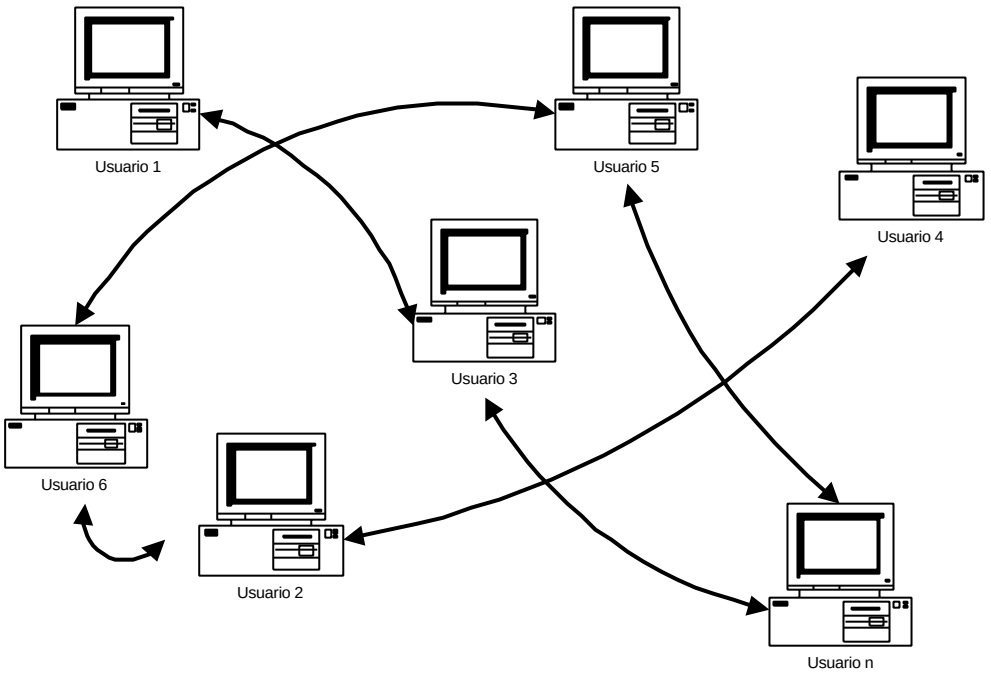


Gráfico 1.3 – Modelo de Operación Descentralizado

El ejemplo más notorio de este modelo de operación es la red Gnutella. Ésta provee un servicio que permite intercambiar archivos entre los usuarios participantes. Cuando algún usuario busca un determinado archivo, envía la búsqueda a sus vecinos, los cuales luego lo reenviarán al resto de la red. Las respuestas solo regresarán de aquellos que posean un recurso que satisfaga la consulta. Luego de recibidas las respuestas, se solicitará la descarga del archivo directamente del nodo que lo tiene disponible (Gráfico 1.4).

La red Gnutella opera de manera completamente descentralizada. Los nodos participantes se conectan entre sí formando una estructura sin jerarquías y sin topología definida. Al enviar una consulta, los nodos actúan de cooperativa funcionando como agentes de reenvío de dicha consulta al resto de los nodos (por sus conexiones) y al generarse una respuesta, colaboran para encaminar la misma hasta el destinatario. Finalmente, la descarga del recurso se realiza directamente del nodo proveedor, sin participar el resto de los mismos. Se puede decir que las descargas se realizan “por fuera” de la red.

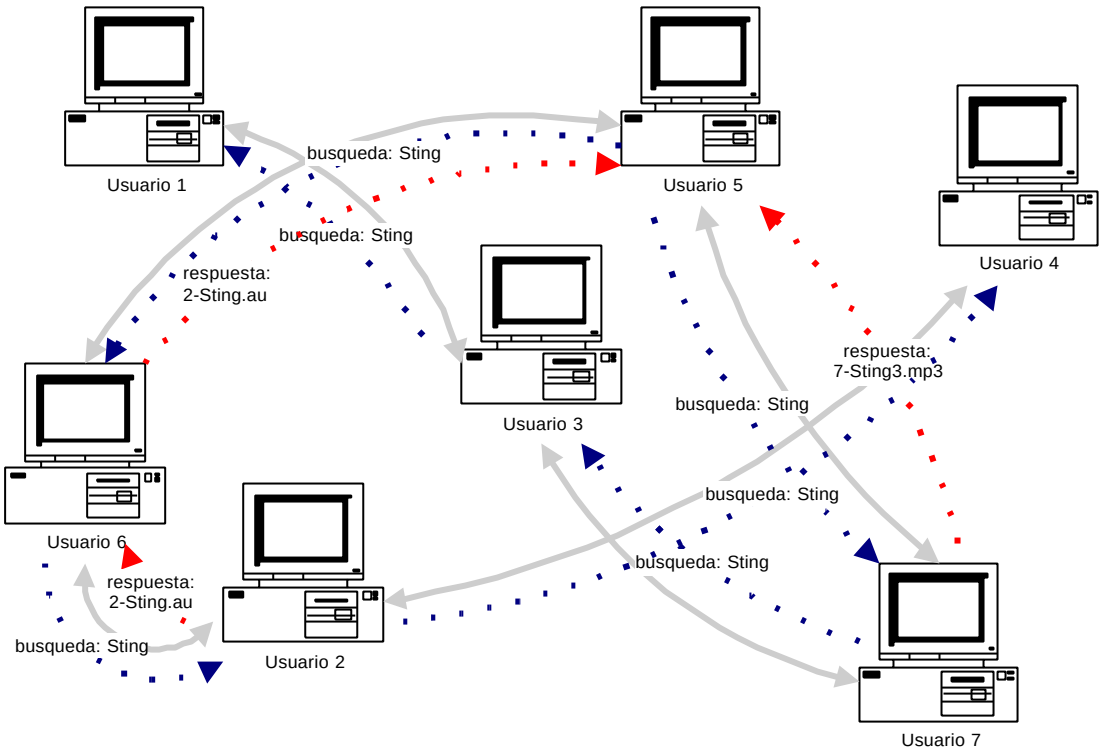


Gráfico 1.4 – Modo de operación de la red Gnutella

1.3.3 – Modelo Jerárquico

Existe un modelo híbrido que permite aprovechar las características más relevantes de los dos modelos anteriores, que permite definir jerarquías entre los nodos participantes de una red P2P. Este modelo define un conjunto de nodos (denominados “super-peers”) que forman una “columna vertebral“ (backbone) ó un hub a los cuales se conectan el resto de los nodos de usuario. Generalmente, éstos facilitan la conectividad y las interacciones ya que poseen alta disponibilidad de conexión a Internet y gran cantidad de recursos, como CPU y ancho de banda, lo que le permite manejar muchos usuarios conectados (Gráfico 1.5). Los “super-peers” pueden verse como nodos con capacidades especiales. Su aporte principal está dado por la necesidad de brindar a la red mayor capacidad en cuanto a performance, estabilidad y escalabilidad (comparativamente a los modelos anteriores).

Desde el punto de vista de las interacciones, los usuarios se conectan con algún nodo de backbone y a éste le envían los requerimientos y reciben las respuestas. Éstos nodos pueden implementar estrategias para reducir el tráfico en la red y optimizar la prestación del servicio, por ejemplo, manteniendo caches locales de usuarios conectados, nodos vecinos, consultas realizadas y respuestas recibidas. Un caso ejemplo de este modelo es el introducido por la empresa Clip2 para la red Gnutella, conocido como Reflector. Los Reflectors mantienen

índices con los nombres de los archivos disponibles en los nodos que se encuentran directamente conectados a cada uno de éstos. Cuando se realiza una consulta, ésta no es retransmitida, sino que se resuelve consultando los índices locales.

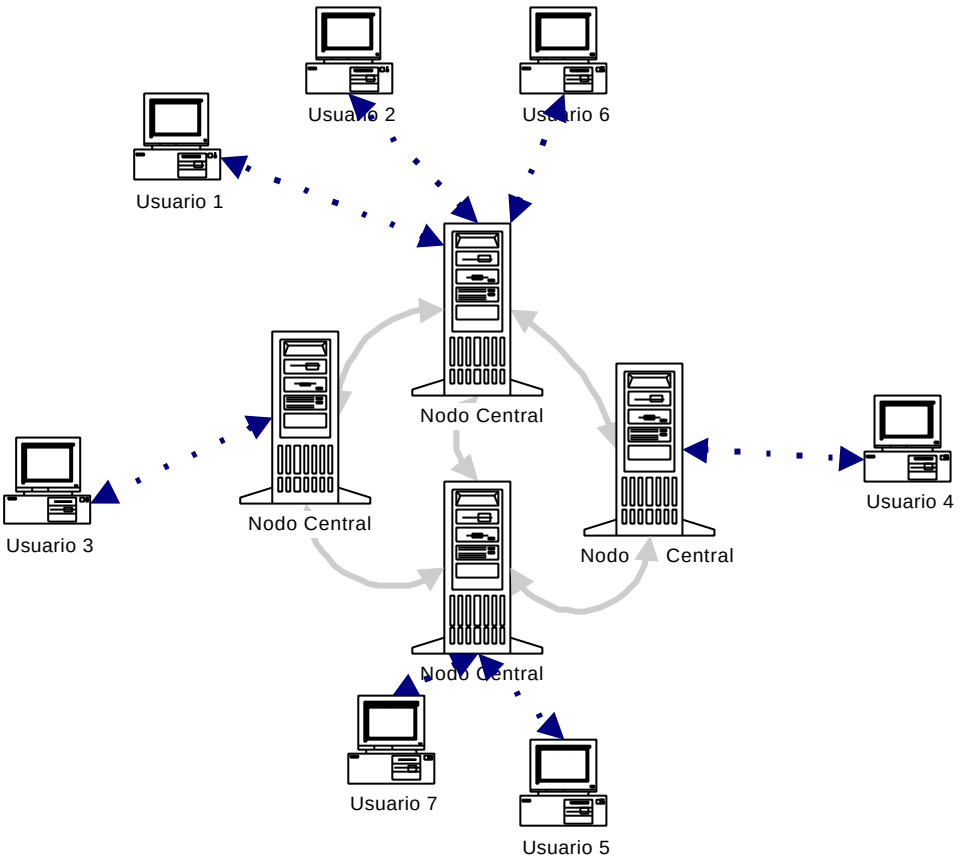


Gráfico 1.5 – Modelo de Operación Jerárquico

1.3.4 – Un caso particular: SETI@Home

Un caso particular de modelo jerárquico es el utilizado por el proyecto SETI@Home (Gráfico 1.6). La idea detrás de este proyecto es disponer de un gran número de usuarios que “presten” ciclos de su CPU cuando están ociosos (más precisamente, cuando su sistema operativo ejecuta un protector de pantallas). De esta forma, cuando cada computadora cooperante detecta que tiene tiempo de CPU libre, dedica tal recurso ocioso a procesar datos a los efectos de buscar señales indicadoras de inteligencia extraterrestre. En este caso, cada uno de los usuarios recibe trozos de datos a procesar por parte de un nodo coordinador central. Al finalizar el procesamiento, devuelven los resultados y solicitan una nueva unidad de datos.

Un trabajo similar se describe en [FRA02], donde se aprovechan ciclos de CPU de

usuarios del espacio web que se encuentran navegando por determinados sitios. El sistema funciona a través de la utilización de pequeños applets Java embebidos en páginas web que realizan tareas de cómputo por breves períodos de tiempo. Básicamente, se pretende utilizar en paralelo, el poder de miles de computadoras de usuario final mientras éstos navegan por un sitio.

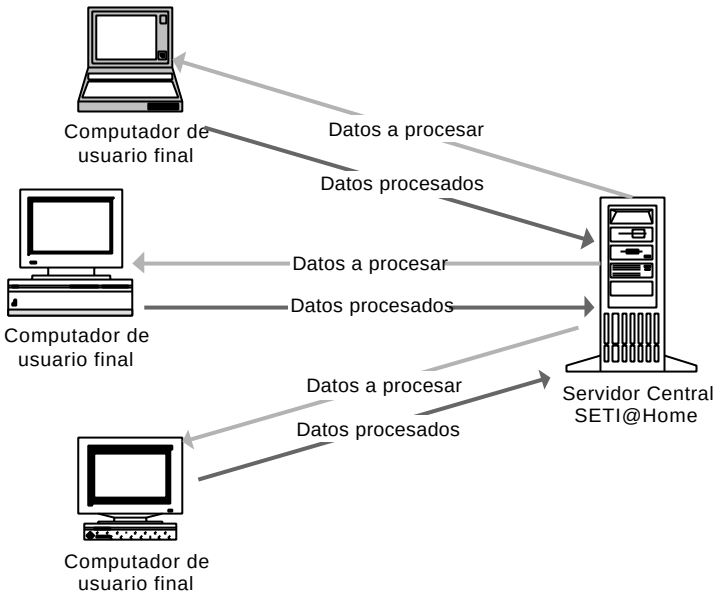


Gráfico 1.6 – Modelo de Interacción en SETI@Home

En ambos casos, el esquema de trabajo es claramente maestro – esclavo, ya que existe un nodo administrador de tareas y un conjunto de nodos de usuarios que las ejecutan. Generalmente, se denomina a este tipo de proyectos como P2P porque utilizan los recursos de computadoras de borde, las cuales poseen conectividad variable y direcciones temporales de red. Las cuestiones básicas y fundamentales del modelo son dos:

- no existe interacción entre nodos de usuarios finales (no hay intercambio directo)

- no se pueden reconocer en el modelo nodos que actúen de forma simétrica (existe un nodo “administrador” ó “gerente” y un conjunto de nodos que colaboran)

Dadas estas características del modo de operación de SETI@Home, no se lo puede considerar como un sistema que opere bajo el modelo compañero a compañero.

1.4 – Potencial de las Aplicaciones P2P

Los sistemas P2P aparecieron brindando un servicio determinado, como por ejemplo, para permitir el intercambio de archivos o procesar unidades de datos de manera distribuida. Algunos rápidamente ganaron un espacio dentro de las aplicaciones utilizadas masivamente en Internet y dieron un punto de partida para nuevos proyectos. Las nuevas aplicaciones P2P normalmente operan en computadoras de usuario final (en los extremos de Internet), asumiendo roles que tradicionalmente (y según el modelo cliente/servidor) tienen los servidores (que se encuentran generalmente con alta disponibilidad conectados a la red). Como ya se mencionó, el servicio Napster permite el intercambio de archivos de música en formato mp3 directamente entre equipos de usuario, en vez de tener que descargarlos de un servidor central. Las primeras aplicaciones emergentes se dividieron básicamente en tres grupos, según sus funciones:

- Compartir archivos (Napster, Gnutella, Freenet)
- Procesamiento distribuido (SETI@Home, Distributed.net)
- Mensajería instantánea (ICQ, AIM)

A partir de estas aplicaciones, toma más fuerza el concepto de P2P, debido a que las arquitecturas prometen gran flexibilidad y escalabilidad, de manera de poder “juntar” un gran número de computadoras en una organización ó – inclusive – en el mundo detrás de un objetivo. Como su ámbito de operación es Internet y cada día más y más gente se conecta a la red, su potencial crece, ya que los nuevos usuarios no solamente consumen los recursos de información, comunicación y entretenimiento disponibles, sino que – además – son potenciales proveedores de recursos computacionales.

El potencial de P2P está en que permite aprovechar recursos distribuidos para resolver problemas y brindar a sus usuarios otras capacidades. Estas capacidades no son generalmente nuevas, sino que están construidas a partir de pensar nuevamente sus objetivos en base a una nueva forma de aprovechamiento de recursos, de interacción entre sus participantes y de asignación de roles. Básicamente, la innovación se presenta en tres diferentes líneas:

- *Acceso directo a la información*: las aplicaciones P2P para compartir y transferir archivos, reorientan a los usuarios a decidir qué contenido está disponible y en qué momento.

- *Acceso directo a poder de procesamiento*: algunas tareas que típicamente requieren de alta capacidad de procesamiento (supercomputadoras) pueden ser divididas a los efectos de distribuir una porción del total entre diferentes nodos. El alto grado de paralelismo que se puede alcanzar (que depende de factores como cantidad y disponibilidad de nodos, tiempo de CPU "prestado" por los usuarios y también del problema mismo) puede equiparar el poder de un gran equipo sin necesidad de disponer de éste.

- *Acceso directo a los usuarios*: seguramente, uno de los servicios más utilizados en Internet es el correo electrónico, que permite el envió de mensajes a personas que no necesariamente estén en línea. Pero, como es posible que un usuario se encuentre en línea al momento que se lo requiere, aplicaciones como chat y mensajería instantánea son altamente útiles. En este último caso, los sistemas P2P de mensajería instantánea permiten vincular usuarios en línea directamente, permitiendo diálogos entre dos ó más usuarios.

Como ejemplo del potencial, se puede plantear que si la arquitectura centralizada de los motores de búsqueda es movida hacia sistemas P2P que permitan coordinar a través de Internet una búsqueda se producen cambios radicales: la búsqueda se realiza de manera distribuida, no se requiere de spiders ó robots que naveguen la web para armar una gran base de datos central, no se requiere de poderosos equipos capaces de atender a miles de usuarios. La red se convierte, entonces, en un motor de búsqueda masivamente paralelo capaz de responder consultas de sus usuarios.

1.5 – Aplicaciones P2P Paradigmáticas

1.5.1 – Gnutella

Gnutella surgió como un proyecto independiente de dos programadores quienes lo plantearon como una herramienta de software para compartir archivos. Sus dos características principales son su carácter descentralizado y su espíritu cooperativo. Bajo Gnutella se trabaja en un ambiente completamente distribuido donde un conjunto de usuarios pueden ofrecer una parte de su sistema de archivos (un conjunto de objetos) que desea compartir con los demás y – simultáneamente – puede descargar desde otros nodos los archivos que desea.

La red Gnutella se compone de numerosos nodos (denominados gnodos) distribuidos. Su topología no indica jerarquía alguna, dado que cada gnodo es igual a los demás en

funcionalidad. Debido a que uno de los atributos que caracterizan a un gnodo es su ancho de banda ofrecido para descarga de archivos, el modelo sufre ciertas variaciones. Aquellos gnodos de mayor ancho de banda son preferidos a otros, formando hubs ó anillos centrales de conexión a la red.

El protocolo es simple y se basa en 5 mensajes:

a) PING, permite anunciar la presencia en la red y solicitar información de otros gnodos

b) PONG, sirve para responder un mensaje PING.

c) QUERY, se genera a instancias de un usuario que efectúa una consulta.

d) QUERY_HIT, se utiliza para responder un mensaje QUERY, cuando un gnodo tiene archivos que satisfacen la consulta.

e) PUSH, este mensaje permite solicitara a un gnodo que establezca una conexión y envíe un archivo determinado. Se utiliza cuando el poseedor del recurso no acepta conexiones entrantes.

En el Apéndice I se brinda un detalle técnico de las estructuras de datos del protocolo y de su funcionamiento.

Cada gnodo de Gnutella sólo “sabe” acerca de los gnodos con los que se conecta directamente. Todos los otros gnodos son invisibles, a menos que ellos se reciba algún mensaje. Cuando un usuario realiza una búsqueda, se envía un mensaje a todos los gnodos con los que se encuentra directamente conectado. La red Gnutella opera bajo el modelo conocido como “propagación viral”. Un cliente envía un mensaje a un gnodo, y éste lo reenvía a todos los gnodos a los cuales esté conectado. De esta forma, con solo conocer la dirección de red de un gnodo ya se puede ingresar a la misma.

El ingreso a la red se realiza indicando la dirección IP y puerto TCP de cualquier gnodo conocido perteneciente a ésta. El gnodo que inicia la conexión anuncia su presencia mediante el envío de un mensaje de exploración, el cual es reenviado a todos los gnodos a los cuales esté conectado directamente y así sucesivamente. Cada uno de estos gnodos, responde a este mensaje indicando cuántos archivos comparte y tamaño total de los mismos. Este mecanismo permite la recolección de información acerca de puntos de acceso alternativos a la red. Si por algún motivo un gnodo sale de servicio, sus vecinos pueden mantener su existencia en la red ya que pueden solicitar conexiones a otros nodos conocidos.

El alcance de todos los mensajes Gnutella está regulado por dos campos de la estructura de datos del protocolo. Un identificador único de gnodo (GUID) permite que los demás receptores de un mensaje puedan evitar la formación de ciclos y un tiempo de vida (TTL) limita la cantidad de reenvíos que se puede realizar de un mensaje. Por cada salto, el gnodo receptor decrementa el valor del campo TTL en 1 y al llegar a 0 descarta el mensaje.

En las implementaciones del protocolo Gnutella se puede definir como parámetro la cantidad de conexiones simultaneas a la red que siempre un gnodo debe tratar de mantener. Cuanto más conexiones tenga, se logrará un mayor espacio de consulta y se asegurará una alta disponibilidad de la red, pero se estarán utilizando mayor cantidad de recursos del gnodo y de la red.

1.5.2 – Freenet

Freenet [CLA00] – al igual que Gnutella – es un sistema compañero a compañero completamente descentralizado. Su función principal es la de un sistema de archivos distribuido, que opera sobre equipos de usuario final, permitiendo la inserción, borrado y búsqueda de archivos. A diferencia de Gnutella, este sistema se basa en el concepto de anonimato. Los objetivos principales de su diseño son:

- Anonimato, tanto para productores como consumidores (Ningún usuario puede saber de dónde está descargando archivos, ni quienes descargan archivos de su repositorio)

- Descentralización de funciones, para evitar un único punto de falla en el sistema.

- Eficiencia, en el mecanismo de almacenamiento y ruteo de consultas.

- Anticensura, ya que impide que terceros nieguen el acceso a la información.

Todos los nodos dentro de la red Freenet mantienen un conjunto de archivos que comparten con el resto de los usuarios, junto con una tabla de ruteo que es mantenida dinámicamente por el sistema conforme opere en el tiempo. Freenet está diseñado para adaptarse al comportamiento de los usuarios e implementa de forma transparente mecanismos para mover y replicar archivos para proveer un servicio de búsquedas que resulte eficiente mediante una estrategia de ruteo (no de propagación como la utilizada en Gnutella) distribuida.

Freenet está implementado como una red de nodos compañeros que tienen la capacidad de comunicarse a los efectos almacenar y recuperar archivos. Los mismos se

identifican en el sistema mediante claves independientes de su localización. Dichas claves son generadas por una función de hash y se asocia a un documento particular. Existen básicamente tres tipos diferentes de claves, que se utilizan, a saber:

- *Keyword-Signed Key (KSK)*, las cuales se basa en la aplicación de una función hash a una cadena ingresada por el usuario al momento de insertar el archivo.

- *Signed-Subspace Key (SSK)*, utilizada para identificar un espacio de usuario, de manera de poder construir su reputación (anónimamente) mediante la publicación de documentos.

- *Content-Hash Keys (CHK)*, funciona como una firma de un archivo y sirve para determinar si un archivo es una copia genuina del original.

El mecanismo de búsquedas se basa en un algoritmo primero en profundidad (deep–first) paso a paso. Cuando un usuario realiza una solicitud, se genera un mensaje que es enviado a diferentes nodos. Inicialmente, se lo envía a un nodo conocido (y confiable). Si dicho nodo no posee el documento buscado, lo reenvía a otro nodo de acuerdo a su información de ruteo. Aquí se utiliza una estrategia de similitud lexicográfica de la clave para elegir el nodo siguiente.

Las respuestas vuelven por el camino original de la consulta hasta el solicitante. De esta manera cada nodo en la cadena puede almacenar una réplica en su caché. Por lo tanto, los documentos más solicitados son cacheados en mayor cantidad de nodos. Para aumentar el grado de anonimato, uno de los nodos en la cadena (que almacenó una copia del archivo original) se elige al azar y responde al solicitante como si fuera el proveedor original.

Cada mensaje generado lleva un identificador único generado al azar y una cantidad máxima de saltos entre nodos que puede realizar (tiempo de vida). Esta información permite a los nodos evitar ciclos infinitos y eliminar mensajes luego de su tiempo de vida.

El sistema no está pensado como para almacenamiento fijo de archivos en la red, de manera tal que un nodo puede determinar eliminar algunos. Además, las búsquedas permitidas son por exactitud de claves y no por el contenido de los archivos. El sistema brinda robustez (debido a que soporta rutas alternativas) y buena escalabilidad. Al igual que en Gnutella, para ingresar a la red se debe conocer una dirección inicial de algún nodo operativo.

1.6 – Arquitectura de un Nodo Compañero

Para cumplir con sus funciones básicas en una red compañero a compañero y teniendo en cuenta la simetría de funciones planteada por el modelo compañero a compañero, los nodos deberían poseer al menos seis módulos principales (Gráfico 1.7), a saber:

a) *Módulo servidor*: Parte de un nodo compañero que provee acceso a datos y recursos del compañero donde se ejecuta.

b) *Módulo cliente*: Parte de un nodo compañero que le permite al usuario comunicarse con otros nodos, iniciando conexiones.

c) *Módulo de datos*: Parte de un nodo que almacena información sobre el estado del nodo y la red. Esta información le permitirá seleccionar el modo de operación en cada instante de tiempo, para poder alcanzar el mayor grado de autonomía posible.

d) *Módulo de encaminamiento*: Parte de un nodo que tiene por función principal propagar mensajes propios y de terceros. Esta tarea será realizada según políticas de ruteo y de acuerdo a la información provista por el módulo de datos (por ejemplo, para la selección de los destinos).

e) *Módulo de comunicaciones*: Brinda soporte de comunicaciones entrantes y salientes tanto al módulo servidor como al cliente.

f) *Módulo de seguridad*: Gerencia la seguridad del nodo, implementando políticas definidas. Permite la validación de mensajes, encriptación y autenticación. Puede existir ó no.

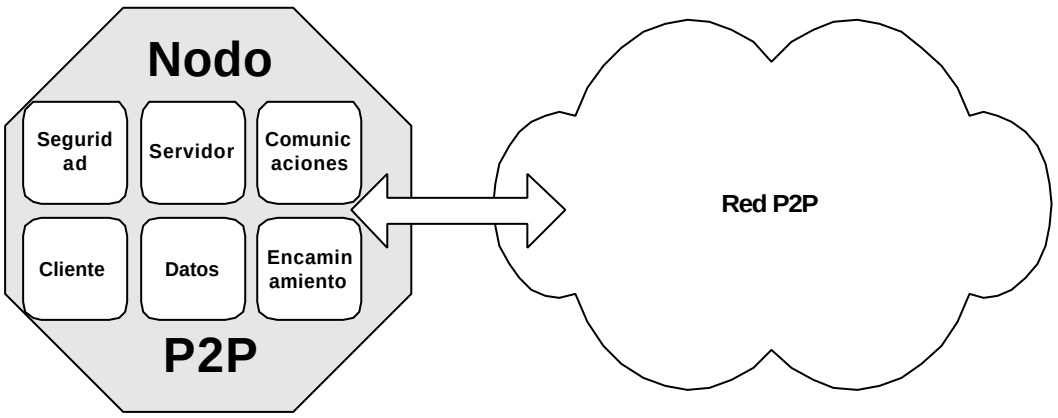


Gráfico 1.7 – Arquitectura típica de un nodo compañero

Funciones del componente cliente

- Proporciona la interfaz de usuario
- Permite enviar requerimientos
- Proporciona conexión con otros nodos compañeros

Funciones del componente servidor

- Detecta, procesa y contesta requerimientos de otros nodos
- Proporciona acceso a sus recursos (Por ejemplo, archivos, tiempo de CPU, espacio en disco)

Funciones del componente datos

- Define un esquema para el almacenamiento de datos y metadatos
- Satisface requerimientos de los componentes servidor y cliente
- Almacena información sobre el estado del nodo y la red (Por ejemplo, una lista de nodos compañeros)

Funciones del componente comunicaciones

- Envía y recibe mensajes de los y a los componentes cliente y servidor
- Administra mensajes duplicados, tiempos de vida de mensajes y trata de optimizar la performance de la red
- Acepta conexiones entrantes
- Gerencia conexiones existentes

Funcionamiento del componente de seguridad

- Garantiza la confidencialidad e integridad de los datos
- Implementa un esquema de autenticidad de nodos compañeros

Estas operaciones se consideran básicas para distintos tipos de aplicaciones P2P. Sin embargo, pueden ser implementadas por diferentes protocolos y diferentes estructuras de datos, pero todas éstas compartiendo la infraestructura subyacente.

Intel proporciona la idea de que debe desarrollarse un modelo de interoperabilidad entre aplicaciones P2P [BAR00], donde tal objetivo puede lograrse a través de un juego común de servicios que proporcionen la funcionalidad necesaria para operar en modo P2P, sobre los servicios proporcionados por el sistema operativo. Estos servicios comunes pueden pensarse como una capa de middleware (Gráfico 1.8).

Una de las ventajas principales de un middleware común es que los diseñadores de aplicaciones ya no tendrán que seguir creando los mismos servicios básicos una y otra vez. Este nivel de interoperabilidad permitirá que equipos con distintas plataformas y con distintas

aplicaciones, programadas en distintos lenguajes de programación se puedan comunicar e integrar.

Gráfico 1.8 – Capa de middleware P2P propuesta por Intel

1.7 – Modelos de Operación

Estos modelos determinan como se comporta la infraestructura P2P para interactuar con sus pares y poder brindar un servicio particular. Una de las cuestiones fundamentales en las arquitecturas compañero a compañero es determinar cómo los nodos "conocen" acerca de la existencia de otros, es decir, cómo realizan el proceso de descubrir cuáles son sus pares en la red (al menos aquellos que se consideran "vecinos"). Las aplicaciones P2P requieren contar con alguna funcionalidad que brinde un "servicio de descubrimiento de compañeros" [SUN02], de manera de poder localizar nodos con los cuales interactuar.

1.7.1 – Modelo Estático

La manera más simple es por configuración explícita. Éste no es realmente un servicio, sino que consiste en la especificación puntual de las direcciones de todos los nodos con los que se puede comunicar y – eventualmente – qué servicios presta cada uno. Esta especificación puede ser parcial ó completa, es decir, se puede indicar cuáles son todos los nodos existentes (situación solo posible en redes de pocos nodos ó redes privadas) ó bien un subconjunto de nodos conocidos (situación típica en redes globales ó públicas).

Un ejemplo de estas formas de trabajo es el sistema de resolución de nombres utilizado en Internet. En los comienzos, se tenía un pequeño grupo de nodos en la red, por lo

que sus nombres y direcciones se asociaban por configuración estática en un archivo (Por ejemplo, en UNIX `/etc/hosts`). Cuando la dimensión y dinamismo de la red determinó que este mecanismo resultaba poco práctico, se implementó el servicio DNS, que opera de manera dinámica.

Por lo general, esta modalidad de configuración explícita tiene problemas de escalabilidad y flexibilidad conforme se agregan más nodos a la red. Como los sistemas P2P pueden ser altamente dinámicos en cuanto a sus componentes, estos inconvenientes determinan que sea necesario y más eficiente contar con un sistema dinámico de descubrimiento de nodos participantes de la red.

1.7.2 – Modelos Dinámicos

A diferencia del enfoque estático de configuración explícita de nodos compañeros, aparecen modelos basados en servicios de directorio y de red que permiten realizar esta tarea de forma dinámica. Éstos se adaptan mejor a las características de las aplicaciones P2P y facilitan la estabilidad, escalabilidad y el funcionamiento del sistema.

1.7.2.1 – Servicio de Directorio

En este modelo se basa en la existencia de un grupo de nodos con funciones específicas que provean un servicio de directorio a los demás nodos compañeros. Cuando un nodo ingresa a la red, debe registrarse en algún servidor de directorio, a los efectos que los demás puedan luego localizarlo (Gráfico 1.9). Los servidores de directorio son también nodos de la red con funciones extendidas. Nótese que un servicio de directorio se puede utilizar también a los efectos de localizar un nodo.

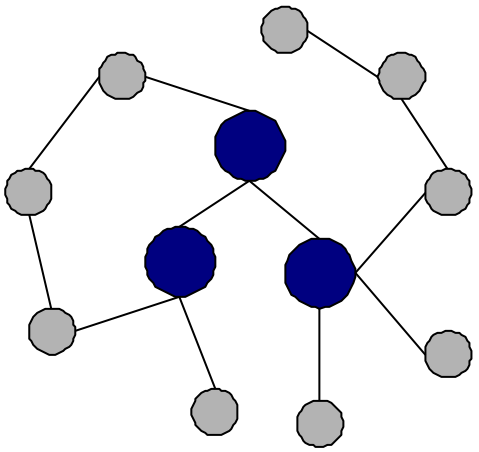


Gráfico 1.9 – Servicio de Directorio

1.7.2.2 – Servicios de Red

Una alternativa a los servicios de directorio consiste en utilizar un conjunto de mensajes que se envían a otros nodos con el objetivo de recolectar información acerca de la existencia de otros nodos ó bien solicitar un determinado recurso ó servicio. Esta tarea se puede realizar básicamente mediante dos mecanismos: difusión (ó inundación) y encaminamiento (ó reenvío directo).

Por difusión: En este caso cuando un nodo requiere recolectar información envía un mensaje a todos los nodos con los cuales se encuentra conectado. Éstos, a su vez, reenvían el mensaje por todas sus conexiones abiertas y así sucesivamente. La intención es que dicho mensaje alcance a todos los nodos dentro de la red (ó al menos, a una cantidad considerable) (Gráfico 1.10). Se lo puede ver como un mensaje tipo broadcast.

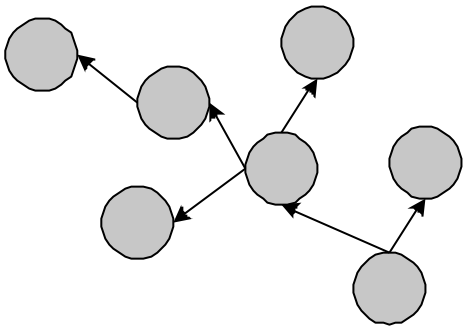


Gráfico 1.10 – Envío por Difusión

Un ejemplo de este mecanismo es la red Gnutella, donde los mensajes de descubrimiento de nodos participantes de la red y los de solicitud de búsqueda de un recurso se envían por inundación. En este caso el alcance del mensaje se encuentra limitado a los efectos de mantener controlada la operación.

Este mecanismo no requiere que los nodos “conozcan” acerca de la topología de la red y de los recursos y servicios disponibles en cada nodo. Si bien puede ser altamente robusto (por redundancia de mensajes), es necesario evaluar su implementación debido a que puede ser un factor limitante en la escalabilidad del sistema (debido a la cantidad de tráfico que genera).

Por encaminamiento: Este mecanismo permite que un mensaje dirigido a un determinado destinatario siga un camino directo desde el origen (Gráfico 1.11). Los nodos intermedios (entre origen y destino) cooperan para propagar dicho mensaje, redireccionándolo

por alguna ruta hasta llegar a destino.

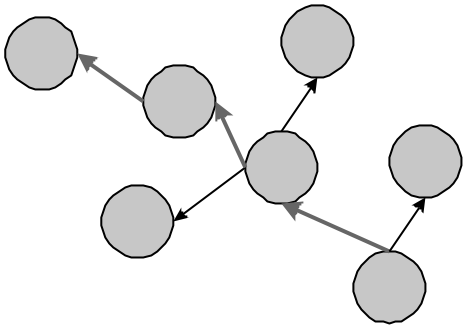


Gráfico 1.11 – Envío por encaminamiento

Para lograr este funcionamiento, los nodos en el camino deben poseer información acerca de la topología de la red ó bien, manejar un esquema de asociación entre nodos y recursos basándose – por ejemplo – en tablas de hash como plantean los proyectos Chord [DAB01], CAN [RAT01] y Pastry [ROW01].

1.8 – Areas de aplicación

Los sistemas P2P pueden aportar soluciones en muchas áreas, principalmente donde la distribución de sus componentes en una red es una característica básica del sistema y donde se pueden aprovechar los recursos disponibles en las “puntas” de Internet. Los equipos de usuario final pueden convertirse en elementos activos dentro de la red.

A continuación se presentan algunas áreas que aparecen como propicias para la implementación de soluciones alternativas al modelo cliente/servidor basadas en redes compañero a compañero:

Grupos y Comunidades: grupos de usuarios con intereses comunes pueden comunicarse directamente a través de sistemas P2P sin intervención de terceros. Esta facilidad puede favorecer el trabajo en grupo y brinda la posibilidad de formar equipos con integrantes distribuidos geográficamente, y disminuir la necesidad de almacenamiento en servidores centrales, utilizando almacenamientos locales con posibilidad de replicación automática.

Computación distribuida: Necesidades de procesamiento en gran escala pueden ser provistas utilizando una red de computadoras donde sus componentes puedan aportar capacidad ociosa de CPU y espacio en disco. Este servicio permite a universidades, institutos de investigación y empresas obtener un gran poder de cómputo a partir de múltiples

computadoras operando en paralelo (procesamiento distribuido sobre una red débilmente acoplada) para aplicar a tareas especificas. Los resultados se pueden obtener en tiempos y costos menores.

Búsquedas distribuidas: Debido a la expansión de la web, los motores de búsqueda tradicionales experimentan problemas de cobertura y de actualización de contenidos. La implementación de servicios de búsquedas distribuidas basadas en redes P2P puede mejorar esta situación, permitiendo a sus usuarios la obtención de resultados de mejor calidad. Si bien puede ser complicado realizar búsquedas de gran cobertura del espacio web, los sistemas P2P ofrecen otra visión de cómo abordar el problema y plantean soluciones para búsquedas extendidas sobre temas específicos.

Comercio Electrónico: Diversas compañías están experimentando con modelos de conexión directa entre usuarios a los fines de implementar sistemas de comercio electrónico. La idea es reemplazar un mercado electrónico centralizado por intercambios directos entre compradores y vendedores.

Replicación de contenidos: Una red de aplicación como la planteada puede ayudar a distribuir contenidos a través de distintas áreas geográficas. Esencialmente, pueden moverse los datos más cerca del punto donde realmente son consumidos, implementando mecanismos de replicación. Las redes de distribución de contenidos (CDN, Contents Distribution Networks), que tienen por objetivo lograr que los usuarios finales accedan más rápidamente a contenidos.

Intercambio de archivos: Permiten que usuarios de computadoras personales intercambien directamente entre ellos archivos de datos. Estas aplicaciones han obtenido una gran popularidad gracias a sistemas como Napster, Gnutella y Freenet. Actualmente, estos sistemas focalizan sus contenidos en la distribución de archivos musicales, videos y fotografías.

Juegos: La infraestructura P2P resulta un ambiente natural para el desarrollo de juegos en línea distribuidos, sin intervención de controles centrales, donde solamente existe interacción entre los participantes del mismo.

Distribución de Datos Multimediales: La distribución de datos multimedia en tiempo real a través de Internet es un área en desarrollo. Generalmente, se requiere operar con modelos multicast a los efectos de hacer eficiente la transmisión. Un modelo alternativo basado en una red P2P se plantea en [TOL02a], donde un conjunto de nodos compañeros colaborar para brindar un servicio de transmisión de radio en tiempo real sobre Internet.

Desarrollos colaborativos: el trabajo en grupo puede verse beneficiado por las

facilidades que se pueden implementar para concretar tareas de manera distribuida. Ejemplos de aplicación son el desarrollo de software y el rendering de gráficos.

1.9 – Futuro de los sistemas P2P

Los sistemas compañero a compañero se encuentran en desarrollo. La mayoría de éstos son construidos con un objetivo único y su implementación se centra en la prestación de un servicio en particular. Si bien las aplicaciones emergentes operan sobre Internet e intentan aprovechar a los usuarios de los "bordes" de la red, es interesante no restringir su aplicación ni a este escenario ni a las aplicaciones hoy conocidas. Por ejemplo, pueden plantearse servicios para organizaciones que se brinden sobre sus redes locales ó redes privadas.

Además, se requiere trabajar en varios aspectos técnicos que permitan aprovechar todas las posibilidades que brinda el modelo, como por ejemplo:

- *Sistemas de Nombres:* Desarrollar mecanismos de identificación de los nodos dentro de la red y de los usuarios de los mismos en un ambiente descentralizado.

- *Mecanismos de Ruteo:* La localización de recursos y el encaminamiento dentro de la red es un área a desarrollar a los efectos de diseñar estrategias que operen de manera eficiente y permitan que el sistema escale en cantidad de nodos conectados.

- *Seguridad, reputación y confianza:* Anonimato, servicios anticensura, y hospedaje descentralizado de bloques de información (una misma información, partirla y distribuirla en n nodos) serán los principios de diseño en las nuevas aplicaciones. El derecho a la privacidad se garantizará implementando como una cualidad estándar de cualquier nueva aplicación.

Actualmente el acceso general a espacios de almacenamiento y a ciclos de CPU se realiza bajo un riesgo de seguridad importante por parte de los usuarios. La computación distribuida necesitará definir nuevos entornos de trabajo, protocolos de comunicación y lenguajes de programación que minimicen los riesgos (ejecución de código no seguro ó acceso a información no compartida) en tales equipos, principalmente:

- *Plataformas*: Modelos de ruteo en redes P2P aportan la posibilidad de ruteo por difusión (multicast). Implementándose mucho más fácil y con menos requisitos que la definida a nivel de red en Internet. Tal característica será mayormente difundida

a los efectos de lograr comunicaciones más veloces y que utilicen un ancho de banda menor.

- *Estándares*: Definir maneras o formas más fáciles de encontrar información compartida, ya sea a través de normalización de formas de almacenamiento, definición de metadatos y sistemas de búsqueda distribuida.

Una iniciativa en la construcción de plataformas de desarrollo de sistemas compañero a compañero es el Proyecto JXTA [JXT03]. Este proyecto pretende brindar un entorno que resuelva algunos problemas comunes al funcionamiento de una red P2P (como por ejemplo, la localización de recursos y el intercambio de mensajes), facilitando a los programadores el desarrollo de nuevos servicios y aplicaciones. En el Anexo II se describe con mayor detalle la plataforma JXTA y se mencionan algunos proyectos existentes sobre la misma.

1.10 – Consideraciones

El modelo compañero a compañero supone un cambio de paradigma, es decir, una forma diferente de intercambio de información y de interacción. Sistemas P2P permiten que los nodos individualmente dialoguen entre sí con el objetivo de proveer ó utilizar un determinado recurso ó servicio, en lugar de interactuar con servidores centrales.

La computación P2P es potencialmente una forma alternativa de aprovechamiento de los recursos computacionales disponibles (propios ó ajenos), brindando un ambiente para proveer servicios desde los extremos (edges), es decir, desde computadoras de usuario final. Dichos equipos tradicionalmente permanecían ocultos para el resto de los usuarios y era imposible nombrarlos y direccionarlos bajo los esquemas clásicos. De esta manera se pueden formar sistemas que brinden servicio a gran escala.

La descentralización y los modelos P2P tienen, probablemente, más aplicaciones que las que al presente es posible imaginar. En el futuro, en cualquier área donde la escalabilidad y la capacidad de cómputo masivo sean importantes, se implementarán sistemas descentralizados.

En las redes P2P, cada usuario construye su propio espacio virtual, elige quienes serán sus interlocutores y lo que desea compartir con ellos [TOL02b]. Este esquema brinda libertad, en base a las posibilidades de elección, dado que los usuarios eligen a que comunidades desean pertenecer.

Existen algunos problemas a resolver (como por ejemplo, confianza mutua, seguridad de las aplicaciones e integridad de los datos) que pueden impedir la masificación de los sistemas P2P. En algunos ambientes, los inconvenientes relacionados con la seguridad y la privacidad resultan importantes barreras para que este tipo de aplicaciones tengan una adopción rápida.

Capítulo 2

Búsquedas Distribuidas en el Espacio Web

La World Wide Web (web) puede ser vista como un gran repositorio de información, completamente distribuido en una red (en este caso, Internet) y accesible por mucha cantidad de usuarios. Por sus orígenes como un espacio público existen millones de organizaciones ó usuarios particulares que incorporan, quitan ó modifican contenido continuamente, por lo que su estructura no es estática. Además, esta gran cantidad de posibles proveedores de información sobre la web hace que su contenido no respete estándares de calidad, ni estilos ni organización. Y como medio de publicación de información de naturaleza diversa se ha convertido en un servicio de permanente crecimiento. Una de las características de la información publicada en la web es su dinamismo, dado que pueden variar en el tiempo tanto los contenidos como su ubicación [BRE00] [LAW99a].

La búsqueda de información de la web es una práctica común para los usuarios de Internet pero debido a las características propias de ésta se ha convertido en un desafío para los sistemas de recuperación de información. Los servicios de búsqueda (motores de búsqueda) aportan una gran solución al problema de encontrar información en la web, pero con el paso del tiempo se requieren de más y mejores herramientas. En algunos casos, debido al crecimiento tanto de la cantidad de información disponible y al crecimiento del número de usuarios que utilizan dichos sistemas. En otros casos, diferentes necesidades requieren diferentes soluciones de búsqueda, o bien, soluciones específicas enfocada a problemas particulares.

Existen diferentes soluciones para dicho problema, con enfoques diferentes, lo que deriva en un amplio espectro de herramientas que permiten ó facilitan a los usuarios la localización de información relevante a sus necesidades. Por ejemplo, los motores de búsqueda de propósito general como Google [BRI98] [GOO03a] ó Teoma [TEO03] aportan servicios de búsquedas acerca de cualquier tema y sus fortalezas se basan en su cobertura y sus estrategias de clasificación y rankeo de respuestas. Otras herramientas limitan su espacio de operación a temas específicos ó bien, a un tipo particular de documentos. Por ejemplo, Excite NewsTracker [EXC03] ó Google Grupos [GOO03b] permiten realizar búsquedas sobre noticias Usenet. Otras herramientas se basan en técnicas de clasificación manuales. Este es el caso de los directorios como Yahoo [YAH03] ó el proyecto Open Directory [OPE03], los cuales

retornan menor cantidad de respuestas irrelevantes. En general, los motores de búsqueda que cubren un tema específico a menudo proveen resultados más precisos y más actualizados, comparado con los motores de búsqueda de propósito general [LAW98].

2.1 – La web como fuente de información

La web es un espacio de cientos de millones de documentos que pueden ser accedidos por los usuarios del sistema. También se la puede verse como una gran base de datos no estructurada. Conviven en ésta documentos de diferente naturaleza y formato, desde páginas HTML hasta archivos de imágenes pasando por gran cantidad de formatos estándar y propietarios.

Como se mencionó anteriormente, una solución ampliamente utilizada son los motores de búsqueda. Actualmente, éstos cubren gran parte de la web pero su funcionamiento no es el ideal. Cuando un usuario realiza una consulta se ve sobrecargado de respuestas, siendo gran parte de dicha respuesta irrelevante. Además, hay que tener en cuenta que los estudios realizados muestran que el 85% de los usuarios solo revisa la primer página de respuestas [BAE99a] y – aunque quisieran – no existe tiempo físico para poder evaluarlas a todas. Por lo tanto, el problema de los motores de búsqueda es tratar de satisfacer las necesidades de información de los usuarios de la forma más eficiente posible. Expresado de otra manera, se puede decir que el desafío es encontrar los objetos de información que un usuario necesita, de acuerdo a su consulta.

El crecimiento que ha tenido la web y sus características plantean a los motores de búsqueda una serie de inconvenientes a resolver, principalmente relacionados con [BAE99b]:

- Datos distribuidos: La web es un sistema distribuido, donde cada proveedor de información publica su información en computadoras conectadas a redes conectadas a Internet, sin una estructura ó topología predefinida.

- Datos volátiles: El dinamismo de sistema hace que exista información nueva a cada momento ó bien que cambie su contenido ó inclusive desaparezca otra que se encontraba disponible. En muchos casos, los motores de búsqueda pierden referencias cuando un archivo cambia de nombre ó de ubicación dentro del sistema.

- Calidad: En general, la calidad de la información publicada en la web es altamente variable, tanto en escritura como en actualización (existe información que puede

considerarse obsoleta), e inclusive existe información con errores sintácticos, ortográficos y demás.

- Heterogeneidad: La información se puede encontrar publicada en diferentes tipos de medios (texto, audio, gráficos) con diferentes formatos para cada uno de éstos. Además, hay que contemplar los diferentes idiomas y diferentes alfabetos (por ejemplo, árabe ó chino).

Estos inconvenientes sumados al tamaño de la web también imponen restricciones a las herramientas de búsqueda en cuanto a la cobertura y acceso a los documentos, exigiendo cada vez mayores recursos computacionales (espacio de almacenamiento, ancho de banda de las redes, ciclos de CPU) y diferentes estrategias para mejorar la calidad de las respuestas entregadas a los usuarios.

2.2 – Búsquedas en el espacio web

Como se mencionó anteriormente, los motores de búsqueda centralizados como Google ó Teoma son una solución válida y ampliamente utilizada para resolver el problema del acceso a los sitios que contienen información de interés para el usuario. Básicamente, su arquitectura y modo de operación se basa en poder recolectar mediante un mecanismo de recolección los documentos existentes en los sitios web. Una vez obtenidos los documentos, se llevan a cabo tareas de procesamiento que permiten extraer los términos contenidos dentro de los mismos a los efectos de construir estructuras de datos (índices) que permitan realizar búsquedas de manera eficiente. Luego, a partir de una consulta realizada por un usuario, un motor de búsqueda extraerá de los índices las referencias que satisfagan la consulta y se retornará una respuesta al usuario.

El modo de funcionamiento de los diferentes motores de búsqueda puede diferir en diversas implementaciones de los mecanismos de recolección de datos, los métodos de indexación y los algoritmos de búsqueda y clasificación. Además, su arquitectura puede ser centralizada ó distribuida.

2.2.1 – Motores de Búsqueda de Arquitectura Centralizada

Básicamente, los motores de búsqueda centralizados se basan en una arquitectura de tres componentes principales: recolector, indexador y motor de consulta. Dicha arquitectura se muestra en la figura 2.1.

- Recolector: también denominado crawler, spider ó robot. Este componente opera estableciendo conexiones con servidores web y solicitándole documentos. Los nombres y direcciones (URLs) de los documentos que debe recuperar se encuentran en archivos de datos que se actualizan a medida que se extraen más referencias de los documentos procesados.

- Indexador: este módulo implementa mecanismos de pre-procesamiento de documentos y construye las estructuras de datos (en general, variantes de archivos invertidos) que soportarán la búsqueda. En el pre-procesamiento se llevan a cabo tareas de normalización de los documentos (palabras en mayúsculas y minúsculas, acentos, signos de puntuación) y de extracción de términos no relevantes. La actualización de los índices depende de la frecuencia con que un crawler obtenga nuevos documentos ó versiones actualizadas de documentos existentes.

- Motor de consulta: es el módulo que interactúa con el usuario aceptando de éste una expresión de consulta para realizar la búsqueda. Con ésta, consultará los índices a los efectos de encontrar los documento que satisfagan la consulta y con ellos armará la respuesta. Para este último paso, implementará alguna estrategia de clasificación ó rankeo para determinar el orden en que las referencias encontradas serán mostradas al usuario.

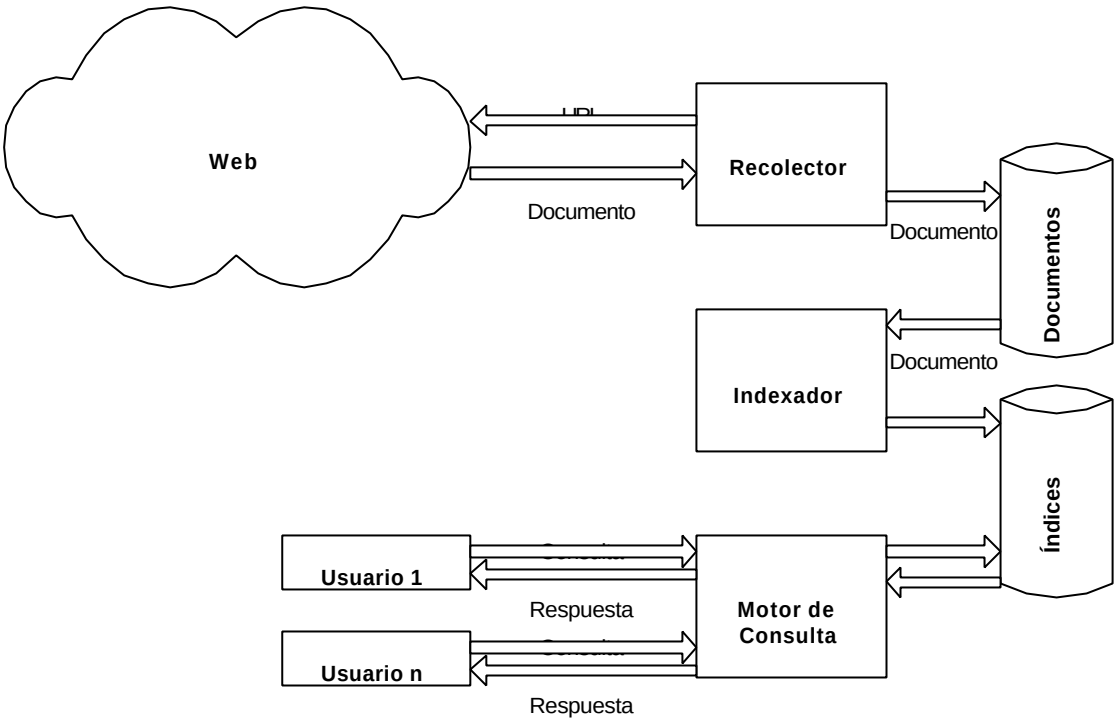


Figura 2.1 – Componentes de un motor de búsqueda de arquitectura centralizada

Uno de los problemas más importantes de esta arquitectura está relacionado con la recolección de los documentos. Esta tarea es la que permite obtener la información para la construcción de los índices. El dinamismo de la web plantea un serio problema en este sentido. El índice de un motor de búsqueda no puede ser una estructura estática, sino todo lo contrario. La frecuencia de actualización de los documentos requiere que el crawler constantemente vuelva a solicitar a los servidores web los mismos a los efectos de poder detectar cambios en el contendido, que determinarán cambios en los índices. Además, la recolección de documentos es una tarea que consume una alta cantidad de recursos computacionales y que se encuentra en cierta medida limitada por el tráfico existente en las redes, la disponibilidad de los servidores web y la carga que tengan en el momento que el crawler les solicita documentos.

Por otra parte, se puede plantear como ventajas de las arquitecturas centralizadas la posibilidad de almacenar información acerca de la ubicación y distancia de determinado contenido al usuario que lo busca y – por tener todas las referencias en un único índice – se pueden implementar mecanismo de clasificación y rankeo por relevancia que abarcan a todo el contenido, como por ejemplo el mecanismo implementado por Google, denominado PageRank [BRI98], tiene este requerimiento.

2.2.2 – Problemas asociados al enfoque centralizado

Las características enunciadas de la web establecen restricciones en cuanto a la cantidad y calidad de la información recolectada y a las respuestas entregadas a un usuario ante una solicitud, en especial utilizando los motores de búsqueda tradicionales (basados en una arquitectura centralizada). De dichas características, el tamaño actual del espacio web, su constante crecimiento y la frecuencia de actualización de contenidos generan una serie de problemas importantes, a saber:

- *Aumento de la proporción de respuestas irrelevantes*. Ante una consulta sobre una temática determinada, los motores de búsqueda de propósito general, retornan – por su propia naturaleza – demasiadas referencias a páginas cuyo contenido no es relevante según la consulta. Esto se debe a la diversidad del espacio web que cubren. Como se mencionó anteriormente, una alternativa para resolver esta problemática la brindan los motores de búsqueda especializados, dado que solo operan en un área temática.

- *Mantenimiento de los índices*. Dados los permanentes cambios e incorporación en los contenidos de los sitios web, la calidad del motor de búsqueda estará condicionada, entre otros factores, a la frecuencia de su esquema de visita y

reindexación. Actualmente, es una necesidad real que este tiempo sea lo más corto posible. Un estudio realizado por la compañía SearchEngineShowdown [SEA02] muestra que el promedio de actualización de los índices de los motores de búsqueda más importantes varía entre 2 semanas y 3 meses, por lo que

- *Cobertura limitada*. Solo una fracción de la web es indexada por los motores de búsqueda y en particular la cobertura de un motor es significativamente limitada [LAW98]. Además, existe una parte de la web conocida como la "web invisible" [BER00] que no es indexada por los motores de búsqueda. En general, se trata de contenido que es generado dinámicamente a partir de contenido almacenado en bases de datos y en selecciones de los usuarios ó bien, páginas que se desean excluir de los índices por alguna política particular.

- Alto requerimiento de recursos. Dada la cantidad de información y usuarios en Internet, se requiere alta cantidad de recursos de hardware como equipos y ancho de banda, tanto para recolectar los documentos a indexar, construir las estructura de datos sobre las que se efectuarán las búsquedas y atender los requerimientos de los usuarios.

Un estudio sobre acceso a información científica en la web [LAW99b], también expone limitaciones de los motores de búsqueda en cuanto a la cobertura, frecuencia de actualización, debilidades en los algoritmos de rankeo y la flexibilidad de los lenguajes de consulta.

Desde el punto de vista de la disponibilidad del servicio, se puede plantear que un motor de búsqueda de arquitectura centralizada tiene un único punto de falla, aunque se implementan técnicas de replicación, balanceo de cargas y tolerancia a fallas para poder manejar la cantidad de información y la cantidad de usuarios existentes en la red.

2.2.3 – Motores de Búsqueda de Arquitectura Distribuida

Existen variantes en motores de búsqueda de arquitectura distribuida. Cada una de éstas se debe a que su naturaleza distribuida puede darse – en general – en el mecanismo de recolección, en la construcción de los índices ó en ambas. En esta arquitectura se plantea la existencia de componentes de recolección de información que operan coordinadamente con agentes de búsqueda para realizar la tarea. (Figura 2.2).

Los recolectores obtienen la información de varios servidores web y pueden llevar a cabo tareas de procesamiento como manejar diferentes formatos de documentos y generar la

información que se enviará a los agentes.

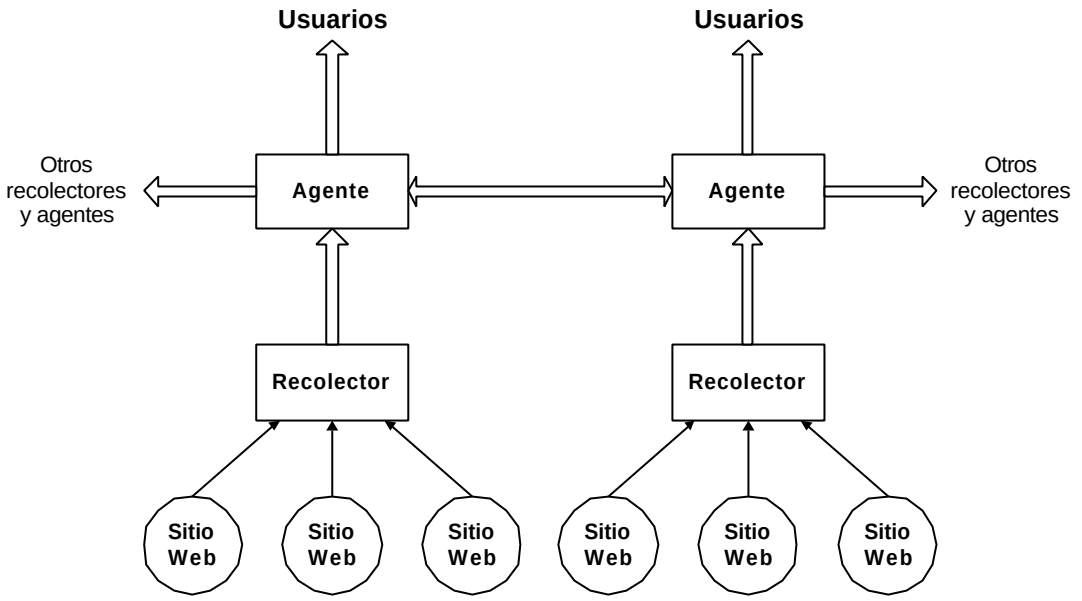


Figura 2.2 – Componentes de un motor de búsqueda de arquitectura distribuida

Los agentes son los encargados de consultar a los recolectores sobre la información obtenida y con ésta construir los índices que soportarán las búsquedas. Además, pueden operar en coordinación con otros agentes a los efectos de intercambiar información. Además, brindan al usuario una interfaz para la especificación de la consulta, realizan la búsqueda sobre las estructura de datos construidas (que almacenan localmente) y entregan los resultados. Opcionalmente, pueden realizar tareas de rankeo ó clasificación de respuestas.

Algunas de las ventajas de operar bajo esta arquitectura están dadas por la reducción del tráfico en las redes, ya que se puede situar a los recolectores “cerca” de las fuentes de información (e inclusive junto a éstas). También se evita el trabajo redundante ya que un mismo recolector puede enviar información a diferentes agentes, lo que reduce la carga de trabajo sobre el servidor web ya que se trabaja coordinadamente.

2.3 – Arquitecturas para búsqueda distribuidas

2.3.1 – Basadas en el modelo cliente/servidor

2.3.1.1 – Harvest

Una de las arquitecturas más importantes es la presentada por Harvest [BOW95]. Esta arquitectura se basa en la idea de separar la actividad de recolección de la distribución de la información recolectada. Utilizada dos componentes principales denominados gatherers y brokers. Un gatherer recolecta información de uno ó más servidores web mientras que los brokers proveen mecanismos de indexación y presentan una interfaz con a los datos recolectados. En la figura 2.2 se muestra la arquitectura de Harvest [BOW95].

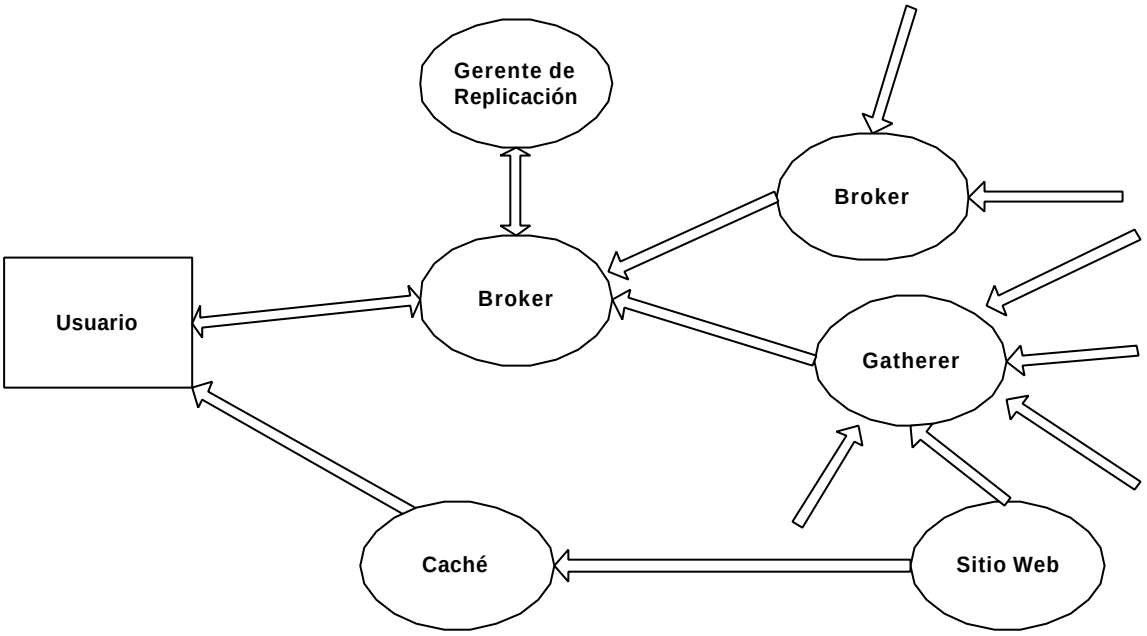


Figura 2.2 – Arquitectura de Harvest

Los gatherers y los brokers se pueden organizar en diferentes configuraciones a los efectos de optimizar el uso de los servidores y la red. Por ejemplo, un broker puede obtener información de varios gatherers ó incluso de otros brokers, minimizando el tráfico. Otra posibilidad es mantener un gatherer en cada servidor web, lo que permite la recolección de información sin generar tráfico externo al servidor.

2.3.1.2 – Oasis

OASIS [BES99] se presenta como un sistema distribuido de motores de búsqueda, proveyendo servicios de búsqueda de texto plano y documentos HTML almacenados en servidores públicos sobre Internet. Funciona de la siguiente manera:

Un usuario contacta un servidor OASIS con un requerimiento (consulta). Dicho requerimiento es procesado localmente y propagado a los demás servidores OASIS en el sistema. La lista de servidores a los que se les reenvía la consulta es obtenida de un Servicio de Directorio. El servidor OASIS que propaga la consulta se comporta como un cliente dentro del sistema. Antes de retornar los resultados al usuario, el servidor OASIS que inició la propagación de la consulta elimina los duplicados y los ordena por un índice de relevancia. La figura 2.3 muestra un diagrama del funcionamiento de OASIS [BES99].

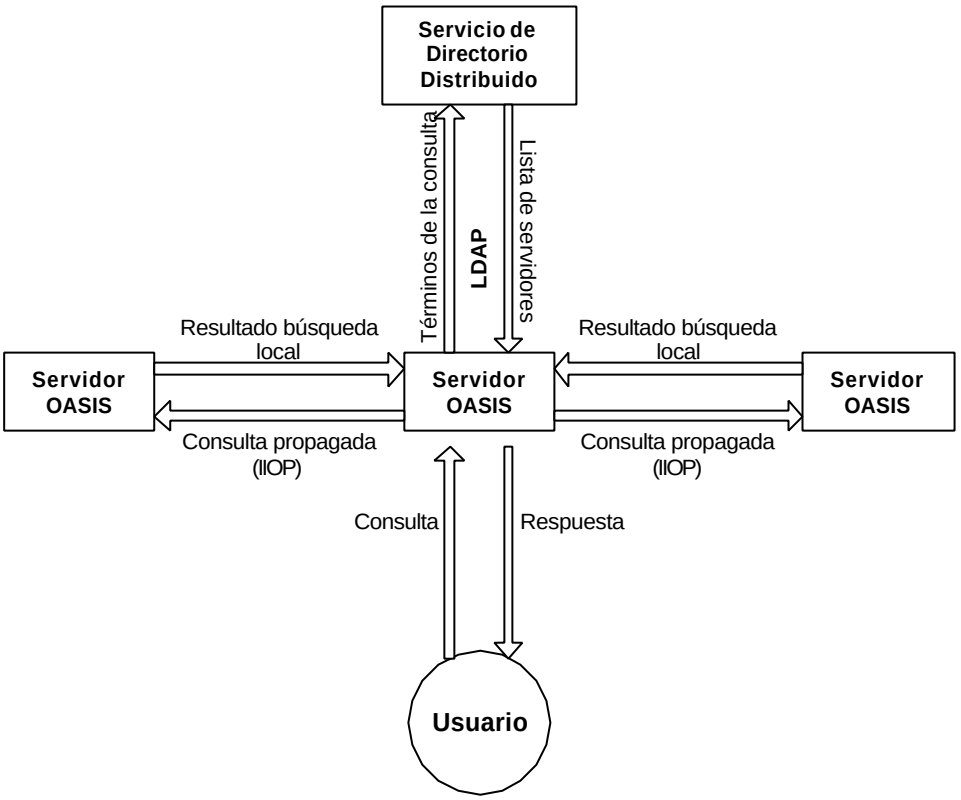


Figura 2.3 – Diagrama de procesamiento de una consulta en OASIS

Además, el sistema permite que los usuarios envíen información de retroalimentación sobre la relevancia de los resultados. Dicha información es propagada a los servidores que participan de la búsqueda para mejorar el procesamiento de futuros requerimientos del usuario.

Cada servidor OASIS mantiene una porción de los documentos disponibles en Internet y se puede especializar en un tema particular ó varios. El área de cobertura de los servidores no es mutuamente excluyente para ninguno, por lo que puede existir solapamiento de contenido indexado.

Los servidores OASIS utilizan un protocolo de comunicación basado en CORBA para

propagar las consultas y un servicio de directorio basado en LDAP (Light Weight Directory Access Protocol) para anunciarse dentro del sistema y anunciar sus colecciones (temas acerca del contenido que cada uno indexa).

2.3.2 – Basadas en el modelo compañero a compañero

2.3.2.1 – JXTA Search

JXTA Search [WAT02] es un proyecto construido sobre la base del Proyecto JXTA [MAI02] [JXT03] que tiene como objetivo principal proveer un mecanismo para soportar consultas distribuidas para dispositivos diversos, desde servidores web hasta computadoras portátiles (En el anexo II se presenta la arquitectura de JXTA). La idea básica es distribuir consultas a una red de nodos compañeros capaces de responderlos. La red de búsqueda de JXTA Search opera mediante un protocolo basado en XML denominado QRP (Query Routing Protocol), el cual define los mecanismos para enviar consultas, recibir respuestas y definir metadatos para cada uno de los nodos de la red. En esta red, los nodos pueden adoptar diferentes roles, como:

- Proveedores de Información, son nodos que pueden responder a consultas utilizando el formato definido por QRP.

- Consumidores de Información, son nodos que realizan consultas ó requerimientos utilizando el lenguaje QRP.

- Hub de Búsqueda JXTA, es un mecanismo que facilita el ruteo eficiente de los mensajes en la red entre proveedores y consumidores de información.

En su modo de operación, JXTA Search plantea la existencia de una serie de nodos hubs, cada uno manejando las consultas de un grupo de compañeros. Se pretende que cada nodo hub pueda manejar nodos proveedores y consumidores con temas afines ó relacionados (por ejemplo, proveedores de información que tengan documentos de un tema específico).

Cuando un nodo ingresa a la red como proveedor de información, debe registrarse en algún hub brindando datos (metadatos) acerca de su descripción. Dichos datos se asocian con la descripción de las consultas que “quieren” (ó “pueden”) responder. Un nodo consumidor se conecta a algún hub de la red y generan requerimientos. El hub, en base a la información disponible acerca de los proveedores conectados a éste, determina a cuáles debe rutear dicha solicitud. Si un nodo hub no puede satisfacer la consulta (ó bien ésta debe expandirse), puede

rutearla a otro nodo hub conocido. Si alguno de los proveedores a los que le llegó la consulta retorna una respuesta, ésta es entregada por el nodo hub al nodo consumidor solicitante.

Aunque un simple hub es un mecanismo escalable, se pretende que éstos sean una forma eficiente de agrupar proveedores por contenido ó atributos similares en una red compañero a compañero.

2.3.2.2 – PeerDB

La aplicación PeerDB [SIO03] se basa en la implementación de un sistema de gerenciamiento de datos distribuidos sobre bases de datos que opera sobre la plataforma BestPeer [SIO02]. Esta plataforma es un sistema P2P genérico que implemente tecnología de agentes móviles, lo que permite extender su funcionalidad. Cada nodo en BestPeer puede comunicarse y compartir recursos con otros nodos dentro de la red.

En PeerDB, el almacenamiento en los nodos está manejado por un sistema de bases de datos (sin mantener un esquema global dentro del sistema) y el procesamiento es realizado por agentes móviles. La arquitectura de un nodo dentro del sistema se basa en los siguientes componentes (figura 2.4):

- Sistema de Administración de Datos: Se encarga del almacenamiento, manipulación y recuperación de los datos contenidos en el nodo. La interfaz con este componente está facilitada por una consulta en SQL. Este componente se complementa con un Diccionario Local, que almacena información referida a las tablas que forman la base de datos como por ejemplo, un conjunto de palabras clave que la describen. Además, un Diccionario de Exportación, almacena metadatos referidos a los objetos que se comparten con el resto de los nodos de la red.

- DBAgent: Es el módulo que provee el entorno para la operación de los agentes móviles.

- Administrador de Cache: El sistema utiliza una política de caché de objetos remotos en almacenamiento secundario.

- Interfaz de usuario: Provee la interfaz para la administración del sistema (por ejemplo, agregado/borrado de objetos) y para especificar las consultas.

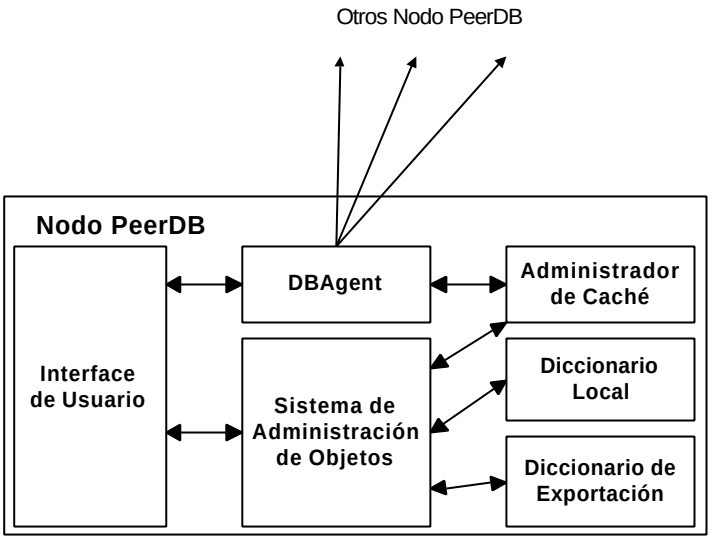


Figura 2.4 - Arquitectura de un nodo PeerDB

Debido a que los nodos no comparten un esquema de base de datos uniforme, son importantes los metadatos asociados a cada tabla creada en cada nodo, a los efectos de poder relacionar una consulta dada (formada por tablas, atributos y condiciones) con el esquema local. Al recibir una consulta desde un usuario, se crea un agente (agente maestro) para evaluarla e interactuar con el sistema de base de datos para responderla. Aquí, PeerDB debe aplicar una estrategia de relación para localizar las posible tablas locales. Si el agente determina que las tablas se encuentran en un nodo remoto, éste es enviado con la consulta al mismo. Cuando el agente llega a un nodo, decrementa un contador de TTL y si TTL > 0, el agente es enviado a los nodos vecinos del nodo remoto.

Las respuestas son enviadas al agente maestro, el cual las retorna al usuario progresivamente, es decir, éste puede estar viendo las primeras respuestas mientras otro agentes se encuentran aún buscando dentro de la red.

2.3.2.3 – Sistema P2P de motores de búsqueda XML

Sobre la idea de que todas las fuentes de información se pueden organizar como un sistema P2P, se plantea un sistema distribuido de procesamiento de consultas escritas en XML [GAL02]. Cada nodo dentro del sistema consiste en un motor de búsqueda XML y una infraestructura de comunicaciones que le permite interactuar con nodos pares.

El motor de búsqueda permite realizar búsquedas basadas tanto en la estructura como el contenido de los documentos no solamente en almacenamiento local, sino también en nodos

vecinos, mediante una estrategia de reenvío de las consultas. Cuando un nodo ingresa al sistema intercambia información con sus nodos vecinos a los efectos de realizar dos tareas necesarias:

a) Anunciar su presencia a los nodos vecinos y publicar sus datos. Esto último lo realiza a través de descripciones pequeñas escritas en XML de sus colecciones de datos.

b) Obtiene información acerca de los archivos de índices de los demás nodos pares a los cuales está directamente conectado.

El sistema usa datos que a menudo cambian constantemente y pueden aparecer en expresiones de consulta ó como metadatos ó palabras características de un determinado sitio. Para ello, se implementan índices que mapean dichos datos con los sitios correspondientes, siendo esta información útil para luego aplicar una estrategia de reenvío de consultas.

Cada nodo además mantiene un índice acerca de información residente en otros nodos del sistema, denominado PI (Peer Inverted Index) que mapea palabras clave con los identificadores (ID) de los del nodo remoto (en este caso, como identificador se utiliza la dirección IP del nodo y el número puerto). La cantidad de información sobre otros nodos está limitada por un concepto de horizonte, definida como una distancia igual a un radio desde un nodo dado.

Este desarrollo soporta consultan que aprovechan la estructura de los documentos XML y – además – facilita la selección eficiente de nodos pares relevantes dada una consulta. El mecanismo de unión de nodos precomputa los destinos de las mismas (esta información se almacena en los PI).

La actualización de los índices se realiza por un mecanismo de encuesta periódica, ó bien, cuando en algún nodo se disparen acciones predeterminadas que requieran que los nodos intercambien sus nuevas ó actualizadas palabras clave.

Los autores pretenden que este desarrollo sirva como una plataforma de bajo nivel para sistemas distribuidos de procesamiento de consultas que permita recuperar los documento XML que satisfagan una consulta dada.

2.3.2.4 – Otros Enfoques

Se han planteado varios enfoques para resolver el problema de las búsquedas

distribuidas en sistemas que operan bajo el modelo compañero a compañero. En [CRE02] se plantea el uso de estructuras de datos que en vez de comportarse como índices tradicionales que determinan el destino ó la ubicación de un recurso, se utilizan “índices de ruteo” (RI - routing indices), los cuales dan una “dirección” hacia el documento en vez de su localización actual. Al utilizar rutas, el tamaño del índice es proporcional al número de vecinos, en vez del número de documentos.

Un enfoque alternativo desarrollado es el uso de tablas de hash distribuidas (DHT – Distributed Hash Tables). La idea básica es especificar una relación entre documentos y nodos dentro de una red. Básicamente se mapean dentro de un mismo espacio (que puede ser n dimensional) un conjunto de nodos y un conjunto de recursos (documentos). Las direcciones dentro del espacio mencionado (IDs) surgen a partir de la aplicación de una función de hash a los identificadores de nodos y de recursos. De esta manera se puede asociar un par <Atributo, Valor> donde <Atributo> es el nombre del recurso y <Valor> la dirección del nodo donde se encuentra. Dicho par se almacena en el nodo cuyo identificador surge de la aplicación de la función de hash sobre el nombre del recurso. Proyectos que han desarrollado esta idea son Chord [STO01], CAN [RAT01] y Pastry [ROW01].

Otros enfoques plantean realizar la tarea de ruteo basándose en el contenido (ruteo semántico). Por ejemplo, en el proyecto Neurogrid [JOS02] se utiliza un sistema adaptivo de búsqueda descentralizada. Con cada búsqueda cambia el conocimiento que cada nodo tiene acerca del contenido de otros nodos. La diferencia de Neurogrid con otros sistemas es que los resultados devueltos se almacenan y son usados para actualizar los metadatos que describen a un servidor. Si un servidor es consultado por un tema y retorna sobre otro tema (por retroalimentación del usuario), baja la confiabilidad en dicho servidor con respecto a la consulta. Neurogrid gradualmente “aprende” la ubicación de contenido diferente, sin mover archivos de lugar. Sirve para aquellas situaciones donde uno no sabe exactamente qué archivo busca.

Capítulo 3

gnutWare: Middleware de acceso a redes P2P

Como se mencionó anteriormente, los sistemas compañero a compañero que surgieron en los últimos años tienen como objetivo brindar un servicio determinado (por ejemplo, mensajería, distribución de archivos y cómputo distribuido) operando principalmente en computadoras de usuario final. Una cuestión importante es que aprovechan recursos existentes en los extremos de la red.

En la mayoría de los casos un determinado proyecto centra su desarrollo en una plataforma de comunicaciones que soporta un determinado servicio [BOR01] [CLA00] [DIS03] [ICQ03]. Sin embargo, estos proyectos pueden ser divididos en dos partes principales: una referida al soporte de comunicaciones y otra referida al servicio propiamente dicho [TOL02c]. Esta división se plantea a partir de pensar en la existencia – por un lado – de diferentes arquitecturas compañero a compañero que utilicen diferentes estrategias para su construcción, envío de mensajes, mantenimiento, interacción entre nodos pares y demás. Por otro lado, la existencia de diferentes aplicaciones que brindan servicios a usuarios finales y que pueden requerir de servicios de comunicaciones para operar en un ámbito completamente distribuido.

Se introducen – entonces – diferentes servicios y aplicaciones sobre diferentes plataformas ó entornos, donde éstos se vinculan a partir de la utilización de una capa de middleware. Si bien no hay una definición consensuada sobre su dignificado y alcances [AIK00] el concepto de middleware se utiliza para denotar un conjunto de servicios que operan sobre el sistema operativo que tienen como objetivo proveer mecanismo de vinculación entre elementos (por ejemplo, por un problema de convergencia tecnológica), lo que permite la interoperabilidad entre éstos. También se lo puede plantear como capas de software que proporciona un nivel de abstracción, por ejemplo a un programador, ocultando detalles inherentes a los protocolos de comunicación, arquitecturas de hardware y software y demás [COU01]. Los servicios típicos implementados como middleware incluyen servicios de directorio, invocación de métodos remotos y monitores de transacciones. Un ejemplo típico es el caso de CORBA [OMG95] (Common Object Request Broker Architecture).

El presente trabajo presenta la construcción de un prototipo de middleware denominado gnutWare que brinda acceso a una red basada en el modelo compañero a compañero a aplicaciones finales que requieren servicios de comunicaciones. En este caso, como arquitectura de propagación de mensajes se utiliza una red basada en el protocolo Gnutella [BOR01]. Tanto la capa de middleware como las aplicaciones distribuidas utilizadas a modo de ejemplo se encuentran codificadas en lenguaje Java.

3.1 – Arquitectura de gnutWare

En la arquitectura propuesta se definen dos niveles ó capas de software (Figura 3.1) para la implementación de servicios basados en redes compañero a compañero. Una capa que brinda los servicios de red P2P, que toma servicio de la capa de transporte (dado que el modelo opera bajo el juego de protocolos TCP/IP), cuyo objetivo es propagar mensajes y permitir que los usuarios accedan a la red, y una capa de aplicación, que toma servicio de la red P2P, que tiene como objetivo implementar servicios de usuario final tales como sistemas de archivos, búsquedas ó procesamiento distribuido.

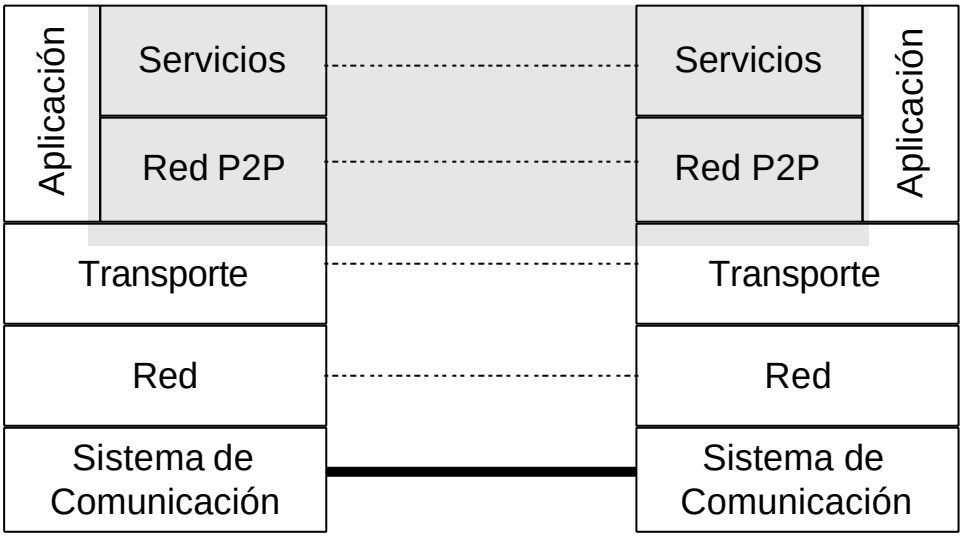


Figura 3.1 – División por capas de la funcionalidad de la red P2P

Este modelo permite separar la funcionalidad de cada capa de manera que se puedan desarrollar aplicaciones que corran sobre una red dada, o bien distintas redes de aplicación (con características propias) que se adecuen de mejor manera a la necesidad del servicio. Por ejemplo podría existir un servicio de mensajería, el cual opere sobre una red de aplicación que implementa mecanismos de protección de privacidad ó sobre otra red que implemente estrategias destinadas a garantizar anonimato. De igual manera una red de aplicación debería soportar distintos servicios, tales como mensajería o computación distribuida.

A partir de dividir esta funcionalidad se puede definir una capa de middleware que permita la interacción entre diferentes aplicaciones y diferentes redes P2P. Este middleware (gnutWare) brinda servicios de acceso a la red P2P a cualquier aplicación que requiera un mecanismo de envío de mensajes para operar en un ambiente distribuido a través de una interfaz normalizada. Una cuestión a considerar es el tipo de servicio que se debe implementar y el mecanismo de envío de mensajes implementado por la red P2P.

3.1.1 – Servicios

En cada nodo participante de la red (nodo compañero), las dos capas de software (el middleware y la aplicación particular) deben proveer un conjunto de servicios ó funciones, de manera tal de poder asegurar el cumplimiento de los objetivos pero manteniendo el principio de división de tareas.

Las funciones a implementar a nivel de middleware persiguen como objetivos:

- Permitir que la aplicación acceda a la red a través de una interfaz normalizada
- Propagar mensajes de terceros (desde la red P2P hacia la aplicación y en sentido inverso)
- Tender a mantener la existencia de la red, obteniendo y brindando información de puntos de acceso a la red alternativos. El middleware puede almacenar en caché información acerca de otros nodos de la red P2P a los cuales acceder en caso de falla
- Mantener conexiones simultaneas con otros pares. Esta función es opcional, pero puede resultar útil de acuerdo al mecanismo de reenvío de la red
- Proporcionar a la capa de servicio (aplicación) información de puntos de acceso alternativos, es decir, de otros módulos de middleware existentes

Para cumplir con estas funciones, el middleware puede analizar los mensajes generados tanto por la red P2P como pos las aplicaciones. Básicamente, el middleware recibirá dos tipos de mensajes:

Mensajes de exploración: Tienen por finalidad el anuncio del ingreso de un nuevo nodo a la red, a los efectos de recopilar información, a través de mensajes respuesta, de otros nodos existentes. Estos mensajes son generados y propagados por la red P2P, siendo de utilidad a los efectos de mantener la estabilidad y conectividad de la misma y poder recuperarse ante caídas de algunos nodos.

Mensajes de servicio: Son generados y respondidos únicamente por los nodos que brindan el servicio, es decir, por aquellos donde corre la aplicación distribuida. Permiten implementar las solicitudes y respuestas de los servicios que se brindan.

A nivel de capa de servicio, la implementación de las funciones requeridas es más sencilla debido a que la complejidad de sistema de comunicaciones está resuelta por el módulo anterior. Dichas funciones están relacionadas con:

- Implementar servicios distribuidos de usuario final
- Comunicarse con las demás aplicaciones pares (a través del middleware y la red P2P)
- Tender a mantener su existencia en la red (obteniendo información de puntos de acceso alternativos). Por ejemplo, la aplicación puede obtener información acerca de varios nodos gnutWare alternativos.
- Mantener conexiones simultaneas con una ó más aplicaciones pares. Esta función es opcional y depende del servicio que se brinda.

Toda comunicación extra necesaria (más allá de los mensajes de consulta y respuesta definidos) se realiza de manera independiente de la red de aplicación, es decir, de manera directa entre dos nodos, ya sea mediante conexiones TCP ó mensajes UDP. Esta característica responde a un criterio de optimización en la prestación de los servicios distribuidos. Como la red es un espacio público de comunicación, solo se permiten mensajes generales y no interacciones directas particulares.

3.1.2 – Nodos gnutWare

El middleware gnutWare debe manejar al menos dos protocolos de comunicaciones,

una para interactuar con la red P2P y otro para hacer lo propio con la aplicación. Básicamente, se encuentra dividido en dos módulos principales (Figura 3.2):

- un módulo Gnodo, que permite conectarse a un nodo de la red Gnutella (Gnodo) a partir de contar la dirección IP y puerto de alguno existente y conocido

- un módulo Network_Sap, que brinda una interfaz para las aplicaciones, posibilitándoles la utilización del servicio

El módulo Gnodo, implementa el protocolo Gnutella a los efectos de conectarse a la red e intercambiar mensajes de exploración (PING/PONG) y de servicio (QUERY/QUERY_HIT), mientras que el módulo Network_Sap, acepta conexiones entrantes de alguna aplicación y luego acepta mensajes de servicio y devuelve las respuestas.

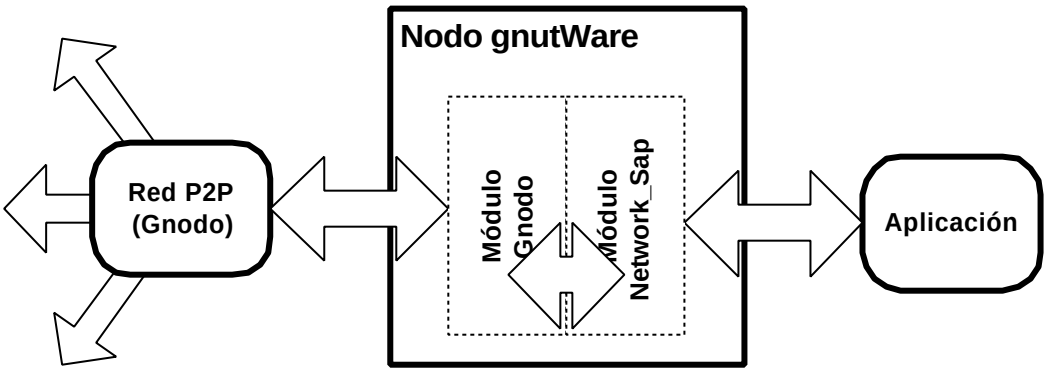


Figura 3.2 – Arquitectura de un nodo gnutWare.

gnutWare se encuentra programado completamente en Java, por lo tanto es portable a cualquier plataforma que soporte la ejecución de la Máquina Virtual Java.

3.2 – Operación

Para ejemplificar el modo de operación de la red de aplicación y su mecanismo de propagación de mensajes, se puede hacer una analogía con el funcionamiento de una red tipo Ethernet. Aquí, cualquier nodo conectado a la red puede emitir un mensaje que alcanzará a todos los destinatarios, y solo aquellos que se hagan eco del mensaje lo procesarán y eventualmente lo contestarán. Esta red de aplicación es un “pasillo virtual”, cuyo ámbito de operación es una intranet ó Internet, y sus receptores todos los nodos participantes de la

misma.

En el prototipo propuesto, la red de aplicación está formada por un conjunto de servents Gnutella (Gnodos), utilizando el software Gnut v.0.4.25 sobre Linux. Esta red P2P opera como una red troncal para permitir prestar diferentes servicios distribuidos, donde una característica importante es la distribución total de las funciones, ya que no existen nodos con características especiales.

El middleware gnutWare brinda servicio a las aplicaciones para acceder a la red de aplicación, la cual sirve como transporte de mensajes genéricos. Los Gnodos se deben configurar para que no posean archivos compartidos en sus sistemas locales, por lo cual se comportan simplemente como ruteadores de mensajes dentro de la red, sin prestar servicio alguno.

Cuando los módulos gnuWare se conectan a algún Gnodo conocido, forman una red de intercambio de mensajes inespecíficos basada en el modelo compañero a compañero, disponible para implementar diferentes servicios distribuidos (Figura 3.3). Inicialmente, el módulo gnutWare se comporta como un Gnodo “normal”, implementando el protocolo según la especificación. Al establecer la conexión con el Gnodo que se encuentra vinculado, a nivel aplicación intercambian las primitivas de protocolo:

GNUTELLA CONNECT\n\n (gnutWare envía al Gnodo)

y

GNUTELLA OK\n\n (el Gnodo responde a gnutWare)

Luego, gnutWare debe enviar un mensaje Gnutella tipo PING, a los efectos de anunciar su presencia en la red y recibirá los correspondientes mensajes tipo PONG. Por protocolo Gnutella, de éstos se extrae información de puntos de acceso alternativos a la red, que pueden servir para mantener conexiones simultaneas con otros Gnodos ó bien, para recuperarse ante la caída de alguno de éstos. Consecuentemente, gnutWare responde (con PONG) mensajes PING provenientes de la red P2P. Si bien esta funcionalidad no tiene demasiado sentido debido a que el middleware no cumple funciones equivalentes a las de un Gnodos (y por ende no puede formar parte de la red P2P troncal), se implementó como un requerimiento de los Gnodos, ya que utilizan esta información como señal de que el proceso en el otro extremo (en este caso el proceso es un módulo gnutWare) se encuentra activo (es una forma de keepalive).

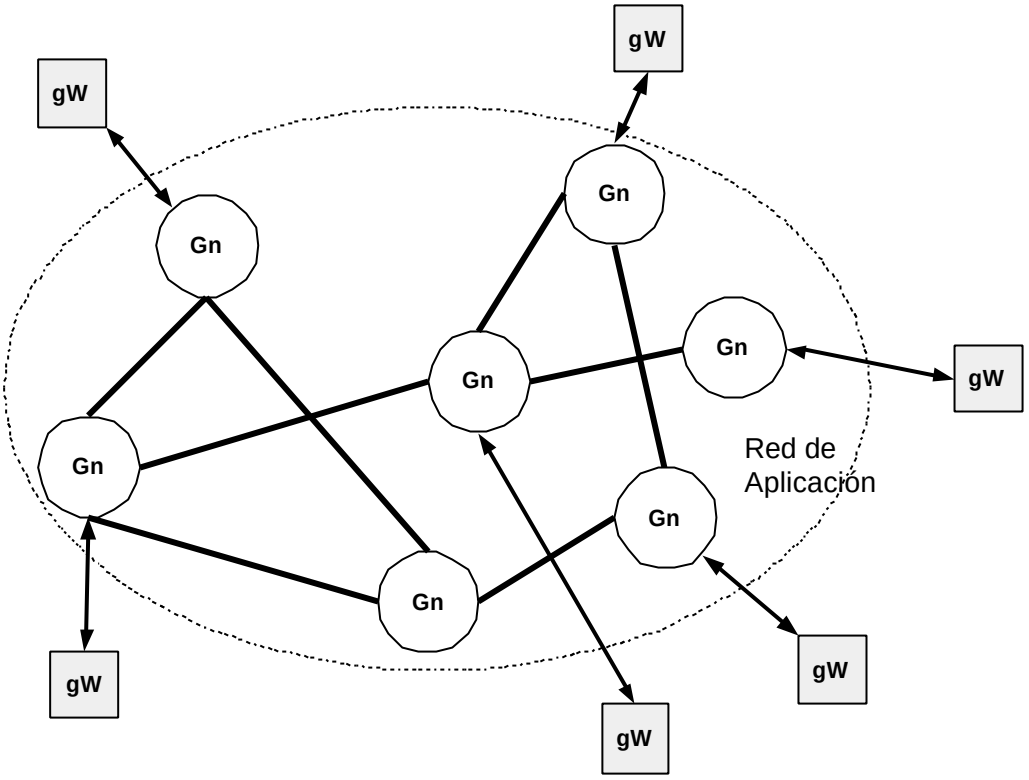


Figura 3.3 – Red inespecífica de intercambio de mensajes (Gn: Gnodos, gW: gnutWare)

Para acceder a dicha red, las aplicaciones debe establecer una conexión TCP con algún módulo gnutWare y ejecutar un simple protocolo de acceso. La aplicación envía una cadena de conexión a nivel aplicación con:

GNUTWARE CONNECT\n\n

Luego, gnutWare solicita a la aplicación que identifique el código de servicio y el identificador de servicio utilizando las primitivas

GNUTWARE SERVICE CODE?\n\n

y

GNUTWARE SERVICE ID?\n\n

en este orden. La aplicación debe contestar con un entero de 2 bytes en el primer caso y con un entero de 4 bytes en el segundo. Estos datos se utilizan para permitir multiplexar a nivel de aplicación diferentes servicios y aplicaciones a los que un mismo módulo gnutWare puede brindar acceso a la red P2P. A partir de finalizar esta etapa de protocolo, la aplicación está en condiciones de enviar solicitudes de servicio y de recibir respuestas.

Cuando un módulo gnutWare recibe una solicitud de servicio, debe armar un mensaje

Gnutella tipo QUERY (manteniendo la estructura de datos del protocolo Gnutella) y enviarlo al Gnodo con el que está vinculado, el cual se encargará de su propagación. Cuando se recibe un mensaje Gnutella tipo QUERY_HIT desde la red P2P, se pasa la respuesta a la aplicación. Las respuestas viajan dentro de la estructura de datos de una QUERY_HIT, como si fuese un mensaje Gnutella típico. Se debe respetar esta estructura ya que los Gnodos de la red de backbone solo propagan mensajes Gnutella bien formados, es decir, aquello que respetan la estructura de datos definida por el protocolo (sin importar su contenido).

A los efectos de validar el comportamiento del middleware y de la red de aplicación propuestos, se codificó una simple aplicación de solicitud de tiempo. Dicha aplicación puede servir como soporte de un mecanismo de sincronización de relojes dentro del sistema.

3.3 – Aplicación ejemplo: Servicio Cooperativo de Tiempos

El servicio cooperativo de tiempos (denominado p2pTime) pretende ser un simple ejemplo de una posible aplicación que puede operar en un ambiente distribuido soportado por el middleware gnutWare y una red P2P (Figura 3.4).

Una aplicación p2pTime requiere de acceso a la red P2P con el objetivo de solicitar los tiempos locales (fecha y hora) de todos los demás nodos p2pTime dentro del sistema. Cuando llegan las solicitudes, todos los módulos p2pTime responden con su tiempo local al solicitante. Nótese que estos módulos no deben preocuparse por determinar quién es el solicitante, ni dónde se encuentra, ya que estas funciones son provistas por la red de aplicación subyacente. Cuando el solicitante recibe todas las respuestas, puede utilizarlas a los efectos de establecer un tiempo global aproximado del sistema ó para sincronizar relojes.

Las aplicaciones se conectan a un módulo gnutWare a través de la interfaz provista por éste, es decir, estableciendo una conexión TCP a un puerto conocido. Cuando una envía una solicitud de tiempo, ésta es propagada hacia todos los demás nodos en la red, los cuales responderán a la misma. Esta estrategia de difusión es propia de la red Gnutella, por lo tanto las respuestas volverán por el mismo camino por el cual se envió la consulta.

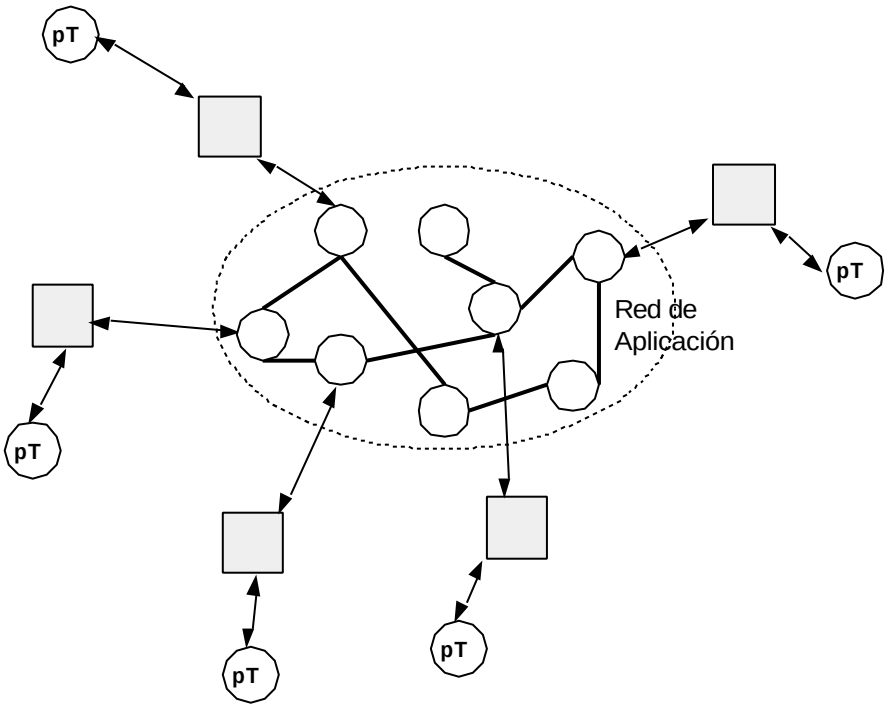


Figura 3.4 – Aplicaciones interactuando con el middleware (pT: p2pTime)

El protocolo de la aplicación es muy sencillo. La primitiva de solicitud incluye dos caracteres que denotan que se trata de una solicitud (QY) y luego la expresión que indica de qué tipo de solicitud se trata, por ejemplo:

QY:TIME, PLEASE!\n\n

De forma similar, la respuesta tiene la misma estructura: dos caracteres que denotan que se trata de una respuesta y luego el tiempo anunciado:

QH:WED 29-11-2002 16:02:11 –300 GMT\n\n

Se pueden recibir tantas respuestas como aplicaciones p2pTime hayan recibido la consulta. Téngase en cuenta que como se trata de una red de transporte de mensajes basada en Gnutella el alcance de las consultas está limitado por el horizonte definido por el valor del campo TTL del encabezado. En la implementación de gnutWare, el TTL es inicializado en 7 pero es decrementado en uno al llegar al primer Gnodo.

Una vez recibido todos los tiempos se puede aplicar un algoritmo de aproximación a los efectos de establecer el tiempo global con el que se va a sincronizar el reloj local. Esta cuestión se puede resolver utilizando los algoritmos típicos de sincronización de relojes utilizados el área

de sistemas distribuidos. Nótese que en esta aplicación ejemplo no se incluye información acerca de tiempos de retardo relativos a la latencia existente en la red al momento de efectuar la consulta, pero esta información puede ser incorporada a la respuesta si el algoritmo de aproximación que se utilice lo requiere.

Finalmente, se puede considerar la posibilidad de que la red a la cual se envían las solicitudes pueden ser un conjunto de servidores de tiempo conocidos.

3.4 – Otros escenarios de aplicación

El middleware gnutWare puede servir como nexo entre una red compañero a compañero y cualquier aplicación que requiera operar de manera distribuida. Se presentan dos escenarios posibles para su aplicación:

El primero, descripto en profundidad en el capítulo final de este trabajo, define un modelo de búsqueda distribuida sobre una comunidad de motores de consulta, donde cada nodo dentro del sistema procesa consultas de manera local y en paralelo la distribuye a sus pares, utilizando la red P2P de recubrimiento.

El segundo servicio está relacionado con el cómputo masivo colaborativo. En este servicio, computadoras de usuario final que poseen recursos ociosos, pueden procesan unidades de trabajo de otros nodos que requieran ciclos de CPU. Tal servicio utilizaría la red de recubrimiento a los efectos de obtener nodos que deseen cooperar aportando sus recursos.

3.5 – Consideraciones

El prototipo de middleware resulta una solución válida a los efectos de brindar un punto de acceso a una red P2P a diferentes aplicaciones que requieren operar en un ambiente distribuido. En este caso, el servicio que se brinda de propagación de mensajes se hace en modo broadcast, ya que la red subyacente se basa en Gnutella.

En esta arquitectura propuesta, se estima que la red de aplicación opera de forma más eficiente, ya que los nodos de los extremos no generan mensajes de exploración. Esta tarea solo la realizan los Gnodos, por lo cual la sobrecarga de la red es menor. Esta situación favorece los límites de escalabilidad encontrados a Gnutella. Como contribución principal de este desarrollo se pueden mencionar:

- Un modelo operativo de utilización de una red basada en el protocolo Gnutella [BOR01] como infraestructura de intercambio de mensajes inespecíficos utilizando la técnica de difusión

- Un modelo de comunicación entre aplicaciones de usuario final, a través de una red de mensajes por difusión

- Una arquitectura sencilla y robusta para implementar servicios distribuidos sobre redes compañero a compañero

- Un prototipo de middleware que presenta una interfaz normalizada a las aplicaciones para acceder a la red de transporte de mensajes

Por otro lado, se debe continuar el estudio de la red Gnutella, a los efectos de lograr mayor escalabilidad y un mejor desempeño. Esta tarea debe orientarse a definir una nueva política de comunicación de los gnodos con sus vecinos y a establecer mejoras en las reglas de reenvío de mensajes.

Además, se puede continuar el desarrollo del prototipo para operar sobre otra arquitectura de red P2P (no solamente basada en Gnutella), a los efectos de proveer servicios diferenciados a las aplicaciones, de acuerdo a las necesidades particulares de cada una.

Capítulo 4

IndiSE: Búsquedas Distribuidas en el Espacio Web

A partir de las características propias del espacio web mencionadas en el Capítulo 2 y de las desventajas que presenta un enfoque centralizado para la construcción de motores de búsqueda, se plantea definir una arquitectura completamente distribuida basada en el modelo compañero a compañero. Para tratar de minimizar estos problemas mencionados, una alternativa a considerar es la posibilidad de cambiar la naturaleza centralizada de los motores de búsqueda tradicionales y plantear modelos donde quien posea la información sea capaz de contestar a una consulta determinada [TOL01]. En éstos, los proveedores de información se convertirían – además – en participantes activos en la tarea de búsqueda de información, operando sobre su propio ámbito ó dominio. Esta modalidad de operación ofrece las siguientes ventajas:

- *Índices y Respuestas actualizados en tiempo real***.** Al ser mantenidos por el proveedor de información, la actualización de los índices será coherente con los cambios que sufra el sitio web. De esta manera, la búsqueda se realizará sobre índices completamente actualizados, que reflejan el estado de los documentos fuente. Además, se puede eliminar la posibilidad de retornar referencias inexistentes ó a páginas cuyo contenido haya variado sustancialmente. En este sentido se establece una ventaja importante respecto de los modelos que operan con recolectores ó los directorios generados manualmente.

- *Indexación de diferentes formatos de información.* Dado que el motor de indexación opera sobre un dominio local de fuentes de información, es posible definir índices sobre distintos formatos de almacenamiento. Si bien los motores de búsqueda actuales indexan formatos alternativos a los archivos tradicionales (HTML y texto plano), como por ejemplo PostScript ó PDF, pueden incorporarse documentos en otros formatos (por ejemplo, de generadores de contenido propietarios), es decir, el mismo productor, que conoce la estructura de su información puede definir cuál es la mejor forma de

obtenerla, generado índices más eficientes.

- *Indexación de la "web oculta".* Como ya se mencionó, existe parte de la web que por naturaleza, estructura y disponibilidad (información almacenada en bases de datos), no siempre es posible indexarla remotamente. El proveedor de la información tiene el control sobre cómo disponer de dicha porción de información para permitir búsquedas sobre ese espacio.

- *Ahorro de recursos.* Debido a que en este modelo no hay actividad de recolección de información (crawling), no se requiere cargar al servidor web y – consecuentemente – se ahorra uso del ancho de banda disponible en la red.

- *Procesamiento distribuido.* Cada proveedor de información es responsable de procesar su propio contenido, de generar sus índices y de responder a las solicitudes. Por lo tanto, no se requieren gran cantidad de recursos comparado con una arquitectura centralizada, donde se utilizan granjas de servidores (servers farm) y dispositivos de almacenamiento masivo.

Otros beneficios que se obtienen con esta modalidad de trabajo están relacionados con la flexibilidad y el control que se puede ejercer sobre el espacio de publicación propio (por ejemplo, se puede definir muy fácilmente qué parte se indexa) definiendo diferentes permisos de acceso. Además, no existe la necesidad de "anunciar" a los diferentes motores de búsqueda de la existencia de nuevo contendido para que lo recolecten y procesen.

Por otra parte, hay que considerar una serie de posibles problemas que pueden existir en este modelo debido a su naturaleza distribuida. Aquellos que poseen información y "necesitan ser encontrados" pueden contestar de manera indebida (sin poseer información relevante) a una consulta con el objeto de que el usuario llegue a ellos (Por ejemplo, para mostrar publicidad ó como una modalidad de ataque al servicio). Además, el hecho de que no exista un mecanismo centralizado la tarea de clasificación y rankeo de las respuestas se vuelve compleja sin no se opera bajo criterios homogéneos y estandarizados.

En este capítulo se presenta IndiSE, un modelo de motor de búsquedas con las características mencionadas, que opera entre nodos de una red compañero a compañero, a través del middleware gnutWare (presentado en el capítulo anterior).

4.1 – Arquitectura de IndiSE

IndiSE define una arquitectura para un servicio de búsqueda distribuida, como alternativa a los motores de búsqueda tradicionales. La diferencia radica en cómo se recolecta la información (en realidad, no hay recolección de información de la forma tradicional conocida) y dónde se ejecutan las búsquedas. Básicamente, se reemplaza una base de datos centralizada de índices por un conjunto de bases de datos distribuidas, situadas donde reside la información (ó cerca de ésta). De esta manera se elimina la necesidad de utilizar un módulo recolector (crawler) para recorrer los sitios, evitando – como se mencionó antes – el problema de la frecuencia de actualización. Se obtiene un servicio de búsquedas en tiempo real, es decir, sobre índices completamente actualizados.

La base para soportar esta arquitectura cooperativa es el middleware gnutWare que brinda acceso a las aplicaciones a una red compañero a compañero, en este caso basada en el protocolo Gnutella. Esta red brinda flexibilidad en cuanto a la dinámica de acceso y permanencia en la red, adaptándose a constantes cambios en su topología. Si bien adolece de problemas de eficiencia, según el modelo planteado, se pueden introducir mejoras solamente en la red de aplicación sin modificar la arquitectura del prototipo IndiSE.

La arquitectura general del sistema se muestra en la figura 4.1. Es ésta se puede observar cómo se relacionan los nodos que pueden proveer servicios de búsquedas con la red P2P (a través del módulo gnutWare) y con los usuarios (a través de una interfaz web normalizada).

Los Nodos de Búsqueda Distribuida son los que soportan el sistema para una organización determinada ó para un usuario particular que requiera participar proveyendo información al mismo. El espacio de búsqueda puede estar formado por un conjunto de nodos de organizaciones afines, formando una red de búsqueda corporativa. Una variante posible es vincular nodos independientes que respondan a un interés temático particular, por ejemplo una comunidad de desarrolladores de software ó un grupo de usuarios de un determinado sistema operativo.

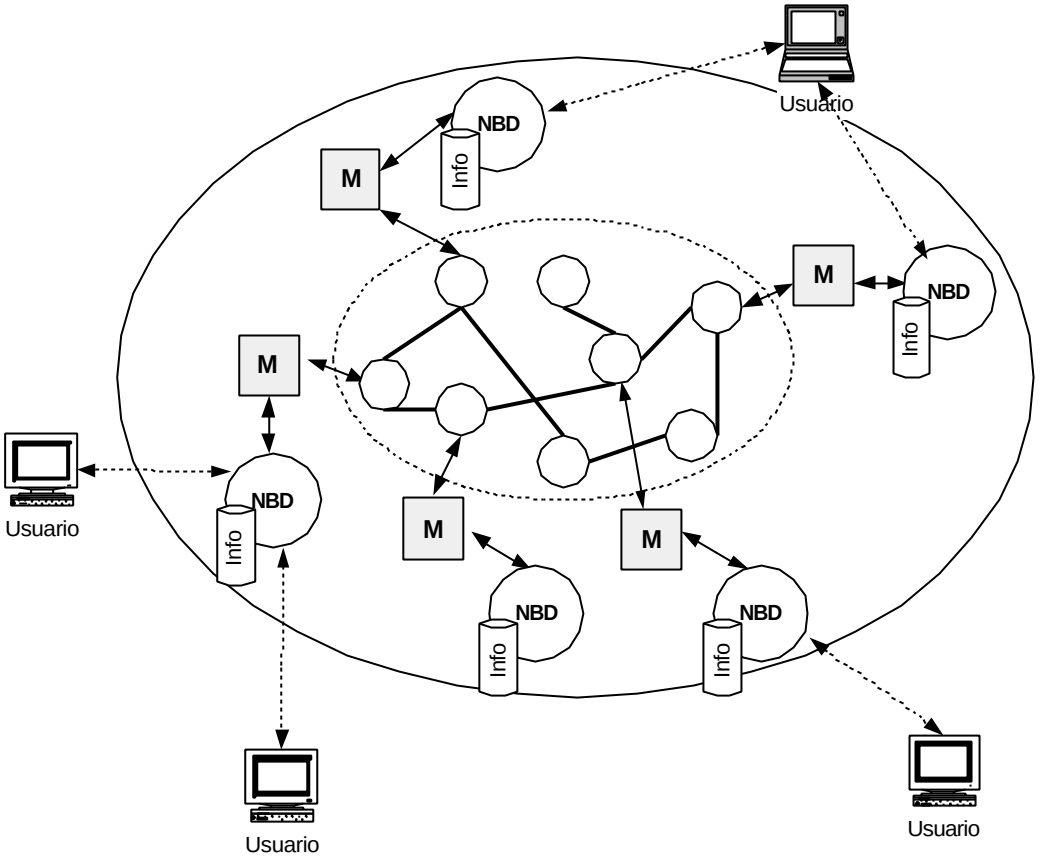


Figura 4.1 – Arquitectura general del sistema
(NBD – Nodo de Búsqueda Distribuida, M – Middleware)

4.1.1 – Nodos de Búsqueda Distribuida

Los nodos de búsqueda distribuida brindan cuatro funciones básicas que permiten que los usuarios interactuen con el sistema. Dichas funciones son:

a) Interfaz de usuario
b) Búsquedas sobre el repositorio (índice) local
c) Búsquedas sobre la red de nodos pares
d) Integración y visualización de resultados

Estas funciones se encuentran implementada en los módulos que forman la arquitectura de los nodos de búsqueda distribuida IndiSE. La misma se muestra en la figura 4.2 y cuenta con los siguientes componentes: Módulo Gerente, Módulo de Búsqueda Local (MBL), Motor de Indexación y Búsquedas genérico (MIB) y Servidor HTTP estándar.

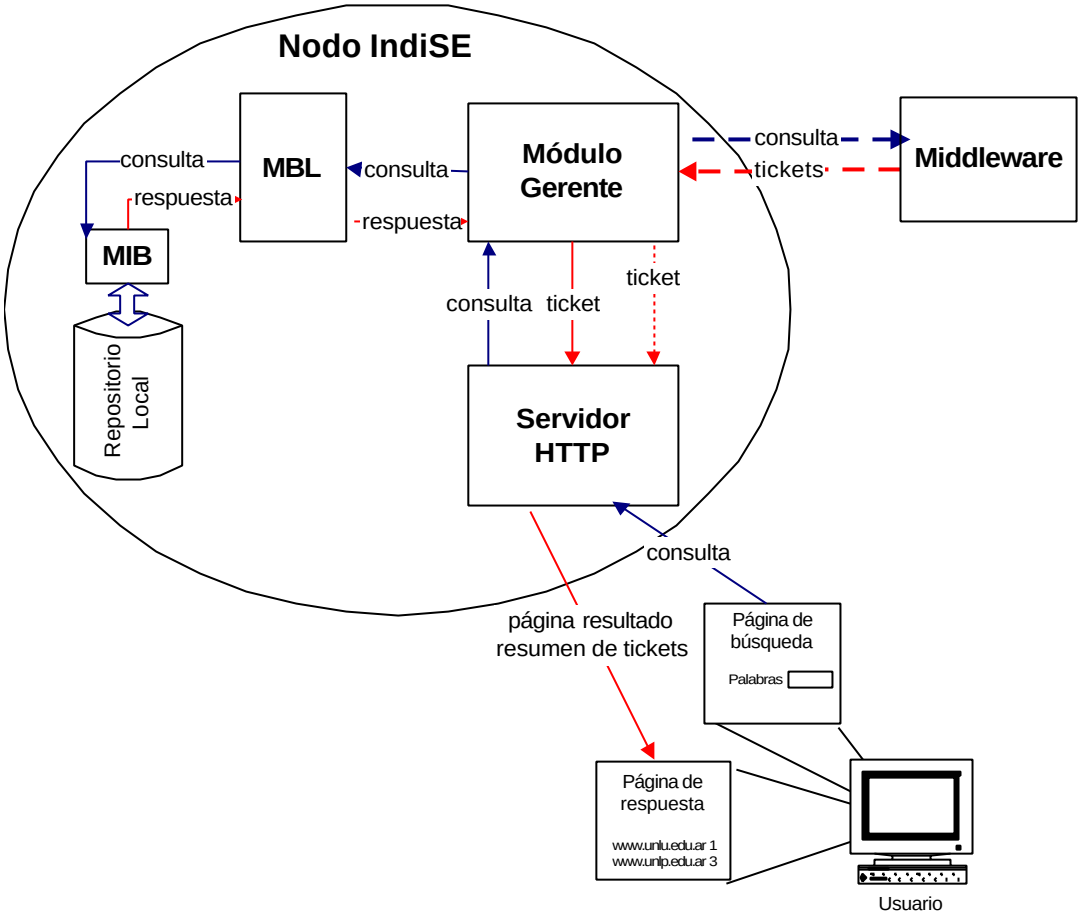


Figura 4.2 – Arquitectura de un nodo de búsqueda distribuida IndiSE

Módulo Gerente: es el encargado de integrar las funciones de todos los demás componentes. Maneja las comunicaciones con el middleware y con los usuarios (a través de una interfaz web), también se encarga de interactuar con el Módulo de Búsqueda Local, ya sea por una solicitud proveniente de un usuario local ó bien, proveniente de la red P2P (a través del middleware). Cuando el Módulo Gerente recibe una consulta, arma una estructura de datos en XML [WWW98] denominada ticket, que es almacenada en un directorio público del servidor HTTP. Como resultado de la búsqueda se entrega solo un identificador único del ticket, a los efectos de minimizar el tráfico por la red P2P.

Motor de Indexación y Búsquedas: Esta arquitectura no define la utilización de un MIB específico sino que permite operar con motores de indexación y búsqueda estándares, ampliamente utilizados para proveer servicios de búsquedas sobre sitios web. Ejemplos típicos de estos productos son Ht://Dig [HTD03], SWISH-E [SIM03], Ksearch [KSE03] ó SiteServer [MSS03]. Esto es posible dado que la interfaz con el MIB es provista por el módulo MBL. Otra posibilidad es que el MIB se construya a partir de un motor de bases de datos y un formato predefinido de respuestas.

Módulo de Búsqueda Local: Su función principal es la de brindar una interfaz normalizada al módulo gerente para acceder al repositorio de datos, a través del motor de indexación y búsquedas que opera en el nodo. Este componente acepta una expresión de consulta y la adapta al formato propio del MIB. Luego realiza una invocación a éste (con la consulta como parámetro) y espera la respuesta, que luego retornará al módulo gerente. Nótese que este componente debe ser modificado para operar con diferentes MIBs, teniendo que adaptar la sintaxis de la consulta éstos. En este sentido debe reconocer expresiones de consulta que incluyan operadores lógicos, frases y demás. En el caso que el MIB sea un motor de base de datos, el MBL debe conocer el esquema de la base de datos para poder adaptar la expresión de consulta a una expresión SQL.

Servidor HTTP: El servidor HTTP brinda acceso al servicio de búsqueda distribuida a través de una interfaz web que se encuentra en una URL normalizada (por ejemplo, http://www.unlu.edu.ar/ds), la cual presenta un formulario de ingreso de la expresión de búsqueda. Además, debido a que las respuestas se encuentran almacenadas en tickets, este servidor los entregará al Módulo Gerente que se lo solicite. La captura de la expresión de consulta, su envío al Módulo Gerente y la recepción de las respuestas se realiza a través de una pequeña aplicación que opera por solicitud del servidor HTTP, a instancias del usuario.

4.1.2 – Modo de Operación

Cuando un Módulo Gerente de IndiSE comienza su ejecución, se conecta a un middleware gnutWare cuya dirección IP y número de puerto recibe como parámetros (el resto de los parámetros de ejecución los toma de un archivo de configuración).

A continuación, se disparan dos hilos de control que manejan la entrada y salida con el middleware. Esta tarea se realiza de manera asincrónica, es decir, el tiempo y la secuencia de mensajes entrantes y salientes no está preestablecida. Cuando llega un mensaje es puesto en una cola de entrada, la cual es consultada regularmente por el proceso principal. Por otro lado, cuando el proceso principal requiere enviar un mensaje a la red, lo introduce en una cola de salida. Esta cola es manejada por un hilo de control que periódicamente chequea la existencia de mensajes y – si no se encuentra vacía – toma cada mensaje y lo envía.

Luego se dispara un tercer hilo de control que maneja las comunicaciones con la interfaz de usuario. Esta tarea se realiza de manera sincrónica, ya que las interacciones están limitadas y preestablecidas. A partir de este estado, el Módulo Gerente entra en el ciclo principal de operación donde interpreta los mensajes recibidos y genera solicitudes propias (ya

sea con la red P2P, como con el MBL).

Cuando se recibe un mensaje proveniente de la red P2P, se analiza si se trata de una solicitud de búsqueda local ó bien de una respuesta a un requerimiento anterior. Si el mensaje es una consulta se crea un objeto LocalSearch, el cual entrega la solicitud al Módulo de Búsqueda Local. El MBL normaliza la expresión de consulta e invoca al MIB correspondiente. La respuesta es devuelta bajo la estructura de un documento XML que contiene una descripción del nodo servidor y la lista de resultados. La definición del tipo de documento XML (DTD – Document Type Definition) se muestra en la figura 4.3. Con esta respuesta se genera un ticket que es almacenado en un directorio público del servidor HTTP (para permitir su recuperación). Con el identificador único del ticket y los datos del servidor HTTP que los almacena (dirección IP y puerto) se arma un mensaje de respuesta que es entregado al middleware para que lo inyecte en la red y se envía al solicitante. La respuesta se completa con un indicador lógico de número de solicitud (que se recibió con la consulta) para permitir la multiplexación de diferentes solicitudes dentro del sistema.

```
<!DOCTIPE ticket [
<!ELEMENT Server (Name, Description)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT ResultSet (Item)+>
<!ELEMENT Item (Url, Title)>
<!ELEMENT Url (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
]>
```

Figura 4.3 – DTD del ticket de respuesta.

Por otro lado, si el mensaje recibido es una respuesta debe existir un usuario aguardando por la misma (esto lo indica el valor de multiplexación, que identifica una conexión con un cliente), de otra manera se descarta. De la respuesta se extrae el identificador de ticket junto con la dirección IP y puerto del servidor HTTP. Con esta información se lo recupera.

Una vez recuperado el ticket, se realiza un parsing del documento XML para convertirlo en la respuesta para el usuario, la cual debe entregarse en formato HTML. Esta tarea se realiza utilizando un parser denominado MinML [MIN03]. Este es un pequeño parser XML pensado para sistemas embebidos que corran Java, ya que utiliza muy pocos recursos del sistema. Se utilizó este componente en la construcción del prototipo (por ser pequeño y simple) pero para un sistema en un ámbito de producción se recomienda utilizar un parser más completo ya que se requieren otras funciones de las cuales MinML carece. Por ejemplo, ignora las definiciones de un archivo DTD, por lo que no se puede determinar si un ticket está bien formado y tampoco

permite generar un archivo XML de salida.

Esta tarea se realiza por cada respuesta recibida, es decir, se procesarán tantos tickets como nodos hayan respondido a la consulta original. Una vez realizado el parsing, se retornan todas las respuestas de acuerdo al orden consignado el cada ticket. Un diagrama simplificado de operación del Módulo Gerente se muestra en la figura 4.4.

Debido que las respuestas pueden provenir de diferentes motores de búsqueda, los cuales pueden utilizar diferentes estrategias para clasificar y rankear, no es posible consolidar todas las respuestas en una sola de acuerdo aun criterio común para entregar al usuario. En tal sentido, se optó por devolver al principio de la página HTML de respuesta las diez primeras referencias de cada nodo que contestó (si hay), dejando las demás al final. Este criterio tiene que ver con la posibilidad de que el usuario pueda acceder primero a los resultados más relevantes de los diferentes nodos. Si bien se estableció un umbral de diez referencias, este valor puede ser ajustado por parámetro del sistema ó – inclusive – por el mismo usuario que realiza la consulta.

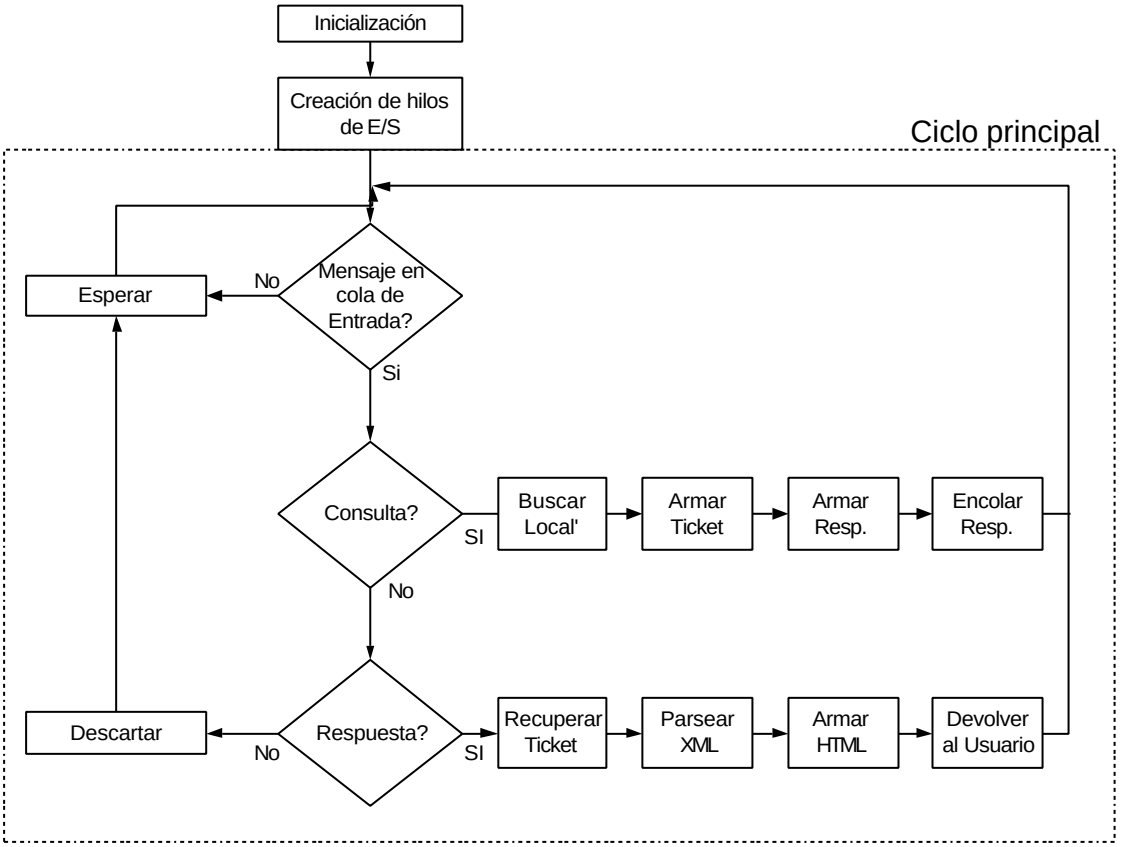


Figura 4.4 – Diagrama simplificado de operación del Módulo Gerente

Finalmente, cabe aclarar que debido a la independencia de cada uno de los nodos y al existir control centralizado de recursos, las respuestas a una misma consulta pueden variar en el tiempo según la disponibilidad de las fuentes de información en la red y la dinámica propia de actualización de cada uno de los participantes de la misma.

4.1.3 – Integración de los componentes

Si bien todos los componentes pueden operar en un mismo nodo, esto no es una condición para su funcionamiento. Cada uno de los mismos se encuentra programado de manera independiente y se comunican entre sí a través de conexiones TCP [TCP81], utilizando la interfaz de sockets [STE98] como API de programación.

El Módulo Gerente se encuentra codificado en Java, lo que permite su portabilidad a diferentes sistemas operativos. La única restricción es que este módulo debe operar en el mismo equipo donde se encuentra el servidor HTTP. Esto se debe a que el Módulo Gerente debe guardar los tickets en un directorio público del mismo y, por simplicidad, se dejó esta restricción. En esta cuestión se podría haber anexado al sistema un módulo que opere de servidor HTTP, pero se prefirió utilizar el software Apache [APA03] debido a su amplia difusión y a que es un producto dedicado a tal tarea.

Los parámetros del sistema se pueden ajustar mediante la edición de un archivo de configuración (IndiSE.conf). A continuación se muestran algunos de las variables del sistema presentes dentro del mismo:

Parámetros referidos a Motor de Búsqueda Local:

```
localSearchIP =192.168.1.10
localSearchPort=4000
maxLocalSearchWait=15
```

Los dos primeros corresponden a la dirección IP y número de puerto donde opera el MBL. Nótese que el mismo puede operar en un equipo diferente al nodo IndiSE, por ejemplo, en un servidor de búsquedas dedicado a tal fin. El último parámetro corresponde al tiempo máximo (en segundos) que se puede esperar por una búsqueda sobre la base de datos local. Este parámetro se debe ajustar de acuerdo a las prestaciones del motor de búsqueda y al tamaño del índice.

Parámetros referidos a Servidor HTTP:

```
webServerIP=192.168.1.10
webServerPort=80
webServerRoot=/www/apache/htdocs
webServerDir=/ds/tickets/
```

Del servidor HTTP se deben configurar dirección IP y número de puerto TCP donde opera y además, los caminos absolutos dentro del sistema de archivos local al directorio principal y al directorio donde se almacenan los tickets.

Parámetros referidos a la interfaz de búsqueda externa y la red P2P

```
searchInterfacePort=4041
maxp2pNetWait=20
```

El primer valor corresponde al número de puerto donde espera conexiones entrantes la interfaz con ds.cgi, mientras que el segundo es el tiempo máximo (en segundos) que se puede esperar por respuestas a una búsqueda sobre la red P2P. Se recomienda que este parámetro se ajuste de acuerdo a la latencia en la red y a las prestaciones del sistema (por ejemplo, este valor no puede ser menor que el parámetro `maxLocalSearchWait`, ya que se “perderían” los resultados locales).

Como los usuarios interactuan con el sistema mediante una interfaz web, se utiliza el mismo servidor HTTP para servir el formulario de ingreso de la expresión de consulta. La consulta es atendida por un script codificado en lenguaje Perl [PER03], denominado ds.cgi, el cual establece una conexión TCP con el Módulo Gerente de IndiSE, le envía el requerimiento y espera por las respuestas.

El Módulo de Búsqueda Local también se encuentra codificado en lenguaje Perl (lo que permite portarlo a diferentes sistemas operativos con modificaciones mínimas). Este módulo es el que brinda la vinculación con el MIB local, por lo que se debe modificar de acuerdo a la interfaz que provea el mismo. Funcionalmente, acepta una conexión en un puerto TCP, normaliza la expresión (de acuerdo a la sintaxis de cada MIB) y realiza una invocación al mismo. Luego, genera la respuesta utilizando el lenguaje de marcas XML y cierra la conexión.

Cabe aclarar que se utiliza como transporte el protocolo TCP debido a que los componentes pueden operar en equipos diferentes y requieren de conexiones confiables. Si todos los componentes operan dentro del mismo entorno (ó en un ambiente confiable) se puede utilizar el protocolo UDP para aumentar la eficiencia en las comunicaciones entre los procesos.

4.2 – Trabajo Experimental

A los efectos de validar la arquitectura propuesta se construyó una experiencia de laboratorio, probando el prototipo IndiSE en un ambiente operativo. Se montaron 10 nodos en una red de área local con el objetivo de validar su funcionamiento. Cada nodo corresponde a una equipo tipo PC (equipos de usuario final, no servidor) donde corría un Gnodo, el middleware, un MIB y – opcionalmente – un motor de base de datos.

Se utilizaron como MIBs los motores de indexación y búsqueda Swish-E y Ksearch y – además – se construyó una interfaz con el motor de base de datos MySQL [MYS03] y una base de datos de ejemplo a los efectos de simular un sitio web dinámico (que normalmente no es indexado por los motores de búsqueda convencionales). Aquí se utilizó la capacidad de indexación y búsquedas en texto completo (full text search) del motor de base de datos para entregar resultados a partir de una consulta SQL. Para esta última configuración se adaptó la aplicación LocalSearch.pl para construir una expresión SQL a partir de una consulta recibida desde el Módulo Gerente de IndiSE y luego generar una estructura de ticket válido con las tuplas de respuesta.

Cada nodo fue provisto de contenido diferente, proveniente de documentos de varios sitios web. Aquí, la intención fue simular distintos sitios web cuyo contenido varía en el tiempo. La base de datos se creó con tablas relativas a productos de un almacén, con atributos textuales.

Las pruebas realizadas consistieron en ejecutar diferentes consultas sobre los distintos nodos IndiSE y analizar los resultados. Se notó que hubo una disparidad en cuanto a los tiempos de respuesta de los diferentes equipos, por lo que se debió ajustar el parámetro `maxLocalSearchWait` a un valor de 15 segundos (en ningún caso, los equipos utilizados tienen características de servidor), mientras que el parámetro `maxp2pNetWait` se ajustó a 20 segundos, para lograr recibir la totalidad de respuestas desde la red. Esta situación se puede mejorar incorporando al sistema una estrategia de buffering para las respuestas, de manera de mostrar las primeras respuestas al usuario mientras se continúan recibiendo otras.

Aleatoriamente, se quitaron de servicio nodos de la red, viéndose los resultados automáticamente reflejados cuando se realizaba una nueva consulta. En este sentido resultó interesante ver los resultados entregados por el nodo que ejecutaba el motor de base de datos ya que el agregado de nuevos registros a las tablas redundaba en la modificación de las respuestas entregadas.

Cabe aclarar que cada vez que se cambió el contenido de los diferentes nodos se

requirió la ejecución del proceso de reconstrucción de los índices, para que se reflejen los cambios. Esta tarea no fue necesaria en el caso de obtener los datos directamente desde la base de datos. Comparado con un motor de búsqueda tradicional, donde los tiempos de reconstrucción total de sus índices varía – en promedio – entre 2 y 6 semanas [INK02], este modelo acelera el proceso de manera notoria. La tabla 4.1 muestra una tabla con los tiempos aproximados de actualización de los índices de los motores de búsqueda más utilizados [WEB03].

Yahoo	4-6 semanas
Google	2-5 semanas
AltaVista	4-6 semanas
Fast / All The Web	9-12 días
Direct Hit	8-10 semanas
Netscape	2-3 semanas
Inktomi	1-3 semanas
HotBot / AOL	1-3 semanas
Open Directory	2 semanas a meses
Looksmart / MSN	7 días (aprox.)
Lycos	6-8 semanas

Tabla 4.1 – Tiempos Aproximados de Actualización de los Indices

En IndiSE la reconstrucción del índice se debe realizar cuando los cambios al sitio web lo justifiquen (cuestión que depende del administrador del mismo), con lo que se podría plantear que el tiempo de actualización es 0 (cero) ó despreciable.

Desde el punto de vista del mantenimiento del sistema, la tarea es muy simple, ya que solo cuenta la reconstrucción del índice (no hay recolección externa). Se puede pensar en un mecanismo automático de reindexado, donde el tiempo entre ejecuciones debe adecuarse a cada caso particular.

4.3 – Escenarios de uso

La arquitectura IndiSE puede servir para comunidades de usuarios que compartan temas ó un ambiente en común. Se plantea que el modelo opere sobre comunidades y no sobre todo el espacio web, debido a la gran escala y a la cantidad de proveedores de información existentes. A continuación, se presenta un ejemplo correspondiente a nodos de organizaciones afines.

Se propone formar un espacio de búsqueda distribuida con las organizaciones integrantes de la Red de Interconexión Universitaria Argentina (RIU) (Figura 4.5). Cada subconjunto representa una Universidad con sus nodos de acceso a la red para búsqueda

distribuida (uno ó varios). Además, se muestran los usuarios del servicio, ya sea internos o externos al sistema.

Cuando un usuario realiza una búsqueda a través del sistema, tendrá la posibilidad de que la misma se resuelva sobre los contenidos de todas las Universidades participantes, por lo que podrá recuperar los enlaces a los diferentes documentos de todas los repositorios de acuerdo a la expresión de consulta.

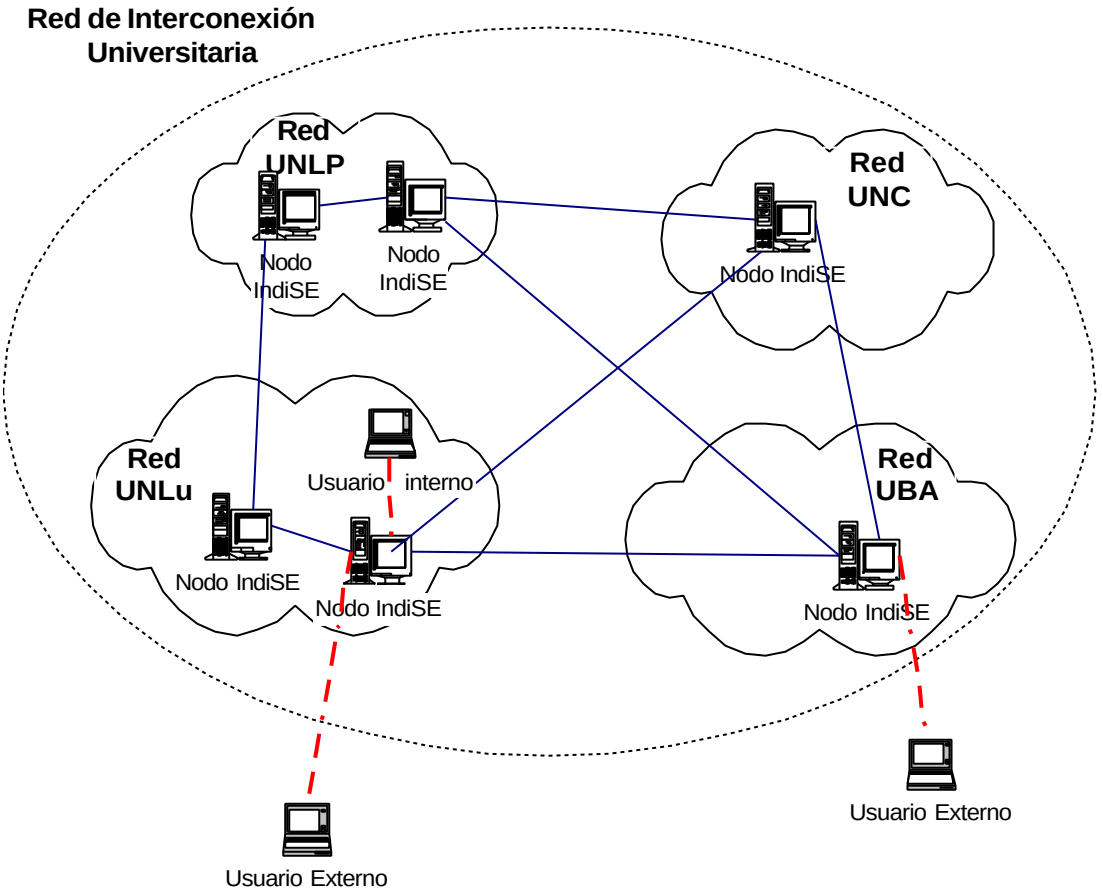


Gráfico 1 – Ejemplo de espacio de búsqueda distribuida RIU

De la misma manera, los participantes de espacio de búsquedas distribuidas pueden ser usuarios con temas afines ó bien, todas las dependencias de una misma organización, distribuidas geográficamente y vinculadas a través de una intranet ó de Internet.

4.4 – Trabajos relacionados

4.4.1 – YouSearch

Un proyecto reciente, de fundamentos y características similares a IndiSE, es el presentado en [BAW03] denominado YouSearch. En éste se plantea un sistema compañero a compañero híbrido que permite búsquedas distribuidas sobre un conjunto de servidores web personales de una organización (en este caso se trata de la intranet de la compañía IBM [IBM03]). Se fundamenta en la existencia de contenido en los equipos personales de los empleados de una organización, el cual está pobremente organizado (lo que dificulta la navegación), cambia constantemente y se almacena en diferentes formatos (no necesariamente en formato HTML).

La arquitectura del sistema se basa en tres componentes: a) Nodos compañeros, que ejecutan el software Yousearch, b) Navegadores (browsers), que brindan la interfaz de usuario final y c) un Servicio de Registro, en el cual los nodos anuncian un resumen de su contenido

En cada nodo se mantiene un índice altamente actualizado. Con esta información, se construye un resumen utilizando filtros de Bloom [BLO70] que son enviados al Servicio de Registro. Cuando un usuario realiza una consulta, se envía un mensaje al Servicio de Registro el cual informará cuáles nodos dentro del sistema posee documentos relevantes de acuerdo a la misma. Luego, cada uno de estos nodos es consultado directamente a los efectos de relevar su contenido. Este mecanismo es utilizado para evitar enviar la consulta a todos los nodos, pero genera un punto centralizado en el sistema (que puede resultar en un punto de falla). Además, para aumentar la eficiencia, se utilizan estrategias de caché de consultas asociadas a las direcciones de los nodos entregadas por el Servicio de Registro. El sistema también permite formar grupos de usuarios (en comunidades), de manera que luego se pueden restringir las consultas a alguno de éstos.

De los estudios realizados en un ambiente real (desde el 16 de septiembre de 2002), con más de 1500 usuarios en 43 países (todos pertenecientes a la compañía IBM), se concluyó que el sistema opera correctamente y satisface las necesidades de los usuarios.

4.4.2 – PlanetP

Otro proyecto que tiene por objetivo resolver el problema de la recuperación de información en comunidades formadas bajo el modelo compañero a compañero es el presentado en [CUE02], denominado PlanetP. La idea principal es proveer un servicio de intercambio de archivos a comunidades de usuarios, sin la necesidad de un servidor central, soportando las funciones de indexación, búsquedas y recuperación sobre el contenido de un conjunto de documentos.

El sistema sigue la siguiente secuencia de operaciones: crear un índice invertido de los documentos que cada usuario desea compartir, sumarizar el índice de una forma compacta y difundirlo al resto de la comunidad. Cada documento es descripto por un “recorte” XML, que contiene texto del cual se extraen las palabras para indexar y referencias a documentos externos. Si el documento a compartir no es XML, se debe anexar un recorte XML con su descripción y un apuntador.

Para realizar el sumario del índice de cada nodo, PlanetP utiliza filtros de Bloom [BLO70], que difunde a la comunidad. Luego, cada nodo puede realizar una consulta a la comunidad a partir de los filtros de Bloom que ha recibido. A partir de éstos, se determinan cuáles nodos tienen información relevante a la consulta, se los rankea calculando una medida denominada IPF (Inverse Peer Frequency) y se decide la cantidad de nodos a consultar. Finalmente se rankean los documentos retornados. La cantidad (y el ranking, que se calcula de acuerdo al modelo vectorial [SAL75]) de los documentos retornados se determinan mediante una heurística de corte.

De la simulación muestra que PlanetP puede escalar a tamaños de comunidades de varios miles. Si bien la principal desventaja es que la información nueva ó cambiante se disemina lentamente (para no consumir demasiado ancho de banda), se puede ajustar la frecuencia de diseminación de los filtros de Bloom para adaptarla a diferentes situaciones.

PlanetP puede servir para compañías que comparten grandes bases de datos, ó repositorios de documentos de investigación ó legales. También, se puede pensar en compartir información estadística de jugadores y equipos. Respecto de archivos en formatos no textuales (como música y películas) aún no se sabe como manejarlas, a menos que se utilicen datos textuales asociados, lo que permitiría la búsqueda y recuperación por contenido.

4.5 – Consideraciones

Como arquitectura para soportar búsquedas distribuidas, el prototipo IndiSE brinda a sus usuarios búsquedas sobre índices completamente actualizados, ya que la diferencia mayor respecto de los motores de búsqueda tradicionales radica en el lugar donde se realiza la búsqueda. También permite un mayor control a los proveedores de información, posibilitando que los usuarios obtengan respuestas provenientes de formatos y fuentes heterogéneas.

Entre las ventajas de este modelo se puede destacar que este mecanismo de búsqueda facilita a los administradores de los sitios web participantes más control sobre el

contenido que sus visitantes reciben como resultados de una búsqueda. Además, se puede adecuar el contenido de acuerdo a lo que los usuarios buscan habitualmente.

Desde el punto de vista del diseño, los principales puntos a favor están dados por su modularidad (con la posibilidad de correr en equipos especializados) y la posibilidad de utilizar el motor de búsqueda que la organización ó el usuario considere más conveniente.

Además, el ingreso ó salida de un participante del sistema no afecta en ningún sentido la operación del resto.

Como punto en contra se puede mencionar que al darle la posibilidad de responder a una consulta a los propios motores de búsqueda de los sitios web se pierde el control del ranking, ya que no se puede asegurar que todos utilicen criterios homogéneos de clasificación. Por otro lado, puede prestarse para enviar información no solicitada (una forma de spam), donde algunos motores (que quieren “ser encontrados”) retornen información no relevante a la consulta, por ejemplo, a los efectos de introducir avisos publicitarios. Esto debería ser autoregulado por los propios usuarios mediante un sistema de reputación.

Para continuar este desarrollo, se debería escribir las interfaces adecuadas que permitan soportar otros motores de búsqueda tradicionales, ya que solamente se escribieron interfaces para los antes mencionados. También se debería analizar la posibilidad de operar bajo un esquema homogéneo de rankeo para permitir la fusión de los resultados provenientes de todos los nodos. Finalmente, se debería definir un mecanismo para que los usuarios puedan incluir ó excluir sitios donde realizar la búsqueda.

Todo el proyecto IndiSE se encuentra distribuido bajo Dominio Público y se encuentra disponible junto con el proyecto gnutWare en la comunidad de Open Source SourceForge [SOU03] en la URL http://gnutWare.sourceforge.net.

5 – Conclusiones

La arquitectura IndiSE presentada se considera una alternativa válida a los motores de búsqueda tradicionales. La diferencia mayor respecto de los anteriores radica en dónde se realiza la búsqueda.

Sus principales beneficios están relacionados con la actualidad de la información recuperada (reduce la posibilidad de encontrar enlaces rotos), el acceso a fuentes normalmente no indexables (por ejemplo, bases de datos) y la utilización eficiente de recursos (no hay recolección externa). Opera con índices altamente actualizados ya que el tiempo de reindexación se puede considerar despreciable (ó cero) respecto a los motores de búsqueda tradicionales (en los cuales dicho tiempo que varía entre 2 y 6 semanas).

La experiencia muestra que el tiempo de respuesta del sistema es alto (15–20 segundos) respecto a los motores de búsqueda tradicionales. Esta situación se la atribuye a que existe una restricción de performance en la búsqueda en los índices locales por tratarse de equipos de usuario final y no equipos dedicados a servidor.

Respecto del diseño de IndiSE, se intenta brindar un mayor control a los proveedores de información, tanto para definir cómo tratar la información que poseen (en cuanto a fuentes, formatos y accesos), qué herramientas utilizar (ya que cada sitio web puede utilizar el motor de indexación y búsqueda que mejor satisfaga sus necesidades) y cómo adecuar el contenido de acuerdo a lo que los usuarios requieren en sus búsquedas habituales (esta información resulta extremadamente útil para mejorar los servicios prestados a través de la web).

Por estar basada en el modelo compañero a compañero plantea una forma diferente de intercambio de información y de interacción entre los nodos participante, que brinda una serie de ventajas tales como: robustez, transparencia (ante cambio de direcciones ó equipos), concurrencia de la operación de consulta y alta disponibilidad. Si bien el prototipo actual opera con una red basada en el protocolo Gnutella como infraestructura de comunicaciones (el cual adolece de problemas de escalabilidad), la arquitectura es abierta y permite que se pueda mejorar ó reemplazar la red troncal para mejorar sus prestaciones.

El middleware presentado brinda una interfaz de acceso a una red P2P sencilla y robusta para implementar servicios distribuidos sobre redes compañero a compañero. Los nodos de la red Gnutella solamente realizan tareas de encaminamiento sin prestar servicios de usuario final, mientras que la aplicación brinda un servicio de usuario final totalmente

independiente de ésta.

Se sugiere que en el estado actual, IndiSE se utilice para comunidades de usuarios ó de organizaciones con intereses comunes, a los efectos de servir como plataforma de búsquedas sobre un repositorio de información distribuida. Operar sobre comunidades facilita el control de posibles abusos provenientes de los proveedores de información mal intencionados ya que aún no se cuenta con un sistema que califique la confiabilidad de cada nodo participante.

También hay que mencionar que el sistema no provee un mecanismo de fusión y rankeo de las respuestas provenientes de diferentes nodos (en el esquema actual, no se puede asegurar que todos los nodos utilicen criterios homogéneos de clasificación).

El desarrollo de IndiSE deja abiertas varias posibilidades para realizar futuros trabajos a los efectos de mejorar el prototipo y brindar mejores prestaciones. En primer lugar, se debe estudiar el comportamiento del sistema en un ambiente real. Para esta tarea, se requiere instalar nodos distribuidos vinculados por una red de área amplia, donde se pueda medir el impacto de la carga de la red en cuanto a la latencia y congestión. Una posibilidad planteada es instalar nodos especializados en las dependencias de la Universidad Nacional de Luján, en Sede Central (Luján) y Centros Regionales (Chivilcoy, Campana y San Miguel), lo que brindaría un ambiente real de pruebas y evaluación.

En cuanto a la continuidad de la investigación y desarrollo, se debe estudiar la red Gnutella, a los efectos de lograr mayor escalabilidad y un mejor desempeño. Esta tarea debe orientarse a definir una nueva política de comunicación de los gnodos con sus vecinos y a establecer mejoras en las reglas de reenvío de mensajes. Además, se puede modificar el prototipo para operar sobre otra arquitectura de red P2P (no solamente basada en Gnutella), a los efectos de proveer servicios diferenciados a las aplicaciones.

Por otro lado, es necesario escribir interfaces para soportar otros motores de búsqueda tradicionales, lo que permitiría la instalación del sistema en mayor cantidad de sitios sin modificar su estructura actual.

También se debería analizar la posibilidad de definir un mecanismo de rankeo homogéneo, ó bien, un esquema de intercambio de información que permita al nodo que disparó la consulta realizar un procesamiento posterior a los efectos de entregar al usuario una respuesta de mayor precisión y relevancia.

Debido a que durante el desarrollo de este trabajo aparecieron proyectos similares como YouSearch [BAW03] sería interesante estudiar si su arquitectura brinda mayores

beneficios que la propuesta. Debido a que – a diferencia de IndiSE – YouSearch utiliza un componente centralizado, habría que determinar bajo qué circunstancias resulta conveniente la utilización de una arquitectura u otra.

Finalmente, se podría estudiar la posibilidad de implementar mecanismos de seguridad, privacidad, confianza e integridad de los datos intercambiados de acuerdo a los requerimientos del ambiente donde se implemente el sistema. Por ejemplo, en un ambiente de producción el sistema puede requerir de mecanismos para incluir ó excluir sitios donde realizar la búsqueda ó de un mecanismo que permita excluir nodos que no se comporten bajo ciertas normas de uso.

6 – Referencias

[AIK00] B. Aiken, J. Strassner, B. Carpenter, I. Foster, C. Lynch, J. Mambretti, R. Moore y B. Teitelbaum. Network Policy and Services: A Report of a Workshop on Middleware. RFC 2768, Febrero 2000.

En el RFC 2768 se discute el concepto de middleware y sus componentes. Se lo examina desde la perspectiva de las aplicaciones y los requerimientos en cuanto a APIs, manejo de recursos, políticas, servicios y demás tópicos relacionados.

[AIM03] American Online Inc. http://www.aim.com. 2003.

[APA03] The Apache Project. http://www.apache.org. 2003.

[BAE99a] R. Baeza Yates y G. Navarro. Recuperación de la Información: Algoritmos, Estructuras de Datos y Búsqueda en la Web. Universidad de Chile. 1999.

El reporte brinda una introducción sobre las técnicas de recuperación de información (RI) y brinda una descripción de los principales modelos y su aplicación. Además, plantea la problemática de la recuperación de información en la Web y enuncia las diferencias con el concepto tradicional de RI.

[BAE99b] R. Baeza Yates y B. Ribeiro-Neto. Modern Information Retrieval. Addison Wesley. Capítulo 13: Searching the Web. 2000.

Se brinda un enfoque acerca de las características de la Web como repositorio de información y como éstas afectan a la tarea de recuperación. Se describen las arquitecturas tradicionales de motores de indexación y búsquedas y se presentan temas como rankeo, interfaces y metabuscadores.

[BAR00] D. Barkai. An introduction to Peer-to-Peer Computing. Intel Developer UpdateMagazine. Febrero, 2000.

El documento presenta una introducción al modelo compañero a compañero y sus diferencias con el modelo cliente/servidor. Sugiere posibles áreas de utilización, en especial, orientadas a la colaboración entre usuarios finales y la formación de comunidades. Además, presenta la idea de proveer una capa de middleware P2P a las aplicaciones sobre el sistema operativo subyacente.

[BAW03] M. Bawa, R. Bayardo, S. Rajagopalan y E. Shekita. Make it Fresh, Make it Quick – Searching a Network of Personal Web Servers. En 12th International WWW Conference. Budapest, Hungría. 20-24 Mayo, 2003

En este documento se presenta un sistema de búsquedas distribuidas denominado YouServ, orientado a la búsqueda de información en servidores web personales de usuarios dentro de una organización. El sistema cuenta con un servidor de registro de palabras clave de cada nodo. Ante una consulta, dicho servidor retorna una lista de posibles nodos que pueden satisfacerla para ser contactados directamente.

[BER00] M. K.Bergman. The deep web: Surfacing hidden value. White Paper. Bright Planet. Julio 2000.

Plantea uno de los problemas actuales que se presentan para los motores de búsqueda al momento de la recolección de información que es el contenido dinámico. Los sitios que generan sus páginas dinámicamente (a partir del contendido de sus bases de datos) imponen restricciones a los crawlers. Se analizan las características y el comportamiento de los motores de búsqueda en cuanto a esta situación.

[BES99] M. Bessonov, U. Heuser, I. Nekrestyanov, A. Patel. Open Architecture for Distributed Search Systems. En Proceedings of the Sixth International Conference on Intelligence in Services and Networks (IS&N 99), Barcelona, España. 27-29 Abril, 1999.

El artículo describe una arquitectura abierta denominada OASIS para soportar búsquedas distribuidas sobre Internet. Dicha arquitectura se basa en un conjunto de nodos recolectores de información y de un servicio de directorio que permite a los mismos determinar cuáles servidores son capaces de responder consultas de sus usuarios.

[BLO70] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. En Communications of ACM, volumen 13(7), páginas. 422-426. 1970.

Se presenta una técnica para codificar claves sobre una cadena de bits, de manera de poder generar resúmenes de contenido. A partir de esta información se puede determinar la membresía de un elemento, donde se permite un determinado nivel de error.

[BOR01] F. Bordignon y G. Tolosa. Gnutella: Distributed System for Information Storage and Searching. Model Description. En JIT – Journal of Internet Technology. Vol. 2, No. 5. Taipei, Taiwan. 2001.
Disponible en:

http://www.tyr.unlu.edu.ar/TYR-publica/paper-final-gnutella-english-v2.pdf

El documento describe el sistema compañero a compañero orientado a compartir archivos denominado Gnutella, basado en el protocolo del mismo nombre. Se presenta técnicamente su modo de operación en cuanto a las funciones de ingreso y salida de nodos de la red, el mantenimiento de la misma y las funciones de búsquedas y descargas.

[BOW95] M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber y M. F. Schwartz. The Harvest Information Discovery and Access System. Computer Networks and ISDN Systems 28, pp. 119-125. 1995.

Se presenta la arquitectura del sistema Harvest, el cual tiene como objetivo brindar una arquitectura abierta para recolectar, indexar y acceder a información de la web de manera distribuida. Dicho sistema opera con dos componentes principales (gatherers y brokers), los cuales se pueden configurar de diferentes maneras para adecuarse a diversos escenarios.

[BRE00] B. E. Brewington y G. Cybenko Thayer. How Dynamic is the Web?. En Proceedings of the Ninth International World Wide Web Conference. 2000.

Los autores intentan determinar el dinamismo de la web a partir del estudio de datos recogidos y de un modelo propuesto. Dicho estudio se plantea a los efectos de determinar la frecuencia de actualización de los índices de los motores de búsqueda y de diseñar sus algoritmos de planificación de recolección.

[BRI98] S. Brin y L. Page. The anatomy of a large-scale hypertextual (Web) search engine. En Proceedings of the 7th International World Wide Web Conference (WWW7)/Computer Networks, 30(1-7):107—117. 1998.

El documento muestra la arquitectura y estructuras de datos del motor de indexación y búsqueda Google, a partir de sus objetivos de diseño. Además, se plantea y ejemplifica el mecanismo de rankeo de resultados denominado PageRank, diseñado por los autores.

[CLA00] I. Clarke, O. Sandberg, B. Wiley y T.W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. En Proceedings of the Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA. Julio, 2000.

Freenet es un sistema de publicación, replicación y recuperación de información con un alto interés en el anonimato. En Freenet no es posible determinar ni el origen de los datos ni el contenido de un nodo. Mediante un mecanismo eficiente de búsqueda que controla la cantidad

de mensajes generados las solicitudes son encaminadas según la ubicación física de los datos. Incorpora – además – estrategias de replicación y caching de respuestas.

[COU01] G. Coulouris, J. Dolimore y T. Kindberg. Sistemas Distribuidos. Conceptos y Diseño. 3ra edición. Addison-Wesley. 2001.

[CRE02] A. Crespo y H. Garcia-Molina. Routing indices for peer-to-peer systems. En Proceedings of the International Conference on Distributed Computing Systems. Julio 2002.

Se aborda el tema de la localización de información en sistemas compañero a compañero a partir de la utilización de estructuras de datos que no se comportan como índices tradicionales. Se proponen los "índices de ruteo" , los cuales determinan una "dirección" dentro del sistema hacia el documento. Básicamente, tres tipos de índices de ruteo son propuestos, analizando las prestaciones de cada uno bajo diferentes escenarios.

[CUE02] F. M. Cuenca-Acuña y T. D. Nguyen. Text-Based Content Search and Retrieval in ad hoc Communities. Technical Report DCS-TR-483, Rutgers University. Abril 2002.

Se presenta la arquitectura de PlanetP, un sistema para búsqueda y recuperación de información en comunidades P2P. El sistema opera diseminando resúmenes de contenido de nodos utilizando filtros de Bloom y rankea los documentos mediante el modelo vectorial. Desarrolla un modelo para determinar a cuáles nodos reenviar las consultas.

[DAB01] F. Dabek, E. Brunskill, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica y H. Balakrishnan. Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. En Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII). Schloss Elmau, Alemania. Mayo 2001.

A partir de la utilización de tablas de hash distribuidas (en este caso hash consistente), se construye un sistema que permite mapear claves en valores dentro de una topología de anillo. En ésta, nodos de una red P2P y objetos comparten un espacio de direcciones. La recuperación de un objeto consiste en aplicar una función hash al nombre del objeto para determinar la dirección del nodo que almacena la información acerca de su localización física.

[DIS03] Distributed.net. http://www.distributed.net. 2003.

[EXC03] Excite NewsTracker. http://news.exite.com. 2003.

[FIP00] FIPA. FIPA Peer-to-Peer Positioning Paper. Technical Report F-OUT-00076, Foundation for Intelligent Pyhsical Agents. Diciembre 2000.

Se plantean las características del modelo compañero a compañero y el rol de la tecnología de agentes inteligentes en la computación P2P. Además, se vislumbran algunas áreas donde puede evolucionar esta tecnología.

[FRA02] H. Fragati, G. Tolosa y F. Bordignon. Una Experiencia en Cómputo Masivo Distribuido Utilizando Applets Java. Póster Jornadas de la Ciencia y Tecnología. Universidad Nacional de Luján. 2002.

El documento describe una experiencia en cómputo masivo distribuido utilizando applets Java embebidos en páginas web. Se utilizan – por breves períodos de tiempo – ciclos de CPU de usuarios que se encuentran navegando por determinados sitios en los cuales – generalmente – se detienen a leer su contenido (por ejemplo, sitios de noticias).

[GAL02] L. Galanis, Y. Wang, S. R. Jeffery, D. DeWitt. Processing XML Containment Queries in Large Peer-to-Peer Systems. En Proceedings of the 28th VLDB Conference, Hong Kong, China. 2002.

Se presenta un sistema de búsqueda de información sobre una arquitectura P2P y un conjunto de documentos descriptos mediante el uso de pequeños resúmenes en lenguaje XML. Las consultas también se expresan en XML y se realizan sobre el almacenamiento local y sobre nodos vecinos mediante una estrategia de reenvío.

[GON01] L. Gong. Project JXTA: A Technology Overview. Sun Microsystems, Inc. White Paper. Abril, 2001.

El documento describe la evolución del modelo compañero a compañero en Internet y la visión del autor con respecto a su evolución. Describe la arquitectura JXTA como un entorno para la creación de servicios y aplicaciones bajo este modelo y plantea una serie de objetivos que se persiguen con su desarrollo.

[GOO03a] Google. http://www.google.com. 2003.

[GOO03b] Google Grupos. http://groups.google.com. 2003.

[HTD03] HT://Dig Search Engine Software. http://www.htdig.org. 2003.

[IBM03] IBM Inc. http://www.ibm.com 2003.

[ICQ03] Sitio Web Oficial. ICQ Inc. http://www.icq.com. 2003.

[INK02] Inktomi Web Search. Press Release: Inktomi Unveils Web Search 9. Disponible en: http://www.inktomi.com/company/news/press/2002/websearch9.html

[JOS02] S. Joseph. NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks. En Proceedings of the International Workshop on Peer-to-Peer Computing (junto a Networking 2002), Pisa, Italia. Mayo 2002.

El sistema Neurogrid utiliza un sistema adaptivo de búsqueda descentralizada. Se aprovechan las respuestas entregadas por un nodo modificando el conocimiento que cada nodo tiene acerca del contenido de otros. El sistema aumenta sus prestaciones a medida que se realizan más consultas y – además – opera con información de retroalimentación entregada por los usuarios.

[JXT01] Project JXTA: Technical Specification. Version 1.0. Sun Microsystems, Inc. White Paper. Abril, 2001.

El documento corresponde a la especificación de los protocolos (estructuras de datos y comportamiento) definidos por la plataforma JXTA.

[JXT03] Proyecto JXTA. Sun Microsystems. http://www.jxta.org. 2003.

[KOR01] E. Korpela, D. Werthimer, D. Anderson, J. Cobb y M. Lebofsky. SETI@home - Massively Distributed Computing for SETI. Computing in Science and Engineering, vol. 3, no. 1. Enero-Febrero 2001.
Disponible en http://computer.org/cise/articles/seti.htm

El proyecto SETI@home, tiene como objetivo detectar civilizaciones extraterrestres a partir del análisis de señales de radio recogidas por un radiotelescopio. Debido al intenso cómputo a realizar se utiliza un modelo de computación masiva distribuida. En éste, un conjunto de usuarios distribuidos por todo el mundo colaboran con el proyecto prestando ciclos de CPU para procesar las señales.

[KSE03] Ksearch Software. http://www.kscripts.com/ksearch. 2003.

[LAW98] S. Lawrence y L. Giles. Searching the World Wide Web. Science, vol.280, pp.98-100. 1998.

El artículo presenta los resultados sobre estudios de cobertura de motores de búsqueda,

respecto del tamaño total de la web indexable. Se incluyeron en el mismo los motores de búsqueda más importantes a la fecha de realización. Se sugiere la utilización de metabuscadores y softbots para mejorar las búsquedas.

[LAW99a] S. Lawrence y L. Giles. Accessibility and Distribution of Information on the Web. Nature, vol.400, n.6740, pp.107-109. 1999.

Se analizan problemas relacionados a la web y los motores de búsqueda. Se plantean inconvenientes en cuanto al crecimiento de la web, cobertura de los motores de búsqueda y limites de recursos.

[LAW99b] S. Lawrence y L. Giles. Searching the Web: General and Scientific Information Access. IEEE Communications, vol.37, n. 1, pp.116-122. 1999.

Los autores muestran un estudio sobre motores de búsqueda, planteando la problemática de la cobertura, cantidad de información y heterogeneidad de formatos. Sugieren técnicas para mejorar la recuperación de información especialmente en el ámbito científico mediante el uso de metabuscadores, librerías digitales y sistemas de citas automáticos.

[MAI02] N. Maibaum y T. Mundt. JXTA: A Technology Facilitating Mobile Peer-To-Peer Networks. En International Mobility and Wireless Access Workshop (MobiWac'02). Fort Worth, Texas. Octubre 12. 2002.

El artículo describe la plataforma JXTA, destinada a facilitar a los desarrolladores la creación de aplicaciones distribuidas. La mencionada plataforma incluye un conjunto de protocolos pensados para facilitar la interacción de dispositivos móviles (PDAs, teléfonos celulares) sin intervención de equipos centrales. Se presentan – además – escenarios de uso de dicha tecnología.

[MIN01] N. Minar y M. Hedlund. A Network of Peers. Peer-to-Peer Models Through the History of the Internet. A. Oram, editor. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Capítulo 1. O'Reilly & Associates. Marzo 2001.

Los autores muestran su visión acerca del modelo compañero a compañero y cómo puede facilitar la creación de nuevas aplicaciones sobre Internet. Plantea como la arquitectura original de Internet era P2P y los motivos por los que migró a cliente/servidor.

[MIN03] MinML a minimal XML parser. http://www.wilson.co.uk/xml/minml.htm. 2003.

[MSS03] Microsoft Site Server. http://www.microsoft.com/siteserver. 2003.

[MYS03] MySQL Database. http://www.mysql.com. 2003.

[NAP00] Napster Inc. http://www.napster.com. 2000.

[OMG95] Object Management Group. The Common Object Request Broker: Architecture and Specification (CORBA), revision 2.0. 1995.

[OPE02] Open Directory Project. http://www.dmoz.org. 2003.

[PER03] Perl Site. http://www.perl.com. 2003.

[RAT01] S. Ratnasamy, P. Francis, M. Handley y R. Karp, A Scalable Content-Addressable Network (CAN). En Proceedings of ACM SIGCOMM. 2001.

A partir de la utilización de tablas de hash, las cuales permiten mapear claves en valores, se construye una infraestructura distribuida de localización de objetos. El sistema mapea claves de objetos en nodos, dentro de un espacio d-dimensional, dividido en zonas, permitiendo la recuperación de las mismas eficientemente. Mediante simulación se probó la escalabilidad del sistema a más de 260.000 nodos.

[ROW01] A. Rowstron y P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. En IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Alemania, pages 329-350. Noviembre 2001.

Se presenta Pastry, un sistema de ruteo a nivel aplicación que permite la ubicación de objetos dentro de una red de recubrimiento. El sistema permite mapear nodos y objetos dentro de un espacio de direcciones. Cuando se solicita a un nodo una consulta por una clave, éste puede retornar la dirección del nodo más cercano a la misma. Según los resultados experimentales, el sistema es escalable, tolerante a fallas y eficiente.

[SAL75] G. Salton, A. Wang, C.Yang. A vector space model for information retrieval. En Journal of the American Society for Information Science, volumen 18, páginas 613-620. 1975.

Se presenta un modelo que permite determinar una medida de relevancia de términos dentro de una colección de documentos y su aplicación en la tarea de recuperación de información como mecanismo de rankeo.

[SEA02] SearchEngineShowdown. Search Engine Statistics: Freshness Showdown. Octubre 2002.
Disponible en http://www.searchengineshowdown.com/stats/freshness.shtml

[SHI01] C. Shirky. What Is P2P and What Isn´t?. The O'Reilly Network. 2001.
Disponible en:
http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

El autor brinda una definición sobre el concepto de compañero a compañero y plantea una serie de preguntas para verificar si una aplicación se corresponde al modelo. Además, plantea una serie de aplicaciones emergentes que aprovechan las capacidades de los equipos de usuario final.

[SIM03] Simple Web Indexing System for Humans – Enhanced. http://swish-e.org. 2003.

[SIO02] W. Siong Ng, B. Chin Ooi, K. Tan. BestPeer: A Self-Configurable Peer-to-peer System. En Proceedings of the 18th International Conference on Data Engineering. San José, CA. Abril 2002. (Poster Paper)

El sistema BestPeer pretende servir como una plataforma de desarrollo de aplicaciones P2P, integrando tecnología de agentes móviles. Éstos permiten expandir la funcionalidad de un nodo incorporando código que se ejecuta en otros componentes de la red, como por ejemplo, filtrado de información ó análisis de datos. Incorpora una estrategia de optimización a partir de prioridades en los nodos de acuerdo a la cantidad de consultas que responden.

[SIO03] W. Siong Ng, B. Chin Ooi, K. Tan y A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. En Proceedings of the 19th International Conference on Data Engineering. Bangalore, India. Marzo 2003. (aceptado para publicar)

El proyecto presentado corresponde a un sistema orientado a compartir archivos entre usuarios de una red P2P, que soporta búsquedas por contenido. Incorpora tecnología de agentes móviles para facilitar el procesamiento de datos en los nodos. Además, el sistema se autoconfigura de acuerdo a un criterio de optimización de comunicaciones.

[SOU03] Sourceforge.net. Open Source Software Development Community. http://www.sourceforge.net. 2003.

[STE98] W. R. Stevens. UNIX Network Programming. Networking APIs: Sockets and XTI. Volumen 1. Prentice Hall. 1998.

[STO01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek y H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. En Proceedings of the ACM SIGCOMM '01 Conference, San Diego, California. Agosto 2001.

Se aborda el problema de la localización de objetos por parte de aplicaciones P2P de manera eficiente. El protocolo presentado, denominado Chord, utiliza la técnica de hash consistente para mapear el nombre de un recurso en una dirección de nodo. En éste se almacena la localización física de dicho recurso.

[SUN02] T. Sunsted. The practice of peer-to-peer computing: Discovery: How peers locate one another. IBM DeveloperWorks. 2002.
Disponible en http://www-106.ibm.com/developerworks/java/library/j-p2pdisc/

El documento plantea la problemática de la localización de nodos dentro de una red compañero a compañero. Muestra las diferentes soluciones para abordar el problema, ya sea mediante mecanismos estáticos ó dinámicos.

[TCP81] Transmission Control Protocol. RFC 793. Septiembre 1981

[TEO03] Teoma. http://www.teoma.com. 2003.

[TOL01] G. Tolosa y F. Bordignon. Propuesta de Arquitectura Cooperativa Destinada a un Servicio de Búsqueda Distribuida en el Espacio Web. Una Alternativa a los Motores de Búsqueda Tradicionales. Póster en Jornadas de la Ciencia y Tecnología. Universidad Nacional de Luján. 2001.

La arquitectura propuesta plantea una solución distribuida al problema de la búsqueda de información en el espacio web, como alternativa a los motores de búsqueda tradicionales. Se utiliza una red P2P basada en el protocolo Gnutella (como una extensión al mismo). El modelo proporciona a los usuarios acceso a recursos dentro de un dominio del conocimiento específico.

[TOL02a] G. Tolosa, F. Bordignon. y C. Rodríguez. Una Arquitectura Cooperativa Destinada a la Distribución de un Servicio de Transmisión de Radio por Internet. Memorias de Informática 2002 – I Congreso Internacional de Tecnologías y Contenidos Multimediales en Ambientes Web. La Habana, Cuba. 2002.

Se define una propuesta de arquitectura de red a nivel aplicación como base de un servicio de retransmisión de un flujo de radio por Internet. Se desarrollan los lineamentos básicos del

servicio en cuestión como una extensión al protocolo Gnutella. Se especifica la arquitectura del sistema, definiendo principalmente la forma de operación de la red de distribución de flujos de audio mientras que la retransmisión multimedial se realiza mediante aplicaciones libres (como por ejemplo Shoutcast).

[TOL02b] G. Tolosa, F. Bordignon y F. Lorge. Aplicaciones P2P en Internet. Hacia un Soporte Cooperativo de Cómputo Masivo y Sistemas de Información. Póster Jornadas de la Ciencia y Tecnología. Universidad Nacional de Luján. 2002.

El póster expone el uso actual de aplicaciones basadas en el modelo compañero a compañero en Internet. Se describen sus orígenes, características, tipos, arquitectura típica y áreas de aplicación, planteando al modelo P2P como infraestructura para la construcción de sistemas de información distribuidos.

[TOL02c] G. Tolosa y F. Bordignon. Red a Nivel de Aplicación como Middleware P2P para Soportar Servicios Distribuidos sobre Internet. Póster en CACIC 2002 – VIII Congreso Argentino de Ciencias de la Computación. 2002.

El trabajo describe la arquitectura de un middleware que permite a las aplicaciones acceder a una red P2P, la cual opera como infraestructura de comunicaciones. Se plantea la división en capas de software de la funcionalidad de la aplicación y del middleware, de manera de generar una arquitectura abierta.

[YAH03] Yahoo. http://www.yahoo.com. 2003.

[YAN01] B. Yang y H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. Proceedings of the 27th VLDB Conference, Roma, Italia. 2001.

Los autores enfocan su estudio a sistemas compañero a compañero híbridos, es decir, aquellos donde existe alguna funcionalidad centralizada en algún nodo de la red. A partir de las aplicaciones orientadas a compartir archivos, desarrollan un modelo probabilístico para describir el comportamiento de las consultas y las respuestas esperadas (en términos de tamaño). Además, desarrollan un modelo para describir la performance del sistema y lo aplican a sistemas existentes.

[WAT02] S. Waterhouse, D. M. Dooling, G. Kan y Y. Faybishenko. JXTA Search: a distributed search framework for peer-to-peer networks. IEEE Internet Computing, v.6, pp 68-73, 2002.

JXTA Search es un motor de búsquedas distribuido que opera sobre redes compañero a

compañero. Ofrece una interfaz para proveedores y consumidores de información utilizando un protocolo propio basado en XML. La red P2P se encuentra formada por nodos con capacidades especiales denominados "hubs", los cuales funcionan como agentes de registro de contenido y ruteadores de consultas.

[WEB03] Web Rank. Search Engine / Directory Indexing Times. 2003. Disponible en: http://www.high-search-engine-ranking.com/search_engine_indexing.htm.

[WWW98] WWW Consortium (W3C). Extensible Markup Language (XML 1.0). Technical Report. 1998. Disponible en http://www.w3.org/XML/

Anexo I

Descripción del Protocolo Gnutella

El protocolo Gnutella define una red a nivel aplicación, bajo la modalidad compañero a compañero donde todo nodo (en adelante gnodo) realiza al mismo tiempo funciones de cliente y servidor. Conceptualmente, es un sistema distribuido para almacenamiento y búsqueda de información.

La red Gnutella se compone de numerosos gnodos distribuidos a lo largo del mundo. Su topología no indica jerarquía alguna, dado que cada gnodo es igual a los demás en funcionalidad. Esta red opera bajo el modelo conocido como “propagación viral”. Un gnodo envía un mensaje a otro gnodo, y éste lo reenvía a todos los gnodos a los cuales esté conectado, y así sucesivamente (hasta un número máximo de saltos).

Tipos de Mensajes

Un mensaje Gnutella se transporta sobre protocolo TCP. Su cabecera siempre consta de 23 bytes y consiste de los siguientes campos: identificador de gnodo (16 bytes), función (1 byte), TTL ó tiempo de vida restante (1 byte), hops ó saltos (1 byte) y largo de la carga en bytes (4 bytes). El valor referenciado en el campo función indica que tipo de acciones deben realizar aquellos gnodos que reciben el mensaje. La cabecera descripta es fija para cualquier tipo de función instanciada, lo que es variable en tamaño es la segunda parte ó carga del mensaje dado que depende directamente de la función. Las funciones definidas son:

Función Ping ó Init (código 0x00)

Los gnodos la utilizan para anunciar su presencia en la red y recolectar información de otros gnodos existentes. No lleva carga de mensaje, es decir que el campo largo de la carga se instancia en cero. Cada gnodo que reciba un mensaje Ping debe responder generando mensajes Pong (pueden ser n) donde anuncien su presencia ó de gnodos que tengan noticia.

Ejemplo:

Mensaje (en hexadecimal)

```
27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 00 07 00 00 00
```

00 00

Detalle

Id. de gnodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 00

TTL : 07

Hops : 00

Largo de carga : 00 00 00 00

Función Pong ó Init_Response (código 0x01)

Un gnodo envía un mensaje Pong como respuesta a un mensaje Ping. La carga de un mensaje Pong se compone de los siguientes campos: puerto (2 bytes), dirección IP (4 bytes), cantidad de archivos compartidos (4 bytes) y kbytes compartidos (4 bytes). Es interesante destacar que un gnodo – al recibir datos como los descriptos – puede abrir nuevas conexiones que le permitan extender su dominio de acción y – a la vez – posibilitarle una mayor estabilidad a su presencia en la red Gnutella.

Ejemplo:

Mensaje (en hexadecimal)

27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 01 07 00 0E 00
00 00 CA 18 AA D2 62 95 03 00 00 00 00 00 00 00

Detalle

Id. de gnodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 01

TTL : 07

Hops : 00

Largo de carga : 0E 00 00 00

Carga

Puerto : CA 18

Dirección IP : AA D2 62 85

Cantidad de archivos compartidos : 03 00 00 00

Kbytes compartidos : 00 00 00 00

Función Push (código 0x40)

Esta función es utilizada para descargar recursos desde gnodos que acceden a la red atravesando un firewall. El gnodo que requiere el recurso enviará un mensaje Push al gnodo que lo tiene, con los siguientes campos: identificador de gnodo (16 bytes), índice (4 bytes), dirección IP (4 bytes) y puerto (2 bytes). Una vez obtenido el mensaje, el gnodo poseedor del recurso establecerá una conexión con el gnodo peticionante y le enviará, a la dirección IP y puerto informado, el archivo referenciado por el campo índice.

Ejemplo:

Mensaje (en hexadecimal)

```
27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 40 07 00 1A 00
00 00 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11 02 00 00
00 AA 12 62 AA CA 22
```

Detalle

Id. de gnodo : 27 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 40

TTL : 07

Hops : 00

Largo de carga : 1A 00 00 00

Carga

Id. de gnodo: 22 11 3F 71 49 B2 D4 11 88 23 00 80 AD 40 22 11

Indice : 02 00 00 00

Dirección IP: AA 12 62 AA

Puerto : CA 22

Función Query (código 0x80)

Esta función permite a un gnodo consultar a los demás gnodos de la red por un determinado recurso. La carga utilizada por esta función se compone de dos campos, a saber: velocidad mínima requerida para descarga (2 bytes) y un criterio de búsqueda de longitud variable.

Ejemplo:

Mensaje (en hexadecimal)

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 80 07 00 06 00 00 00 00 00 2A 2E 61 00

Detalle

Id. de gnodo : 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 80

TTL : 07

Hops : 00

Largo de carga : 06 00 00 00

Carga

Velocidad mínima : 00 00

Criterio de búsqueda : 2A 2E 61 00 (patrón seguido de un caracter nulo)

Función Query_Hit (código 0x81)

Se utiliza como respuesta a un mensaje tipo Query. Los campos que componen un Query_Hit son: cantidad de hits ó coincidencias (1 byte), puerto (2 bytes) y dirección IP de descarga (4 bytes), velocidad de operación (4 bytes), resultados (longitud variable) e indentificador del gnodo que responde (16 bytes). Los resultados (que pueden ser n) se estructuran como una lista, con la siguiente estructura: índice ó número de respuesta (4 bytes), tamaño del archivo en kilobytes (4 bytes), nombre del archivo (longitud variable) y delimitador de registro (0x0000). El gnodo que responde incluye en la respuesta su identificador a los efectos de que lo pueda utilizar el gnodo peticionante si decidiera solicitar un recurso mediante un mensaje Push.

Ejemplo:

Mensaje (en hexadecimal)

28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62 81 07 00 57 00 00 00 03 CA 18 AA D2 62 95 1C 00 00 00 00 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 61 00 00 01 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 62 00 00 02 00 00 00 70 00 00 00 31 34 39 61 72 63 68 69 2E 6D 00 00 A0 3D 8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66

Detalle

Id. de gnodo : 28 08 3F 71 49 B2 D4 11 88 23 00 80 AD 40 4E 62

Función : 81

TTL : 07

Hops : 00

Largo de carga : 57 00 00 00

Carga

Cantidad de hits: 03

Puerto : CA 18

Dirección IP : AA D2 62 95

Velocidad : 1C 00 00 00

Estructura resultados

Id. de gnodo : 3D 8D 3D 7D 84 D1 11 85 8E 52 54 00 DC 37 66

Estructura resultados

1er elemento

Indice : 00 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 61 (149archi.a)

Terminador : 00 00

2do elemento

Indice : 01 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 62 (149archi.b)

Terminador : 00 00

3er elemento

Indice : 02 00 00 00

Tamaño de archivo : 70 00 00 00

Nombre de archivo : 31 34 39 61 72 63 68 69 2E 6D (149archi.m)

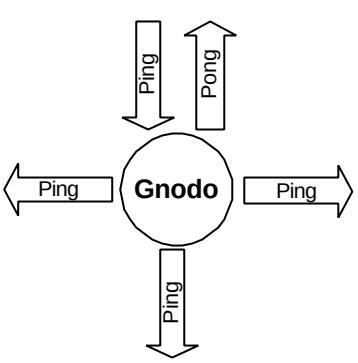
Terminador : 00 00

Reglas de Propagación

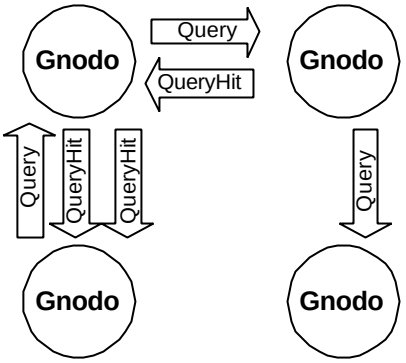
El modelo de propagación de mensajes implantado en Gnutella requiere que los gnodos tengan la capacidad de reenviar los mensajes recibidos (funciones Ping, Pong, Query,

Query_Hit y Push) a través de sus conexiones establecidas. Esta tarea se realiza de acuerdo a las siguientes reglas de propagación:

Regla A: Un gnodo propagará mensajes tipo Ping y Query a todos los gnodos conectados directamente, excepto al que entregó dicho mensaje.



Propagación de Ping



Propagación de consultas

Regla B: Los mensajes tipo Pong, Query_Hit y Push solo deben ser propagados por el mismo camino por el que viajó el mensaje inicial (Ping ó Query) asociado. Esto es posible debido a que antes de propagar un mensaje, los gnodos revisan en tablas internas por aquel que generó tal respuesta. De estas tablas obtienen la información de la conexión por donde deben enviarlos.

Regla C: Un gnodo decrementará en 1 el valor del campo TTL de un mensaje e incrementará el valor del campo Hops en 1 antes de propagar dicho mensaje por las conexiones pertinentes. Si al decrementar el valor de TTL el resultado obtenido es cero, debe desechar el mensaje.

Regla D: Si un gnodo recibe un mensaje con idéntico identificador de gnodo y mismo valor de función que uno recibido anteriormente (en un período breve de tiempo, no especificado formalmente) debería evitar propagarlo.

En un instante de tiempo puede haber miles gnodos en la red, pero para un gnodo en particular su dominio de acción estará restringido a la cantidad de conexiones establecidas y al alcance de sus mensajes (antes de que el valor del campo TTL llegue a cero y sea descartado). Debido al dinamismo, la topología y la existencia de usuarios temporales, es posible que las respuestas a una misma consulta varíen en el tiempo. Cada vez que un usuario ingresa en la red puede hacerlo a una parte diferente de la misma, según el estado actual de sus gnodos vecinos.

Descarga de Archivos

Una vez que un gnodo recibe un resultado (Query_Hit) a una consulta previamente realizada, puede optar por seleccionar un archivo para su descarga desde el gnodo remoto. Los archivos se recuperan directamente, sin intervención de gnodos intermedios y por lo tanto tampoco de la red Gnutella. La descarga se realiza mediante el uso del protocolo HTTP 1.0.

A los efectos de iniciar la descarga, el gnodo que requiere el recurso abrirá una conexión TCP contra el gnodo remoto en el puerto anteriormente obtenido y, mediante la primitiva GET solicitará el recurso seleccionado, indicando índice y nombre de archivo, por ejemplo:

GET /<índice>/<nombre de archivo>/ HTTP/1.0
Connection: Keep-Alive
Range: bytes=0-

Al recibir una petición, un gnodo remoto valida que el recurso solicitado está disponible y – en caso afirmativo – transmitirá el archivo, previo envío de una cabecera estándar de HTTP con el siguiente formato:

HTTP 200 OK
Server: Gnutella
Content-type: application/binary
Content-length: <tamaño del archivo en bytes>

Anexo II

Descripción de la plataforma JXTA

JXTA es una plataforma de programación, impulsada por la compañía Sun Microsystems, orientada a facilitar el desarrollo de aplicaciones sobre ambientes distribuidos, en particular, operando bajo el modelo compañero a compañero.

Sus objetivos principales están relacionados con resolver – de forma general – cuestiones como [GON01]:

- *Interoperabilidad*, entre los nodos componentes de una red, facilitando la ubicación de los recursos, la comunicación entre distintos nodos compañeros, la organización en grupos y la prestación de servicios.

- *Independencia de plataforma*, debido a la heterogeneidad de arquitecturas de hardware y software. Por esta razón, la plataforma se encuentra programada en lenguaje Java y los protocolos construidos a partir de mensajes escritos en XML.

- *Ubicuidad*, ya que la intención es que sea implementable en cualquier dispositivo digital, no solamente computadoras.

Tecnológicamente, la arquitectura de la plataforma (Figura 8.1) está formada por un conjunto de conceptos y protocolos [GON01] organizados en tres niveles ó capas y definidos – como se mencionó anteriormente – mediante mensajes XML. Las capas se organizan en tres niveles de abstracción desde un núcleo (Core ó Platform, la capa interna), de la siguiente manera:

- *Plataforma (Platform, Core)*: Provee un conjunto de primitivas comunes a todos las funciones de red compañero a compañero.

- *Servicios (Services)*: Está formado por un conjunto de servicios de red P2P que no son fundamentales, pero que pueden ser utilizados como soporte de otros desarrollos, por ejemplo, servicios de directorio, de almacenamiento y autenticación.

- *Aplicaciones (Applications)*: Este nivel es similar al concepto de Servicio. Aquí se incluyen otras funciones que se pueden implementar como otros servicios de una red P2P, por ejemplo un sistema de votación distribuido.

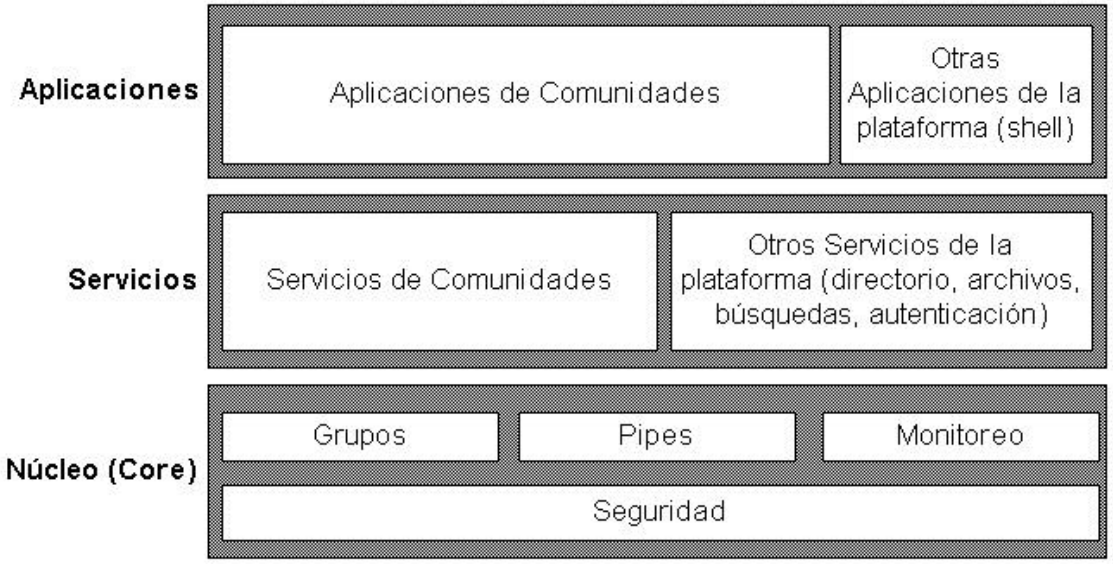


Figura 8.1 – Arquitectura de la plataforma JXTA

Componentes de la Arquitectura

Los nodos dentro de una red JXTA reciben el nombre de Peers y tienen un identificador de 128 bits dentro de la red (UUID). Operan independientemente de otros nodos. Dichos nodos se vinculan entre sí formado la red y – a su vez – se pueden agrupar en comunidades. Estas operaciones las realizan intercambiando mensajes de acuerdo a uno de los protocolos definidos.

Los Peers se pueden agrupar de acuerdo a intereses comunes formado grupos denominados PeerGroups identificados por un PeerGroupID. Los grupos permiten que los nodos se intercambien información solo si pertenecen al mismo grupo, se puedan formar ambientes acotados de búsquedas y se implementen – por ejemplo – grupos seguros ó se pueda monitorear un determinado servicio. Un nodo dentro de JXTA puede pertenecer a más de un grupo. Dichos servicios pueden estar asociados a un nodo particular ó a un grupo. En este último caso, el servicio es provisto por todos los nodos integrantes del grupo, brindando mayor disponibilidad del mismo.

Los recursos dentro de la plataforma están descriptos mediante documentos escritos en XML denominados *Advertisements*. Éstos sirven para publicar la existencia de un recurso

en JXTA y están organizados jerárquicamente en elementos (*Elements*) que poseen atributos (*Atributes*) consistentes en cadenas de pares <nombre-valor>. Existen tipos de advertisements definidos a nivel del núcleo de JXTA y se pueden definir subtipos propios. Los tipos definidos son: Core, Peer Group, Pipe, Service, Content y Endpoint.

El intercambio de información entre nodos se realiza mediante Mensajes (Messages), los cuales están formados por un sobre (Envelope) y una pila (Stack) de encabezados y cuerpos de protocolo (Figura 8.2). La estructura del sobre incluye datos como versión, dirección origen, dirección destino y resumen, mientras que la pila depende del protocolo involucrado.

Figura 8.2 – Estructura de los mensajes JXTA

A los efectos de proveer una abstracción del canal de comunicaciones entre dos servicios ó aplicaciones, JXTA define el concepto de Pipe como un canal virtual asincrónico utilizado para enviar y recibir mensajes. Los Pipes se vinculan a un nodo mediante un protocolo de unión (Pipe Binding) en tiempo de ejecución y proveen la abstracción de un punto donde un Peer recibe mensajes, independiente de su localización física.

Los mensajes no se propagan de acuerdo a reglas predefinidas, sino que cada nodo los reenvía a su ámbito. El ámbito de un nodo está definido por uno ó más grupos a los cuales pertenece. Si una aplicación o servicio particular requiere de un ámbito diferente de propagación, se deberá definir un grupo nuevo.

Protocolos Base

Básicamente, existen seis protocolos definidos por la arquitectura de JXTA y son los descriptos a continuación [JXT01]:

- *Peer Discovery Protocol (PDP)*: Este protocolo permite a los nodos encontrar los Anuncios de otros Peers ó de otros Grupos de Peers. Es posible enviar mensajes destinados a localizar recursos conociendo ó no el nombre de otro nodo (el mensajes es unicast ó broadcast respectivamente). La respuesta es uno ó más Advertisements por parte del receptor ó los receptores.

- *Peer Resolver Protocol (PRP)*: Permite que un nodo envíe un mensaje de consulta genérico a un servicio en otro nodo, con diversas funciones como por ejemplo: localizar un Peer ó el estado de un servicio.

- *Peer Information Protocol (PIP)*: Mediante PIP se puede obtener información sobre las capacidades y estado actual de un nodo. Un mensaje típico de este protocolo es Ping, que permite determinar si un nodo está activo.

- *Peer Membership Protocol (PMP)*: Este es un protocolo utilizado para participar de grupos de pares. Mediante PMP, un nodo puede consultar acerca de los requerimientos para pertenecer a un determinado grupo y – eventualmente – solicitar ó cancelar la membresía al mismo.

- *Pipe Binding Protocol (PBP)*: Se utiliza para que los nodos establezcan un canal virtual de comunicaciones con otros componentes de software (aplicaciones ó servicios) de nodos ó grupos de nodos.

- *Peer Endpoint Routing Protocol (PEP)*: Funciona como un protocolo de ruteo, habilitando a los Peers ruteadores el reenvío de mensajes a otros Peers ruteadores a los efectos de alcanzar el destino de los mismos.

Estos protocolos no son necesarios de implementar en todos los nodos, sino que cada uno de los Peers utiliza solamente aquellos que sean requeridos por la funcionalidad que tiene dentro de la red.

Desarrollos sobre JXTA

Debido a que la plataforma se encuentra disponible para cualquier desarrollador (bajo una licencia especial) existen actualmente una gran cantidad de aplicaciones y servicios que se pueden evaluar (en diferentes etapas de desarrollo) y – opcionalmente – contribuir con alguno de éstos.

A continuación se mencionan brevemente alguno de los proyectos que utilizan la plataforma para el desarrollo de aplicaciones y servicios distribuidos.

Aplicaciones	
allhands	Este proyecto pretende utilizar JXTA como un entorno de mensajería, a partir de unir – mediante gateways – sistemas de mensajería instantánea, correo electrónico y buscapersonas.
gnougat	Gnougat es un sistema de caching de archivos completamente distribuido. Permite la localización de contenido en un ambiente descentralizado a partir de una red P2P.
p2pconference	Es un sistema colaborativo para organizar y llevar a cabo conferencias y exposiciones de expertos en un tema a partir de una interfaz textual, si necesidad de realizar un encuentro real entre las personas.
radiojxta	Es un experimento para distribuir contenido de audio en una red, donde cada nodo colabora con la tarea. Conforme aumenta la cantidad de nodos en la red, se tiene mayor ancho de banda y recursos para llevarla a cabo.
sentinel	La idea es desarrollar una herramienta de monitoreo de redes y sistemas que resulte precisa, escalable, altamente flexible (configurable) y eficiente.

Servicios	
gisp	GISP (Global Information Sharing Protocol) provee un servicio de tabla de hash distribuida. Mediante esta técnica, se permite el mapeo de claves en determinados nodos facilitando luego la recuperación.
jxta-rmi	JXTA-RMI permite la programación sobre la API de Invocación a Métodos Remotos (RMI) del lenguaje Java, facilitando una abstracción que – de manera dinámica – decide si un objeto debe ser contactado utilizando RMI ó JXTA Pipes.
jxtaspaces	Este proyecto pretende diseñar un servicio de memoria compartida distribuida para la plataforma JXTA, que facilite la programación de aplicaciones distribuidas.
p2p-email	JXTA P2P email es un entorno abierto de comunicación para ambientes colaborativos. Facilita la creación de herramientas para manejar mensajería entre pequeños grupos de usuarios con intereses comunes.
Proxyservice	En este proyecto se pretende crear un servicio de servidor proxy para otras aplicaciones, junto a un servicio de resolución de nombres. La idea es facilitar a los desarrolladores el acceso a servicios externos a través de este servicio, manejando solamente el envío de mensajes a un proxy.

Anexo III

Código Fuente de gnutWare

Clase gnutWare.java

```
import java.io.*;
import java.net.*;
import java.lang.*;
import java.text.*;
import java.util.*;

public class gnutWare
{
   //Streams para conexión Gnut
   public static DataOutputStream os;
   public static DataInputStream is;
   public static List msg_received;
   public static List msg_tosend;
   public static String nodename;
   public static int listen_port;
   public static GUID guid;
   public static GUID rcv_guid;

   static boolean exit;

   public static void main(String args[])
   {
      byte[] b = new byte[65535];
      byte[] r = null;
      try
      {
         //Leo configuración
         String ipaddr = args[0];
         int port = Integer.parseInt(args[1]);
         nodename = args[2];
         listen_port = Integer.parseInt(args[3]);

         //Genero un GUID para este el nodo gnutWare
         guid = new GUID();
         System.out.println ("El GUID del nodo " + nodename + " es " +
         guid.getGUID());

         //Abro una conexión con el Gnut
         Socket clientSocket = new Socket(ipaddr, port);
         System.out.println("[" + nodename + "] Conexion: " +
         clientSocket);
         connectServent(clientSocket);

         //Inicializo dos listas: una para manejar los mensajes
         //entrantes y otra para manejar los mensajes salientes
```

```
        msg_tosend   = Collections.synchronizedList((List)(new
        Vector()));
        msg_received = Collections.synchronizedList((List)(new
        Vector()));

        new Send_Message().start();
        new Receive_Message().start();
        new Terminal_Control().start();
        new Network_Sap().start();

        send(new PingMessage());

        String code;
        String id;
        String str;
        String payload;
        GUID g;
        int plen;

        while (!exit)
        {
           try
           {
              synchronized (msg_received)
              {
                 if (!msg_received.isEmpty())
                 {
                    b = (byte[]) msg_received.remove(0);
                    byte id_fun = b[Message.PAYLOAD_ID_POS];
                    Message msg = new Message(b);

                    switch (id_fun)
                    {
                       case Message.PING:
                          PingMessage msgPING = new PingMessage(b);
                          g = msg.getGUID();
                          System.out.println("----------------------
------------------------------------------------");
                          System.out.println("Ping RECIBIDO desde: "
+ Util.byteArray2string(g.toByteArray()));
                          System.out.println("----------------------
------------------------------------------------");
                          send(new PongMessage(g));
                          break;

                       case Message.PONG:
                          PongMessage msgPONG = new PongMessage(b);
                          System.out.println("----------------------
------------------------------------------------");
                          System.out.println("Pong RECIBIDO desde: "
+ msgPONG.getIPAddress() + ":" + msgPONG.getPort() + " con " +
msgPONG.getFileCount() + " archivos, " + msgPONG.getSharedSize() + "
Kb");
                          System.out.println("----------------------
------------------------------------------------");
                          break;

                       case Message.QUERY:
                          /*
```

```
                           Llegó un QUERY, tengo que enviarlo a la
                           aplicación que
                           corresponda de acuerdo al code y al id. Por
                           ahora va a la única existente
                           */
                           rcv_guid = msg.getGUID();
                           String str_guid =
Util.byteArray2string(rcv_guid.toByteArray());
                           payload = msg.getPayload();
                           System.out.println("----------------------
----------------------------------------------");
                           System.out.println("[gnutWare, QUERY
RECIBIDO <- p2p ] *" + payload + "* recibido con GUID: *" + str_guid +
"*");
                           plen = (int) msg.getPayloadLength();

                           int delimiter = payload.indexOf("~");
                           String speed = payload.substring(0, 2);
                           System.out.println("Del.pos: " + delimiter
+ "*");
                           System.out.println("Speed  : " + speed +
"*");
                           str = payload.substring(2, delimiter);
                           System.out.println("str    : " + str +
"*");
                           code = payload.substring(delimiter + 1,
delimiter + 5);
                           System.out.println("code   : " + code +
"*");
                           id   = payload.substring(delimiter + 5,
delimiter + 10);
                           System.out.println("id     : " + id + "*");
                           System.out.println("----------------------
----------------------------------------------");
                           System.out.println("[gnutWare: QUERY ->
p2pTime] *" + str);

                           byte mqy[] = Util.string2byteArray(str +
"\n");
                           Network_Sap.nsos.write(mqy);
                           break;

                        case Message.QUERY_HITS:
                           ServiceResponseMessage msgRESP = new
ServiceResponseMessage(b);
                           System.out.println("----------------------
----------------------------------------------");
                           System.out.println("[gnutWare, QUERY_HIT
RECIBIDO <- p2p]*" + msgRESP.getResult(1, 3) + "*");
                           /*
                           Llegó un QUERY_HITS, tengo que enviarlo a
                           la aplicación que corresponda de acuerdo al
                           code y al id
                           */
                           str = msgRESP.getResult(1, 3);

                           byte mqh[] = Util.string2byteArray(str +
"\n");
                           Network_Sap.nsos.write(mqh);
```

```
                           System.out.println("----------------------
------------------------------------------------");
                           System.out.println("[gnutWare, QUERY_HIT ->
p2pTime] * " + str);
                           break;
                        }
                     }
                  }
                  Thread.sleep(1);
               }
               catch (Exception e)
               {
                  e.printStackTrace();
               }
            }
            clientSocket.close();
         }
         catch (Exception e)
         {
            System.out.println(e);
         }
         System.exit(-1);
      }

      public static void connectServent(Socket clientSocket)
      {
         try
         {
            clientSocket.setSoTimeout(7000);
            clientSocket.setTcpNoDelay(true);
            is = new DataInputStream(clientSocket.getInputStream());
            os = new DataOutputStream(clientSocket.getOutputStream());

            byte[] b = Util.string2byteArray("GNUTELLA CONNECT/0.4\n\n");
            os.write(b);
            is.skip("GNUTELLA OK\n\n".length());
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }

      public static void send(Message message)
      {
         send(message.toByteArray());
      }

      public static void send(byte[] message)
      {
         synchronized (msg_tosend)
         {
            msg_tosend.add(message);
            System.out.println(message);
         }
      }

   }

   class Send_Message extends Thread
```

```
{
   Send_Message()
   {
   }

   public void run()
   {
      while (!gnutWare.exit)
      {
         try
         {
            synchronized (gnutWare.msg_tosend)
            {
               if (!gnutWare.msg_tosend.isEmpty())
               {
                  byte[] m = (byte[]) gnutWare.msg_tosend.remove(0);
                  String s = Util.byteArray2string(m);
                  int mlen = s.length();
                  gnutWare.os.write(m, 0, mlen);
                  gnutWare.os.flush();
               }
            }
            Thread.sleep(1);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
   }
}


class Receive_Message extends Thread
{
   Receive_Message()
   {
   }

   public void run()
   {
      while (!gnutWare.exit)
      {
         try
         {
            if (gnutWare.is.available() > 0)
            {
               receive();
            }
            Thread.sleep(1);
         }
         catch (IOException e)
         {}
         catch (InterruptedException e)
         {}
      }
   }

   public void receive()
   {
```

```
      try
      {
         byte[] b = new byte[65535];
         long off, len;
         long wish_len, read_len, read_len_sum;

         //Comienzo al leer el header
         wish_len = Message.HEADER_LEN;
         read_len_sum = 0;
         read_len = 0;

         while (read_len_sum < wish_len && read_len != -1 &&
!gnutWare.exit)
         {
            if (wish_len > b.length)
            {
               read_len = gnutWare.is.skip(wish_len - read_len);
            }
            else
            {
               off = 0 + read_len;
               len = wish_len - read_len;
               read_len = gnutWare.is.read(b , (int)off, (int)len);
            }
            read_len_sum += read_len;
         }

         if (gnutWare.exit)
         {
            return;
         }

         if (read_len == -1)
         {
            System.out.println("Error en lectura de mensaje!");
            return;
         }
         //Finalizo lectura del header
         //Comienzo a leer el payload
         wish_len = new Message(b).getPayloadLength();
         if (wish_len > b.length)
         {
            System.out.println("Mensaje omitido por longitud: " +
wish_len);
         }
         //
         read_len_sum = 0;
         read_len = 0;
         while (read_len_sum < wish_len && read_len != -1 &&
!gnutWare.exit)
         {
            if (wish_len > b.length)
            {
               read_len = gnutWare.is.skip(wish_len - read_len);
            }
            else
            {
               off = Message.HEADER_LEN + read_len;
               len = wish_len - read_len;
               read_len = gnutWare.is.read(b , (int)off, (int)len);
```

```
            }
            read_len_sum += read_len;
         }

         if (gnutWare.exit)
         {
            return;
         }

         if (read_len == -1)
         {
            System.out.println("Error en lectura de mensaje!");
            return;
         }
         //Finalizo lectura del payload
         received(b);
      }
      catch (java.net.SocketException e)
      {
         if (gnutWare.exit)
         {
              System.out.println("Conexion perdida!");
         }
         else
         {
            e.printStackTrace();
         }
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }


   public static void received(Message message)
   {
      received(message.toByteArray());
   }

   public static void received(byte[] message)
   {
      synchronized (gnutWare.msg_received)
      {
         gnutWare.msg_received.add(message);
      }
   }
}

/*
Clase: Terminal_Control
Permite el ingreso de una cadena por terminal (consola). Si es "exit"
setea una bandera de finalización del programa, sino, arma un mensaje
ServiceMessage y lo envía a la red. En este caso utiliza el código de
servicio 00 y de identificación de servicio 0000.
*/
class Terminal_Control extends Thread
{
   BufferedReader input;
```

```
   Terminal_Control()
   {
      input = new BufferedReader(new InputStreamReader(System.in));
   }

   public void run()
   {
      try
      {
         while (true)
         {
            String cadena = input.readLine();

            if (cadena.equals("exit"))
            {
               gnutWare.exit = true;
            }
            else
            {
               gnutWare.send(new ServiceMessage(gnutWare.guid, 00,
0000, cadena));
               System.out.println("[Terminal_Control] Se envió un
mensaje por terminal...");
            }
         }
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }

   }
}

/*
Clase: Network_Sap
Permite prestar sevicio a cualquier aplicación que quiera enviar y
recibir mensaje a través de la red p2p de servents gnut. Acepta
conexiones entrantes en el puerto 4000 e intercambia mensajes entre la
aplicación y el gnut que brinda el punto de acceso a la red.
Esta versión opera de modo iterativo, pero futuras versiones debe
operar de modo concurrente, multiplexando por servicio.
*/
class Network_Sap extends Thread
{
   int port;
   ServerSocket server_socket;
   public static PrintStream nsos;
   public static DataInputStream nsis;

   Network_Sap()
   {
      try
      {
         server_socket = new ServerSocket(gnutWare.listen_port);
         System.out.println("[Network_Sap] Esperando un cliente en
puerto " + server_socket.getLocalPort());
      }
      catch (Exception e)
      {
```

```
            System.out.println(e);
         }
      }

      public void run()
      {
         try
         {
            // Loop infinito
            while(true)
            {
               Socket socket = server_socket.accept();
               System.out.println("[Network_Sap] Conexion acceptada: " +
socket.getInetAddress() + ":" + socket.getPort());

               nsos = new PrintStream(socket.getOutputStream());
               nsis = new DataInputStream(socket.getInputStream());

               byte b[] = new byte[65535];
               int bdis;
               int blen;

               try
               {
                  nsis.skip("GNUTWARE CONNECT\n\n".length());

                  b = Util.string2byteArray("GNUTWARE SERVICE
CODE?\n\n");
                  nsos.write(b);

                  //Leo código de servicio
                  bdis = 0;
                  while (bdis < 2)
                  {
                     bdis = nsis.available();
                  }
                  blen = nsis.read(b, 0, 2);
                  byte sc[] = new byte[blen];
                  System.arraycopy(b, 0, sc, 0, blen);
                  String service_code = Util.byteArray2string(sc);

                  Util.byteArray2string(sc);

                  System.out.println("[Network_Sap] Service code:*" +
service_code + "*");

                  b = Util.string2byteArray("GNUTWARE SERVICE ID?\n\n");
                  nsos.write(b);

                  //Leo ID
                  bdis = 0;
                  while (bdis < 4)
                  {
                     bdis = nsis.available();
                  }
                  blen = nsis.read(b, 0, 4);
                  byte si[] = new byte[blen];
                  System.arraycopy(b, 0, si, 0, blen);
                  String service_id = Util.byteArray2string(si);
```

```
                System.out.println("[Network_Sap] Service id:*" +
service_id + "*");

                int nsc = Integer.parseInt(service_code);
                int nsi = Integer.parseInt(service_id);

                //Leo los mensajes
                String caracter = "";
                String service_msg = "";
                boolean salir;
                boolean salir_total;

                salir_total = false;
                while (!salir_total)
                {
                   salir = false;
                   while (!salir)
                   {
                      try
                      {
                         byte q[] = new byte[65535];
                         blen = nsis.read(q, 0, 1);
                         byte sm[] = new byte[blen];
                         System.arraycopy(q, 0, sm, 0, blen);
                         caracter = Util.byteArray2string(sm);

                         if (caracter.equals("\n"))
                         {
                            salir = true;
                         }
                         else
                         {
                            service_msg = service_msg + caracter;
                         }
                         bdis = nsis.available();
                      }
                      catch (Exception e)
                      {
                         System.out.println(e);
                         salir = true;
                         salir_total = true;
                      }
                   }

                   System.out.println("[gnutWare <- p2pTime] *" +
service_msg + "*");

                   String msgs = service_msg.substring(0, 2);

                   if (msgs.equals("QY"))
                   {
                      System.out.println("-----------------------------
-----------------------------------------");
                      System.out.println("[gnutWare QUERY -> p2p] *" +
service_msg + "*");
                      gnutWare.send(new ServiceMessage(gnutWare.guid,
nsc, nsi, service_msg));
                   }
                   else
                   {
```

```
                    String ctrl = service_msg;
                    gnutWare.send(new
ServiceResponseMessage(gnutWare.rcv_guid, ctrl));

                    System.out.println("----------------------------
----------------------------------------");
                    System.out.println("[gnutWare: QUERY_HIT -> p2p]
*" + ctrl + "*");
                 }
                 service_msg = "";
              }
           }
           catch (Exception e)
           {
              System.out.println(e);
           }
        }
     }
     catch (Exception e)
     {
        System.out.println(e);
     }
  }
}
```

Clase GUID.java

```
import java.util.Random;
import java.util.Date;

public class GUID
{
   protected byte[] guid_ = new byte[16];

   public GUID()
   {
      Random r = new Random((new Date()).getTime());
      for (int i = 0; i < guid_.length; i++)
      {
         guid_[i] = (byte)(r.nextInt(256) - 128);
      }
   }

   public GUID(byte[] guid)
   {
      System.arraycopy(guid, 0, guid_, 0, guid_.length);
   }

   public byte[] toByteArray()
   {
      return guid_;
   }

   public String getGUID()
   {
      return Util.byteArray2string(guid_);
   }
}
```

Clase Message.java

```
import java.lang.StringBuffer;

public class Message
{
   protected byte[] message_;

   public Message(byte[] message)
   {
      message_ = new byte[message.length];
      System.arraycopy(message, 0, message_, 0, message.length);
   }

   public GUID getGUID()
   {
      if (message_.length < HEADER_LEN)
      {
         return null;
      }
      byte[] guid = new byte[GUID_LEN];
      System.arraycopy(message_, GUID_POS, guid, 0, GUID_LEN);
      return new GUID(guid);
   }

   public void setGUID(GUID guid)
   {
      if (message_.length < HEADER_LEN)
      {
         return;
      }
      System.arraycopy(guid, 0, message_, GUID_POS, GUID_LEN);
   }

   public byte getPayloadID()
   {
      if (message_.length < HEADER_LEN)
      {
         return 0;
      }
      return message_[PAYLOAD_ID_POS];
   }

   public void setPayloadID(byte id)
   {
      if (message_.length < HEADER_LEN)
      {
         return;
      }
      message_[PAYLOAD_ID_POS] = id;
   }

   public int getTTL()
   {
      if (message_.length < HEADER_LEN)
      {
         return 0;
      }
      return (int)message_[TTL_POS];
   }
```

```
public void setTTL(int ttl)
{
   if (message_.length < HEADER_LEN)
   {
      return;
   }
   message_[TTL_POS] = (byte)ttl;
}

public int getHops()
{
   if (message_.length < HEADER_LEN)
   {
      return 0;
   }
   return (int)message_[HOPS_POS];
}

public void setHops(int hops)
{
   if (message_.length < HEADER_LEN)
   {
      return;
   }
   message_[HOPS_POS] = (byte)hops;
}

public long getPayloadLength()
{
   if (message_.length < HEADER_LEN)
   {
      return 0;
   }
   byte[] len = new byte[PAYLOAD_LEN_LEN];
   System.arraycopy(message_, PAYLOAD_LEN_POS, len, 0,
   PAYLOAD_LEN_LEN);
   return Util.byteArray2int(len);
}

public void setPayloadLength(int len)
{
   if (message_.length < HEADER_LEN)
   {
      return;
   }
   byte[] len_b = Util.int2byteArray(len);
   System.arraycopy(len_b, 0, message_, PAYLOAD_LEN_POS,
   PAYLOAD_LEN_LEN);
}

public String getPayload()
{
   String payload;

   if (message_.length < HEADER_LEN)
   {
      payload = "";
   }
   else
```

```
      {
         int PAYLOAD_LEN = (int) this.getPayloadLength();
         byte[] payload_b = new byte[PAYLOAD_LEN];
         System.arraycopy(message_, 23, payload_b, 0, PAYLOAD_LEN);
         payload = Util.byteArray2string(payload_b);
      }
      return payload;
   }

   public byte[] toByteArray()
   {
      return message_;
   }
}
```

Clase PingMessage.java

```
import java.io.ByteArrayOutputStream;

public class PingMessage extends Message
{
   public PingMessage()
   {
      super();
      message_ = this.create();
   }

   public PingMessage(byte[] message)
   {
      super(message);
   }

   public static byte[] create()
   {
      ByteArrayOutputStream bos = new ByteArrayOutputStream();
      try
      {
         GUID tmp_guid = new GUID();
         // header
         byte[] guid = tmp_guid.toByteArray();
         bos.write(guid);
         byte[] fid = { PING };
         bos.write(fid);
         byte[] ttl = { 0x7 };
         bos.write(ttl);
         byte[] hops = { 0x0 };
         bos.write(hops);
         byte[] payloadLength = { 0x0, 0x0, 0x0, 0x0 };
         bos.write(payloadLength);
         // payload
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
      return bos.toByteArray();
   }
}
```

Clase PongMessage.java

```
import java.io.ByteArrayOutputStream;

public class PongMessage extends Message
{
  public static final int MIN_PAYLOAD_LEN = 14;
  public static final int PORT_POS        = 23;
  public static final int PORT_LEN        =  2;
  public static final int INET_ADDR_POS   = 25;
  public static final int INET_ADDR_LEN   =  4;
  public static final int FILE_COUNT_POS  = 29;
  public static final int FILE_COUNT_LEN  =  4;
  public static final int SHARED_SIZE_POS = 33;
  public static final int SHARED_SIZE_LEN =  4;

  public PongMessage(GUID guid_)
  {
    super();
    message_ = this.create(guid_);
  }

  public PongMessage(byte[] message)
  {
    super(message);
  }

  public static byte[] create(GUID guid_)
  {
      ByteArrayOutputStream bos = new ByteArrayOutputStream();
      try
      {
         GUID tmp_guid = guid_;
         byte[] guid = tmp_guid.toByteArray(); bos.write(guid);
         byte[] fid = { PONG }; bos.write(fid);
         byte[] ttl = { 0x7 }; bos.write(ttl);
         byte[] hops = { 0x0 }; bos.write(hops);
         byte[] payloadLength = { 0x0E, 0x0, 0x0, 0x0 };
         bos.write(payloadLength);
         byte[] payload = { 0x2, 0x10, 0x10, 0x15, 0x1, 0x12, 0x14,
         0x0, 0x0, 0x0, 0x32, 0x0, 0x0, 0x0 };

         bos.write(payload);
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
      return bos.toByteArray();
   }

   public int getPort()
   {
      if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
      {
         return 0;
      }
```

```
      byte[] b = new byte[PORT_LEN];
      System.arraycopy(message_, PORT_POS, b, 0, b.length);
      return Util.byteArray2int(b);
   }

   public void setPort(int port)
   {
   }

   public String getIPAddress()
   {
      if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
      {
         return null;
      }
      byte[] b = new byte[INET_ADDR_LEN];
      System.arraycopy(message_, INET_ADDR_POS, b, 0, b.length);
      return Util.byteArray2address(b);
   }

   public void setIPAddress()
   {
   }

   public int getFileCount()
   {
      if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
      {
         return 0;
      }
      byte[] b = new byte[FILE_COUNT_LEN];
      System.arraycopy(message_, FILE_COUNT_POS, b, 0, b.length);
      return Util.byteArray2int(b);
   }

   public void setFileCount(int fileCount)
   {
   }

   public int getSharedSize()
   {
      if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
      {
         return 0;
      }
      byte[] b = new byte[SHARED_SIZE_LEN];
      System.arraycopy(message_, SHARED_SIZE_POS, b, 0, b.length);
      return Util.byteArray2int(b);
   }

   public void setSharedSize()
   {
   }
}
```

Clase ServiceMessage.java

```
import java.io.ByteArrayOutputStream;

public class ServiceMessage extends Message
```

```
{
   public static final int MIN_PAYLOAD_LEN  = 14;
   public static final int MIN_SPEED_POS    = 23;
   public static final int MIN_SPEED_LEN    =  2;
   public static final int PARAM_KEY_POS    = 25;
   public static final int DELIMITER_LEN    =  1;
   public static final int SERVICE_CODE_LEN =  4;
   public static final int SESION_ID_LEN    =  4;
   public static final int PARAM_END_LEN    =  1;

   public ServiceMessage()
   {
      GUID dummy = new GUID();
      message_ = this.create(dummy, 0, 0, "");
   }

   public ServiceMessage(byte[] message)
   {
      super(message);
   }

   public ServiceMessage(GUID guid, int scode, int sid, String param)
   {
      message_ = this.create(guid, scode, sid, param);
   }

   public static byte[] create(GUID xguid, int xscode, int xsid,
String xparam)
   {
      ByteArrayOutputStream bos = new ByteArrayOutputStream();
      try
      {
         GUID tmp_guid = new GUID();
         System.out.println("QUERY sale con GUID: *" +
         Util.byteArray2string(tmp_guid.toByteArray()) + "*");

         // header
         byte[] guid = tmp_guid.toByteArray(); bos.write(guid);
         byte[] fid = { QUERY }; bos.write(fid);
         byte[] ttl = { 0x7 }; bos.write(ttl);
         byte[] hops = { 0x0 }; bos.write(hops);
         byte[] payloadLength = Util.int2byteArray(xparam.length()
                                                   + MIN_SPEED_LEN
                                                   + SERVICE_CODE_LEN
                                                   + DELIMITER_LEN
                                                   + SESION_ID_LEN
                                                   + PARAM_END_LEN
                                                   );
         bos.write(payloadLength);
         byte[] minSpeed_ = { 0x0, 0x0 }; bos.write(minSpeed_);
         byte[] param = Util.string2byteArray(xparam);
         bos.write(param);
         byte[] delimiter = Util.string2byteArray("~");;
         bos.write(delimiter);
         byte[] scode = new byte[4];
         scode = Util.int2byteArray(xscode);
         bos.write(scode);
         byte[] sid   = new byte[4];
         sid = Util.int2byteArray(xsid);
         bos.write(sid);
```

```
            bos.write(0x0);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
         return bos.toByteArray();
      }

      public int getMinSpeed()
      {
         if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
         {
            return 0;
         }
         byte[] b = new byte[MIN_SPEED_LEN];
         System.arraycopy(message_, MIN_SPEED_POS, b, 0, b.length);
         return Util.byteArray2int(b);
      }

      public void setMinSpeed(int minSpeed)
      {
         if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
         {
            return;
         }
         byte[] b = Util.int2byteArray(minSpeed);
         message_[MIN_SPEED_POS]     = b[0];
         message_[MIN_SPEED_POS + 1] = b[1];
      }
   }
```

Clase ServiceResponseMessage.java

```
import java.io.ByteArrayOutputStream;
import java.net.URL;
import java.util.Set;
import java.util.HashSet;
import java.util.Vector;

public class ServiceResponseMessage extends Message
{
   public static final int MIN_PAYLOAD_LEN = 27;
   public static final int HIT_COUNT_POS   = 23;
   public static final int HIT_COUNT_LEN   =  1;
   public static final int PORT_POS        = 24;
   public static final int PORT_LEN        =  2;
   public static final int INET_ADDR_POS   = 26;
   public static final int INET_ADDR_LEN   =  4;
   public static final int SPEED_POS       = 30;
   public static final int SPEED_LEN       =  4;
   public static final int SHARED_INFO_POS = 34;

   public ServiceResponseMessage(GUID tmp_guid, String response)
   {
         super();
         message_ = this.create(tmp_guid, response);
   }

   public ServiceResponseMessage( byte[] message)
```

```
{
      super(message);
}

public static byte[] create(GUID tmp_guid, String response)
{
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   try
      {
         //Header
         byte[] guid = tmp_guid.toByteArray(); bos.write(guid);
         byte[] fid = { QUERY_HITS }; bos.write(fid);
         byte[] ttl = { 0x7 }; bos.write(ttl);
         byte[] hops = { 0x1 }; bos.write(hops);
         byte[] payloadLength = Util.int2byteArray(37 +
         response.length());
         bos.write(payloadLength);

         byte[] payload1 = {0x1,                      //Hits  (1)
                            0x10, 0x10,               //Port  (2)
                            0x15, 0x1, 0x12, 0x14,    //IP    (4)
                            0x32, 0x0, 0x0, 0x0,      //Speed (4)
                            0x1, 0x0, 0x0, 0x0,       //Index (4)
                            0x18, 0x0A, 0x0, 0x0,     //F.Size(4)
         byte[] payload2 = Util.string2byteArray(response);
         byte[] payload3 = {0x0, 0x0,                 //Delimiter
                            0x5, 0x6, 0x7, 0x8,
                            0x9, 0x10, 0x11, 0x12,
                            0x13, 0x14, 0x15, 0x16 };
         bos.write(payload1);
         bos.write(payload2);
         bos.write(payload3);
}
   catch (Exception e)
   {
      e.printStackTrace();
   }
   return bos.toByteArray();
}

public int getHitCount()
{
   if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
   {
      return 0;
   }
   return message_[HIT_COUNT_POS] & 0xff;
}

public int getPort()
{
   if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
   {
      return 0;
   }
   byte[] b = new byte[PORT_LEN];
   System.arraycopy(message_, PORT_POS, b, 0, b.length);
   return Util.byteArray2int(b);
}
```

```
public String getIPAddress()
{
   if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
   {
      return null;
   }
   byte[] b = new byte[INET_ADDR_LEN];
   System.arraycopy(message_, INET_ADDR_POS, b, 0, b.length);
   return Util.byteArray2address(b);
}

public int getSpeed()
{
   if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
   {
      return 0;
   }
   byte[] b = new byte[SPEED_LEN];
   System.arraycopy(message_, SPEED_POS, b, 0, b.length);
   return Util.byteArray2int(b);
}

public GUID getClientGUID()
{
   if (message_.length < HEADER_LEN + MIN_PAYLOAD_LEN)
   {
      return null;
   }
   byte[] b = new byte[16];
   System.arraycopy(message_, message_.length - b.length - 1, b, 0,
   b.length);
   return new GUID(b);
}

public byte[] getResult()
{
   int plen = (int) getPayloadLength();
   int rlen = plen - MIN_PAYLOAD_LEN;
   byte[] b = new byte[rlen];
   System.arraycopy(message_, SHARED_INFO_POS, b, 0, rlen);
   return b;
}

public String getResult(int row, int col)
{
   byte[] rs = getResult();
   int hits = getHitCount();
   int size = 0;
   int index= 0;
   int q = 0;
   int p = 0;
   int i = 0;
   String file = "";
   boolean salir = false;

   try
   {
      for (; hits > 0; hits--)
      {
```

```
                index = Util.bytes2int(rs[p++], rs[p++], rs[p++],
                rs[p++]);
             size  = Util.bytes2int(rs[p++], rs[p++], rs[p++],
             rs[p++]);
             q = p;
             i = 0;

             while (true)
             {
                if (rs[p] == 0x0)
                {
                   p++;
                   p++;
                   break;
                }
                else
                {
                   i++;
                }
                p++;
             }
             byte[] f = new byte[i];
             System.arraycopy(rs, q, f, 0, i);
             file = Util.byteArray2string(f);

             if (index == row)
             {
                break;
             }
          }
       }
       catch (Exception e)
       {
          e.printStackTrace();
       }

       String s = file;

       if (col == 1)
       {
          s = Integer.toString(index);
       }
       else if (col == 2)
       {
          s = Integer.toString(size);
       }
       return s;
    }
}
```

Anexo IV

Código Fuente de IndiSE

Clase IndiSE.java

```
import java.io.*;
import java.net.*;
import java.lang.*;
import java.text.*;
import java.util.*;

public class IndiSE
{
   //Objeto de recuperación de parámetros de configuración
   public static IndiSE_Cfg_Reader cfg;
   //Propiedades para mensajes de I/O con gnutWare
   public static String cadenaSnd;
   public static String cadenaRcv;
   //Streams para I/O con gnutWare
   public static DataOutputStream os;
   public static DataInputStream is;
   //String para da la cadena de respuesta de un QH
   public static String queryHitRcv;
   boolean exit;

   public static void main(String args[])
   {
      try
      {
         String ipaddr = args[0];
         int port = Integer.parseInt(args[1]);

         //Se leen los parámetros de configuración en el objeto cfg
         cfg = new IndiSE_Cfg_Reader("IndiSE.conf");

         //Abro una conexión con el gnutWare
         Socket clientSocket = new Socket(ipaddr, port);
         System.out.println("Conexion: " + clientSocket);

         String service_code = "80";   // Se define que el 80 es
                                       "Búsquedas Distribuidas"
         String service_id  = "8000";

         connectGnutWare(clientSocket, service_code, service_id);

         /*
         Ya se abrió una conexión con el middleware gnutWare. Por ésta
         se pueden enviar ó recibir solicitudes de búsqueda.
         */
         cadenaSnd = "";
         cadenaRcv = "";
         queryHitRcv = "";
```

```
new Send_Query().start();
new Receive_Query().start();
new Search_Interface().start();

/*
Comienzo del ciclo principal del programa que chequea por
mensajes entrantes desde la red y actúa según corresponda
*/
while (true)
{
   synchronized (cadenaRcv)
   {
      if (!cadenaRcv.equals(""))
      {
         String xRcv = cadenaRcv;
         cadenaRcv = "";

         String comm = xRcv.substring(0, 2);
         String prms = xRcv.substring(3, xRcv.length());

         if (comm.equals("QY"))
         {
            /*
            Aquí se pueden aplicar políticas sobre qué buscar
            y dónde buscarlo (por ejemplo en BD privadas)
            */
            if (prms.equals(""))
            {
                  //Nada que buscar, la cadena de QUERY está
                  vacia.
            }
            else
            {
               synchronized (cadenaSnd)
               {
                  /*
                  Aquí recibí un query desde la red p2p,
                  entonces ejecuto la búsqueda
                  sobre el web server a través de la
                  interface localsearch.pl
                  Si hay resultados, se genera un ticket con
                  información acerca de la
                  respuesta y se envía de vuelta
                  */

                  System.out.println("QUERY recibido con: " +
                  prms);

                  int posi = prms.indexOf("\t");
                  String rcQuery = prms.substring(0, posi);
                  String rcPort  = prms.substring(posi + 1,
                  posi + 5);

                  System.out.println("Conectandose a
                  localsearch en: " + cfg.localSearchIP + ":"
                  + cfg.localSearchPort);

                  LocalSearch ls = new
                  LocalSearch(cfg.localSearchIP,
```

```
                        cfg.localSearchPort, rcQuery,
                        cfg.maxLocalSearchWait);
                        String respuesta = ls.getResponse();

                        if (!respuesta.equals(""))
                        {
                           Ticket t = new Ticket(rcPort);
                           String filename = t.getTicket();
                           String serverIP = cfg.webServerIP;
                           int serverPort = cfg.webServerPort;

                           //A la respuesta le agrego un encabezado
                           de identificación y un cierre

                          t.save(cfg.webServerRoot +
                          cfg.webServerDir, filename, respuesta);

                           /*
                           Una vez guardado el archivo con las
                           respuestas retorno
                           el ticket que permitirá recuperarlas vía
                           HTTP
                           */

                           String msgQH = serverIP + "*" +
                           serverPort + "*" + filename;
                           cadenaSnd = "QH:" + msgQH + "\n";
                        }
                     }
                  }
               }
               else
               {
                  //La respuesta recibida es un ticket que guardo
                  en una variable global para IPC
                  System.out.println("----------------------------
                  ------------------------------------------------
                  -");
                  System.out.println("RESPUESTA RECIBIDA");
                  queryHitRcv = prms;
                  System.out.println("queryHitRcv = " + prms);
                  System.out.println("----------------------------
                  ------------------------------------------------
                  -");
               }
            }
         }
         Thread.sleep(1);
      }     }
   catch (Exception e)
   {
      System.out.println(e);
   }
}

public static void connectGnutWare(Socket clientSocket, String sc,
String si)
{
   try
   {
```

```
            clientSocket.setTcpNoDelay(true);
            is = new DataInputStream(clientSocket.getInputStream());
            os = new DataOutputStream(clientSocket.getOutputStream());

            byte[] b = Util.string2byteArray("GNUTWARE CONNECT\n\n");
            os.write(b);

            is.skip("GNUTWARE SERVICE CODE?\n\n".length());

            b = Util.string2byteArray(sc);
            os.write(b);

            is.skip("GNUTWARE SERVICE ID?\n\n".length());

            b = Util.string2byteArray(si);
            os.write(b);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
   }

   class Send_Query extends Thread
   {
      Send_Query()
      {
      }

      public void run()
      {
         while(true)
         {
            synchronized (IndiSE.cadenaSnd)
            {
               if (!IndiSE.cadenaSnd.equals(""))
               {
                  try
                  {
                     byte[] b = Util.string2byteArray(IndiSE.cadenaSnd);
                     IndiSE.os.write(b);
                     IndiSE.cadenaSnd = "";
                  }catch (Exception e){}
               }
            }
            try
            {
               Thread.sleep(1);
            }
            catch (Exception e)
            {
            }
         }
      }
   }

   class Receive_Query extends Thread
   {
      Receive_Query() {}
```

```
public void run()
{
   int bdis;
   int blen;

   String caracter;
   boolean salir = false;
   boolean salir_total = false;

   while (!salir_total)
   {
      int i = 0;
      String msg = "";

      salir = false;
      while (!salir)
      {
         try
         {
            byte b[]  = new byte[65535];
            blen = IndiSE.is.read(b, 0, 1);
            byte r[] = new byte[blen];
            System.arraycopy(b, 0, r, 0, blen);
            caracter = Util.byteArray2string(r);

            if (caracter.equals("\n"))
            {
               salir = true;
            }
            else
            {
               msg = msg + caracter;
            }
            bdis = IndiSE.is.available();
         }
         catch (Exception e)
         {
            System.out.println(e);
            salir = true;
            salir_total = true;
         }
      }
      i++;

      synchronized (IndiSE.cadenaRcv)
      {
         IndiSE.cadenaRcv = msg;
      }

      try
      {
         Thread.sleep(1);
      }
      catch (Exception e)
      {
         salir_total = true;
      }
   }
}
```

```
}

/*
Clase: Search_Interface
Acepta una conexión TCP en un puerto (de una script CGI, normalmente)
y dirige la búsqueda en dos direcciones: primero, genera un mensaje
QUERY, para enviar a la red de pares de manera distribuida y también
realiza la búsqueda en la base de datos local usando el script
LocalSearch.pl
*/

class Search_Interface extends Thread
{
   //Streams para I/O con la interface localsearch.pl
   public DataOutputStream bos;
   public DataInputStream bis;

   int port = IndiSE.cfg.searchInterfacePort;
   ServerSocket ss;

   public void run()
   {
      try
      {
         ss = new ServerSocket(port);
      }
      catch (Exception e){}

      System.out.println("Search interface inicializada...");

      while (true)
      {
         try
         {
            String localResponse = "";

            Socket s = ss.accept();
            InputStream is = new
            BufferedInputStream(s.getInputStream());
            PrintStream ps = new PrintStream(s.getOutputStream());
            s.setTcpNoDelay(true);

            System.out.println ("Cliente conectado: " + s.getPort());

            int serverPort = 0;
            String serverIP = "";
            String filename = "";

            int bdis;
            int blen;
            String rs = "";
            byte b[] = new byte[65535];

            try
            {
               while (rs.equals(""))
               {
                  bdis = is.available();
                  while (bdis > 1)
                  {
```

```
      if (bdis == 0) {bdis = 1;}
      blen = is.read(b, 0, bdis - 1);
      byte r[] = new byte[blen];
      System.arraycopy(b, 0, r, 0, blen);
      bdis = is.available();
      System.arraycopy(b, 0, r, 0, blen);
      rs = rs + Util.byteArray2string(r);
      System.out.println("rs: " + rs);
      IndiSE.cadenaSnd = "QY:" + rs + "\t" +
      s.getPort() +"\n";
   }
}

/*
Busco en el localsearch local. Esto debe ser un thread?
Inicio un objeto fecha para calcular la cantidad de
segundos máxima que puede esperar. El query ya fue a la
red, por lo tanto a la espera total le resto los
segundos que tardó la búsqueda local
*/

Date d0 = new Date();

System.out.println("Conectandose a localsearch en: " +
IndiSE.cfg.localSearchIP + ":" +
IndiSE.cfg.localSearchPort);

LocalSearch ls = new
LocalSearch(IndiSE.cfg.localSearchIP,
IndiSE.cfg.localSearchPort, rs,
IndiSE.cfg.maxLocalSearchWait);

String respuesta = ls.getResponse();

if (!respuesta.equals(""))
{
   serverPort = IndiSE.cfg.webServerPort;
   Ticket t = new Ticket(Integer.toString(serverPort));
   filename = t.getTicket();
   serverIP = IndiSE.cfg.webServerIP;
   //A la respuesta le agrego un encabezado de
   identificación y un cierre
   t.save(IndiSE.cfg.webServerRoot +
   IndiSE.cfg.webServerDir, filename, respuesta);
   /*
   Una vez guardado el archivo con las respuestas
   retorno el ticket que permitirá recuperarlas vía HTTP
   */
   localResponse = serverIP + "*" + serverPort + "*" +
   filename;
}

/*
Este es el ticket de la búsqueda local. Recupero el
archivo via HTTP porque la búsqueda local no
necesariamente corre en el mismo servidor
*/

System.out.println("Recuperando: " + serverIP + ":" +
serverPort + "" + IndiSE.cfg.webServerDir + filename);
```

```
HttpClient lhc = new HttpClient(serverIP, serverPort,
IndiSE.cfg.webServerDir + filename);

//Aquí se realiza el parsing del archivo XML (contenido
en la string
//ResponseText del objeto hc. Con ésto se arma la
página de respuesta.

Ticket_Parser ltp = new
Ticket_Parser(lhc.getResponse());
ltp.parse();

ps.print(ltp.HTML_PrintServer());
ps.print(ltp.HTML_PrintLinks());
ps.print("<hr>");

lhc = null;
ltp = null;

/*
Acá se pone en espera por respuestas durante 30
segundos. Cada respuesta será un ticket, que ingresa de
la red al Hash queryHitRcv y debe ser recuperado
mediante un objeto Http_Client. Luego, su contenido en
XML (parsing mediante) es impreso al socket con el
cliente (ds.cgi)
*/

String recv = "";
Date d1 = new Date();

long secondsWait = ((d1.getTime() - d0.getTime()) /
1000);

System.out.println("Respuesta local recibida en: " +
secondsWait + "s");

while (secondsWait < IndiSE.cfg.maxp2pNetWait)
{
   int v = 0;
   while (recv.equals("") && secondsWait <
   IndiSE.cfg.maxp2pNetWait)
   {
      Date d2 = new Date();
      secondsWait = ((d2.getTime() - d0.getTime()) /
      1000);
      synchronized(IndiSE.queryHitRcv)
      {
         recv = IndiSE.queryHitRcv;
      }
   }

   System.out.println("Respuesta recibida en: " +
   secondsWait + "s");

   recv = IndiSE.queryHitRcv;
   IndiSE.queryHitRcv = "";

   if (!recv.equals(""))
   {
```

```
                        // Vino un ticket, entonces lo recupero del web
                        server
                        int ippos;
                        String restoRecv;
                        ippos = recv.indexOf("*");
                        serverIP = recv.substring(0, ippos);
                        restoRecv = recv.substring(ippos + 1,
                        recv.length());
                        ippos = restoRecv.indexOf("*");
                        serverPort =
                        Integer.parseInt(restoRecv.substring(0, ippos));
                        restoRecv = restoRecv.substring(ippos + 1,
                        restoRecv.length());
                        filename = restoRecv.substring(0,
                        restoRecv.length());

                        System.out.println("Recuperando: " + serverIP +
                        ":" + serverPort + "" + IndiSE.cfg.webServerDir +
                        filename);
                        HttpClient rhc = new HttpClient(serverIP,
                        serverPort, IndiSE.cfg.webServerDir + filename);

                        Ticket_Parser rtp = new
                        Ticket_Parser(rhc.getResponse());

                        rtp.parse();
                        ps.print(rtp.HTML_PrintServer());
                        ps.print(rtp.HTML_PrintLinks());
                        ps.print("<hr>");

                        rhc = null;
                        rtp = null;
                     }
                  }
                  ps.print ("\n\n");
                  ps.close();
               }
               catch(Exception e)
               {
                  System.out.println(e);
               }
            }
            catch(Exception e)
            {
               System.out.println(e);
            }
         }
      }
   }
```

Clase IndiSE_CFG_Reader.java

```
import java.io.*;
import java.util.*;

public class IndiSE_Cfg_Reader
{
   public String localSearchIP;
   public int localSearchPort;
```

```
public String webServerIP;
public int webServerPort;
public String webServerRoot;
public String webServerDir;
public int maxLocalSearchWait;
public int searchInterfacePort;
public int maxp2pNetWait;

public IndiSE_Cfg_Reader(String file)
{
   try
   {
      BufferedReader infile = new BufferedReader(new
      FileReader(file));

      while (infile.ready())
      {
         String line = infile.readLine();

         if (!line.equals("") && !line.substring(0, 1).equals("#"))
         {
            int posi = line.indexOf("=");
            String cfgKey = line.substring(0, posi);
            String cfgValue = line.substring(posi + 1,
            line.length());

            if (cfgKey.equals("localSearchIP"))
            {
                  localSearchIP = cfgValue;
            }
            else if (cfgKey.equals("localSearchPort"))
            {
               localSearchPort = Integer.parseInt(cfgValue);
            }
            else if (cfgKey.equals("webServerIP"))
            {
                  webServerIP = cfgValue;
            }
            else if (cfgKey.equals("webServerPort"))
            {
                  webServerPort = Integer.parseInt(cfgValue);
            }
            else if (cfgKey.equals("webServerRoot"))
            {
                  webServerRoot = cfgValue;
            }
            else if (cfgKey.equals("webServerDir"))
            {
                  webServerDir = cfgValue;
            }
            else if (cfgKey.equals("maxLocalSearchWait"))
            {
                  maxLocalSearchWait = Integer.parseInt(cfgValue);
            }
            else if (cfgKey.equals("searchInterfacePort"))
            {
                  searchInterfacePort = Integer.parseInt(cfgValue);
            }
            else if (cfgKey.equals("maxp2pNetWait"))
            {
```

```
                        maxp2pNetWait = Integer.parseInt(cfgValue);
                    }
                    else
                    {
                        System.out.println("Parámetro no reconocido: " +
                        cfgKey);
                    }
                }
            }
            infile.close();
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Clase LocalSearch.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class LocalSearch
{
    //Socket cliente para conexión con la interface localsearch.pl
    Socket searchSocket;
    //Streams para I/O con la interface localsearch.pl
    public DataOutputStream bos;
    public DataInputStream bis;
    //Propiedad que almacena la respuesta
    public String respData;
    //Propiedad que determina si hay respuesta lista
    public boolean respReady;
    //Propiedad que almacena el tiempo máximo de espera por una
    respuesta
    public int maxLocalSearchWait;

    public LocalSearch(String ip, int port, String query, int
    secondsWait)
    {
        maxLocalSearchWait = secondsWait;
        respData = "";
        respReady = false;

        try
        {
            searchSocket = new Socket(ip, port);
            connectLocalSearch(searchSocket);
            //Envío la consulta denominada "local"
            byte[] q = Util.string2byteArray(query + "\n\n");
            bos.write(q);
            //Recibo la respuesta "local"
            String respuesta = readSocket(bis);
            //System.out.println("Resp: " + respuesta);
            searchSocket.close();
        }
        catch (Exception e)
```

```
   {
      e.printStackTrace();
   }
}

public void connectLocalSearch(Socket searchSocket)
{
   try
   {
      searchSocket.setSoTimeout(7000);
      searchSocket.setTcpNoDelay(true);
      bis = new DataInputStream(searchSocket.getInputStream());
      bos = new DataOutputStream(searchSocket.getOutputStream());
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
}

public String readSocket(DataInputStream xbis)
{
   int bdis;
   int blen;
   String rs = "";
   byte b[] = new byte[65535];

   try
   {
      long secondsWait = 0;
      Date d1 = new Date();

      while (rs.equals("") && secondsWait < maxLocalSearchWait)
      {
         bdis = xbis.available();
         while (bdis > 1)
         {
            if (bdis == 0) {bdis = 1;}
            blen = xbis.read(b, 0, bdis - 1);
            byte r[] = new byte[blen];
            System.arraycopy(b, 0, r, 0, blen);
            rs = rs + Util.byteArray2string(r);
            bdis = xbis.available();
         }
         Date d2 = new Date();
         secondsWait = ((d2.getTime() - d1.getTime()) / 1000);
      }

      respData = rs;
      respReady = true;
   }
   catch (Exception e)
   {
      System.out.println(e);
   }
   return rs;
}

public String getResponse()
{
```

```
      while (!respReady)
      {
      }
      return respData;
   }
}
```

Clase HTTPClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class HttpClient
{
   private String ResponseText;
   private String ResponseHeaders;
   private String StatusText;

   HttpClient(String ipaddr, int port, String filename)
   {
      try
      {
         get(ipaddr, port, filename);
      }
      catch (Exception e) {}
   }

   public void get(String ip, int port, String filename)
   {
      Socket clientSocket = null;

      OutputStream os;
      OutputStreamWriter osw;
      PrintStream out = null;
      InputStream is;
      BufferedReader in = null;

      String request;
      String line;
      int len = 0;

      ResponseText = "";
      ResponseHeaders = "";
      StatusText = "";

      request = "GET " + filename + " HTTP/1.0 \n";
      request += "Content-Length: " + len + "\r\n";
      request += "Connection: Close\r\n";
      request += "\n\n";

      try
      {
            clientSocket = new Socket(ip, port);
            is = clientSocket.getInputStream();
            os = clientSocket.getOutputStream();
            in = new BufferedReader(new InputStreamReader(is));
            out = new PrintStream(os);
      }
```

```
      catch(Exception e) {}

      out.print(request);

      try
      {
         StatusText = in.readLine();
         // Se leen headers HTTP
         line = in.readLine();
         while(line.length() > 0)
         {
            if(line != null) ResponseHeaders += line + "\n";
            line = in.readLine();
         }

         // Se leen los datos del archivo recuperado
         while(in.ready())
         {
            line = in.readLine();
            if(line != null) ResponseText += line + "\n";
         }
         clientSocket.close();
      }
      catch(Exception e) {}
   }

   public String getResponse()
   {
      return ResponseText;
   }
}
```

Clase Ticket.java

```
import java.io.*;
import java.util.*;

public class Ticket
{
   private Random rn = new Random();
   private String vl;

   public Ticket(String tk)
   {
      setTicket(tk);
   }

   public String getTicket()
   {
      return vl;
   }

   public void setTicket(String tk)
   {
      vl = randomnumber(5, 5) + randomstring(5, 5) + "." + tk + ".tk";
   }

   public void save(String path, String name, String data)
   {
```

```
      FileOutputStream fos;
      PrintStream ps;
      try
      {
         fos = new FileOutputStream(path + name);
         ps = new PrintStream(fos);
         ps.println(data);
         ps.close();
      }
      catch (Exception e)
      {
         System.err.println (e);
      }
   }

   public String randomstring(int lo, int hi)
   {
      int n = rand(lo, hi);
      byte b[] = new byte[n];
      for (int i = 0; i < n; i++)
         b[i] = (byte)rand('a', 'z');
      return Util.byteArray2string(b);
   }

   public String randomnumber(int lo, int hi)
   {
      int n = rand(lo, hi);
      byte b[] = new byte[n];
      for (int i = 0; i < n; i++)
         b[i] = (byte)rand('0', '9');
      return Util.byteArray2string(b);
   }

   public int rand(int lo, int hi)
   {
      int n = hi - lo + 1;
      int i = rn.nextInt() % n;
      if (i < 0) i = -i;
         return lo + i;
   }
}
```

Clase TicketParser.java

```
import java.io.IOException;
import java.io.StringReader;

import org.xml.sax.*;
import uk.co.wilson.xml.MinML;
import java.util.*;
import java.io.BufferedInputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.Writer;

public class Ticket_Parser extends MinML
{
   private LinkedList lstItems = new LinkedList();
   private LinkedList urlItems = new LinkedList();
```

```
private String data = new String("");
private String strXML = new String("");
private String serverName = new String("");
private String serverDesc = new String("");

public Ticket_Parser(String xmlText)
{
   strXML = xmlText;
}

public void parse()
{
   try
   {
      StringReader r = new StringReader(strXML);
      parse(new InputSource(r));

      setServer();
      setLinks();
   }
   catch (Exception e)
   {
      System.out.println(e);
   }
}

public void startDocument() {}
public void endDocument() {}

public void endElement (String name)
{
   if (!data.equals(""))
   {
      //lstItems.add(name + "::"+ data);
      lstItems.add(data);
      data = "";
   }
}

public void characters (char ch[], int start, int length)
{
   data = new String(ch, start, length);
}

public void fatalError (SAXParseException e) throws SAXException
{
   throw e;
}

public void list()
{
   for (int j = 0; j <= lstItems.size() - 1; j++)
   {
      System.out.println(lstItems.get(j));
   }
}

public void setServer()
{
   serverName = (String) lstItems.get(0);
```

```
      serverDesc = (String) lstItems.get(1);
   }

   public void setLinks()
   {
      for (int j = 2; j <= lstItems.size() - 1; j = j + 2)
      {
         String link = (String) lstItems.get(j);
         String desc = (String) lstItems.get(j + 1);
         urlItems.add("<a href=" + link + ">" + desc + "</a>");
      }
   }

   public void HTMLlist()
   {
      System.out.println(serverName);
      System.out.println(serverDesc);
      for (int j = 0; j <= urlItems.size() - 1; j++)
      {
         System.out.println(urlItems.get(j));
      }
   }

   public String HTML_PrintServer()
   {
      String txt = "";
      txt = txt + "<font face=Arial size=3 color=blue> " + serverDesc
      + "</font>";
      txt = txt + "<br>";
      txt = txt + "<font face=Arial size=2 color=black><b>" +
      serverName + "</b></font>";
      txt = txt + "<br><br>";
      return txt;
   }

   public String HTML_PrintLinks()
   {
      String txt = "";
      for (int j = 2; j <= lstItems.size() - 1; j = j + 2)
      {
         String desc = (String) lstItems.get(j + 0);
         String link = (String) lstItems.get(j + 1);

         System.out.println("Link: " + link);
         System.out.println("Desc: " + desc);
         System.out.println("------------------------------");

         txt = txt + "<li><font face=Arial size=2 color=blue>";
         txt = txt + "<a href=" + link + ">" + desc + "</a>";
         txt = txt + "</font></li>";
         txt = txt + "<br>";
      }
      return txt;
   }
}
```

Script ds.cgi (Perl)

```
#!/usr/bin/perl

open(CFG, "<./ds.conf");
$host = <CFG>;
$port = <CFG>;
close(CFG);

# Get the CGI input variables
%in= &getcgivars;

# Print the header information
print "Content-type: text/html \n\n";
print "<html>";
print "<body>";

$query = $in{"q"};
&openConnection($host, $port);
print "<br>";

&sendQuery($query);

$r = &receiveResponse();

&closeConnection();

print "</body>";
print "</html>";

exit;

sub openConnection
{
      my($minombre) = shift;
      my($puerto)   = shift;

      ($nombre,$alias,$prototipo) = getprotobyname("tcp");
      ($nombre,$alias,$tipo,$largo,$rawcliente) =
      gethostbyname($minombre);
      ($nombre,$alias,$tipo,$largo,$rawserver)  =
      gethostbyname($minombre);

      $dircliente = pack("Sna4x8",2,0,$rawcliente);
      $dirserver  = pack("Sna4x8",2,$puerto,$rawserver);

      socket(SOCKET,2,1,$prototipo) || die("Error en construcción de
      socket");
      bind(SOCKET, $dircliente)     || die("Error en bind");
      connect(SOCKET, $dirserver)   || die("Error no puedo
      conectarme");

      select(SOCKET);
      $| = 1;
      select(STDOUT);
}

sub sendQuery
{
      my($query) = shift;
```

```
      print SOCKET $query . " ";
}

sub receiveResponse
{
      $r = "";
      while (<SOCKET>)
      {
            $r = $r . $_;
      }
      return $r;
}

sub closeConnection
{
      close (SOCKET);
}


sub getcgivars
{
      local($in, %in);
      local($name, $value);

      if (($ENV{'REQUEST_METHOD'} eq 'GET') || ($ENV{'REQUEST_METHOD'}
      eq 'HEAD'))
      {
            $in= $ENV{'QUERY_STRING'};
      }
      elsif ($ENV{'REQUEST_METHOD'} eq 'POST')
      {
            if ($ENV{'CONTENT_TYPE'}=~ m#^application/x-www-form-
            urlencoded$#i)
            {
                  length($ENV{'CONTENT_LENGTH'})|| print "No Content-
                  Length sent with the POST request.";
                  read(STDIN, $in, $ENV{'CONTENT_LENGTH'});
            }
            else
            {
                  print "Unsupported Content-Type:
                  $ENV{'CONTENT_TYPE'}";
            }
      }
      else
      {
            print "Script was called with unsupported REQUEST_METHOD.";
      }

      foreach (split(/[&;]/, $in))
      {
            s/\+/ /g;
            ($name, $value)= split('=', $_, 2);
            $name=~ s/%([0-9A-Fa-f]{2})/chr(hex($1))/ge;
            $value=~ s/%([0-9A-Fa-f]{2})/chr(hex($1))/ge;
            $in{$name}.= "\0" if defined($in{$name});
            in{$name}.= $value;
      }
      return %in;
}
```

Script LocalSearch.pl (Perl)

```
#!/usr/local/bin/perl

$minombre = "localhost";
$puerto   = 4000;

($nombre,$alias,$prototipo)              = getprotobyname("tcp");
($nombre,$alias,$tipo,$largo,$rawserver) = gethostbyname($minombre);

$dirserver = pack("Sna4x8",2,$puerto,$rawserver);

socket(SERVERSOCKET,2,1,$prototipo) || die("Error en construcción de
socket");
bind(SERVERSOCKET, $dirserver)      || die("Error en bind");

$x = 0;
while($x++ >= 0)
{
      listen(SERVERSOCKET, 1)  || die("Error no puedo escuchar");
      select(CLIENTESOCKET);
      $| = 1;

      print STDOUT "Esperando por un cliente en $nombre:$puerto...\n";

      ($dircliente = accept(CLIENTESOCKET, SERVERSOCKET)) ||
      die("Error en accept");

      print STDOUT "Atendiendo a cliente...\n";

      ($af,$puerto, $inetaddr) = unpack("Sna4x8",$dircliente);
      @inetaddr = unpack('C4',$inetaddr);
      print STDOUT "Conexión $x desde @inetaddr:$puerto\n";

      $query = <CLIENTESOCKET>;

      chop($query);
      print STDOUT "Server recibió consulta por $query \n";

      $cmd = "perl /www/apache/cgi-bin/ds/ksearch.cgi terms=" . $query
      . "";

      @salida = `$cmd`;

      shift(@salida);   #Desapilo los dos primeros elementos
      shift(@salida);   #para eliminar el encabezado HTTP

      print CLIENTESOCKET @salida;
      print STDOUT "Comando: $cmd    \n";
      print STDOUT "Salida : @salida \n";
}
close (SERVERSOCKET);
```