Peer 2 Peer

Sistemas Operativos Distribuidos

Abel Santín González

INDICE

Introducción P2P

El término anglosajón P2P (Peer-to-Peer) suele ser traducido al castellano como "entre pares". Sin embargo, y según el diccionario, peer significa en inglés "par, igual". Luego la traducción correcta es **entre iguales o de igual a igual**, aunque lo más normal es referirse directamente el acrónimo (pedospe).

Básicamente una red informática P2P se refiere a una red que **no tiene clientes y servidores fijos**, sino una serie de nodos que se comportan a la vez como clientes y como servidores de los demás nodos de la red. Este modelo de red contrasta con el modelo cliente-servidor tradicionalmente empleado en las aplicaciones de Internet.
Así, **en una red P2P todos los nodos se comportan igual** y pueden realizar el mismo tipo de operaciones; pudiendo no obstante diferir en configuración local, velocidad de proceso, ancho de banda y capacidad de almacenamiento.

El P2P no es un concepto nuevo aunque gracias a muchos factores como la gran explosión de ordenadores conectados a Internet, el rápido incremento de ancho de banda disponible por los usuarios, la mayor potencia de cálculo y capacidad de almacenamiento de los ordenadores personales y la proliferación de fuentes de información y contenidos diversos a través de la Red han hecho que esta tecnología sea por fin conocida por la inmensa mayoría de los internautas.

Elementos de las redes P2P

El elemento fundamental de toda red P2P es un par o un igual, y es la unidad de procesamiento básico de cualquier red P2P. Un par es una entidad capaz de desarrollar algún trabajo útil y de comunicar los resultados de ese trabajo a otra entidad de la red, ya sea directa o indirectamente.

Existen dos tipos de pares

- **Pares simples**: Sirven a un único usuario final, permitiéndolo proporcionar servicios desde su dispositivo y empleando los servicios ofrecidos por otros pares de la red. Los pares suelen tener una naturaleza dinámica y heterogénea, es decir se conectan a la red de forma intermitente y tienen capacidades muy distintas.
- **Superpares**: Ayudan a los pares simples a que encuentre otros pares o a otros recursos de los pares. Los pares lanzan solicitudes de búsqueda de recursos a los superpares y los superpares les indican donde conseguirlos. Generalmente los superpares tienen una naturaleza estática, se encuentran conectados normalmente a la red y son fácilmente accesibles.

Otro elemento es el concepto de **grupo de pares**, Un grupo de pares es un conjunto de pares formado para servir a un interés común u objetivo dictado por el resto de pares implicados. Los grupos de pares pueden proporcionar servicios a sus pares miembro que no son accesibles por otros pares de la red P2P. Considerando un sistema P2P en el que todos los clientes pueden hablar el mismo conjunto de protocolos (JXTA) el concepto de grupo de pares es necesario para dividir el espacio de la red.

Los servicios proporcionan una funcionalidad útil que se consigue mediante la comunicación de los distintos pares. Esta funcionalidad o aplicación para el usuario puede ser transferir un fichero, proporcionarle información de estado, realizar un cálculo o comunicarse con otro usuario.
Los servicios se pueden clasificar en servicios de pares y servicios de grupo de pares:

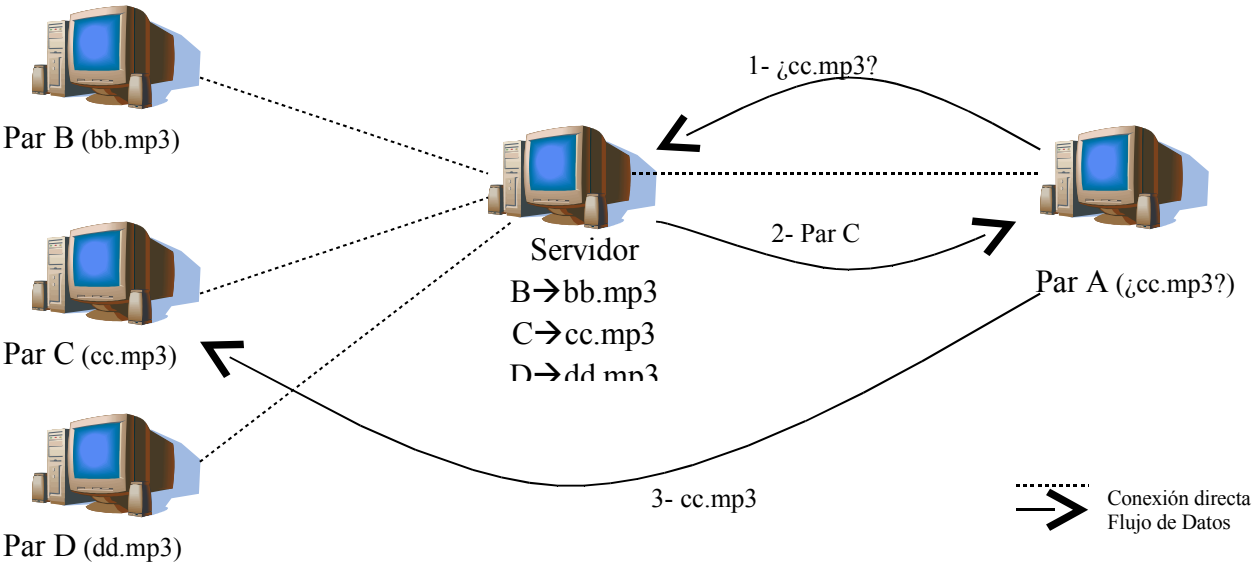
- Servicios de pares: funcionalidades ofrecidas por un par concreto de la red a otros pares, si el par se desconecta el servicio se cae.
- Servicios de Grupo de pares: funcionalidades proporcionadas por varios miembros del grupo consiguiendo así acceso redundante al servicio. Si un par del grupo se cae el servicio sigue estando disponible.

Arquitectura de las redes P2P

Modelo Híbrido o Centralizado

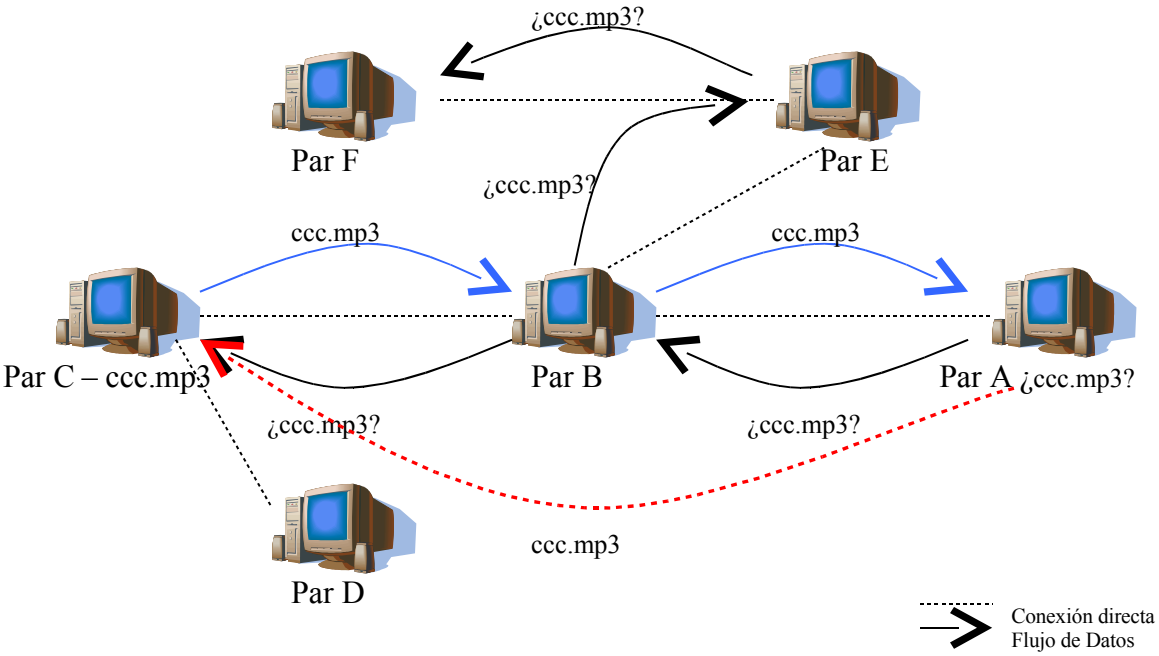
La primera generación de P2P (Napster) empleaba una **estructura de red cliente-servido**r. El servidor central mantiene una base de datos con información de los ficheros servidos por cada par. Cada vez que un cliente se conecta o desconecta de la red, la base de datos se actualiza. Todos los mensajes de búsqueda y control son enviados al servidor centralizado. El servidor centralizado compara la solicitud de sus clientes con el contenido de su base de datos y envía las correspondencias al cliente en cuestión. Una vez que es informado de las correspondencias, el cliente contacta con el par directamente y accede al recurso solicitado.

Proporciona un **rendimiento muy elevado a la hora de localizar recursos** siempre y cuando el servidor esté bien dimensionado, sin embargo **es muy costoso**.



Modelo P2P Puro o totalmente descentralizado

La segunda generación de P2P (Gnutella) usa **un modelo distribuido** donde no existe ningún servidor central y **todos los nodos tienen el mismo estatus**. Cada nodo actúa como servidor y como clientes en la red. Como es evidente cada par dentro de esta arquitectura trata de mantener un cierto número de conexiones con otros pares durante todo el tiempo. Este conjunto de pares conectados transporta el tráfico de red, que está conformado esencialmente por peticiones y respuestas a esas peticiones, y varios mensajes de control que facilitan el descubrimiento de otros nodos (mensajes ping en Gnutella)



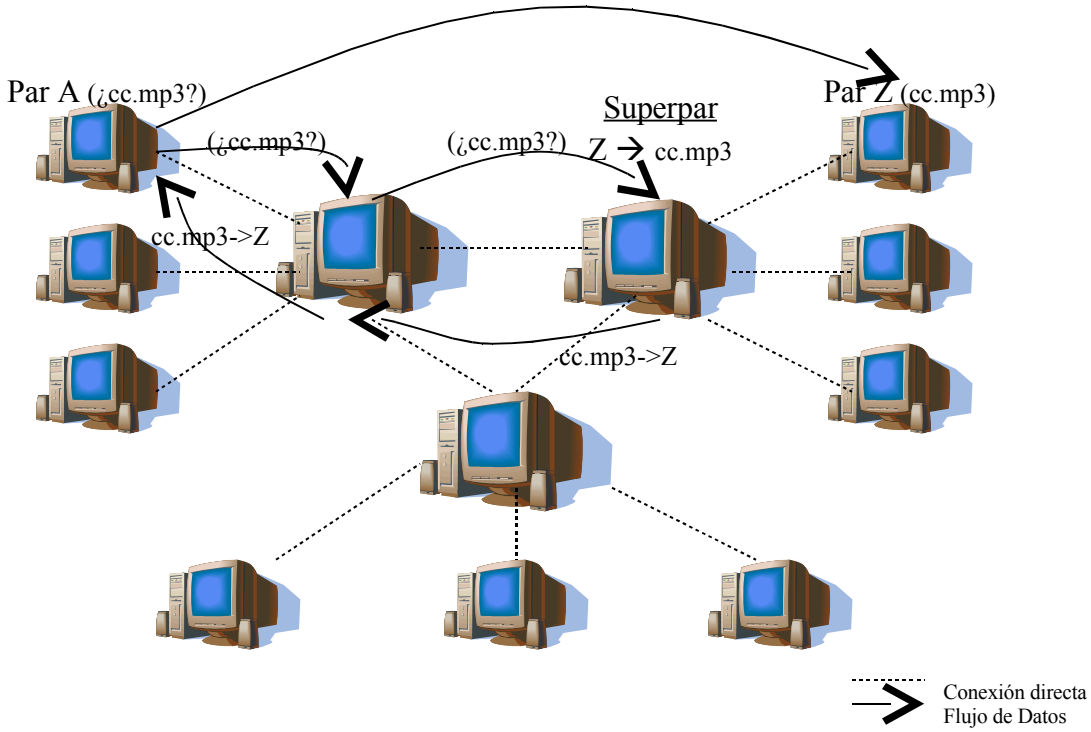
A pesar de que el número de saltos de la red es potencialmente infinito, permanece limitado por un tiempo de vida máximo o TTL, relacionado con el máximo número de saltos que puede dar un mensaje. Por cada nodo o par por el que circula el mensaje de petición se decrementa en una unidad el TTL descartándose la petición si esta llega a cero. En la figura se ve como el par A pide el recurso ccc.mp3 al par B que es el único al que está conectado. Este a su vez distribuye la petición a todos los pares con los que tiene conexión, y así seguidamente hasta que la petición alcance al dueño del recurso o se descuente su TTL. Cada nodo que recibe la solicitud y puede atenderla, responde incluyendo información de interés adicional (nombre del fichero, tamaño, extensión, etc.) y todas las respuestas son reenviadas de vuelta al origen de la solicitud (línea azul). El par A una vez con la respuesta puede establecer una conexión con alguno de los nodos que respondan (Par C).

El modelo P2P puro es más robusto al no depender de un servidor central, además **más económico**. La principal desventaja es el **elevado tiempo y sobrecarga de ancho de banda** que suponen las búsquedas de información en la red. Además puede ser que el recurso buscado y existente ni siquiera pueda ser encontrado.

Modelo P2P Mixto o semicentralizado

Hoy en día la mayoría de aplicaciones P2P consideradas de tercera generación emplean un modelo mixto. Dentro de este modelo, ciertos pares de la red son seleccionados como superpares y ayudan a gestionar el tráfico dirigido hacia otros pares. Los superpares cambian dinámicamente a medida que nuevos pares se conectan.

En este modelo cada nodo cliente mantiene sólo un pequeño número de conexiones abiertas y cada una de esas conexiones es a un superpar. Así mismo los superpares están conectados entre si
Esta nueva topología virtual tiene el efecto de hacer la **red escalable**, mediante la **reducción del número de nodos involucrados en el encaminamiento** y manejo de los mensajes, así como la disminución del volumen de tráfico entre ellos.
Además la **velocidad de respuesta** a las solicitudes dentro de un entorno mixto es comparable al de un entorno P2Pcentralizado.



Comunicación en las redes P2P

Para que la comunicación tenga lugar es importante poder identificar los pares, grupos de pares y contenidos en la red. En los sistemas P2P tradicionales, algunos de estos identificadores se basaban en detalles específicos de los protocolos de transporte; por ejemplo, un par puede ser identificado mediante su dirección IP y puerto. Sin embargo, esta representación es inflexible e incapaz de proporcionar un sistema de identificación independiente de la red de transporte subyacente.

Búsqueda de pares, contenidos y servicios

La búsqueda de información (pares, contenidos y servicios), dada la ausencia de un conocimiento global de los datos y recursos involucrados, es un aspecto fundamental en entornos P2P, así como uno de sus grandes problemas. Un sistema de búsqueda de información en una red P2P debe soportar búsquedas flexibles, eficientes tolerantes a fallos, y ofrecer garantías de que todo lo que existe puede ser encontrado; también debe ser capaz de tratar con escalabilidad, dinamismo y heterogeneidad de estos entornos distribuidos.

La búsqueda se realiza en tres niveles:

- **Búsqueda en caché** (Sin descubrimiento): Cada par mantiene una caché de recursos previamente descubiertos. Este método es muy fácil de implementar y reduce enormemente la cantidad de tráfico generado por cada par en la red. La caché debe eliminar las entradas que han superado un tiempo de vida máximo o aquellos recursos que dejan de estar disponibles. Se suele implementar como una cola FIFO. Este método es el más usado para encontrar superpares y pares en el momento previo de conexión a la red. En la mayoría de las aplicaciones P2P, la forma más sencilla de asegurar que un par en concreto pueda encontrar a un superpar es descargar en el par un listado actualizado de superpares.
- **Búsqueda directa**: Los pares en caso de no encontrar la información en su caché, pasará a preguntar directamente a otros pares de la red con los que tenga conexión directa, usando métodos de broadcast y multicast. Es la forma de búsqueda usada en modelos P2P puros o totalmente descentralizados.
- **Búsqueda indirecta:** Los superpares actúan como fuente de información de localización de pares y otros recursos conocidos. Además esos hacen la búsqueda en nombre del par. Es la búsqueda usada en arquitecturas mixtas o semidescentralizadas.

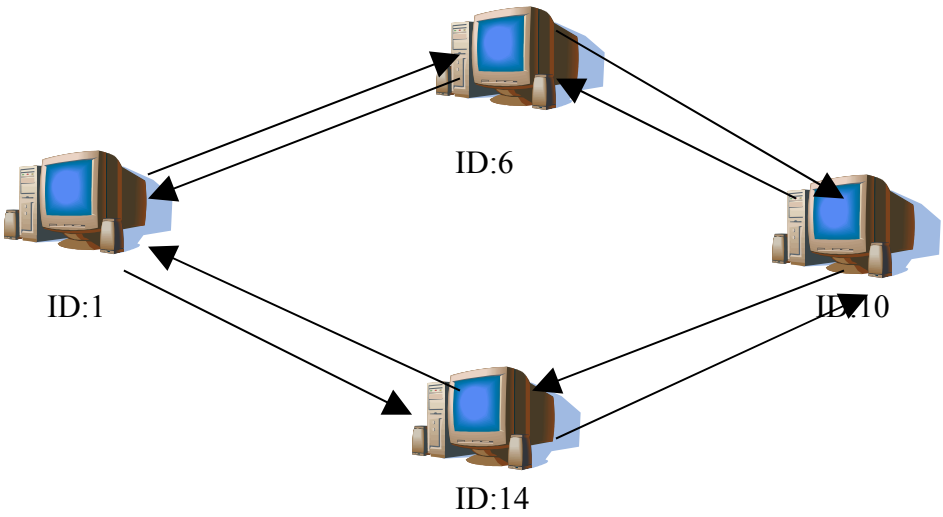
Tablas Hash Distribuidas (DHT)

Las tablas hash distribuidas son un componente esencial de las redes p2p, y han tenido un efecto revolucionario en la descentralización de estas redes. Las topologías caóticas de red de la primera generación de protocolos p2p ha sido sustituida por arquitecturas más escalables y con mejores propiedades gracias a las DHT.

Una DHT básicamente realiza las funciones de una tabla Hash, que básicamente se resumen en dos operaciones, almacenar el par valor y clave en la tabla, y dada una clave buscar su valor. Un ejemplo típico de Tabla Hash es un diccionario donde las palabras son claves y sus definiciones los valores.

Lo que se pretende con las DHT es distribuir el almacenamiento y la búsqueda de la tabla hash a múltiples máquinas. A diferencia de un modelo cliente servidor en el que se basan las arquitecturas de replicación de datos, todos los nodos son iguales que pueden unirse y dejar la red libremente. A pesar del aparente caos producido por el cambio de los miembros en la red, DHT garantiza su funcionamiento

Para comenzar nuestro estudio sobre el diseño de una DHT, comenzaremos con una lista doblemente enlazada. Cada nodo en la lista es una máquina en la red y cada nodo mantiene una referencia al siguiente y al nodo previo. Para definir este orden a cada nodo de la red se le asigna un Identificador (ID) de k bits, de forma aleatoria y único (En la mayoría de los protocolos es un ID de k=64 bits, suficientemente grande para no producirse colisiones). Por lo tanto se define el nodo siguiente como aquel que tiene el ID superior más cercano al suyo propio y el nodo anterior el que tiene el ID inferior mas cercano al suyo propio. La única excepción es para el nodo con ID mayor que tiene como siguiente al ID menor, y al nodo ID menor que tiene como anterior al nodo con ID mayor



También es importante definir la distancia entre dos nodos, que puede ser en pseudo código:

```
Def distance (a,b):
        If a==b:
                Return 0
```

```
Else if a<b:
        Return b-a;
Else:
        Return (2^k) + (b-a)
```

En este código se calcula la distancia entre dos nodos con identificadores a y b, como la diferencia entre sus identificadores, aunque más adelante se mostrará una forma mejor de medir la distancia.
Para construir una DHT, cada nodo es en si una tabla hash estándar, y las operaciones de almacenar y consultar el valor, se realizarán en el nodo apropiado de la red. Una manera simple par determinar cual es el nodo apropiado para una clave en particular es el mismo para determinar cual es el sucesor de un nodo con un ID particular.
Primero tomamos la clave (para poner ejemplos la clave puede ser el nombre de un recurso, como el contenido de un fichero, y el valor su ubicación) y realizamos una función hash para generar una clave de exactamente k bits (mismo número de bits que el ID). Tratamos ese numero como un ID de nodo y determinamos cual es su nodo sucesor empezando en cualquier punto en el anillo y desplazándose en sentido horario hasta que se encuentre el nodo cuya ID sea la más cercana pero todavía superior que la clave en particular. El nodo encontrado es el responsable de almacenar y buscar para esa clave en particular. Las funciones hash para generar la clave deben tener la propiedad de generar claves con igual probabilidad en todo el rango de IDs, para que la distribución sea lo mayor posible y no sobrecargar un nodo en particular.

```
Def findNode (Start, key)
        Current=start
        While distance (current.id , key) > distance (current.next.id, key)
                Current=current.next
        Return current

Def lookup(start, key)
        Node=findNode(start, key)
        Return node.data[key]

Def store (start, key, value)
        Node=findNode(start, key)
        Node.data[key]=value
```

Este diseño DHT es simple pero suficiente para el propósito de una tabla hash distribuida para una red de nodos estáticos. Una cosa a tener en cuenta en el código es que en una DHT real cada nodo debería estar en una máquina diferente y todas las llamadas entre ellas se deben comunicar por algún protocolo de red.

Para hacer más útil nuestro diseño, vamos a añadir la posibilidad de que los nodos puedan unirse y dejar la red, ya sea intencionadamente o en caso de fallo. Para ello vamos a establecer un protocolo de join / leave (unión y desunión) a nuestra red.

El primer paso para unirse a la red es buscar el sucesor de el nuevo nodo ID usando el protocolo de búsqueda que hemos visto. El nuevo nodo debería insertarse entre el nodo sucesor y antecesor y hacerse responsable de la parte de claves que le corresponde de su predecesor. Para garantizar el correcto funcionamiento durante la unión del nodo, los nodos no cambiarán sus punteros hasta que no se hayan copiado todas las claves que le corresponden al nuevo nodo.

La desunión también es muy simple, el nodo que deja la red copia toda su información a su predecesor y este cambia su puntero de nodo siguiente al mismo del nodo que abandona la red. En caso de caída del nodo, el nodo anterior al nodo caído, tendrá que iniciar un proceso de unión para volver a establecer el anillo.
También cabe destacar que en caso de caída de un nodo la información almacenada en dicho nodo deja de estar disponible temporalmente. Los distintos protocolos establecen mecanismos de almacenamiento de la información pasado un tiempo para que la información vuelva a estar disponible.

Sin embargo este protocolo no está del todo optimizado, ya que para encontrar un recurso en la red se pueden dar tantos saltos como número de nodos como máximo, siendo el número de saltos medio n/2 (siendo n el nº d nodos).

Para resolver esto, una modificación al protocolo es que cada nodo en lugar de apuntar únicamente al siguiente nodo, tenga una tabla donde apunte a k nodos. La distancia entre el ID del nodo que almacena la tabla y los IDs de los nodos apuntados incremente de forma exponencial.

Para que la búsqueda "logarítmica" funcione, la tabla de punteros necesita estar actualizada. Una tabla de punteros a nodos mal actualizada no tiene porque perjudicar a la búsqueda siempre y cuando todos los nodos tengan el puntero al siguiente nodo actualizado, aunque en el caso de estar mal actualizada no funcionará la búsqueda logarítmica.

Como hemos dicho, cada nodo tiene k punteros a otros nodos (k=nº de bits del ID). Sea x uno de estos punteros (1<=x<=k). El contenido del puntero x se determina tomando el ID del nodo y buscando el nodo responsable de la clave $(id + 2^{(x-1)}) \bmod (2^k)$

```
Def update (node):
    For x in range (x)
        oldEntry=node.finger[x]
        node.finger[x]=findNode (oldEntry, (node.id + (2**x)) % (2**k))
```

Cuando se hace una busqueda, hay que decidir cual de los k nodos apuntados elegir en el siguiente salto. Para cada nodo , se buscará en la tabla de punteros a nodos aquel que tenga la menor distancia a la clave.

```
Def findFinger (node,key)
    Current=node
    For x in range (k)
        If distance (current.id, key) > distance (node.finger[x].id, key)
            Current=node.finger[x]
    Return current

Def lookup (start, key)
    Current=findFinger (start, key)
    Next=findFinger(current, key)
    While distance (current.id, key) > distance (next.id, key)
        Current=next
        Next=findFinger(current,key)
    Return current
```

Mas o menos tal y como está definido el protocolo hasta este punto es la versión original de Chord DHT, descrito por el MIT.

Otra propiedad útil para una DHT es la posibilidad de actualizar la tabla de punteros de forma pasiva, haciendo búsquedas de forma pasiva para refrescar la tabla.
Otra de las mejoras que se le puede añadir al protocolo es la posibilidad para un nodo de añadir mas punteros a nodos en su tabla, cuando contacta con otros nodos durante la búsqueda, ya que en el proceso de búsqueda se establece la comunicación con el nodo reduciendo la sobrecarga en la red.

Desafortunadamente, las tablas de punteros que hemos visto son unidireccionales, o lo que es lo mismo, la distancia no es simétrica entre dos nodos y por lo tanto un nodo como norma general no estará en la tabla de nodos apuntados de los nodos a los que el apunta.
Una solución a este problema es remplazar la distancia entre nodos definida anteriormente, por la operación XOR entre las IDs de los nodos siendo así la distancia(A,B)=distancia(B,A). Si A está en la tabla de punteros de B entonces B está en la tabla de A. Esto significa que los nodos pueden actualizar sus tablas de punteros guardando la tabla de los nodos a los que consulta, reduciendo así la cantidad de tráfico.

Una cuestión con el diseño presentado es que si un nodo en la ruta no quiere cooperar. Entre dos nodos hay solo un camino, así que hacer búsquedas entre nodos caídos es imposible. Kademlia DHT resuelve esto ampliando la tabla de punteros a nodos conteniendo j referencias para cada nodo, en lugar de sólo uno. Además Kademlia da preferencia en su tabla de nodos a aquellos que más tiempo llevan, añadiendo tan solo nuevas referencias en caso de que no tenga la tabla llena.

Características y Beneficios de las Redes P2P

- Descentralización
- Escalabilidad
- Anonimato
- Propiedad compartida
- Rendimiento
- Seguridad
- Tolerancia a Fallos
- Interoperabilidad

JXTA

Introducción

Desde la aparición de Napster, conocido como el primer programa P2P, han surgido multitud de programas que implementan dicha funcionalidad. Sin embargo la gran cantidad de aplicaciones P2P distintas, ha resultado en una gran inflexibilidad y falta de interoperabilidad entre ellas.

El proyecto JXTA comenzó en el año 2001 dentro de Sun Microsystems e intenta convertirse en una plataforma modular que provee bloques de construcción simples y esenciales para el desarrollo de un amplio rango de servicios y aplicaciones distribuidas basadas en P2P.

La tecnología JXTA está basada en estándares como XML y Java, aunque el uso de Java no es requerido, ya que los protocolos JXTA pueden ser implementados en C++, Perl o cualquier lenguaje de programación. Además JXTA está pensado para ser ejecutado en todo tipo de dispositivos, teléfonos móviles, PDAS, ordenadores.

La plataforma provee de un ambiente descentralizado que minimiza los puntos únicos de fallo y no es dependiente de ningún servidor centralizado.
Los servicios JXTA pueden ser implementados para interoperar con otros servicios dando nacimiento a nuevas aplicaciones P2P. Por ejemplo, un servicio de comunicación P2P de mensajería instantánea puede ser fácilmente agregado a una aplicación P2P de compartición de ficheros si es que ambos soportan protocolos JXTA.

La forma de funcionamiento se basa en un conjunto de protocolos P2P simples y abiertos que permiten que cualquier dispositivo de red se comunique, colabore y comparta recursos. Los nodos JXTA crean una red virtual sobre las redes existentes, escondiendo la complejidad de sus capas.

Pila de Protocolos JXTA

Todos los protocolos JXTA son asíncronos y están basados en el modelo petición respuesta. Un par JXTA envía una petición a uno o más pares en el grupo de pares y puede recibir cero, una o varias respuestas.

- Endpoint Routing Protocol: Usado para encontrar las rutas de destino a otros pares a través de identificadores únicos de pares (relay peers) que pueden ser usados para enviar un mensaje a su destino
- Peer Resolver Protocol: Permite a los pares enviar peticiones genéricas a uno o más pares y recibir una o múltiples respuestas. Las peticiones pueden estar dirigidas a todo el grupo de pares específicos del grupo.
- Peer Discovery Protocol: Usado para publicar sus propios recursos y descubrir donde se encuentran los recursos de otros pares
- Peer Binding Protocol: Establece un canal de comunicación virtual entre dos o más pares

- Peer Information Protocol: Usado por los pares para obtener información de estado de otros pares.
- Peer Membership Protocol: Protocolo para suscribirse a un servicio de propagación.

Bibliografía

Domine las redes P2P : "Peer to Peer" orígenes, funcionamiento y legislación del P2P, selección y configuración del acceso de banda ancha a internet"
Millán Tejedor, Ramón Jesús
Editor: Creaciones Copyright
Fecha de pub: 2006
Páginas: XVIII, 330 p.
ISBN: 849630020X

Kademlia: A Peer-to-peer Information System Based on the XOR Metric
Petar Maymounkov and David Mazieres
New York University

Artículo Distributed Hash Tables, Part I
Brandon Wiley
Linux Journal http://www.linuxjournal.com/article/6797