



Diseño de una DHT

Práctica opcional



+ Diseño de una DHT

Objetivo

Directrices y opciones

Ejemplo

Consideraciones y defensa

+ Diseño de una DHT

Objetivo

- Diseñar un sistema para compartir datos mediante una DHT
- Libertad para decidir las opciones de diseño e implementación, según
 - Estándares conocidos (Kademlia, Chord, etc.)
 - Recursos disponibles (código fuente, participantes, etc.)
 - Tiempo disponible
- El sistema no tiene por qué ser totalmente funcional. P. ej:
 - Puede no usar un método de encriptado
 - Puede no tener una escala global
 - Puede no compartir ficheros, si no sólo las URLs donde están

+ DHT

Directrices generales

■ Métodos

- Subida (alta) y bajada (búsqueda) de recursos
 - Uso de distancias entre nodos
 - Opcionalmente, encriptación
- Opcionalmente, alta y baja de nodos, y baja de recursos

■ Clases

- Una única clase para los peers, que actúa como cliente y servidor
- Opcionalmente, cualquier otra clase

■ Despliegue

- Al menos 3 nodos, cada uno con uno o más peers

+ DHT

Opciones de diseño

- **Middleware:** Java RMI, REST, SOAP, Sun RPC, etc.
- **Encaminamiento:** Kademlia, Kad, Chord, etc.
 - Encriptado, distancia, etc.
- **Lenguaje:** recomendado Java, pero también C#, Python, etc.
- **Escala:** n° de nodos y máquinas donde se ejecuta
 - n° máximo en que se podría ejecutar
- **Información compartida:** ficticia (sólo se transmiten urls o paths) o real (varios tipos de archivos o solo uno, troceados o no)

+ DHT

Métodos: ejemplo (Kademlia)

- Métodos básicos (tema 8, diap. 42)
 - **guid put(key, value)**
 - Almacena el par <key,value> en el nodo más cercano a key
 - **value get(key)**
 - Retorna el valor asociado a key
- Métodos adicionales
 - **d distance(key, guid)**: determina la distancia xor entre dos claves de 160 bits (una cadena de 20 bytes en Java)
 - **key sha1(value)**: dada una cadena de caracteres, obtiene una clave de 160 bits, correspondiente a su cifrado mediante SHA1
 - **guid getNode(key)**: obtiene el guid del nodo más cercano a la clave

+ DHT

Clases: ejemplo (Kademlia)

- Todos los nodos implementarán la misma clase **Peer**
 - Contiene los métodos necesarios para obtener/subir valores
 - Los valores serán simplemente URLs, simulando la localización
 - Mantiene su porción correspondiente de la DHT mediante un **HashMap**<byte[], String>
 - Mantiene su tabla de encaminamiento con los nodos que conoce
 - Tema 8 (P2P) diap. 48 y sigs
 - Conoce su **GUID**
 - Utilizaremos para ello una URI: <http://host:port/Peer?id=n>
 - host y port indican el host y puerto en el que se aloja
 - alojaremos varios Peer en un mismo host

+ DHT

Opciones: ejemplo

- Lenguaje: Java
- Middleware: REST (Jersey)
- Encaminamiento: Kademlia
 - SHA1, XOR
- Información compartida: ficticia (URLs)
- Escala: Kademlia (2^{160} nodos)
 - Probada en 3 ordenadores, 3 Peer por ordenador



+ DHT

Consideraciones

- Cada grupo de prácticas constará de 2 a 10 personas
 - A mayor nº de personas, más se espera del trabajo
 - Cada grupo deberá enviar el nombre y apellidos de sus miembros a rodri@usal.es, y recibirá un identificador de grupo
 - Sólo se admitirán grupos hasta el 20 de abril
- Se permite el uso de código/fuentes externas
 - Siempre que se CITE y EXPLÍCITE adecuadamente
 - No debe constituir la base del trabajo
 - El plagio está penalizado con el suspense de la asignatura
- Entrega:
 - Plazo indicado y subida habilitada en Studium
 - Se debe entregar un fichero con nombre *grupoN.tar.gz*



DHT

Defensa



- La defensa se llevará a cabo en la semana posterior a la entrega
- Para la defensa se usará el fichero *tar.gz* entregado
 - Se pueden usar bibliotecas o software adicional previamente instalado (p. ej. Eclipse, Tomcat, Jersey)
- Requisito mínimo: el sistema debe ser capaz de localizar nodos y recursos en cualquier proceso del despliegue
 - Sin inanición, espera ocupada o interbloqueos.
- Consideraciones adicionales
 - Extensibilidad, escalabilidad, arquitectura, rendimiento, sencillez, documentación, real/ficticia, n° miembros.



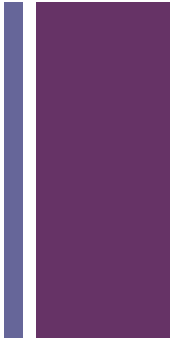


DHT: FAQ

¿Cómo descubrir peers al inicio?



Cómo descubrir peers al inicio?



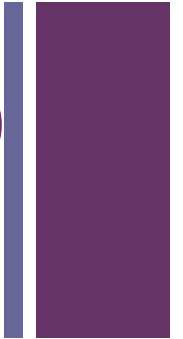
- ¿Si todavía no he poblado mi DHT de ningún modo, cómo hago para conocer a los nodos que hay en la red?
 - La respuesta simple es que, sólo con DHT, **no puedes**
 - Este es un punto crítico de DHT, necesitamos de alguna manera que, de forma centralizada, nos den un primer peer o “cabo”
 - A partir de ese momento, podemos tirar del cabo y continuar poblando nuestro DHT mediante un enlazado o ‘**bootstrap**’
 - Bootstrap se refiere a las tiras de una bota, a partir de las que podemos agarrar y ponérselas
 - Es decir, podemos tirar de ese peer para buscar más pares que nos vayan dando información, o tomarla de este primero
 - En SuperUser hay un artículo excelente al respecto*



* <https://superuser.com/questions/592238/in-simple-terms-how-does-a-bittorrent-client-initially-discover-peers-using-dht>



Cómo descubrir peers al inicio? (ii)



- Cómo proveer ese 'cabo' inicial queda a vuestro criterio
 - Puedo proveerlo como argumento o directamente en el código
 - O leer de algún fichero de configuración centralizado
 - O hacer un broadcast a un rango de direcciones IP para ver si hay alguien escuchando
 - O bien directamente poblar la DHT con información suministrada por nosotros, sin necesidad de este proceso de bootstrap
 - O cualquier otra cosa que se os ocurra y funcione!

