Kademlia

Implementación del algoritmo Kademlia para Sistemas Distribuidos

> Nicolás García Martín, Mario Alonso Nuñez

Resumen

Se ha realizado un ejecutable para los pares que pertenecerán a una red Kademlia.

Como base para el estandar de Kademlia se ha utilizado esta definición. Y se ha implementado todo requisito de esa especificación, si bien se han tomado algunas decisiones, detalle, de diseño de implementación, los requisitos han sido satisfechos en su máxima.

El programa es funcional y se provee de una interfaz interactiva que permite observar el comportamiento del nodo.

Modo de funcionamiento

Se encuentra en el directorio ./dist ejecutables para distintas plataformas. Estos ejecutables se han de lanzar con dos parámetros: *ip puerto*, que son la interfaz por la cual el nodo **ofrece el servicio**.

Una vez accedido a la aplicación, se necesitará adherirse a una red de Kademlia. Para ello puede utilizarse el comando *join* dentro de la interfaz. y se han de expresar, la interfaz por la que otro nodo ofrece el servicio (en el código le hemos denominado amigo, *friend*). Si se está iniciando la red, simplemente hay que conectar de esta forma un nodo a otro, y en cadena comunicar los siguintes nodos a alguno que ya pertenezca a la red.

> La red, por propia definición de Kademlia, permite autodescubrimiento, el primer nodo no es especial ni es necesario configurarlo de manera especial.

Una vez conectado a una red, puede utilizarse el comando *publish* <*valor*> que permite publicar información dentro de la red.

Se devolverá el hash generado (SHA1) que puede ser utilizado en cualquier nodo de la red para recuperar la información mediante el comando *get* <*clave*>.

Puede utilizarse el comando *logs* para observar el comportamiento del nodo ante los eventos que suceden.

Arquitectura

Si bien no hemos realizado una Clean Architecture (por falta de experiencia + falta de tiempo), la arquitectura del software si se ha visto afectada por su gravedad. Se tienen componentes *Application* que definen eventos (casos de uso), capa de *Domain* que refleja el modelo de negocio, y se han relevado los detalles de implementación a *Infraestructure*. Si bien no hicimos spaguetti, al menos hicimos macarrones.

Lib (Bibliotecas externas)

Las bibliotecas que aquí se encuentran, son importadas pero de fabricación propia.

BinaryStringLib

Esta biblioteca está diseñada para la manipulación de cadenas binarias de longitud indiferente. Que mediante sobrecargas de operadores permiten realizar operaciones complejas de forma transparente. Su principal función es permitir realizar ordenaciones y operaciones de xor sin conocer los detalles.

Gracias a esta biblioteca, Kademlia puede calcular cercanía de nodos y claves, generar claves aleatorias y transformar la información a un string hexadecimal y volverla a convertir en BinaryString.

No se encontró forma de almacenar la información en arrays de bytes que fuese mejor que diseñar internamente bloques de booleanos con representación directa a hexadecimal y permitiese todas estas operaciones, y es más, lo permitiese de forma indistinta al *Endian* del hardware del nodo.

TaskExtensionsLib

Esta biblioteca si está ~~copiada~~ inspirada en el software de una charla de *John Thiriet*, público en github y referenciado en el código, que consiste en un método de extensión para la clase Task que facilita en gran medida el diseño asíncrono basado en eventos de este servidor.

TCPLayer

Por cuestiones explicadas más abajo, se decidió utilizar TCP para las comunicaciones en la red, y esta biblioteca permite de forma asíncrona y no dependiente del código, manejar comunicaciones por sockets.

Incluye tanto un servidor que mediante un callback maneja peticiones de forma asíncrona según llegan, y un cliente que también de forma asíncrona permite realizar peticiones.

src (El código interno de la aplicación)

Configuration

Básicamente, un singleton que permite acceder a la configuración de forma global y desacoplada.

Kademlia

Este es el núcleo del proyecto, se tienen:

- Application (los eventos, casos de uso)
    - Network (eventos de red)
        - Se definen los 4 RPC de Kademlia (store, find node, find value, ping) y uno extra que consideramos conveniente, dada la naturaleza de Kademlia, los nodos se inventan una identificación para si mimos, y resultaba util una forma de obtener la tupla del contacto solo conociendo la ip y puerto del servicio, este método es (Identify)
    - User (eventos de usuario)
        - Se definen los 3 casos de uso que permite la aplicación, publish, join, get.
- Domain (Las reglas de negocio de la aplicación)
    - Bucket
        - BucketContainer almacena la información sobre los contactos conocidos, los mantiene según las reglas de Kademlia (buckets de K contactos ordenados por tiempo sin contactar)

y expone métodos para manipular los contactos abstrayendo esta lógica (dame los k contactos más cercanos a esta clave, actualiza el bucket de este contacto).
    - Clock
        - Clock permite manejar los eventos de reloj, de manera abstracta, se define un comportamiento programado para tras un periodo específico. No se definen aquí los eventos de Kademlia porque no son su responsabilidad. Por ejemplo, se define que tRepublish hace a un nodo republicar una tupla tras ese tiempo, entonces es responsabilidad del caso de uso publish, esta biblioteca no sabe ni quiere saber sobre el detalle.
    - Database
        - Expone las **acciones** sobre una agnóstica base de datos. Se ha abstraido el comportamiento sobre la base de datos del detalle de implementacion, mediante la definición de la interfaz IDatabase. Por ejemplo, Store programa la expiración para una tupla, y después quiere almacenarla. Esto es lógica de negocio.
    - Interfaces
        - Define que necesita un IClient para comunicarse y la excepción que utiliza si falla.
    - Iteratives
        - Define las acciones iterativas de Kademlia, por ejemplo, IterativeFindNode, que es altamente utilizado, termina haciendo de fachada a un enorme algoritmo construido mediante una clase.
    - Models
        - Define FoundResult, pues es un tipo complicado, al buscar un valor, se puede retornar el valor o una colección de contactos cercanos, y este modelo lo define.
    - Network y User, define interfaces para los casos de uso diseñados en la capa de aplicación.
- Infraestructure
    - Define el detalle de implementación de (IDatabse)InMemoryDatabase e (IClient)TCPClient.

LinesUI

Biblioteca que provee de la interfaz gráfica a Kademlia. Utiliza un controlador para, mediante una clase Observer que implementa un evento, mostrar la información de un controlador en tiempo real y pasarle los comandos que escribe el usuario.

Solo es necesario implementar un controlador basado en *IController*, y sindicalizar el controlador a una ruta a traves de una instancia singleton de *Router*. Toda la biblioteca es código nuestro.

Logger

Permite realizar eventos de logging. Ofrece dos fachadas, un *Logger<T>* que permite realizar logs, y un *LoggerForConsumer* que permite obtener los logs realizados a lo largo de todo el sistema.

Router

No confundir con la clase Router de LinesUI, aquel es un Router para las instancias de las vistas, esta biblioteca Router se encarga de enrutar los eventos de red a los casos de uso de Kademlia. Es decir, encapsula la capa de presentación del programa del dominio de red.

Protocolo de red

Se ha decidido utilizar TCP, la especificación de Kademlia, en principio, asume que se utilizará UDP, pero se ha visto contraproducente. Los RPC formulados por Kademlia son de ida y vuelta, petición y respuesta. Si se pide un find_node, hay que devolver nodos. Siendo inherente la necesidad de un servicio de sesión, no dabamos con un diseño de UDP que no fuera una reinvento de TCP. Y además, BitTorrent hace uso de TCP de hecho. Bajo nuestra poca experiencia, solo observamos util a UDP para aquellas comunicaciones de un solo sentido donde la pérdida de paquetes no sea catastrófica, como retransmisión de vídeo. Pero para una conexión de funciones remotas, nos parece más sensato utilizar TCP.

El protocolo que hemos diseñado es el siguiente:

```
PING;origen<EOF>
```

Enviar ping con ID de origen.

```
PONG<EOF>
```

Responder a ping.

```
FIND_NODE;origen;id<EOF>
```

Solicitar búsqueda de nodo con ID.

```
FOUND_NODES;contacto;*<EOF>
```

Retornar nodos encontrados con ID, IP y puerto: ;ABCD,192.168.1.0,4200

```
FIND_VALUE;contacto;key<EOF>
```

Encontrar valor con Key.

```
FOUND_VALUE;[FOUND_NODES;contacto;*|VALUE;value]<EOF>
```

Retornar o nodos más cercanos indicando NODES o el valor para esa key con FOUND.

```
STORE;contacto;Tupla<EOF>
```

Guardar un par indicando su Key y su Value.

```
IDENTIFY -> IDENTIFICATION;contact
```

Identificarse

Diseño basado en eventos

Se ha diseñado el programa entorno a eventos.

- Eventos de red
  - Los RPC ya mencionados
- Eventos de usuario
  - Publish
  - Get
  - Join
- Eventos de reloj
  - tRepublish
  - tReplicate
  - tExpire
  - tRefresh

Como ya se ha mencionado, los eventos de red y usuario se han expuesto como casos de uso en la capa de aplicación para poder ser accedidas por distintos adaptadores. Sin embargo, los eventos de reloj proveían desde circunstancias internas y su programación está dispersa a lo largo del dominio, por ello se realizó la clase Clock de manera abstracta y reutilizable.

Dependencias externas

Siendo un trabajo de tal gran tamaño, se ha hecho uso de un contenedor de dependencias, en específico, AutoFac. Esta biblioteca, y la extensión de Task son el único código importado al proyecto.