Proyecto de Sistemas Distribuidos.

Karel León Manresa, C312 Rainel Fernández Abreu C312, Karlos Alfonso Rodríguez C311

Facultad de Matemática y Computación
Universidad de la Habana

Descripción:

Una plataforma de agente permite la creación de soluciones distribuidas utilizando recursos y conocimientos distribuidos. Para ello es necesario que se puedan registrar los distintos agentes y su funcionalidad, así como la identificación, localización y búsqueda de estos acorde a sus funcionalidades.

Especificaciones:

Estos se recursos y conocimientos se implementarán en forma de agentes. Su principal característica es que son **autónomos**. La plataforma debe lograr **conectar** dichos agentes. Debe ser posible **negociar** con estos agentes y que estos realicen acciones si así lo desean. La plataforma debe ser capaz de **registrar** agentes, así como sus puntos de acceso y funcionalidades, cada agente debe tener un **identificador único** dentro del sistema. Se debe poder realizar una búsqueda de agentes por su funcionalidad y se debe garantizar **tolerancia a fallas**.

Diseño:

El proyecto está dividido en cuatro paquetes, *NodeAP*, donde se recibe y procesan todos lo pedidos del cliente al servidor, *ChordNode*, en el cual se estructura todo el sistema de la tabla de hash distribuida sobre la cual se sostendrá el sistema, *ServerAP*, que es quien se encarga de la comunicación de nuestro servidor con los agentes y por último el paquete *Client*, separado completamente de los otros tres y pensado para la interacción de los usuarios con la plataforma.

Expliquemos como están formados estos paquetes en nuestra aplicación para luego exponer como es el flujo de los datos.

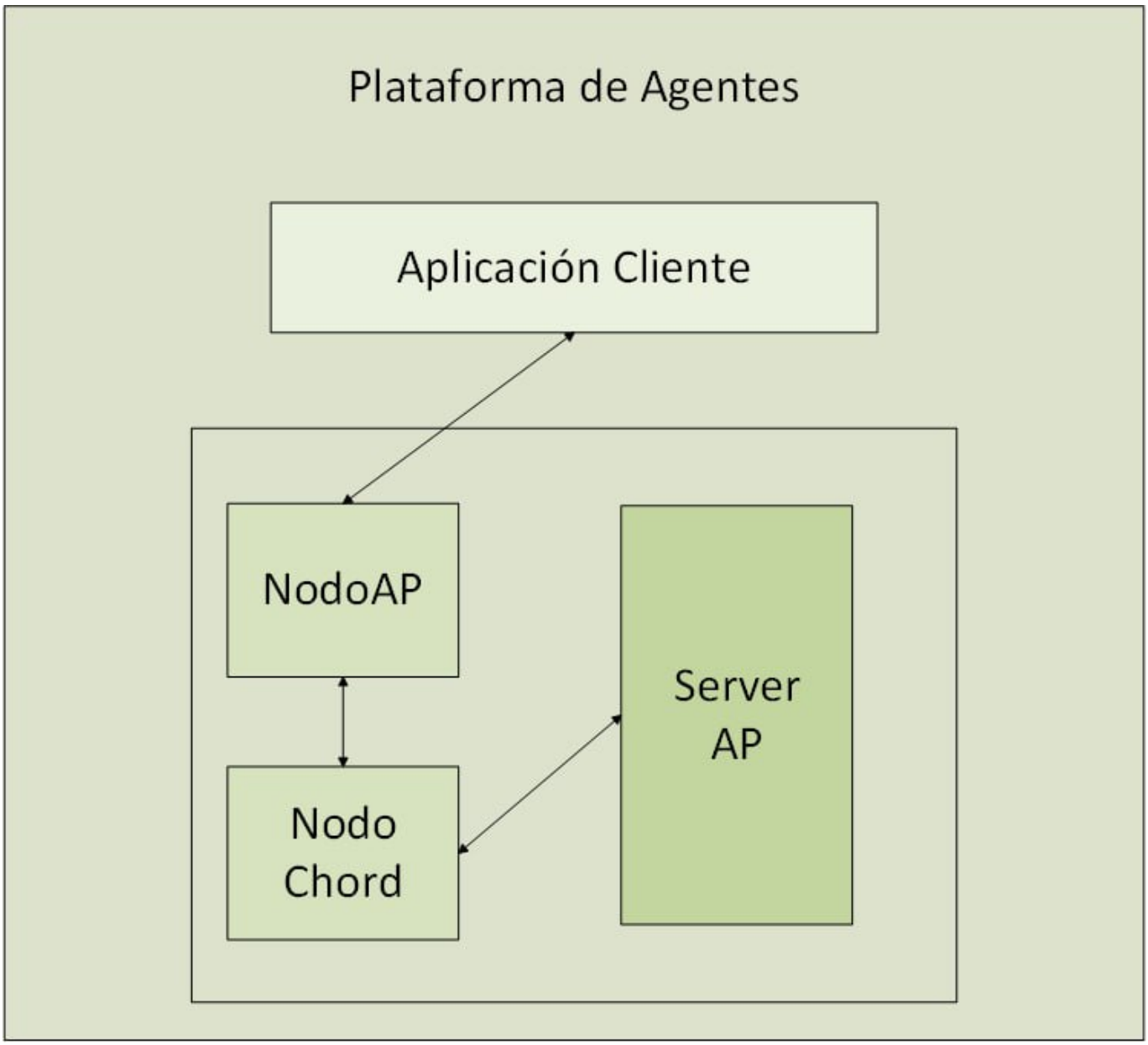
NodeAp: Este objeto es el encargado de recibir las peticiones hechas desde el cliente. Por tanto este objeto mantiene comunicación con el cliente mediante un `socket`, que es la vía de comunicación entre todas las componentes de la plataforma. Tiene una dirección ip y un puerto que deben ser únicos para cada nodo en la red, por convenio en nuestra plataforma decidimos

que estos nodos siempre debían tener puerto par. Además tiene un 'dispatch' para procesar las peticiones del cliente.

ChordNode: Este objeto no "vive" en la red de manera autónoma, sino que existe en cada uno de los *NodeAP*, estos tienen al igual que NodeAP un ip y un puerto asociado a cada uno, estos tienen el mismo número ip que el NodeAP que los contiene, y su puerto coincide con el puerto de su NodeAP sumado 1, de manera que cada NodeAP tiene un puerto par y los ChordNode puertos impares. Estos objetos ChordNode también poseen un 'dispatch', solo que este se encarga de procesar las peticiones de otros ChordNode de la red o de algún NodeAP, estos nodos poseen un `socket` para la comuniación, que como se mencionó es la única vía de comunicación entre componentes.

ServerAP: En esta componente tratamos solamente las comunicaciones con los agentes, se hacen los pedidos `get` y en consideración de la respuesta a nuestra solicitud se envía una respuesta al nodo que hizo dicha petición.

Client: En esta componente se brinda al cliente las funcionalidades que brinda el servidor. Se conecta al servidor deseado usando ip y puerto. Las peticiones y respuestas al servidor se hacen mediante la consola.



Implementación:

Para la implementación se utilizaron agunas herramientas que brinda el lenguaje que hacen mas cómodo el trabajo. En la parte de la comunicación se utilizó solamente sockets para enviar y recibir mensajes. Los mensajes se escriben en formato `json` para una mejor organización en el momento de leer.

DHT.

Para lograr que los agentes funcionen de manera distribuida se utilizó el protocolo **Chord** como *DHT*. La implementación del mismo se basa este documento. Nuestra implemetación recoge:

1. `join()` : Este método se usa para agregar un nuevo nodo al anillo Chord. El nuevo nodo debe encontrar su sucesor y predecesor en el anillo y actualizar su tabla de enrutamiento.

2. `get()` : Este método se utiliza para buscar un valor en el anillo Chord. El nodo que inicia la búsqueda envía una solicitud de búsqueda al nodo responsable del rango de claves que contiene el valor. Si el nodo responsable no tiene el valor, reenvía la solicitud al nodo responsable del siguiente rango de claves.

3. `set()` : Este método se utiliza para insertar un valor en el anillo Chord. El nodo que inicia la inserción busca el nodo responsable del rango de claves que contiene el valor y lo almacena allí.

4. `delete()` : Este método se utiliza para eliminar un valor del anillo Chord. El nodo que inicia la eliminación busca el nodo responsable del rango de claves que contiene el valor y lo elimina de allí.

5. `successor()` : Este método devuelve el sucesor del nodo en el anillo Chord.

6. `predecessor()` : Este método devuelve el predecesor del nodo en el anillo Chord.

7. `notify()` : Este método se usa para notificar a un nodo de que su predecesor ha cambiado. El nodo predecesor envía una notificación al nodo sucesor para que actualice su tabla de enrutamiento.

8. `fix_fingers()` : Este método se utiliza para actualizar la tabla de enrutamiento del nodo. El nodo selecciona un índice aleatorio en su tabla de dedos y busca el sucesor correspondiente en el anillo Chord.

9. `stabilize()` : Este método se utiliza para garantizar que el sucesor del nodo sea correcto y actualizar su predecesor si es necesario. El nodo obtiene el sucesor de su sucesor y verifica si todavía es su sucesor. Si no lo es, se actualiza la tabla de enrutamiento.

10. `replicate()` : Este método se utiliza para replicar un valor en múltiples nodos en el anillo Chord. El nodo que inicia la replicación busca sus nodos sucesores y en dependencia del factor de replicación almacena la llave en una cantidad x de nodos, siendo x el factor de replicación. Esto garantiza que el valor esté disponible en múltiples nodos y mejora la disponibilidad.

Motor de búsqueda.

Otra herramienta usada, que fue construida por los autores también, es un pequeño motor de búsqueda para hacer eficiente y satisfactorias las búsquedas de agentes por su funcionalidad. Para determinar la similitud entre la consulta y las funcionalidades de los agentes se utilizó la fórmula de similitud del coseno que provee la biblioteca `sklearn.metrics.pairwise`. Por convenio se determinó que tanto las funcionalidades de los agentes como las consultas deben ser en inglés.

Cliente-Servidor.

Las funciones que brinda el servidor que pueden ser utilizadas desde la aplicación **Cliente** se muestran a continuación, asi como ejemplos testados por los creadores.

```
def console():
    ip, port = get_ip_port()

    while True:
        print("************************MENU************************")
        print("PRESS ***********************************************")
        print("1. TO ENTER AGENT ************************************")
        print("2. TO SHOW AGENT *************************************")
        print("3. TO DELETE *****************************************")
        print("4. TO USE AGENT **************************************")
        print("5. TO FIND AGENTS BY FUNCTIONALITY *******************")
        print("6. TO SHOW ALL AGENTS ********************************")
        print("7. TO EXIT *******************************************")
        print("*****************************************************")

        choice = input()
```

1. TO ENTER AGENT:

   Para ingresar un nuevo agente a la plataforma el usuario debe enviar a la pataforma un mensaje que consta de dos partes.

   ENTER THE KEY: 'nombre del agente'

   ENTER THE VALUE: 'nombre del agente' 'nombre del endpoint' '[param,...,param]' 'url' 'descripción'

   si se desea añadir un agente con mas de un endpoint, para más de uno se debe continuar con el signo '-' seguido del proximo endpoint

   ejemplo 1: Hola hola [] http://127.0.0.1:8001/ "Dice hola"

   ejemplo 2: ApiCalculator add [x,y] http://127.0.0.1:8002/add/ "Suma dos numeros"-ApiCalculator sub [x,y] http://127.0.0.1:8002/sub/ "Resta dos numeros"

2. TO SHOW AGENT:

Introduce el nombre del agente y obtendrá a información de dicho agente.

ENTER THE KEY: 'nombre del agente'

3. TO DELETE:

   A partir del nombre del agente y de su id se eliminarán todas las existncias de mismo dentro de la plataforma.

   ENTER THE AGENT NAME: 'nombre del agente'

   ENTER THE AGENT ID: 'id único asociado a su agente'

4. TO USE AGENT:

   A partir de que se conoce como funciona un agente, se le brinda a la plataforma el nombre del mismo, con su endpoint y los parámetros.

   ENTER THE API_NAME ENPOIN_NAME PARMAS: 'nombre del agente' 'endpoint' 'params'

   ejemplo1: Hola hola

   ejemplo2: ApiCalclualtor add [x=5,y=5]

5. TO FIND AGENTS BY FUNCTIONALITY:

   A partir de una consulta se obtiene una lista de todos los agentes con funcionalidad similar a la consulta. La lista está ordenada descendentemente, de modo que el primer agente es quien tiene la funcionalidad más parecida a la consulta. Se recorre todo el anillo analizando la similitud de los agentes con la consulta.

6. TO SHOW ALL AGENTS:

   Se recorre todo el anillo para obtener el nombre de todos los agentes que participan en la plataforma para luego mostrarlos.

7. TO EXIT:

Se cerrará la comunicación del cliente y el servidor.