Chord

Ejecución de desarrollo

1. Abrir Docker Desktop para que se inicie el servidor de Docker.
2. Tener descargada y disponible alguna imagen de python en Docker.
3. Abrir una terminal interativa de la imagen de python, utilizando como volumen el directorio del proyecto

   Ejecutando el siguiente comando en una terminal abierta en el directorio del proyecto

   ```
   docker run --rm -it -v <path_>:/app <python_img> /bin/bash
   ```

   - <path_>: la ruta completa de ubicación del proyecto. (Ej: C:\Documents\GitHub\Tags-based-file-system)
   - <python_img>: la imagen de python descargada y su versión. (Ej: python:3)
4. Una vez abierta la terminal interactiva de linux, vamos al directorio `/app`

   ```
   cd app
   ```

5. Creamos un primer nodo de Chord. De esta forma tendríamos ununn anillo donde solo hay un nodo

   ```
   python ChordNode.py
   ```

6. Creamos el segundo nodo repitiendo los pasos anteriores a partir del 3. Pero esta vez el nodo se crea (en sustitución al paso 5) utilizando el flag: `-c`

   ```
   python ChordNode.py -c
   ```

7. De esta misma forma se pueden incluir la cantidad de nodos deseados a la red de chord

En el directorio `logs` se crea un archivo `.txt` por cada nodo de chord que se crea, y en él se muestra de manera continua la información de cada nodo, incluidos su sucesor y predecesor. Esto es posible a la clase `Logger`.

ChordNodeReference (CNR)

ChordNodeReference es una clase auxiliar, que sirve para facilitar la comunicación entre los nodos de chord. Una instancia de ella se asocia a un ChordNode (su referenciado), que es un nodo de la red de chord. Cada instancia de un CNR almacena la dirección IP de su referenciado, y un conjunto de funciones básicas de comunicación, para que otros nodos interactúen con el referenciado. O sea, si tengo un nodo de chord A, entonces un CNR de A tiene su misma IP, y tiene un conjunto de funciones para que otro nodo cualquiera interactúe con él; si B quiere decirle algo a A, tiene que hacerlos a través de las funciones del CNR de A.

Miembros de ChordNodeReference:

- `_send_data`: Esta funcion es privada dentro de la clase, y la van a utilizar otras funciones dentro de la misma clase. Es la funcion que se comunica directamente con el nodo referenciado. Lo que hace es abrir un socket a la propia IP del nodo, y se manda un mensaje en el formato: `'{op},{data}'` . Donde `op` es un entero que representa una constante, mapeada a una operación que el nodo puede satisfacer. (La lista de constantes están en `const.py`), y `data` es un string con la información que se envía en la operación.
- `find_successor`: Es la función a través de la cual otros nodos pueden preguntar al referenciado, cuál es el sucesor correspondiente a un identificador. Donde el sucesor de un ID es el nodo existente en el anillo, con su ID más próximo (en el sentido de las manecillas del reloj). Lo que hace esta función es preguntar al nodo referenciado cual es el sucesor de ese ID.
- `find_successor`: Análoga a la anterior
- `succ` y `pred`: Son dos propiedades de ChordNodeReference que piden directamente el valor del sucesor y predecesor del nodo q se referencia, al referenciado.
- `check_node`: Es una función que pregunta al referenciado si se encuentra conectado a la red. Es una función de comprobación en caso de fallas. En función de eso, reponde con un booleano.
- `notify`, `not_alone_notify`, `get_leader` y `closest_preceding_finger`: Funcionan de forma análoga a las anteriores.

ChordNode

Un ChordNode corresponde a una computadora distribuida, por tanto tiene una IP propia. Esa IP se pasa al constructor de ChordNode, a partir de la cual se deduce el identificador de ese nodo, mediante una función de hash. También se almacenan: una referencia del propio nodo (un CNR), el nodo predecesor (un CNR, por defecto en None), el sucesor (un CNR, por defecto es una referencia a él mismo), m es el numero de bits del anillo (si m=3, en el anillo hay 2^3 nodos), finger y next por ahora en desuso.

Luego en el constructor instanciamos el logger, que será quien muestre el estado del nodo en el tiempo, y creamos varios hilos, sobre los que corren funciones del anillo que actúan de forma ininterrumpida y se explicarán a continuación.

Funciones de ChordNode:

- `start_server` y `request_handler`: La función `start_server` es una de las funciones que corren en hilos ininterrumpidos, y es la encargada de que el nodo actúe en forma de servidor, recibiendo peticiones de otros nodos para posteriormente procesarlas. En ella se abre un socket que escucha hasta 10 clientes a la misma vez, y por cada solicitud pide a `request_handler` que la procese. `request_handler` funciona para cada solicitud en un hilo separado del anterior, de esta forma se evitan tiempos de espera innecesarios. La función recibe el socket del cliente y los datos que este mandó. Posteriormente identifica que operación es la que el cliente pide al nodo, y da una respuesta en consecuencia con ello.
- `_inbetween`: Es una funcion auxiliar de la clase para determinar si un k está entre dos otros numeros, $start$ y end. Lo que hace es comprobarlo de forma circular, porque es para los ID del anillo.

- `find_pred`: Lo que tiene q hacer es, dado un ID, determinar cual es el primer CN que precede a ese ID, y luego devuelve una referencia de él. El recorrido lo hace caminando por los sucesores (esto sale de forma lineal, más adelante hay que usar la finger table para q sea logaritmico)
- `find_succ`: Su funcionamiento se basa en la siguiente idea. El sucesor de ese ID, debe ser el sucesor del predecesor de es ID.
- `join`: Función del propio nodo que se encarga de incluirlo en el anillo de chord, utilizando al *node* que se pasa como parámetro como punto de entrada. Sea A el nuevo nodo que entra al anillo, podemos diferenciar 3 casos:
  1. Es el primer nodo del anillo: entonces el parámetro *node* es None, y A define a sí mismo como sucesor, y sin predecesor de momento.
  2. Es el segundo nodo del anillo (cuando el sucesor de mi sucesor es él mismo): Defino mi sucesor pidiéndoselo al nodo por el que entro al anillo. Luego le aviso a ese nodo que ya no estará solo en el anillo a través de `not_alone_notify`, y finalmente defino como mi predecesor al mismo que es mi sucesor, pues solo somos 2 nodos.
     - `not_alone_notify`: Actualiza el sucesor y predecesor del nodo que es notificado con el nodo que notifica.
  3. Es al menos el 3er nodo: Entonces se pasa el parámetro *node* con el CNR. Defino mi sucesor pidiéndoselo al nodo por el que entro al anillo, pues ese nodo por el que entro es el único nodo que conozco del anillo.
- `stabilize`: Es otra función que corre en un hilo ininterrumpido, y es la encargada de comprobar y asegurar constantemente la estabilidad de la red. Decimos que la red de chord es estable si se encuentran conectadas correctamente todas referencias en los nodos hacia sus sucesores y predecesores

  > El **sucesor** de un nodo de Chord es el siguiente nodo de Chord existente en la red en favor de las manecillas del reloj
  >
  > El **predecesor** de un nodo de Chord es el siguiente nodo de Chord existente en la red en sentido contrario a las manecillas del reloj

  Entonces lo que hace `stabilize` es actuar en un nodo, si no es el único de la red (es lo que comprueba la primera condicional). Sea A un nodo que hace `stabilize`, y sea X el predecesor del sucesor de A. Comprueba si X es un nodo que repentinamente se ha colocado entre A y su sucesor, en cuyo caso lo asume como su sucesor. Finalmente, sin importar si hubo alguien entre A y su sucesor, A notifica a su sucesor para sugerirle que él debe ser su predecesor.
- `notify`: Es una función de un nodo, que es llamada por otro nodo, para sugerirle que él puede ser su predecesor. Cuando al nodo llega la notificación desde el parámetro *node*, comprueba q no se haya autonotificado, y entonces:
  - si el nodo no tiene predecesor, se asigna el nodo que le notificó, como predecesor
  - sino, comprueba que ese nodo q le notificó siga estando vivo, y que esté entre él y su predecesor, para entonces asignárselo.
- `check_predecesors`: Esta función es otra función que se ejecuta en un hilo ininterrumpido en el constructor del nodo. Es la de cada nodo, encargada de hacer comprobaciones constantes sobre su propio predecesor, para comprobar que sigue vivo. En caso de no estarlo, tiene q asignarse alguien de predecesor, y va a ser el nodo que era predecesor de su predecesor. Se asigna dicho predecesor y luego le sugiere al mismo que lo tome como sucesor, utilizando la función `reverse_notify`.

- `reverse_notify`: Función de cada nodo encargada de asignarse como sucesor, el nodo que lo notifica.

En caso de que el predecesor encontrado sea el propio nodo, significa que el nodo se ha quedado solo en el anillo, por lo que debe definir su predecesor como `None`.

Leader:

En todo instante de tiempo del anillo de Chord, va a existir un nodo denominado líder. El concepto de líder centraliza parte del proceso de control de errores. Cuando algún nodo quiere acceder a algún recurso compartido, envía un mensaje al líder solicitando permiso para acceder al recurso, solo con el permiso del líder puede acceder. El líder garantiza el permiso solo cuando otros procesos no estén utilizando ese recurso.

LeaderElection:

Inicialmente, cuando tenemos un solo nodo, él se autoproclama líder. Al unirse más nodos al anillo, el líder permanece siendo el mismo. Al estar los nodos pidiendo constantemente permiso al líder para acceder a recursos, estos son capaces de notar cuando cae el líder. En ese caso, el nodo que lo detecta convoca al resto para elegir un nuevo lider. El algoritmo de elección de líder utilizado es el Bully. En el Bully, el nodo que se proclama líder es el nodo con mayor valor respecto al resto (mayor ip). El nodo que detecta la caída del lider envía el mensaje ELECTION por broadcast. Los nodos mayores que él que reciben el mensaje le responden con un mensaje OK, y luego hacen broadcast con ELECTION. Este proceso se hace en todos los nodos, y si un nodo recibe un OK, es porque existe un nodo mayor que él, vivo en el anillo, por lo que su tarea en la elección terminó. Finalmente, el nodo de mayor valor nunca recibirá una respuesta de OK a su broadcast de ELECTION, por lo que al esperar un tiempo definido, se proclama líder, enviando un broadcast de LEADER. Todos los nodos reciben el mensaje de LEADER, y lo asumen como el nuevo líder.

La implementación está en `leader_election.py`. La clase `LeaderElection` se instancia en cada uno de los nodos del anillo cuando se crean, los cuales ejecutan en un hilo ininterrumpido la función `loop` de la misma, encargada de determinar, en caso de estar en elección, cuándo el nodo termina su trabajo en la elección. Las funciones de

- `_server`: Ejecuta un servidor para escuchar los mensajes ELECTION, OK y LEADER. `
- `_broadcast_msg`: Función para mandar un mensaje por broadcast.
- `_bully`: Función para determinar si el primer elemento es mayor que el segundo.

  Además tenemos las funciones que utiliza el nodo para interactuar con la clase LeaderElection.
- `get_leader`: Para preguntar cuál es el líder actualmente.
- `adopt_leader`: Utilizada por el nodo que se unen al anillo, para adoptar el líder que se encuentra elegido actualmente en el anillo.
- `leader_lost`: Utilizada por el nodo cuando detectan que se perdió el líder, para comenzar el proceso de elección.

> La función `_leader_checker` es una función que se ejecuta en un hilo ininterrumpido que controla periodicamente la existencia del lider en el anillo, que en caso de no existir este, convoca a elección. La existencia de esta forma de detectar la pérdida del líder es temporal, pues en la práctica, la comprobación se hace al pedir permisos de acceso a recursos al líder.

Autodescubrimiento:

Con el autodescubrimiento logramos que cuando un nodo se une al anillo, no tenga que conocer la dirección IP de ningún nodo específico. El nodo que se une logra esto emitiendo un mensaje de broadcast: DISCOVERY, junto al IP y puerto propios, que es captado por todos los nodos del anillo. Los nodos le responden con el mensaje ENTRY_POINT por un socket TCP al nodo solicitante a través de la IP y puerto enviados. El nodo está esperando las respuestas, pero solo le interesa la primera que llegue, pues esta ya contiene la ip del nodo que le va a servir de punto de entrada al anillo. Luego se une al anillo a través de `join`.

SelfDiscovery:

Cuando al inicializar el `ChordNode.py` utilizando el flag `-c`, definimos que queremos que el nodo se una a un anillo existente en la red local. Entonces creamos una instancia de la clase `SelfDiscovery` con el IP del nuevo nodo, y llamamos al método `find` de la instancia. El resultado de `find` debe ser la IP de algún nodo del anillo, que sirva como punto de entrada. La clase está compuesta por un método `_recv`, que es ejecutado en un hilo y que sirve como servidor para capturar las solicitudes TCP de un socket del nodo. Esta función es la que se encarga de recibir el mensaje de algún nodo del anillo que responda al nodo que se une. Por otro lado `find` es la encargada de enviar el mensaje inicial por broadcast a todos los nodos utilizando `_send`, y espera a que la función `_recv` ofrezca la IP del nodo que respondió al broadcast.