Distributed Tag-Base System

Descripción

El Sistema de Ficheros con Etiquetas Distribuido es un sistema distribuido que permite a los usuarios almacenar y recuperar archivos utilizando etiquetas. Está diseñado para ser escalable y tolerante a fallos. El sistema está construido utilizando una arquitectura peer-to-peer, donde cada nodo en la red actúa como cliente y servidor. Esto permite un sistema descentralizado que puede manejar grandes cantidades de datos y tráfico. El sistema está escrito en `Python`.

Instalación

Paquetes de `Python` utilizados

- `pickle`
- `umgspack`
- `rpyc`
- `asyncio`

Protocolo Kademlia

El sistema utiliza el protocolo `Kademlia` para comunicación peer-to-peer y almacenamiento de la tabla hash distribuida (`DHT`).

El protocolo `Kademlia` es un protocolo de tabla hash distribuida que permite a los nodos en una red localizar y recuperar datos de manera eficiente. Utiliza una estructura de árbol binario para organizar los nodos en la red, donde cada nodo es responsable de un subconjunto del espacio de ficheros. Los nodos se comunican entre sí para compartir información sobre las llaves de las que son responsables, lo que permite el enrutamiento y búsqueda eficientes de datos.

En el Sistema de Etiquetas Distribuido, cada nodo en la red utiliza el protocolo Kademlia para almacenar y recuperar archivos y sus etiquetas asociadas. Cuando se carga un archivo en la red, se genera una clave hash, que luego se utiliza para determinar qué nodo en la red es responsable de almacenar el archivo. El nodo almacena el archivo y sus etiquetas asociadas en su almacenamiento local.

Cuando un usuario busca archivos según etiquetas, el cliente envía una solicitud a uno de los nodos en la red para buscar archivos con la etiqueta especificada. El nodo luego utiliza el

protocolo `Kademlia` para localizar los nodos que son responsables de las claves asociadas con la etiqueta. El nodo luego envía solicitudes a estos nodos para recuperar los archivos y sus etiquetas asociadas. Una vez que se han recuperado los archivos y las etiquetas, el nodo los envía al cliente.

Tolerancia a fallos

El protocolo `Kademlia` asegura la tolerancia a fallos al permitir que los nodos se unan y abandonen la red de manera dinámica sin afectar el sistema en general.

Cuando un nodo se une a la red, contacta a otros nodos en la red para determinar su posición en la estructura de árbol binario. Luego, el nodo almacena información sobre los otros nodos que ha contactado en su almacenamiento local. Esta información incluye los IDs de nodo, las direcciones IP y los números de puerto de los otros nodos.

Si un nodo falla o se vuelve inaccesible, los otros nodos en la red aún pueden localizar y recuperar datos utilizando el protocolo `Kademlia` para enrutar alrededor del nodo fallido. Cuando un nodo necesita localizar una clave que no está en su almacenamiento local, envía una solicitud al nodo que está más cerca de la clave de los que conoce. Si ese nodo no tiene la información que se está buscando envía la solicitud al nodo más cercano a la clave que él conoce, y así sucesivamente, hasta que se encuentra la clave.

La tolerancia a fallos se logra mediante la replicación de archivos y sus etiquetas asociadas en varios nodos en la red. Esto asegura que si un nodo falla o se vuelve inaccesible, los archivos y etiquetas aún pueden ser recuperados de otros nodos en la red.

Arquitectura

El sistema consta de los siguientes componentes:

- Nodos: Cada nodo en la red es responsable de almacenar y brindar archivos. Los nodos se comunican entre sí para compartir información sobre los archivos que tienen y las etiquetas asociadas con ellos.
- Cliente: El cliente es responsable de interactuar con el usuario y enviar solicitudes a los nodos en la red. El cliente puede cargar, descargar, eliminar y buscar archivos según etiquetas en la red.
- Parser de comandos: Es responsable de analizar la entrada del usuario y generar un comando que puede ser ejecutado por el cliente.

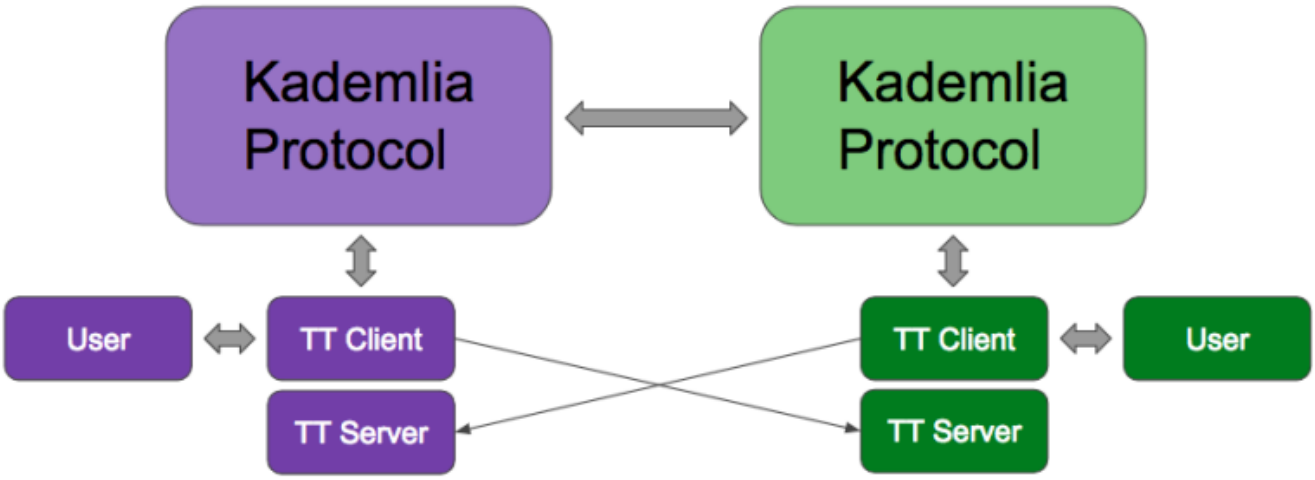
Estructura general

El sistema utiliza un cliente-servidor bidireccional que utiliza RPyC y TCP para enviar archivos.

Cada nodo que se une a la red de intercambio de archivos arranca el cliente-servidor bidireccional, lanzando tanto un servidor como un cliente, este último solo cuando es necesario. El usuario solicita o publica un archivo a través del cliente. Para publicar un archivo en la red, el cliente utiliza la API de Kademlia. Después de la publicación, los nodos conectados a la red Kademlia pueden solicitar a Kademlia el/los nodo(s) que alojan el archivo deseado.

Para solicitar la ubicación de un archivo deseado, el cliente utiliza la API de Kademlia para obtener el archivo especificando su etiqueta. El cliente recibirá la dirección IP y el puerto de otro servidor en la red que tenga ese valor. Una vez que el cliente recibe el par de dirección IP y puerto adecuado para descargar el archivo deseado, se comunica con el servidor al que pertenece ese par. En el caso de que varios nodos alojen el mismo archivo, el cliente recibirá una lista de pares de compañeros para contactar en caso de que uno falle.

Kademlia está estructurado para tener su propio protocolo que consta de un conjunto de instrucciones RPC sobre UDP.



Estructura del sistema

Flujo de datos:

- Los archivos se cargan en los nodos de la red y se almacenan de manera distribuida.
- Las etiquetas se asocian con los archivos y se almacenan de manera distribuida.
- Cuando un usuario busca archivos según etiquetas, el cliente envía una solicitud a uno de los nodos en la red para buscar archivos con la etiqueta especificada. El nodo luego devuelve una lista de archivos que coinciden con la etiqueta.

- Cuando un usuario descarga un archivo, el cliente envía una solicitud al nodo que tiene el archivo. El nodo luego envía el archivo al cliente.

Lenguaje de Comandos

La clase `CommandParser` en `client.py` es responsable de analizar la entrada del usuario y generar un comando que puede ser ejecutado por el cliente.

El método `parse` recibe una cadena como entrada y devuelve un objeto `Command`. El objeto `Command` tiene dos atributos: `name` y `args`. `name` es una cadena que representa el nombre del comando (por ejemplo, "add", "delete", "list", etc.), y `args` es un diccionario que contiene los argumentos del comando.

El método `parse` primero divide la cadena de entrada en tokens usando el espacio en blanco como delimitador. Luego verifica si el primer token es un nombre de comando válido. Si lo es, establece el atributo `name` del objeto `Command` en el nombre del comando. Si no lo es, genera una excepción.

El método `parse` luego itera sobre los tokens restantes y los agrega al diccionario `args`. Se espera que los argumentos estén en la forma "-arg value", donde "-arg" es el nombre del argumento y "value" es el valor del argumento.

Una vez que se han agregado todos los argumentos al diccionario `args`, el método `parse` devuelve la instrucción.

Instrucciones

add -f file-list -t tag-list

Copia uno o más ficheros hacia el sistema y estos son inscritos con las etiquetas contenidas en tag-list.

delete -q tag-query

Elimina todos los ficheros que cumplan con la consulta tag-query.

list -q tag-query

Lista el nombre y las etiquetas de todos los ficheros que cumplan con la consulta tag-query.

add-tags -q tag-query -t tag-list

Añade las etiquetas contenidas en tag-list a todos los ficheros que cumpan con la consulta tag-query.

delete-tags -q tag-query -t tag-list

Elimina las etiquetas contenidas en tag-list de todos los ficheros que cumplan con la consulta tag-query.

Funcionalidades adicionales

get -q tag-query

Descarga todos los ficheros que cumplan con la consulta tag-query. Los ficheros serán almacenados en la carpeta 'secure' del proyecto.