Proyecto de Sistemas Distribuidos: BitTorrent

Jesús Santos Capote, Kenny Villalobos Morales, Diamis Alfonso Pérez

Facultad de Matemática y Computación, Universidad de La Habana, La Habana, Cuba

Keywords: BitTorrent · FastAPI · Chord · Python · Docker · Replication · Distributed system

1. Funcionamiento General del Proyecto

Se propone una implementación del protocolo BitTorrent sobre una red peer to peer, donde cada peer es un Cliente BitTorrent o un Tracker, entidades que son a su vez servidores de FastAPI, utilizando la biblioteca de python fastapi, con el protocolo de comunicación HTTP.

1.1. Tracker

Servidor encargado de guardar información sobre que peer contiene que archivo, o parte del archivo. Los clientes BitTorrent hacen peticiones a estos servidores para saber que peer contiene, potencialmente, partes del archivo. El Tracker responderá a estas peticiones con una lista de tuplas (IP, Puerto) de peers a los cuales el cliente se puede conectar para lograr su objetivo, entre ellos, el peer que tiene el archivo completo. La lógica de esta entidad es implementada en la clase Tracker del archivo tracker.py.

1.2. Cliente BitTorrent

Un Cliente BitTorrent es un peer que sirve y descarga archivos. Un cliente es puede iniciar la descarga de un archivo a partir de tener el .torrent correspondiente al archivo que se quiera descargar. Todo cliente que se encuentra compartiendo un archivo es capaz de enviar el mismo (o partes de este, véase sección 1.5). Una vez que un cliente tiene una parte de un archivo inmediatamente este la comparte en la red para que otros clientes puedan descargar de él dicha pieza.

1.3. Archivo Torrent

Un torrent es un archivo de texto con extensión .torrent que contiene información sobre el archivo que se quiere descargar. Dicha información esta dispuesta en una serie de campos:

1. announce: IP y Puerto del Tracker que coordina la transferencia de archivos.

2. announce-list: IPs y Puertos de Trackers que tambien pueden coordinar la transferencia.
3. info: un diccionario que describe los archivos que se van a descargar, incluyendo su nombre, tamaño, número de piezas, etc.
4. piece length: el tamaño de cada pieza en bytes.
5. pieces: una cadena que contiene el hash SHA-1 de cada pieza de los archivos que se van a descargar.
6. name: el nombre de la carpeta o archivo que se va a descargar.
7. length: el tamaño del archivo que se va a descargar.
8. private: un valor opcional que indica si la descarga es privada o no.

1.4. Compartir un archivo

El proceso de intercambio en la red BitTorrent inicia cuando un cliente comienza a compartir un archivo local. Para esto se utiliza el método **upload_file** con la dirección de un archivo local que se quiere comenzar a compartir y con la dirección de algún tracker (potencialmente más de uno) al que se va a notificar que se está compartiendo el archivo. Se genera a partir de esta información el .torrent correspondiente al archivo a compartir. Posterior a esto se lo notifica al tracker que este cliente identificado por su ip:port está compartiendo el archivo reconocible por el identificador creado en el .torrent.

1.5. Descarga y transferencia de Archivos

Para iniciar la descarga de un archivo un cliente debe tener el .torrent correspondiente al archivo que se quiere descargar. A partir de las direcciones de trackers en el announce-list de este .torrent el cliente realiza la petición a un tracker de que peers de la red están compartiendo el archivo, a través del método **get_peers_from_tracker**, la respuesta a esta petición será una lista de peers que poseen al menos una parte del archivo. Para la transferencia de archivos, un archivo se desglosa por piezas, de un tamaño prefijado, y estas a su vez se desglosan en bloques, de un tamaño igualmente prefijado, los clientes se transfieren entre sí una pieza y esta a su vez se transfiere un bloque a la vez. Una vez que el cliente tiene la lista de peers brindada por el tracker necesita saber que piezas del archivo tienen cada uno de estos peers, para esto se utiliza el campo bitfield. Dado el .torrent de un archivo se conoce el número de piezas que tiene el mismo, y por ende los índices de cada una de ellas en el archivo, el bitfield de un archivo (potencialmente incompleto) es un array booleano donde cada casilla le corresponde a una pieza del archivo, y está será True solo si en el archivo ya está la pieza correspondiente a dicha casilla, esto es verificable usando el hash de cada pieza brindado en el .torrent. El cliente pide a cada uno de los peers en la lista brindada por el tracker sus bitfield respectivos al archivo en cuestión, luego con esa información calcula cual es la pieza más rara en la red (o sea, la pieza que menos peers poseen), e inicia la descarga de esta pieza desde alguno de los peers que conoce que la poseen. La prioridad de descarga de las piezas es por rareza, para contribuir a que la pieza más rara se disemine primero por la

red. Una vez que están descargados todos los bloques de la pieza, estos se unen y se construye la misma, luego se verifica si el hash de la pieza recibida coincide con el definido en el .torrent y, de ser así, se considera descargada la pieza, pasando a escribir esta en la ruta local correspondiente al archivo en descarga. Este proceso se repite hasta tener todas las piezas del archivo. Además desde el momento en que se descarga la primera pieza, el cliente notifica al tracker de que ya se encuentra listo para compartir el archivo.

2. Evitando puntos de falla. Protocolo CHORD entre los trackers y replicación de la información.

Cuando un cliente notifica a un tracker de que está compartiendo un archivo(o al menos partes de este), el tracker guarda en su base de datos(un diccionario de Python) el ip:port del peer que la está compartiendo bajo la llave del identificador del archivo, de esta manera si un peer quiere descargar el archivo le realiza la petición al tracker con el identificador del archivo. De esta forma si tenemos un solo tracker en la red la información sobre que peer tiene que archivo se encuentra centralizada en un solo servidor Tracker, lo cual constituye un punto de falla en la red. Con la implementación del protocolo Chord sobre los Trackers se evita esto, pues la información estará diseminada por el anillo Chord.

Cada Tracker posee un ID, que es el entero resultante de aplicar el hash sha256 a la concatenación de su IP y Puerto. El anillo estará ordenado a partir del orden entre los ID de los tracker que lo componen. Cada tracker conoce el IP y el Puerto de su sucesor, predecesor y de el predecesor de su predecesor en el anillo. La información que distribuyen es la almacenada en la Base de Datos de cada Tracker.

A un Tracker le corresponde en un anillo guardar la información sobre todos los pares (identificador de un archivo(hash de sha256), lista de peers que lo comparten), tal que el identificador del archivo sea menor o igual que el ID de dicho Tracker y mayor que el ID de su Tracker predecesor. Al Tracker con ID más bajo le corresponde los pares con hash menor que su ID y los pares con hash mayor que el ID del Tracker con ID más grande.

Se implementó una función **find-successor**, la cual a partir de un ID, ya sea de un Tracker o de un par, encuentra el sucesor de ese ID en el anillo Chord, es decir, el Tracker con ID mas cercano por abajo al ID pasado como argumento. Cuando un nuevo Tracker se va a unir al anillo Chord, llama a la función find-successor del Tracker que le sirve como entrada, pasándole su ID como parámetro. Así encuentra su sucesor en el anillo y se coloca en el lugar que le corresponde. Luego le pide a su sucesor los pares que le pertenece, reparte sus pares a sus respectivas posiciones en el anillo y finalmente el anillo queda en un estado consistente.

Cuando un Tracker pertenece a un anillo CHORD, además de contener su base de datos con los pares del anillo que le corresponden, contiene una base de datos de replicación, en la cual guarda todos los pares que le corresponden a su sucesor en la red. Cada tracker del anillo se encarga de hacer ping periódicamente

con su predecesor en el anillo y en el caso en que este falle (o sea, que su predecesor se desconectó de la red), toma los pares que le pertenecían a su predecesor(el que se desconectó) a través de la base de datos de replicación del predecesor de su predecesor, pues si su predecesor se va del anillo dichos pares le corresponden a él, y luego actualiza el estado de sucesor, predecesor y predecesor de su predecesor a todos los tracker que se ven afectados en la desconexión, finalmente actualiza la base de datos de replicación de su nuevo predecesor.

Los clientes BitTorrent son ajenos al funcionamiento de todo este proceso. Cuando un cliente hace una solicitud de peers o notifica que está descargando una pieza, envía el identificador de su archivo al Tracker que le sirve como entrada al anillo. Este llama a find-successor con ese valor para encontrar el Tracker al que le corresponde la información. Finalmente el Tracker al que se le hizo la solicitud responde al cliente o manda a actualizar la base de datos del Tracker correspondiente.