MINUTAS DEL TRABAJO PRÁCTICO FINAL DE SISTEMAS DISTRIBUIDOS

‘TORRENTE’

Integrantes del grupo:

- Valentina Fernández
- Enzo Nogueira Barria
- María Josefina Oller
- Juan Federico Santos Di Leo

Fecha de entrega: 6/12/2021

SEMANA DE 27 DE SEPTIEMBRE

Se empezó a implementar la estructura del servidor web a partir del código proporcionado por la cátedra.

SEMANA DE 4 DE OCTUBRE

Se implementó el diseño de cliente web.

SEMANA DE 11 DE OCTUBRE

Se agregaron funcionalidades al servidor web.

Se crearon interfaces gráficas de Alta de archivos y Torrent.

Se realizaron modificaciones necesarias en el componente de Cliente para utilizar la api Fetch.

Se realizó la gestión de las peticiones HTTP recibidas del navegador en el servidor web.

Se realizaron modificaciones necesarias en el componente de Cliente para utilizar la api Fetch.

Se realizaron actualizaciones en el diseño del cliente.

Se decidió utilizar la librería Express por su simplicidad por ende se hizo la modificación del servidor web.

Se realizaron actualizaciones en el Cliente.js

SEMANA DE 25 DE OCTUBRE

Durante la clase con el profesor y los otros grupos, se decidió que no es un objetivo del trabajo permitir la interoperabilidad entre trackers.

Se hizo unos arreglos en el Cliente.js

Se hicieron pequeños cambios en el servidor web, cliente y los HTML.

Se implementó la comunicación del servidor con el navegador.

Se implementaron pruebas para independizar al servidor del cliente.

Se limpiaron algunos archivos viejos que eran para pruebas que se fueron haciendo, y se reorganizó todo para las pruebas del cliente-servidor independientes. Además, quedaron

fijos el servidor implementado con 'express', y las peticiones HTTP cliente implementadas con 'fetch()'.
.
Se discutió sobre si era mejor que el cliente y el servidor sean dos componentes independientes o que el cliente y el servidor sean un mismo componente.Consultando al profesor, se decidió que el cliente y el servidor sean un mismo componente.

Se hicieron cambios necesarios para la estructura final de cliente-servidor (mismo componente) . El servidor cuenta con todos los HTML/CSS/JS que conformarían al 'cliente', ya que nuestro 'cliente' como tal realmente es el navegador. Entonces cuando en el navegador escribimos las URL's 'localhost:8080/' o 'localhost:8080/formulario', el servidor responde con los archivos correspondientes para que el navegador muestre las páginas.

Por todo esto, ya no hay una carpeta 'Cliente-Web', y dentro de 'Servidor-Web' están los HTML/CSS/JS, para que el servidor se los envíe al navegador cuando éste se los solicite. Y se actualizaron las rutas de cambio de páginas ('href') en los HTML para que se realice bien la comunicación con el servidor.

Además, se renombraron algunos archivos y ya quedaron actualizadas dichas referencias en los HTML y en el servidor.

Con todo lo mencionado anteriormente, damos por terminado el componente cliente-servidor.

Se hizo la creación de clase Nodo Tracker. Y se utiliza el módulo 'dbly-linked-list' para la lista doblemente enlazada de nodos tracker.

SEMANA DE 1 DE NOVIEMBRE

Se implementó la DHT como un arreglo de elementos Map para poder respetar el orden de los elementos, de esta forma para obtener el índice del arreglo se utiliza el parseado de key ,que es el hash, al decimal.

Se hicieron algunos cambios en Nodo Tracker tales como:

- Se agregó un archivo JSON (arreglo de objetos) con la configuración inicial de cada nodo tracker.
- Se definió que al iniciar el sistema, se instancien 4 nodos tracker.
- El concepto de 'lista doblemente enlazada circular' describe cómo será la comunicación UDP entre nodos tracker, por ende no habrá una 'lista' como tal en memoria.

Se hicieron un par de correcciones y se implementó la función hash en el servidor.

Se agregó un método a la clase NodoTracker donde se comienza a escuchar en el puerto UDP correspondiente (se podrán recibir mensajes de otro nodo tracker que sea vecino, o del servidor web).

Se agregó la implementación de la tabla Hash en la clase NodoTracker.

Se realizó la comunicación UDP de servidor a tracker.

SEMANA DE 22 DE NOVIEMBRE

Durante la clase, se discutió sobre el funcionamiento de nodos pares y descarga de archivos (definición de interacción entre pares).

Se realizó una actualización en la comunicación Servidor-Tracker.

Se definió la estructura del nodo par y se agregaron sus funciones.

Se consultó al profesor sobre los detalles de la implementación de la tabla Hash y se aplicaron las siguientes correcciones:

- Ahora el arreglo de Map's es dinámico.
- Se corrigió la forma de tratar los hashes, ya que siempre hay que manejarse con el hash SHA-1 completo, ya sea para insertar nuevos pares {clave:valor} a la tabla, o hacer búsquedas, etc. El tema de usar los 2 primeros chars del hash es únicamente para los índices del arreglo, y se maneja internamente en la clase HashTable. Por esto último, también se cambió en el servidor web la forma de retornar el hash cuando lo genera (ahora retorna el hash completo, y no solo los 2 primeros chars).

SEMANA DE 29 DE NOVIEMBRE

Se realizó la gestión de comunicación UDP de los trackers y se plantearon las funciones como store,count,search y found. Se realizó la implementación de scan y una función que analiza el tipo de petición.

Se implementó el ingreso del ID por consola en el Nodo Tracker, es decir, cada Nodo Tracker se ejecutará en consolas distintas. Para esto, se le pide al usuario que ingrese el identificador del tracker a instanciar (1, 2, 3 o 4). Además se acomodaron las pruebas de agregar archivos a la tabla hash del tracker.

Se realizó la corrección de nomenclatura ‘pair’ en Nodo Tracker para respetar el nombre que se les dio a los nodos pares en la interfaces del sistema, todo lo que decía 'pair' se cambió por 'par'. Y la lista 'pair_nodes' se cambió por 'pares'.

Se creó un módulo donde se ubicó la implementación de la tabla hash, para así reducir un poco el archivo del nodo tracker y que el código se enfoque más al nodo tracker como tal.

Se hicieron pequeños cambios en Nodo Par.

Se definió que el valor del atributo MessageId que colocaría el primer nodo tracker antes de comunicarse con el siguiente nodo tracker (y así saber cuándo se da toda la vuelta). De esa forma sería:

- scan: 1

- search:2
- store: 3
- count: 4

Se implementaron las funciones que faltaban del Nodo Tracker como count, search, found, la función que calcula el tipo de destino (adónde enviar el mensaje) y la función que cambia el atributo MessageId . Además se agregó unas funciones necesarias ,para el tracker, en la tabla Hash.

Se completó el Nodo Par con búsqueda de archivo en el directorio y descarga del archivo solicitado a partir del Hash. Además se cambió la forma de lectura de los archivos utilizando ‘fs’ que es más sencillo.

Se agregó una función que lista los archivos en el HTML.

Se implementó la función para cortar el recorrido en el nodo Tracker.

Se implementó la comunicación UDP con el nodo Tracker en el servidor web.

Se incluyó la require(‘sha1’) en nodo Par.

Se realizó corrección en la función que gestiona la vuelta de los mensajes en el nodo Tracker.

Se agregó la configuración inicial en el nodo Par.

Se realizaron algunas correcciones en el nodo Tracker y un posible planteo de la función ‘count’.

Se realizaron algunas correcciones en el servidor web donde ahora tanto en las peticiones de [Listar Archivos] como en las de [Dar de Alta un Nuevo Archivo], la respuesta que le manda el servidor web al cliente web (navegador) es directamente el atributo que necesita, en vez de mandarle el objeto completo que nos devolvió el nodo tracker.

Las respuestas son respectivamente el arreglo con la información de todos los archivos de la red, y la variable booleana que determina el éxito (true) o no (false) de la carga del nuevo archivo.

Además se hizo un cambio mínimo en la apariencia del objeto 'store' (dentro de solicitud_descarga()), para que sea más fiel a las interfaces del sistema (de todas formas no afecta al funcionamiento).

Se hizo la gestión de la respuesta de peticiones ‘Listar archivos’, es decir cuando el cliente web realiza un 'fetch()' al servidor web con la petición HTTP GET /file, el servidor le responde con un arreglo que contiene la información de todos los archivos de la red. Los elementos de dicho arreglo se insertan adecuadamente en la página HTML mediante una tabla dinámica (cada fila representa a un archivo).

Se hizo la gestión de la respuesta de peticiones ‘Descargar’ y ‘Subir’, es decir cando el cliente web realiza un 'fetch()' al servidor web con la petición HTTP GET /file/<hash> para solicitar la descarga del .torrente asociado a un archivo de la red, el servidor le responde con el contenido del .torrente para que pueda generarse dicho archivo (veremos que el navegador lo empezará a "descargar"). En caso de no encontrarse el archivo, el servidor nos devuelve un mensaje de error que mostraremos en una 'alerta'.

Cuando el cliente web realiza un 'fetch()' al servidor web con la petición HTTP POST /file/ para solicitar el alta de un nuevo archivo en la red (a partir de los datos escritos en el formulario), el servidor le responde con un valor booleano (true/false) indicando el éxito o no del alta (que luego lo traduciremos en un mensaje mostrado en una 'alerta').

Se realizó la implementación de ‘store’ .Ante una petición 'store', se definió que cada nodo tracker tiene un rango de índices disponibles en su tabla hash para guardar archivos (la tabla hash distribuida tendrá en total 255 índices, ya que se indexa con los 2 primeros bytes del hash SHA-1 de los archivos, por ende en nuestro caso inicialmente cada tracker tendrá 255/4 índices para usar). Para hacer esto se agregó en el archivo de configuración de los trackers, la cantidad de nodos tracker en la red (en nuestro caso inicialmente son 4). Entonces, cuando a un nodo tracker le llega un 'store', se fija si el archivo a almacenar corresponde a su rango de índices o no. Si le corresponde, lo guarda, y sino, le manda el 'store' a su nodo tracker vecino.

Se terminó de implementar los mensajes 'search-found', teniendo en cuenta que la petición de 'search' puede venir del servidor web, o de un nodo par.

Se aplicaron correcciones en el Nodo Tracker.

Se decidió que no utilizaremos la función ‘count’ en el Nodo Tracker.

Se resolvieron los conflictos de comunicaciones entre componentes. Ahora los nodos trackers responden correctamente tanto a las peticiones de 'search' del servidor web, como a los de los nodos pares. Con este arreglo también se solucionaron las respuestas a las demás peticiones ('scan' y 'store'). Se resolvió asignando ('bind') un puerto UDP específico a los clientes UDP del servidor web y de los nodos tracker, que es lo que se coloca en los '.originPort' de los mensajes.

También se hicieron correcciones en los nodos pares para que se pueda correctamente descargar un archivo, o empezar a compartir los archivos en la red (se pide por consola que acción realizar).