Distributed file system
Sistemas distribuidos
2020

Alumnos de Sistemas Distribuidos, Ing. En Computación, UNS

Fig. 1.

COORDINADOR

Será quien se encarga de resolver las peticiones de los nodos. Conoce la ubicación de los archivos, versiones y permisos. Al recibir una petición de un archivo por parte de un nodo, responde qué nodos poseen el archivo.

NODOS

Cuando un nodo ingresa en el sistema, reporta los archivos que posee al coordinador principal y este actualiza su lista de archivos asociados. Cuando un nodo sale del sistema, reporta los archivos que posee al coordinador y este, marca como inaccesibles los archivos asociados. Un nodo es a su vez un cliente, es decir, ejecutará la capa de aplicación para poder efectuar tareas como “ls”, “cd”, “mkdir”, “rm”, “cp”, “mv” y editar archivos mediamente un editor de texto, “editor”.

DECISIONES DE DISEÑO

¿Cómo se resuelve la caída del coordinador?

Existe en el sistema un coordinador de respaldo, el cual esta sincronizado con el coordinador principal y mantiene en consistencia la tabla de direcciones, permisos, archivos y nodos asociados. Ante cada modificación en la tabla del coordinador, este reporta la actualización al coordinador de respaldo. Entre los coordinadores, la comunicación será por sockets por conexión.

¿Cómo solucionamos la transparencia de ubicación?

Tanto el cliente como el programador desconocen la ubicación de los archivos.

¿Cómo solucionamos la consistencia de información?

El coordinador accede mediante un wrapper a una tabla que mantiene de cada archivo en el sistema su ubicación (IP del nodo), nombre, carpeta en la que se encuentra y permisos de lectura/escritura. Ante la creación o modificación de un archivo por parte de un nodo, este debe informarlo al coordinador para actualizar la tabla.

El coordinador se encarga de mantener una tabla con información sobre la ubicación de los archivos, esto es, el nombre del archivo y la dirección del nodo donde está almacenado. Ahora bien, para mantener la consistencia en los datos, las nuevas subidas de archivos en un nodo, deben actualizar también la tabla del coordinador.

La secuencia normal de funcionamiento detallada es:

- Un nodo cliente ejecuta el sistema pudiendo acceder a todas las funciones anteriormente mencionadas. En caso de que no las conozca, o no sepa como invocarlas, accede a las instrucciones con “help”.
- Un nodo cliente pide por la consola del sistema, por ejemplo, “ls”. El programador dispone de un wrapper de las llamadas RPC, de las cuales hace uso para solicitar al servidor RPC que le indique al coordinador que necesita que le resuelvan la petición “ls”.
- El coordinador recibe la solicitud por “ls” y accede a su tabla asociada otra vez mediante un wrapper (para no necesitar conocer cómo se guardan efectivamente los datos en la tabla) pidiendo listar los archivos correspondientes.
- El coordinador empaqueta la respuesta y la asigna a la respuesta RPC.
- El servidor RPC media entre el coordinador y el nodo, reenviando la respuesta.
- El nodo recibe luego de la llamada al wrapper RPC, los archivos a listar y mostrar por pantalla.

Tabla de indexado

Para el manejo de la tabla de indexado, como primera solución se planteó un manejador por medio de archivos, abriendo el archivo y recorriéndolo hasta identificar el objetivo.

Como se plantearon ciertos objetivos, se trató de seguir estos mismos en lo que sería la tabla de indexado. Los objetivos fueron tolerancia a fallas, migración, eficiencia, abstracción y consistencia. Para esto, se cambió el diseño del manejo de archivos por un manejo con una base de datos Mysql/MariaDB.

Tolerancia a fallas

Se implemento la base datos y el formato de la tabla de indexado con la idea de que la funcionalidad permita tener servidores replicados, esto permitiendo al coordinador en caso de fallar, poder levantar de otro lado la base de datos, con los datos actualizados al corriente y con la mejor integridad posible, teniendo siempre en cuenta que las perdidas deben ser las menores posibles. Cabe destacar que, por falta de tiempo, no pudimos implementar esta replicación en el coordinador, por lo tanto, la funcionalidad y la forma de trabajar con la base de datos quedo en solo implementación.

Migración

Se utilizo una base de datos con la idea de tener mayor control de los datos, permitiendo tener la posibilidad de que la base de datos este en otra locación, en este caso, se utilizó una librería capaz de permitir conectarse a una base de datos de forma remota. Ante esta utilidad, nos permite hacer replicación de datos, conectándonos a dos bases, o bien desde un manejo o administración de la base de datos original, podemos hacer una migración cada cierto tiempo hacia una base de datos de respaldo, idealmente 1 vez por semana.

Eficiencia

Con las primeras implementaciones de la tabla de indexado de la forma de archivos, notamos que el tiempo de llamado, lectura y escritura de un archivo de la tabla de indexado, tardaba suficiente tiempo para ser considerado ineficiente ante la consulta concurrente de diferentes nodos. Por esto se migro a un manejo de base de datos, el cual nos permitía mejorar los tiempos de consulta, mejor orden de los datos, búsquedas prácticamente instantáneas por el motor de la DB, y una utilidad que se implemento es la simulación de una cache. Esta cache, fue simulada mediante arreglos, los cuales eran llenados con los métodos de implementación, un ejemplo de esto, era la búsqueda de archivos o la lectura de un LS, llenando los arreglos, y filtrándolos de manera “local” según el pedido que hizo.

Abstracción

Se realizaron los métodos de consulta con la idea de mantener una abstracción a la hora del pedido de datos de la base. Los métodos devuelven datos, con filtros según el tipo de parámetros pasados, mejor dicho, un método de consulta en la base de datos, puede traer más de una información, con diferente significado, según los parámetros establecidos.

Consistencia

La base de datos se empleó con la idea de facilitar y hacer de manera más eficiente el manejo del estado del sistema, esto en archivos nos iba a resultar muy complicado, y el diseño transaccional de una base de datos, nos permite mantener este estado y acceder al mismo de forma atómica. Esta ventaja antes mencionada, lo que permite es eliminar el cuello de botella que produce la lectura y escritura de un archivo que se hace más notoria si muchos clientes realizan consultas constantemente.

Como última acotación sobre la base de datos, se logró hacer una buena conexión entre el lenguaje C y MySQL. Se implementaron varias librerías, pero la que mejor nos pareció, después de un análisis, y con la cual actualmente se suele trabajar en producción se seleccionó la librería: *libmysqlclient.dev* y *zlib1g-dev.*

Comunicación nodo-coordinador

La comunicación nodo-coordinador se realiza mediante RPC. Decidimos envolver las funciones RPC en un wrapper para poder abstraer a los otros módulos. De esta manera, ellos solamente necesitan pasar los parámetros, siendo el wrapper el encargado de armar el mensaje a enviar correctamente.

Por ejemplo, los módulos utilizan la función exists(CLIENT* clnt, char tipo, char* nombre, char* ubicacion) en la cual mandan los parámetros por separado, y el wrapper se encarga de juntar todo en un solo string y enviarlo utilizando la estructura de mensaje definida en protocolo.x

De manera similar, la función RPC llamada no resuelve la petición directamente, sino que vuelve a desarmar el mensaje y obtiene los parámetros originales, y con estos, llama a una función implementada en el coordinador que se abstrae de la noción de RPC y solo lidia con la tabla de indexado.

Comunicación nodo-nodo

La comunicación nodo-nodo se realiza mediante sockets. Y cumplen el rol de resolver los pedidos entre nodos entre ellas se encuentran los métodos rm, co ,download y mv. Decidimos encapsular y modularizar las conexiones del socket en un módulo *socketNodos.c* que provee 5 funciones:

- **startListening():** Ubica un hilo a escuchar pedidos de los demás nodos, además de resolver las peticiones según el caso(remove, copy ,download y move).
- **downloadFile(ip,ruta)**: Descarga un archivo desde otro nodo con IP “ip” y el archivo en la ruta “ruta”.
- **copyFile(ip,rutaOrigen,rutaDestino)**: Notifica al nodo con IP “ip” que tiene que copiar un archivo en la ruta “rutaOrigen” a la ruta local “rutaDestino”.
- **moveFile(ip,rutaOrigen,rutaDestino)**: Notifica al nodo con IP “ip” que tiene que mover un archivo en la ruta “rutaOrigen” a la ruta local “rutaDestino”.
- **removeFile(ip,rutaOrigen):** Informa al nodo con IP “ip” que tiene que eliminar un archivo en la ruta “rutaOrigen” a nivel local.

Cabe aclarar que cada una de las funciones mencionadas anteriormente son invocadas desde el cliente.

- **stopListening(): T**ermina la ejecución del hilo que escucha pedidos de los demás nodos. Esta función se invocará al momento de terminar la ejecución de un nodo.

Con estas funciones, los demás componentes del sistema se abstraen de cómo está implementada la comunicación. Solo basta con incluir el header asociado a socketNodos e invocar a alguna de estas funciones.

Coordinador: detalle

El coordinador es el que se encuentra directamente conectado con la tabla de indexado, y es el encargado de recibir las peticiones de los servidores y modificar la tabla de indexado de manera acorde.

Al ser un módulo separado del servidor, el coordinador se abstrae totalmente de la comunicación con RPC con la que se comunican los clientes y el servidor, y se encarga de implementar funciones de servicio para el servidor.

Las funciones que el coordinador implementa son las siguientes:

- **esValido(Tipo, Nombre, Ubicacion):** Retorna si una carpeta o archivo es válido (existe) en una determinada ubicación.
- **funcionListar(Ubicacion):** Retorna todos los elementos que existen en una carpeta (otras carpetas y archivos)
- **obtenerIP(Archivo, Ubicacion):** Retorna la IP de un determinado archivo.
- **insert(Tipo, Nombre, IP, Ubicacion):** Inserta un nuevo elemento (archivo o carpeta) en la tabla de indexado.
- **delete(Tipo, Nombre, IP, Ubicacion):** Elimina un elemento (archivo o carpeta) en la tabla de indexado.
- **carpetaVacia(Nombre):** Retorna si una carpeta contiene algun elemento o no.
- **modificarIP(NuevaIP, Archivo, Carpeta):** Cambia la IP en la que se encuentra un determinado archivo.
- **isFile(Nombre, Ubicacion):** Verifico si un determinado elemento es un archivo o una carpeta.
- **modificarDirectorio(Nombre, CarpetaVieja, CarpetaNueva):** Cambia la ubicación en la que se encuentre un determinado archivo.

Utilizando estas funciones, el servidor maneja el comportamiento de los pedidos de los clientes.

Nodo: detalle

El nodo es donde se encuentra el cliente y la shell. Al iniciar se lleva a cabo el proceso de inicialización y conexión con el servidor. Se pide la dirección de la carpeta a sincronizar remotamente (Esta es la carpeta que elegimos donde se van a guardar los nuevos archivos y carpetas), y luego el módulo inicializador entra en acción. Este obtiene todos los datos cargados por el nodo en la base de datos (Si ya había entrado previamente) e inserta aquellos archivos que no se encuentren ingresados. Si ya existe un archivo con el mismo nombre, pero cargado por otro nodo, entonces se lleva a cabo un renombrado lógico en la base de datos, de forma que los nombres reales de los archivos queden libres de cambios.

Una vez finalizado el proceso de sincronización, el cliente puede usar las operaciones que desee.

Operaciones provistas

- **mv** [archivo origen] (directorio destino): mover archivo a otro directorio.
- **cp** [archivo origen] [directorio destino absoluto]: copiar archivo a otro directorio.
- **rm** [-f/-d] [archivo/directorio]: eliminar un directorio vacío o un archivo.
- **ls**: listar el directorio actual.
- **cd** [directorio]: cambiar de directorio.
- **editor** [archivo]: editar un archivo existente o nuevo.
- **mkdir** [directorio]: crear un directorio nuevo.
- **exit**: salir del DFS.

Función CP (COPY)

La función CP contempla comunicación nodo-coordinador, como así también comunicación nodo-nodo. El cliente utilizará el comando de la siguiente manera:

cp *SOURCE DEST*

El source es el archivo origen, y el destino es la ruta absoluta donde se pegará el mismo. Por decisión de diseño no se podrá reemplazar un archivo que se encuentre en la ruta destino, y, además, no se contempla realizar la operación dentro de una misma carpeta.

El proceso de copiado es el siguiente:

- Se valida que el usuario haya ingresado correctamente los argumentos, es decir, que el archivo origen exista y que el directorio destino también sea parte del sistema.
- Se verifica que no haya un archivo con el mismo nombre en el directorio destino
- Una vez que haya pasado todos los controles se procede a realizar la copia física:
    - Se obtiene la IP del nodo que contiene el archivo origen.
    - Se le envía "la orden" de realizar la copia.
    - El nodo que recibe el comando realiza la copia de manera local en el nivel que corresponda, y responde el resultado de su ejecución. Esto lo realizamos de esta manera ya que asumimos que las carpetas no tienen "propiedad", por lo que es indistinto que la copia se realice de esta manera, y tiene el mismo significado para el DFS. Adicionalmente, nos evitamos enviar un archivo a través de la red.
- Una vez que recibimos que la ejecución en el otro nodo fue exitosa, comunicamos la creación del archivo al coordinador.

Función MV (MOVE)

La función mv contempla comunicación nodo-coordinador, como así también comunicación nodo-nodo. El cliente utilizará el comando de la siguiente manera:

mv *SOURCE [DEST]*

El source es el archivo origen, y el destino es la ruta a donde se moverá el mismo. Para mover un archivo a la raíz no es necesario ingresar el argumento destino.

El proceso del move es el siguiente:

- Se valida que el usuario haya ingresado correctamente los argumentos, es decir, que el archivo origen exista y que el directorio destino también sea parte del sistema o que sea la raíz.
- Una vez que haya pasado todos los controles se procede a realizar el move físico:
    - Se obtiene la IP del nodo que contiene el archivo origen.
    - Se le envía "la orden" de realizar el move.
    - El nodo que recibe el comando realiza el move de manera local en el nivel que corresponda, y responde el resultado de su ejecución. Esto lo realizamos de esta manera ya que asumimos que las carpetas no tienen "propiedad", por lo que es indistinto que la copia se realice de esta manera, y tiene el mismo significado para el DFS. Adicionalmente, nos evitamos enviar un archivo a través de la red.
- Una vez que recibimos que la ejecución en el otro nodo fue exitosa, comunicamos al coordinador que debe actualizar la carpeta padre del archivo movido.

Función RM (REMOVE)

La función RM contempla comunicación nodo-coordinador, como así también comunicación nodo. El cliente utilizara el comando de la siguiente manera:

```
rm option SOURCE
```

-f, en caso de intentar eliminar un archivo

-d, en caso de intentar eliminar una carpeta

El proceso de eliminado es el siguiente:

- Se valida que el usuario haya ingresado correctamente los argumentos, es decir, que el archivo o directorio exista y sea parte del sistema.
- Si se va a eliminar una carpeta, se verifica que este vacía.
- Si la carpeta a eliminar está vacía, entonces avisamos al servidor que debe eliminarla. También la eliminamos en nuestro directorio físico.
- Si por el otro lado, se debe eliminar un archivo, primero se verifica que de verdad sea un archivo lo ingresado por parámetro.
- Una vez verificado, se pide al servidor la ip del dueño de ese archivo.
- Una vez con la ip, comparamos con la nuestra para ver si somos nosotros los dueños del archivo.
- Si el archivo es nuestro, se envía una actualización al servidor y se elimina el archivo localmente.
- Si el archivo es de otro nodo, se le envía un mensaje a ese nodo pidiendo que borre el archivo.
- Una vez que recibimos que la ejecución en el otro nodo fue exitosa, se comunica el borrado del archivo al coordinador.

Función MKDIR (CREAR DIRECTORIO)

```
mkdir nombre_carpeta
```

La función mkdir, así como en Linux, crea una carpeta tanto localmente como en la base de datos. Primero se verifica que la carpeta no exista actualmente y luego le avisa al coordinador que inserte una nueva carpeta. En el sistema solo se admite un único nivel de carpeta, por lo tanto, todos los directorios creados serán hijos de la carpeta raíz. Si se intenta crear un nuevo nivel de carpetas, un mensaje será lanzado al usuario.

Función LS (LISTAR)

```
ls
```

La función ls, así como en Linux, lista los contenidos de una carpeta. No se pide argumento, sino que funciona en base a la carpeta actual. Cuando se invoca el comando, se chequea en la base de datos si existe la carpeta actual, por si se eliminó mientras nosotros estábamos dentro. Luego se le pide al coordinador que nos devuelva una lista con los nombres de las carpetas y archivos dentro de la carpeta actual, y luego se muestra por pantalla.

Función CD (CAMBIAR DIRECTORIO)

```
cd nombre_carpeta
```

La función cd, así como en Linux, cambia el directorio actual al indicado. Para ejecutarse, el nodo envía un pedido al coordinador preguntando si la carpeta ingresada por parámetro existe en la base de datos. Si el coordinador responde que sí, entonces se actualiza el estado local de la carpeta actual.

Función EDITOR

`editor` *`nombre_archivo`*

La función editor sirve para poder crear archivos, leerlos y modificar su contenido. Para ejecutarse, el nodo envía un pedido al coordinador preguntando la ip del nodo dueño del archivo. Una vez que el coordinador responde, se inicia una comunicación con el nodo correspondiente para poder obtener el contenido del archivo. Si el archivo es uno nuevo, entonces no es necesario obtener sus datos, sino que basta con crearlo.

Una vez terminado ese proceso se abre una pantalla con el contenido del archivo, el cual podremos leer o modificar.

El uso del editor es libre para todos los nodos en cualquier momento, todos pueden abrir el archivo y modificarlo, pero solo la última copia guardada prevalecerá en el sistema.

Cabe aclarar que la creación de archivos nuevos se dará luego de guardar el archivo, para evitar ciertos inconvenientes.

Función EXIT

`exit`

La función exit termina la ejecución del nodo, llevando a cabo el borrado de archivos cargados por ese nodo, de forma que no aparezcan accesibles para otros nodos que aun sigan activos. Luego de eliminarlos, cierra la conexión y termina la ejecución del DFS.

Repositorio