Lab 2: Como los nodos de un sistema se coordinan mediante el uso de dos algoritmos de exclusión mutua

Raúl Álvarez 201773010-2 Cristian Bernal 201773026-9

Antes de explicar todo, se deja el link del repositorio github: https://github.com/sirbernal/lab2SD

¿Qué y cómo se hizo?

Lo que se realizó en este laboratorio fue un sistema en el cual pueda subir y descargar archivos (en este caso son libros pdf). Por un lado en la subida del archivo, este primero será dividido en chunks y estos serán distribuidos en los distintos nodos que existen, donde dependiendo del algoritmo, la manera y forma de la distribución cambiará, aunque habrá un nodo donde llevará el registro de donde están distribuidos los chunks. Este último nodo nos servirá para cuando se necesiten descargar libros, ya que a él le pediremos la lista de libros en el sistema y donde están ubicadas las partes, para luego descargar todas las partes y armar el libro.

Para explicar cómo funciona todo este sistema, nos centraremos en cada componente:

Cliente

Este será el encargado de descargar o subir algun libro, para ello dispondrá de un menú con el cual establecerá primero al sistema el tipo de distribución (se explica más adelante). Luego, dispondrá de la opción de subir o descargar libros

- Para la opción de subida seleccionada en el menú, el menú preguntará por el datanode a contactar y el nombre del archivo que debe de estar en la carpeta de cliente. Lo que se realizará es llamar a la función ***`SubirArchivo()`*** que dentro de ella creará los chunks del archivo, primero enviará la información del archivo (nombre del libro, cantidad de chunks) mediante ***`Upload()`*** al datanode seleccionado y luego enviará cada chunk mediante el canal de grpc ***`UploadChunks()`*** a este mismo datanode
- Para la opción de descarga en el menú, lo primero que realizará es contactar a datanode mediante ***`SolicitarLibros()`*** que lo que hará es pedirle la lista de libros disponibles en sistema. Luego, el cliente desplegará la lista de libros y al seleccionar uno a descargar, pues se contactará nuevamente a Namenode para pedirle la ubicación de los chunks del libro mediante ***`SolicitarUbicacion()`*** para luego llamar a ***`DescargarChunks()`*** que contactará a los Datanodes específicos para pedirle los chunks específicos y luego crear el libro con todos los chunks. Este estará en la carpeta cliente.

Datanode

Estos son 3 nodos que básicamente están encargados de almacenar los chunks de los libros. Para que estos funcionen, se debe tener un cliente activo que les mande el tipo de distribución que debe de aplicar el sistema. El comportamiento será distinto dependiendo del algoritmo de distribución cuando se sube un archivo

Algoritmo Exclusión Mutua Centralizada :

Cuando el cliente se contacta con algún datanode para subir algún archivo, lo primero que hará es recibir la información del libro (nombre y cantidad de chunks) recibidas en ***`Upload().`*** Luego, empezará a recibir los chunks en forma de bytes que el cliente le enviará mediante ***`UploadChunks()`***, entonces cuando detecte que recibió el último chunk, este verificará la disponibilidad del Namenode, cuando este esté libre, se plantea una propuesta de envio mediante ***`GenerarPropuesta()`***, ésta considera que todos los nodos están conectados. Esta propuesta es enviada a Namenode, quien rechazará o aceptará la propuesta (se explicara en su sección). Luego de concretado este paso, se procede a liberar al Namenode. En el caso de ser aceptada la propuesta se empieza a distribuir los chunks acorde a la misma, en caso de ser rechazada, se distribuye de acorde a la nueva propuesta recibida por el Namenode. Esta será enviada a través del canal ***`Distribucion(),`*** que al recibir un chunk usará la función ***`SaveChunk()`*** que leerá el byte recibido en el mensaje de distribución, transformándolo en un chunk (archivo).

Algoritmo Exclusión Mutua Distribuida:

Al igual que en el algoritmo anterior, recibirá siempre antes la información del archivo a subir (nombre y cant. de chunks) y luego los bytes de chunks. Cuando reciba todos los chunks necesarios, el Datanode que recibió los chunks (de aquí en adelante lo llamaré encargado) contactará a sus pares de datanodes, con una propuesta de distribución generada por ***`GenerarPropuesta()`***, ésta considera que todos los nodos están activos. Cada

nodo recibe la propuesta mediante ***Propuesta()***, lo que hará es revisar la propuesta comprobando que los nodos que implican la propuesta estén conectados mediante ***AllAlive()***, para así enviar de vuelta un true o un false. Entonces, si el Datanode encargado recibe de parte de sus pares true's, hará que el encargado pueda continuar al siguiente paso, pero basta con que uno de sus pares rechace la propuesta y esto hará que el encargado vuelva a plantear una propuesta nueva mediante ***GenerarPropuestaNueva()***, esta vez viendo que nodos no considerar ya que si la propuesta es rechazada es porque algún nodo no está conectado, por lo que nuevamente enviará esta propuesta hasta que reciba una aceptación de sus pares. El siguiente paso es entonces enviar la propuesta a Namenode para que haga registro de ella, para ello lo primero es mediante ***RicartyAgrawala()*** lo que realiza esta función es contactar a los pares de datanodes preguntándoles si están contactando a Namenode (ya que tienen una variable de activación llamada "ocupado" que si es true es porque están o quieren contactar al datanode), por lo que hay 2 posibilidades: que los demás nodos no esten interesados en entrar a Namenode dandole permiso a este datanode para hacerlo, y la otra es que uno de ellos si lo este, por lo que él obtendrá la id de aquel nodo (los nodos poseen un valor de id, que entre mayor sea más prioridad tiene, la prioridad definida fue datanode1>datanode2>datanode3), procederá a comprobarla con la suya y si es mayor podrá acceder al namenode, si no no podrá hacerlo y por lo tanto ***RicartyAgrawala()*** retorna falso. Como nosotros llamamos a esta función en un for (como un while !***RicartyAgrawala()***) esto hará que vuelva a preguntar a los nodos si es que puede acceder, hasta que los nodos no esten interesados.
Procederemos entonces a contactar a Namenode si es que los demás Datanodes no están contactando a Namenode, acá también usaremos el canal de ***Propuesta()***. Cuando recibimos respuesta de Namenode, entonces nuestro paso final es distribuir los chunks a cada Datanode mediante el canal ***Distribucion(),*** al igual que en algoritmo centralizado, cada datanode al recibir el byte de chunk correspondiente usará la función ***SaveChunk()*** para escribir el chunk a archivo.

Namenode

Es el único nodo que se encargará de mantener el registro de la ubicación de los chunks en cada datanode. Puede que tenga una función adicional dependiendo del tipo de distribución.

Algoritmo Exclusión Mutua Centralizada :

Cuando recibe una propuesta del Datanode encargado mediante el canal de ***Propuesta()***, luego de actualizar su estado a ocupado, ésta verificará que todos los nodos estén conectados mediante la función ***AllAlive()***, esto debido a que se asumió que la primera propuesta considera que todos los nodos están conectados. En el caso de que estén todos los nodos vivos, pues hará registro de la propuesta mediante ***GuardarPropuesta()*** y enviará de vuelta al DataNode con un true que le indica que aceptó la propuesta. En el caso contrario, cuando alguno de los datanodes no está disponible, Namenode realizará una nueva propuesta mediante ***GenerarPropuestaNueva()*** tomando en consideración los nodos que sí están conectados para la nueva propuesta, enviará esta nueva propuesta y un false que le indicaría al Datanode encargado que la propuesta fue rechazada. Esta nueva propuesta antes de ser enviada se guarda también con ***GuardarPropuesta()*** para tener el registro de esta. Finalmente, el namenode esperará el mensaje de respuesta por parte del datanode para confirmar la llegada de la respuesta anterior actualizando su estado a disponible para permitir el acceso a otra propuesta.

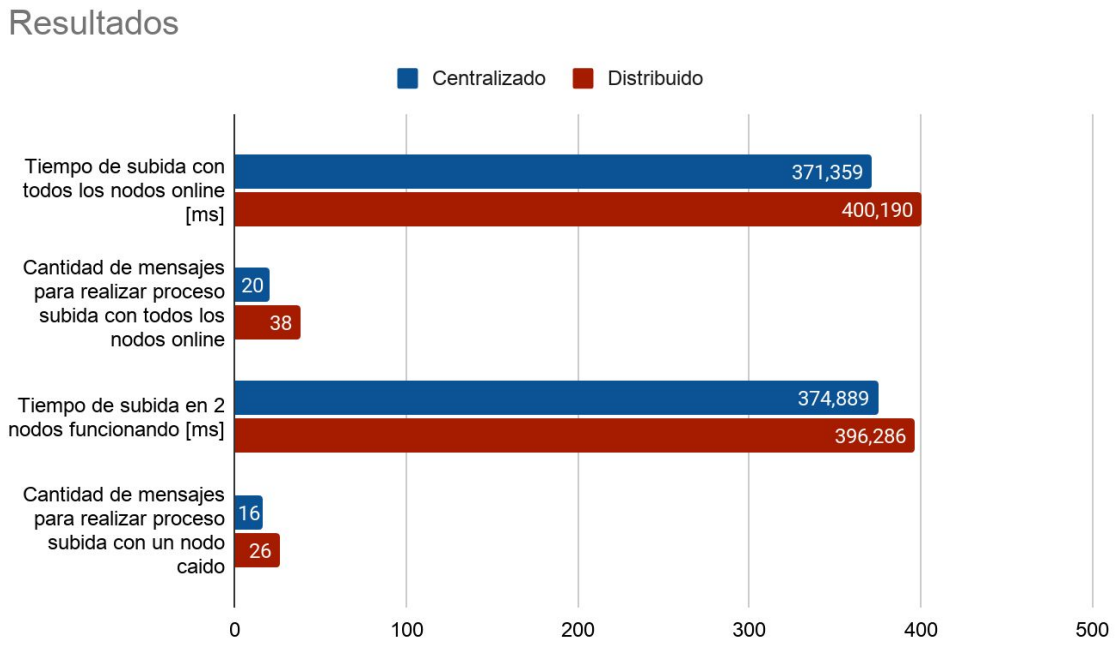
Algoritmo Exclusión Mutua Distribuida:

Aquí el datanode solo estará encargado de recibir la propuesta ya aceptada por los Datanodes, por lo que al recibir una propuesta simplemente se guarda mediante ***GuardarPropuesta()*** y se envía un true informando que se guardó la propuesta.

Los detalles más en profundidad de las funciones mencionadas están en el propio código comentado, ya que si se detalla todo el largo de este informe sería mayor a 4 hojas. En el readme además se menciona como ejecutar el sistema.

Resultados

Se realizaron mediciones de tiempo de subida de archivos respecto a los algoritmos de distribución



Para la toma del tiempo de ejecución se tomó el tiempo antes de realizar el envío del archivo en chunks, osea antes de usar la función ***`SubirArchivo()`*** dentro de cliente y apenas termine de usar esta función se toma el tiempo final, para luego hacer la comparación de los tiempos, con la librería time de golang. Para contar los mensajes, para centralizado contamos cada vez que se realizaba un mensaje en el datanode encargado por lo que llevábamos un contador, además añadimos uno en namenode pues este hace envíos de mensajes en ***`AllAlive()`***, por lo que al final imprimimos estos contadores y sumamos estos valores. Luego, para el distribuido, como cada datanode hace el llamado de ***`AllAlive()`*** para verificar que cada nodo esté conectado para aceptar o rechazar la propuesta enviado por el datanode encargado, entonces además a la llamada de esta función imprimimos los mensajes en cada datanode. Entonces, finalmente sumamos todos los mensajes obtenidos

Análisis

Notamos primero que el algoritmo de exclusión mutua centralizada se demora menos en comparación al de exclusión mutua distribuida, como también que la cantidad de mensajes de este último es mayor que al centralizado. Si bien, se puede atribuir que entre menor cantidad de mensajes enviados, menor es la cantidad de tiempo de demora, notamos que en nuestro segunda experimentación, cuando tenemos un nodo caído, notamos que esto no se puede afirmar al 100%, puesto que al ver el algoritmo centralizado, que su cantidad de mensajes se disminuyó de 20 a 16 al tener un nodo caído, pero el tiempo aumento. Esto puede ser explicado debido a que al tener un nodo menos, la primera propuesta que se genera considera que los 3 nodos están conectados y esto no es cierto, por lo que Namenode debe generar una nueva propuesta, implicando que el proceso se demore más.

Discusión

Como grupo, antes de realizar las pruebas ya teníamos la idea de que el algoritmo centralizado iba en general a tener menos demora como también menos cantidad de envío de mensajes en comparación al distribuido, puesto que este último para que recién se acepte una propuesta se debe contactar con los demás nodos y estos verifican que sus pares estén conectados, lo que se mencionó en clases, el multicast (que es una desventaja de este método de distribución). Al analizar los datos, notamos que claramente se cumplía lo que esperábamos, como mencionamos anteriormente en el análisis, el algoritmo centralizado tuvo menor cantidad de mensajes y de tiempo de demora que el distribuido.

Quizás si pudo haber existido una falla en las métricas, es el hecho de contar con exactitud la cantidad de mensajes en total que hacen cada tipo de distribución, con el algoritmo centralizado si era preciso pues era bastante más simple el proceso (de hecho este es una ventaja de este proceso) en comparación al algoritmo distribuido, ya que por ejemplo, no podíamos poner a prueba el algoritmo de Ricart y Agrawala al 100% ya que no encontrábamos alguna forma en que se pudieran realizar 2 propuestas al mismo tiempo, como bien se notó en los resultados, el proceso tomaba milisegundos en completarse, por lo que sí se generaban 2 propuestas al mismo tiempo nunca existiría teóricamente un problema de acceso a Namenode porque todo se hacía demasiado rápido.

Conclusión

Con la realización del Laboratorio 2 de Sistemas Distribuidos al programar un sistema con 2 tipos de distribución pudimos ver en práctica cómo se desempeñan estos algoritmos. Por un lado, el algoritmo de exclusión mutua centralizada fue la más fácil de implementar, ya que solo uno de los nodos el cual era el namenode era el encargado de aceptar y rechazar propuestas, esto es lo que en clases se aprendió de la simplicidad de este algoritmo. Esta misma simplicidad hace que si el Namenode esta caido, seria imposible de hacer una distribución con este algoritmo, pues el trabajo más importante lo hace el. Por el otro lado, el algoritmo de exclusión mutua distribuida tomó más tiempo en poder implementarlo, puesto que había que comunicar a todos los datanodes entre sí para que se coordinarán en aceptar una propuesta. Con los resultados obtenidos, claramente notamos la desventaja de este último algoritmo, que el hecho de que todos los datanodes se tengan que comunicar entre sí producía un multicast y mayor tiempo de ejecución de la distribución, por lo que si se tiene un sistema con una gran cantidad de nodos, puede producir un problema con el tiempo de ejecución. Finalmente, independientemente del algoritmo a utilizar, se puede ver que la solicitud de descarga de un archivo es imposible si el namenode se encuentra fuera de servicio, ya que este es el encargado de guardar la ubicación de cada parte necesaria para generar el archivo nuevamente.