Biblioteca online distribuida

Rodrigo Cayazaya Marín / 201773538-4
Jean-Franco Zárate González / 201773524-4

Que se hizo y Cómo se hizo

Para realizar el laboratorio, se utilizaron 4 máquinas, las cuales se comunican entre ellas mediante gRPC utilizando Protocol Buffers, ocupando funciones de cliente y servidor, puesto que deben enviar y recibir mensajes por medio de este protocolo en el lenguaje de programación Go.

Se implementó una interfaz, la cual el cliente puede ver y escoger diferentes opciones. Primero, el cliente podrá elegir si decide subir o descargar un libro. En caso de querer subir un libro, se le despliega una lista con los libros que puede subir, ya que estos se ubican en su carpeta “libros_cliente”, una vez escoge el libro, se le da la opción al cliente de escoger el algoritmo de repartición de chunks para subir el libro, estos son el algoritmo centralizado y el distribuido. Luego de escoger, se generarán propuestas que dependen del algoritmo utilizado.

Para el caso centralizado, la propuesta es generada por un DataNode definido de manera aleatoria y enviada directamente al NameNode, este último es el encargado de revisar si es una propuesta factible, viendo el estado de los Data Nodes, para ver si la acepta o no. En caso de que no se acepte, el NameNode genera una nueva propuesta considerando los Data Nodes activos. Una vez se acepta la propuesta, se envían los chunks a los Data Nodes correspondientes y se escriben los datos del libro (como el nombre, la cantidad de partes y las ubicaciones de cada parte) en un registro “log.txt” ubicado en la carpeta del NameNode, el cual se encarga de manejar este registro.

Para el caso distribuido, la propuesta se genera en un DataNode definido de manera aleatoria, el cual distribuye las propuestas hacia los otros Data Nodes para que estos acepten. La primera diferencia notable entre este algoritmo y el centralizado, es que las propuestas no las acepta el NameNode, sino cada DataNode por separado. En el caso de que se pierda la conexión de un DataNode, el DataNode seleccionado para generar la propuesta, genera una nueva propuesta considerando los Data Nodes activos y luego, realiza la misma acción de escribir en el registro “log.txt”.

Para prevenir choques de Data Nodes al momento de escribir en el registro, se utiliza el algoritmo de “Ricart & Agrawala”. Este consiste en preguntar, al momento de querer escribir, si algún otro DataNode quiere o está escribiendo en el registro. Si ya está escribiendo, el DataNode debe esperar a que termine para poder escribir, sin embargo si

existe otro DataNode que quiera escribir, el DataNode con una “Id” mayor podrá escribir, mientras que el otro deberá esperar. En cualquier otro caso el DataNode puede escribir sin ningún problema.

Luego de aceptar la propuesta, ya sea en el algoritmo centralizado o el distribuido, se procede a enviar los chunks a los Data Nodes correspondientes, los cuales son almacenados en memoria.

Para el caso de querer descargar un libro, se despliega una interfaz con la lista de libros que se encuentran en el registro “log.txt” manejado por el NameNode. El cliente puede elegir el libro que desea descargar. Una vez elija el libro, el cliente pedirá las partes correspondientes del libro, ubicadas en sus respectivos Data Nodes, los cuales enviarán las partes que estos mantienen almacenados, para luego el cliente une estos chunks y reconstruye el libro completo con extensión “.pdf”, para luego guardarlo y finalmente estar disponibles para la lectura.

Resultados

Se ejecutó el cliente y se utilizaron los algoritmos distribuido y centralizado, subiendo el libro de Alicia en el País de las Maravillas, el cual tiene 2 partes. En cada ejecución de una función gRPC se sumaba un contador, ya que se consideró que un mensaje es ida y vuelta. Además, se implementó la función time.Now() al mandar el primer mensaje desde el cliente, hasta que este mensaje sea respondido. Para calcular el intervalo de tiempo de ejecución se utilizó la función time.Since(), este tiempo considera desde el momento en que el cliente realiza la solicitud hasta se logra escribir en el log.
Dado esto, se obtuvieron las siguientes cantidades de mensajes y tiempos:

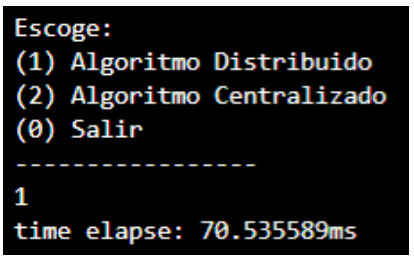
<u>Distribuido:</u>

```
-----------------
Escoge:
(1) Algoritmo Distribuido
(2) Algoritmo Centralizado
(0) Salir
-----------------
1
2020/12/02 18:29:53 La cantidad de mensajes enviados es: 2
```

Cantidad de mensajes del cliente

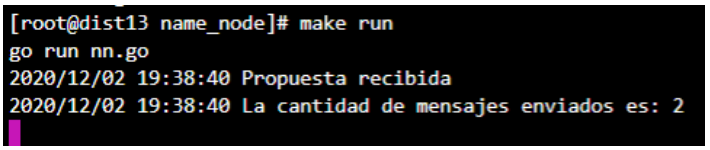
```
[root@dist16 data_node_3]# make run
go run dn_3.go
2020/12/02 18:29:54 La cantidad de mensajes enviados es: 0
2020/12/02 18:29:54 La cantidad de mensajes enviados es: 7
```

Cantidad de mensajes del Data Node

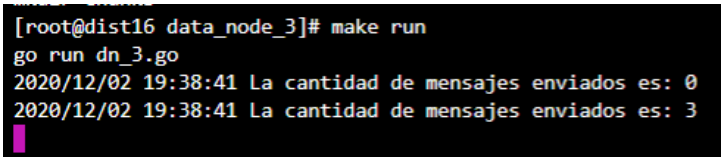


Tiempo de ejecución

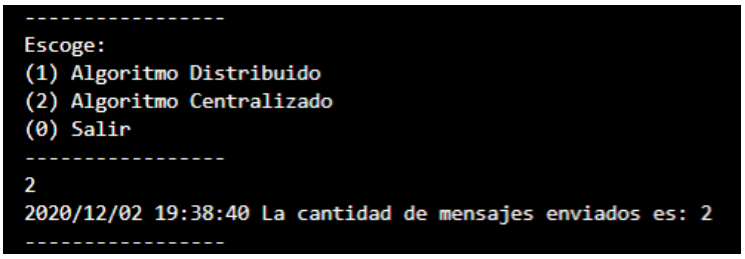
Centralizado:



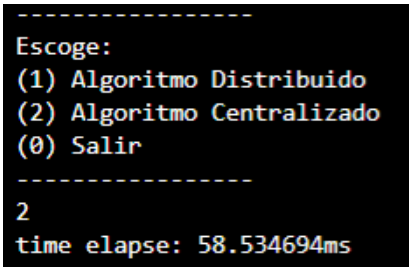
Cantidad de mensajes del Name Node



Cantidad de mensajes del Data Node



Cantidad de mensajes del cliente



Tiempo de ejecución

Análisis

Para el caso del algoritmo distribuido, se puede ver que la cantidad de mensajes es de 9, sumando los del cliente y los del Data Node. Cabe destacar que un mensaje es considerado ida y vuelta. La cantidad de mensajes se ve afectada por la cantidad de ejecuciones de funciones gRPC las cuales se comunican entre máquinas. El tiempo de ejecución que arrojó el test fue de 70.5 [ms] aproximadamente.

Para el caso del algoritmo centralizado, se puede ver que fueron 7 mensajes, 2 del Name Node, 3 del Data Node y 2 del cliente. Podemos ver que la cantidad de mensajes del algoritmo distribuido es mayor que del algoritmo centralizado. El tiempo de ejecución que arrojó el test de centralizado fue de 58.5 [ms] aproximadamente, el cual equivale a un 82% del distribuido.

Discusión

Como se pudo observar, la cantidad de mensajes en el algoritmo distribuido es mayor que en el algoritmo centralizado, esto es debido a que, en el algoritmo distribuido, un Data Node genera una propuesta que deben aceptar los otros Data Nodes, es por ello que tenemos 2 mensajes más que en el centralizado. Por otro lado, en el algoritmo centralizado es el Name Node el encargado de ver las propuestas y aceptarlas dependiendo del estado de los Data Nodes, limitando la cantidad de mensajes.

Para el caso del tiempo de ejecución, se pudo observar que en el caso del distribuido fue mayor que el centralizado, debido a que los tiempos también van de la mano con la cantidad de mensajes, los cuales vimos que en el distribuido eran mayores, causando que este último se demorara unos milisegundos más que su contraparte.

Conclusión

Como conclusión, podemos decir que el algoritmo centralizado es un tanto más eficiente que el algoritmo distribuido, dados los resultados anteriores en temas de cantidad de mensajes y tiempos desde la solicitud a la escritura en el registro. Sin embargo, existe un trade-off: En el algoritmo centralizado, si se cae el Name Node, el sistema directamente deja de funcionar, ya que este se encarga de manejar las propuestas, mientras que en el distribuido, si se cae un Data Node, se puede utilizar otro para continuar con la solicitud del cliente. En resumen, el trade-off es el siguiente: El algoritmo distribuido sacrifica eficiencia a cambio de seguridad del sistema, mientras que el algoritmo centralizado sacrifica seguridad a cambio de eficiencia.