# Informe Biblioteca Distribuida

Ignacio Aedo, 201773556-2
Ethiel Carmona, 201773533-3

## ¿Qué hicimos?

La idea del sistema es implementar una biblioteca que almacena la información de forma distribuida, esto es, en distintos servidores. Para esto es necesario definir algoritmos de coordinación, distribución y control entre servidores. El trabajo realizado tiene la siguiente estructura:

- **Cliente:** Se divide en Client Downloader y Client Uploader.
    - **Client Uploader:** Se encarga de entregarle las partes de un libro a un Data Node aleatorio, proporcionando el tipo de algoritmo a utilizar para la repartición de las mismas (centralizado o distribuido).
    - **Client Downloader:** recupera la IP de cada parte del libro para luego reconstruirlo. También entrega el catálogo de libros guardados en el sistema.
- **Name Node:** Encargado de recibir solicitudes de escritura en el log, el cual es un archivo de texto encargado de mantener toda la información de los libros y donde se ubican sus partes. En el caso de distribuir las partes entre Data Nodes de forma centralizada, el Name Node funciona como coordinador, lo cual quiere decir que acepta o entrega propuestas además de manejar la concurrencia de clientes que quieren escribir en el log.
- **Data Node:** Existen tres data nodes. En una primera instancia, uno de los tres nodos es elegido de forma aleatoria para recibir todas las partes de un libro y las reparte según el algoritmo que indicó el cliente: centralizado o distribuido. Los data nodes funcionan como sistemas que almacenan las partes de los libros y uno de ellos actúa de coordinador en el caso del algoritmo distribuido.

## ¿Cómo lo hicimos?

- **Cliente:** Se realizan las conexiones con todas las demás entidades (Name Node y los tres Data Nodes) y se muestra por pantalla si quiere utilizar el Client Uploader o el Client Downloader:
    - **Client Uploader:** Se pide por pantalla el nombre del libro sin extensión ".pdf" y el algoritmo a utilizar. Se utilizó como supuesto que los nombres de los libros no tengan caracteres especiales ( /, #, _ , etc) para evitar problemas al separar cadenas de caracteres. Luego procede a dividir el libro en partes de tamaño 250 kB y enviar las partes mediante un stream a un data node elegido de forma aleatoria.
    - **Client Downloader:** Le entrega las opciones de 'mostrar catálogo' y 'descargar un libro' al usuario. La primera simplemente le pide al Name Node el catálogo de libros disponibles y lo muestra por pantalla. La segunda opción le pide el nombre del libro sin extensión ".pdf" al usuario para luego pedir la ip de cada parte de ese libro al

Name Node. Este último le devuelve las IPS de cada parte en un string con cierto formato que el cliente utiliza para ir a buscar chunk por chunk al Data Node correspondiente. Una vez terminado el proceso de descarga de las partes de cada Data Node, el programa rearma el libro.

- **Name Node:** Se levanta el servidor del Name Node para que se puedan conectar el Cliente y los Data Nodes. Se definen las funciones tanto locales como gRPC encargadas de manejar el log.txt, ya sea su escritura, lectura e inicialización. En el caso de que el algoritmo sea centralizado, el Name Node actúa como coordinador. Primero recibe una propuesta de repartir los chunks entre todos. Verifica si las conexiones funcionan correctamente. Si funcionan correctamente, entonces la propuesta se acepta, si no, entrega una propuesta basándose en las conexiones que logró realizar y el data node que envía la propuesta. También maneja la concurrencia de clientes con una variable global que guarda el libro que se está trabajando y si es el primer cliente o no. Si entra un cliente mientras se trabaja con otro, se le responde que está ocupado. En este caso el segundo cliente debe seguir insistiendo hasta que termine el primer cliente.

- **Data Node:** Se levantan los Data Nodes para permitir almacenar los chunks de los libros en ellos. Pueden estar funcionando los 3 o una menor cantidad, la resolución de gestión de escritura en log y decisión sobre que Data Node almacenará un chunk se da mediante la siguiente forma:
    - Cuando el Data Node escogido aleatoriamente por el cliente recibe el stream de chunks, el estado del mismo cambia de estar libre a buscado, cuando esto sucede, el Data Node seleccionado se comunica con los demás Data Nodes solicitando permiso para poder acceder al log y comenzar a procesar los chunks, ante esto sucede:
        - Cuando los DN receptores están libres, se le concede el permiso al DN emisor y su estado cambia de buscado a tomado, lo que indica que está ocupando el log (es decir, se encuentra procesando los chunks para distribuirlos).
        - Si algún DN receptor quiere también acceder al log, se comparan los instantes de tiempo en que los estados de los DN entraron en conflicto, y aquel con un tiempo mayor será el privilegiado. Es decir, si el DN emisor pidió acceder antes, se le concederá el permiso, si el DN receptor fue quién pidió permiso antes, entonces lo dejará esperando.
        - Si algún DN receptor está tomado (ya está ocupando el log), dejará la petición en espera.
        - Dejar al DN emisor en espera significa que este esté consultando constantemente por el permiso para acceder al log, hasta que se le otorgue.
    - El proceso anterior se describe mediante la gestión con el uso del algoritmo de Ricart y Agrawala.
    - Por otra parte, una vez que se empiezan a procesar los chunks para distribuirlos, nos basamos en la explicación que los ayudantes dieron sobre “Cuando se rechaza una propuesta”, así, la propuesta siempre se podrá hacer cuando los DN se puedan conectar, y tendrá que cambiar en caso de que algún DN no esté funcionando. Este proceso se realiza por cada chunk, es decir, se chequean las conexiones con los demás DN y posterior a esto se procede a distribuir con aquellos que lograron responder a la comunicación.
    - Así, los chunks son distribuidos. Para esto nos basamos también en las ideas sugeridas por los ayudantes en el pdf de Q&A:
        - Si la cantidad de chunks es igual a la cantidad de DNs, entonces cada chunk se almacenará en un DN distinto. Para esto nuestra lógica fue guardar la primera parte en el DN que recibió el stream, y repartir los otros en órdenes

de IP de menor a mayor (Es decir, la parte 2 va al DN con IP 10.6.40.x y la parte 3 va al DN con IP 10.6.40.y, para x < y)

- Si la cantidad de chunks es menor a la cantidad de DN, entonces estos chunks se reparten de tal forma que ningún DN tenga todas las partes. Para esto nos basamos en la misma lógica antes mencionada: Primera parte se almacena en DN que recibe el stream y las demás acorde a las IP de menor a mayor.
- Si la cantidad de chunks es mayor a la cantidad de DNs, para las primeras N partes (con un total de N DNs) se sigue la lógica mencionada en el primer subíndice (1:1), para las demás partes, se selecciona un DN de forma aleatoria.

## Resultados obtenidos

Para las pruebas realizadas, se sometieron ambos algoritmos al envío de un libro, este libro de prueba se llama libro.pdf, además, dado el tamaño, se dividió en 9 partes, donde las primeras 8 de ellas fueron de 250kB y la restante con el resto (inferior a 250kB). Además, cada prueba se hizo con una cantidad distinta de Data Nodes funcionando.

| Data Nodes activos: 3 | Tiempo de ejecución [s] | Cantidad de mensajes intercambiados |
|---|---|---|
| Algoritmo Centralizado | 2.556870334 | 24 |
| Algoritmo Distribuido | 2.627620781 | 35 |

| Data Nodes activos: 2 | Tiempo de ejecución [s] | Cantidad de mensajes intercambiados |
|---|---|---|
| Algoritmo Centralizado | 2.227824388 | 25 |
| Algoritmo Distribuido | 2.250429674 | 35 |

| Data Nodes activos: 1 | Tiempo de ejecución [ms] | Cantidad de mensajes intercambiados |
|---|---|---|
| Algoritmo Centralizado | 80.788702 | 18 |
| Algoritmo Distribuido | 43.260796 | 27 |

## Análisis

Para los resultados mostrados, se puede observar que los tiempos de ejecución del algoritmo Centralizado son menores que el Distribuido para cuando se usa más de un Data Node. En el caso de un solo Data Node activo, el algoritmo Distribuido fue mucho más rápido al finalizar. Los tiempos de ejecución se midieron por algoritmo, es decir, una vez que se recibía un stream del cliente y se especificaba el algoritmo, se comenzaba a considerar el tiempo en que se implementaron los códigos. Por otra parte, se observa que la cantidad de mensajes intercambiados en el algoritmo centralizado es menor en los 3 casos, la discusión de esto se presentará en el apartado de Discusión.

## Discusión

Dado los resultados obtenidos y lo mencionado en análisis se esperaba que el algoritmo centralizado tomase menos tiempo de ejecución, ya que la gestión de propuesta se debe realizar solo con el Name Node en vez de consultar con los demás Data Nodes disponibles, ya que la distribución de los chunks entre Data Nodes es similar, todos deben balancear la carga. Por otra parte, la cantidad de mensajes que presenta el algoritmo distribuido es mayor, ya que al momento de realizar las conexiones entre DNs al repartir los chunks, independiente a si la máquina estaba arriba o no, la conexión se realizaba, por lo que se tuvieron que implementar funciones de mensajería que comprobaran que esta conexión se hiciese, aumentando considerablemente la cantidad de mensajes, los cuales son 2*<cantidad_partes_libro> mensajes extra. Esto explica porqué para un solo Data Node el tiempo disminuye considerablemente, pues la gestión se realiza directamente entre los Data Node, y como estos no funcionan (solo está activo el que posee los chunks del cliente), se decide de inmediato por cada chunk que se almacenará en el mismo.

## Conclusiones

Logramos implementar dos algoritmos de distribución de datos a diferentes servidores de forma exitosa. Gracias a esto, se logró comparar el rendimiento de ambos algoritmos utilizando métricas de tiempo y cantidad de mensajes haciendo variar la cantidad de data nodes disponibles con el objetivo de ver el comportamiento de ambos en condiciones diferentes. Al observar los resultados, es posible decir que el algoritmo distribuido presenta mayor cantidad de mensajes y mayores tiempos para una cantidad igual o mayor a los 2 data nodes activos, mientras que se muestra más eficiente en tiempo cuando hay un solo data node activo. Para mantener una cantidad de mensajes menor y un tiempo levemente menor de ejecución se recomienda el uso del algoritmo centralizado.