# Laboratorio 2
# Sistemas Distribuidos

Martín Salinas 201773557-0
Zhuo Chang 201773617-8

December 2, 2020

## 1 Configuración del Sistema

A continuación se presenta como esta distribuida la arquitectura del sistema en cada maquina.

| Arquitectura | Maquina |
|---|---|
| Cliente Uploader (CU) | dist30 |
| Cliente Downloader (CD) | dist30 |
| Data Node 1 (DN1) | dist31 |
| Data Node 2 (DN2) | dist30 |
| Data Node 3 (DN3) | dist32 |
| Name Node 1 (NN) | dist29 |

## 2 Lo que se hizo y como se hizo

### 2.1 Uploads

- **Subir un libro**: Para este caso, se inicia la comunicación entre cliente uploader y un DataNode que es aleatoriamente seleccionado gracias a la función Select(), encontrada en cada uno de los DataNodes. Cuando uno de ellos recibe dicha solicitud, enseguida recibe los chunks del libro deseado (mediante la función gRPC ReceiveChunks), guardandolos en un map creado específicamente para este momento. Luego de ello, al detectar que los tiene todos, envía un proposal al NameNode con la configuración de distribución deseada.
- **Almacenamiento de información de cada libro**: Una vez aceptada o realizada una contra-propuesta, el NameNode procede almacenar la información según se especifica en ella. Esto se hace mediante el uso de un map que registra dicha propuesta en forma de un struct.
- **Envío de chunks hacia otros Datanodes**: Se ejecutan los envíos en base al DataNode que esté enviando, es decir, es el mismo código en todos los DataNodes ordenado de distinta manera. Todos hacen uso de la función SendToDataNode (que utiliza protocol buffers) enviando los chunks e información relevante de cada libro.
- **Recepción de chunks a guardar**: Cada DataNode tiene la función StoreChunk que permite almacenar en un archivo un arreglo de bytes, que sería cada chunk de cada libro. Esta funcionalidad es llamada por la función SendToDataNode.

## 2.2 Downloads

- **Obtener ubicación de chunks**: Para realizar esta funcionalidad, el cliente downloader, hace una solicitud RequestBook a través de la función downloadBookInfo al NameNode, adquiriendo información relevante para la descarga de un libro, que es la cantidad de chunks que se almacenan en cada DataNode.

  Dicha función (RequestBook) se encuentra implementada en el NameNode y su trabajo es enviar el proposal aceptado anteriormente para el libro especificado, encontrado en el map librería.

- **Descargar un libro**: Una vez recibida la información, el cliente descargas se conecta a cada DataNode llamando a la función connectAndDownload, cuyo trabajo es hacer una iteración para descargar cada chunk encontrado en cada DataNode (uno a la vez), llamando a la función downloadChunks que se encarga de solicitar un chunk de un libro en la IP correspondiente mediante una llamada gRPC usando protocol buffers.

- **Reconstrucción del libro**: Una vez recibidas todas las partes del libro, el cliente downloader hace un llamado a la función JoinBook, la cual se encarga de juntar el libro implementando el algoritmo sugerido en el enunciado de la tarea.

## 2.3 Algoritmo Exclusión Mutua Centralizada

Para este algoritmo, se tiene en consideración la configuración del sistema dónde el NameNode es el encargado de aceptar propuestas de repartición de chunks por cada libro procesado. El algoritmo se realiza en función de las siguientes situaciones.

- **Generación de propuestas**: Una propuesta define cómo se van a repartir los chunks de un libro entre los DataNodes. La estructura de esta es la siguiente: "titulo**c1**c2**c3**partes"
  - titulo: Nombre del libro.
  - c1: Cantidad de chunk a repartir al DN1.
  - c2: Cantidad de chunk a repartir al DN2.
  - c3: Cantidad de chunk a repartir al DN3.
  - partes: Partes totales del libro.

  Al momento de realizar una propuesta el DataNode que recibe los chunks hace la repartición aplicando un modulo 3 al total de chunks y si sobran estos son asignados al DN1. En el caso de que un DataNode este caído, una nueva propuesta es hecha por el NameNode, donde tomara en cuenta el fallo presente. La propuesta de como distribuir los nodos seguirá las siguientes reglas:
  - Si el DN1 esta caído todos sus chunks se pasan al DN2.
  - Si el DN2 esta caído todos sus chunks se pasan al DN3.
  - Si el DN3 esta caído todos sus chunks se pasan al DN1.
  - Se consideró que siempre hay al menos un DataNode activo.

## 2.4 Algoritmo Exclusión Mutua Distribuida

Para este algoritmo, se tiene en consideración la configuración del sistema dónde los DataNodes llegan a un consenso para aceptar una propuesta de repartición de chunks por cada libro procesado. El algoritmo se realiza en función de las situaciones anteriores y unas pequeñas diferencias.

- **Generación de propuestas**: La propuesta de este algoritmo tiene la misma estructura del algoritmo anterior, lo distinto sería que se envía el proposal a los demás DataNodes para ser aprobada (dataNodeProposal), luego se manda la misma propuesta en forma de WriteRequest al NameNode para registrarlo en el Log.

- **WriteRequest**: Antes de hacer el writerequest, se llama a CallRichardAgrawalla para cada nodo y la propuesta de escritura espera hasta que los demás nodos respondan usando el algoritmo de Ricart - Agrawala mediante llamadas gRPC.

# 3 Resultados, Análisis y Discusión

Los resultados se obtuvieron a partir de subidas de libros a los DataNodes y se vieron cuántos mensajes se intercambiaban, además de ver el tiempo que se tarda en agregar las partes del libro y su ubicación el el log.

## 3.1 Resultados

| Test 1 (5 Partes) | Mensajes | Tiempo [$\mu s$] |
|---|---|---|
| Centralizado | 10 | 221,556 |
| Distribuido | 12 | 219,021 |

| Test 2 (5 Partes) | Mensajes | Tiempo [$\mu s$] |
|---|---|---|
| Centralizado | 10 | 632,314 |
| Distribuido | 12 | 159,331 |

| Test 3 (1 Partes) | Mensajes | Tiempo [$\mu s$] |
|---|---|---|
| Centralizado | 3 | 163,253 |
| Distribuido | 5 | 155,769 |

| Test 4 (4 Partes) | Mensajes | Tiempo [$\mu s$] |
|---|---|---|
| Centralizado | 8 | 181,753 |
| Distribuido | 10 | 159,091 |

## 3.2 Análisis y Discusión

Con los resultados obtenidos, se puede apreciar que el algoritmo de exclusión mutua distribuida envía una mayor cantidad de mensajes que el otro algoritmo. De esto se puede notar que el algoritmo distribuido siempre manda dos mensajes más que el algoritmo centralizado, esto son los avisos del DataNode para ver si los otros DataNodes están activos. Además, se puede apreciar que es ligeramente más rápido al escribir el log.

Estos resultados pueden deberse a la cantidad de procesos que están en ejecución en el NameNode cada vez, pues existe el tiempo de 632,314[$\mu s$], el cual debe haber tenido relación directa con esto. Pues, en un caso similar (Test 1), la escritura en el caso Centralizado no fue muy diferente del caso Distribuido.

Es posible realizar optimizaciones dentro del código, como por ejemplo el paso de mensajes, pues al implementar alguna forma de compresión de texto podría hacer que los mensajes viajaran más rápido.

# 4 Conclusión

Al realizar este trabajo se aprendió a utilizar el algoritmo de Ricart-Agrawala para resolver las posibles colisiones que se generan al intentar contactar al NameNode en el algoritmo distribuido. Por otro lado, fue posible expandir los conocimientos de golang al implementar el uso de la separación de archivos en chunks. También se pudo lograr una mejor implementación de las estructuras de datos que el lenguaje dispone y hacer uso de buenas prácticas tales como documentar el código, chequeo de errores, entre otros.