.UBAfiuba
FACULTAD DE INGENIERÍA

Universidad de Buenos Aires

TP Tolerancia a Fallos

Movies Analysis

Ingeniería en Informática

Junio, 2025

Grupo 16:

Chávez Cabanillas, José Eduardo	jchavez@fi.uba.ar	96467
Davico, Mauricio	mdavico@fi.uba.ar	106171
Lardiez, Mateo	mlardiez@fi.uba.ar	107992

Índice

Alcance

Requerimientos funcionales:

Se solicita un sistema distribuido que analice la información de películas y los ratings de sus espectadores en plataformas como iMDb.
Los ratings son un valor numérico del 1 al 5. Las películas tienen información como género, fecha de estreno, países involucrados en la producción, idioma, presupuesto e ingreso.

Se debe obtener:

1. Películas y sus géneros de los años 00' con producción Argentina y Española.
2. Top 5 de países que más dinero han invertido en producciones sin colaborar con otros países.
3. Película de producción Argentina estrenada a partir del 2000, con mayor y con menor promedio de rating.
4. Top 10 de actores con mayor participación en películas de producción Argentina con fecha de estreno posterior al 2000
5. Average de la tasa ingreso/presupuesto de películas con overview de sentimiento positivo vs. sentimiento negativo, para películas de habla inglesa con producción americana, estrenadas a partir del año 2000

Requerimientos No funcionales:

- El sistema debe estar optimizado para entornos multicomputadoras
- Se debe soportar el incremento de los elementos de cómputo para escalar los volúmenes de información a procesar
- Se requiere del desarrollo de un Middleware para abstraer la comunicación basada en grupos.
- Se debe soportar una única ejecución del procesamiento y proveer graceful quit frente a señales SIGTERM.

Datos Necesarios:

Para construir la simulacion: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset
Valores como resultados patron:
https://www.kaggle.com/code/gabrielrobles/fiuba-distribuidos-1-the-movies

Arquitectura

Vista Física

Se detallan en esta vista todos los componentes físicos del sistema así como las conexiones físicas entre esos componentes que conforman la solución.

Diagrama de Robustez

El diagrama representa la comunicación e interacción de mensajes entre nodos. Los boundary objects que se definen son la Terminal y el Client Process. El cliente se conecta con el Gateway y envía un primer mensaje en el que detalla que Query / Querys quiere resolver, y luego envía todos los datos necesarios (3 archivos csv) de manera secuencial al Gateway. En este caso el Gateway comienza a distribuir los datos dependiendo de qué Query quiere resolver el usuario. Los datos son preprocesados por los nodos cleaner, que limpian cada batch de los documentos movies_metadata.csv, raitings_data.csv y credits_data.csv, manteniendo únicamente la información necesaria para resolver las Querys. Estos comunican el resultado del procesamiento a los demás nodos.

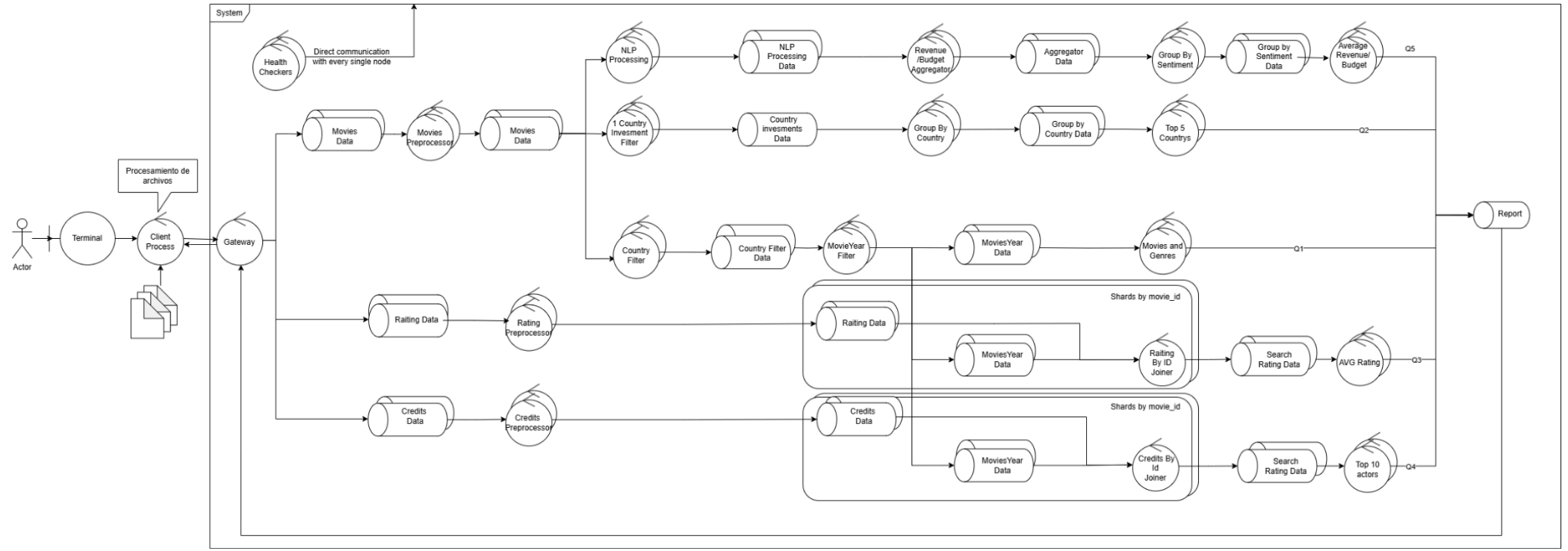
Siguiendo el flujo por el camino superior y resolviendo la Query 5, el nodo NLP Processing recibe data del archivo movies_data, pero únicamente las columnas que se van a utilizar en su procedimiento, para reducir el volumen de data, las cuales son “id”, “title”, “overview”, “budget”, “revenue”. Estas últimas 2 se van a utilizar en el siguiente nodo del flujo. El nodo NLP Processing procesa con lenguaje natural las reseñas de las películas y envía a través de la cola toda esa información al siguiente nodo. El siguiente nodo Revenue/Budget hace el cálculo de revenue/budget para cada película y envía toda la información al siguiente nodo GroupBySentiment, que agrupa por cada batch los valores de sentimiento POSITIVO y NEGATIVO, calculando la suma de Revenue/Budget y la cantidad de películas en el batch para cada sentimiento. Por último, GroupBySentiment envía la data agrupada al nodo sinker de esta Query utilizando la lógica de enrutar al sinker dependiendo del id del cliente y de la cantidad de sinkers. **En todo momento se envían sólo las columnas que se van a utilizar,** no hay columnas extra, y por ende se reduce mucho la información enviada entre los nodos. El nodo Sinker de la Query 5 va almacenando y guardando en disco los resultados hasta que llega el EOF. Una vez que llega el EOF, ya tiene todos los datos necesarios (“sentiment”, “suma de revenue/budget”,”cantidad de películas”) y calcula el average budget de todas las películas, finalmente envía al Gateway la respuesta de la Query 5. El Gateway envía la respuesta al Cliente. Cabe aclarar que todos los nodos de este procedimiento son escalables. El nodo sinker necesita recibir toda la información del cliente para poder realizar el cálculo final y comunicar el resultado.

Para el flujo de la Query 2, también se espera recibir sólo la información necesaria para resolverla y siguiendo con el razonamiento del flujo anterior, el primer nodo FilterByYear filtra todas las películas que tengan un único país de producción. El siguiente nodo GroupByCountry agrupa por cada país sumando dentro del batch la inversión. Al final envía la información del resultado a un nodo sinker que se le asigna a un cliente usando la misma lógica de ruteo hacia un sinker dependiendo del client_id descrita en la query anterior. El nodo sinker Top 5 Countries procesa todos los clientes que se le asignen, y guarda la

información que recibe para cada cliente en un archivo y espera recibir el EOF. Una vez que llega el EOF realiza la operación correspondiente y encola el resultado en la queue de reportes, de la que luego el Gateway va a consumir para finalmente enviar los resultados al cliente.

El nodo de tipo filter, Movie Year Filter filtra los datos por el año correspondiente y se utiliza para resolver las 3 Queries restantes. Luego un siguiente nodo de tipo filter, Country Filter, toma los datos y los filtra por el país correspondiente.
En este punto la Query 1 ya está completada, y el nodo de tipo sink Movies and Genres recibe los datos y los encola en la queue de reportes, de la que luego el Gateway va a consumir para finalmente enviar los resultados al cliente.

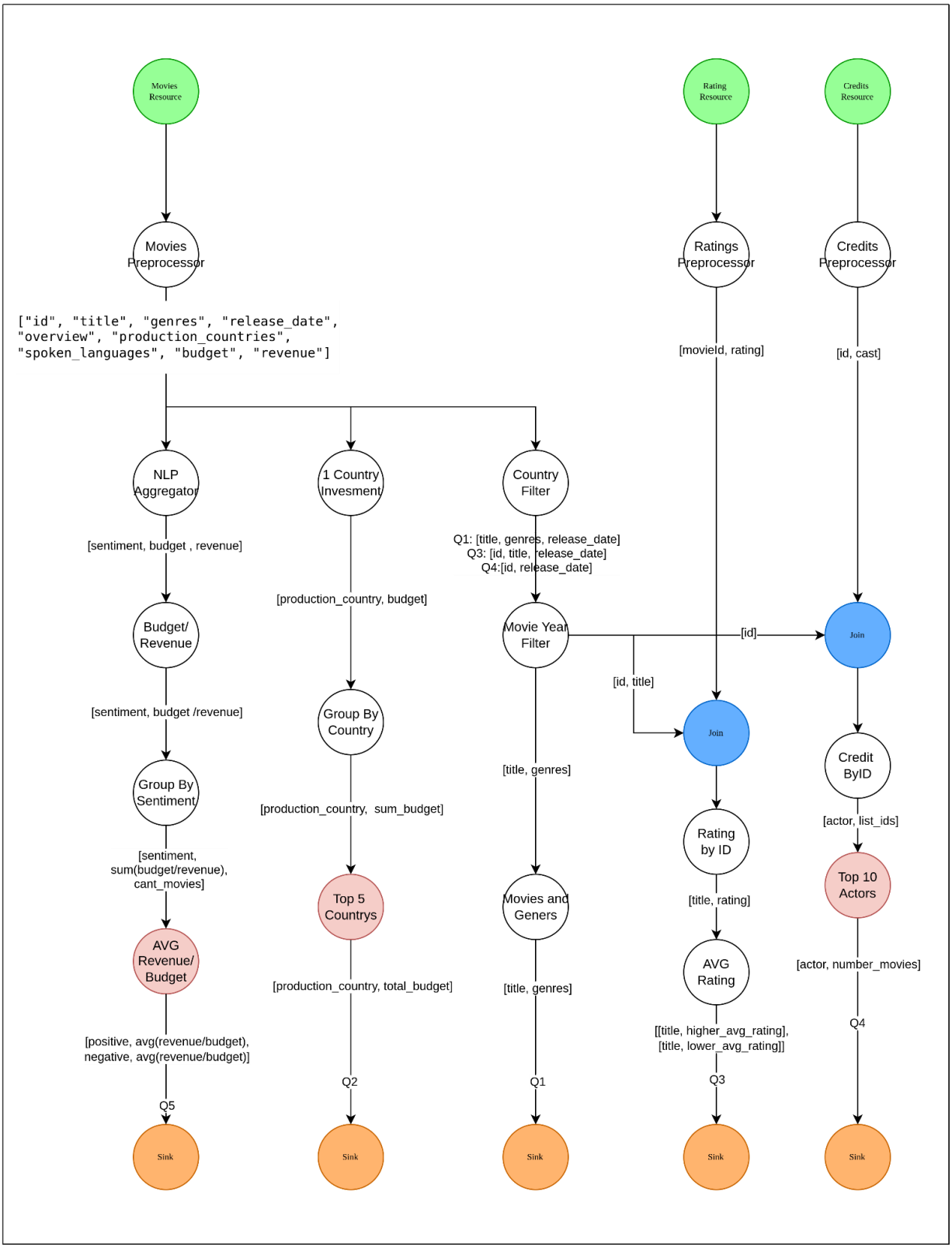
Para las Querys 3 y 4 se tienen sus respectivos nodos de tipo joiner, que conectan los datos recibidos por los Preprocessors de Ratings y Credits con las movies filtradas según cada Query. Luego se tienen nodos de tipo sink, que nuevamente esperan hasta que llega el EOF para poder realizar respectivamente la operación de average rating o top 10 de actores. Una vez que llega el EOF realizan la operación correspondiente y encolan el resultado en la queue de reportes, de la que luego el Gateway va a consumir para finalmente enviar los resultados al cliente.


System
Health Checkers
Direct communication with every single node
Actor
Terminal
Procesamiento de archivos
Client Process
Gateway
Movies Data
Movies Preprocessor
Movies Data
Raiting Data
Rating Preprocessor
Credits Data
Credits Preprocessor
NLP Processing
NLP Processing Data
Revenue /Budget Aggregator
Aggregator Data
Group By Sentiment
Group by Sentiment Data
Average Revenue/ Budget
Q5
1 Country Invesment Filter
Country invesments Data
Group By Country
Group by Country Data
Top 5 Countrys
Q2
Country Filter
Country Filter Data
MovieYear Filter
MoviesYear Data
Movies and Genres
Q1
Shards by movie_id
Raiting Data
MoviesYear Data
Raiting By ID Joiner
Search Rating Data
AVG Rating
Q3
Shards by movie_id
Credits Data
MoviesYear Data
Credits By Id Joiner
Search Rating Data
Top 10 actors
Q4
Report

Vista Lógica

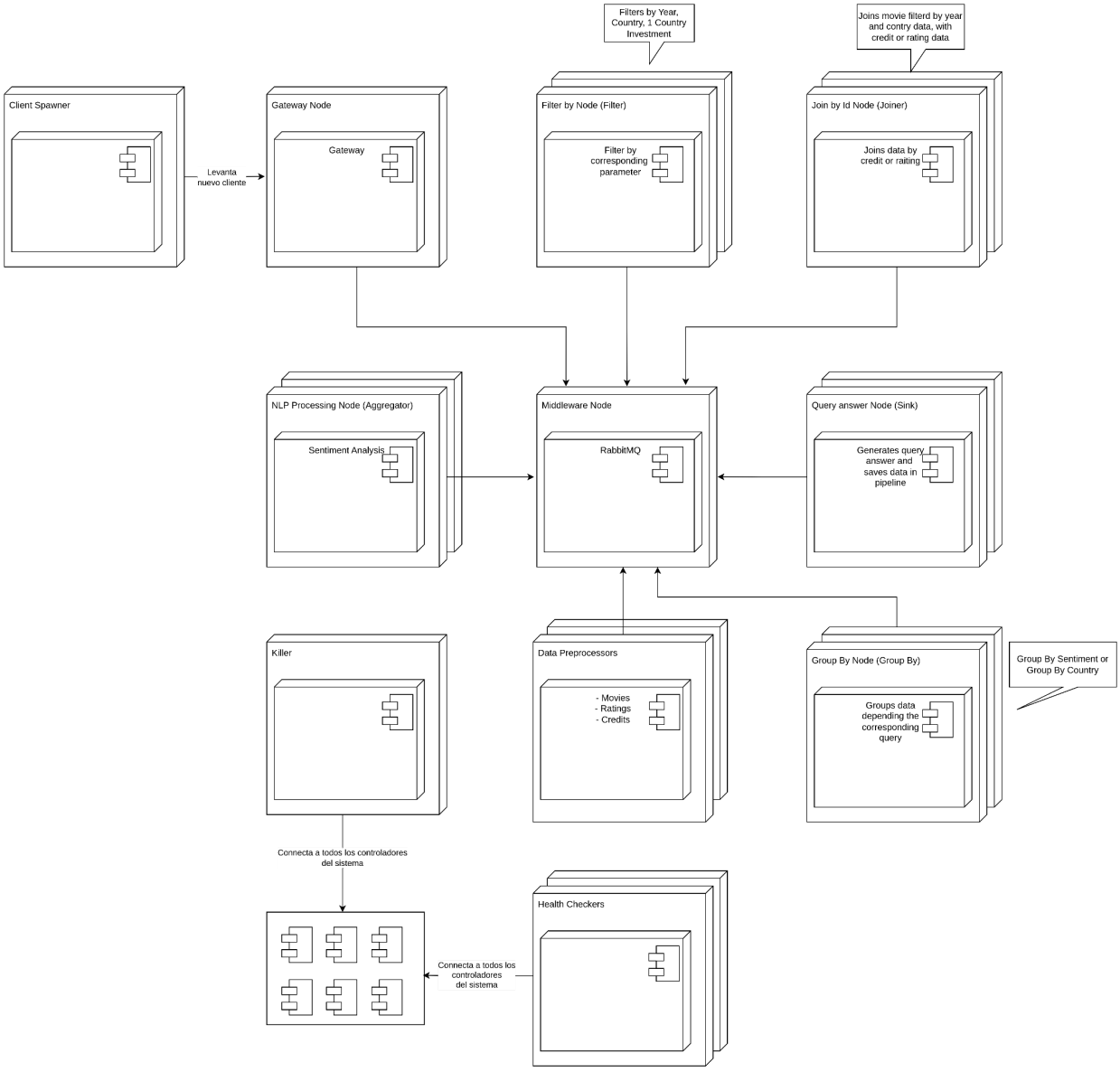
Esta sección explica la estructura y funcionalidad del sistema. A continuación se muestra un directed acyclic graph que evidencia el flujo de datos:

DAG



En este diagrama se pueden ver todos los nodos de nuestro sistema y su interacción. Se ven los distintos nodos Filter, Join, Agregator, Sink y Group By. El flujo de datos comienza en las fuentes, las cuales son los distintos datasets, y atraviesa distintos filtros que permiten ir acotando y refinando la información hasta llegar al resultado final de cada consulta. Las listas entre la conexión entre 2 nodos define las columnas que se van a transferir de un nodo a otro, ilustrando cómo se reduce el tamaño de transferencia de información. En ciertos casos, es necesario combinar datos provenientes de distintas fuentes, y ahí es donde entran en juego los nodos Join.
Por otro lado tenemos nodos que son de stop (nodos de color rojo) que indican que el nodo tiene que esperar a toda la información antes de poder devolver la información procesada.

Diagrama de Despliegue



El diagrama representa cómo se distribuyen los distintos procesos en múltiples computadoras. La comunicación entre nodos se realiza mediante el Middleware, con la excepción del vínculo entre el cliente y el gateway, que se gestiona de forma directa utilizando TCP.

Para facilitar la visualización, se agrupan varios procesos en nodos únicos. Por ejemplo, el nodo *Filter* abarca todas las tareas relacionadas con el filtrado de películas por año, por País o por Inversión de 1 País. También para el nodo Joiner se agrupan distintos nodos que realizan tareas de unificación de datos, como por ejemplo la unificación de data filtrada por País y por año, con la de los ratings, entre otros. De la misma idea se agrupan los nodos Sink en uno.

Los nodos Data Processors se agrupan entre sí, ya que realizan tareas de reducción de columnas dependiendo los archivos movies, credits y ratings respectivamente.

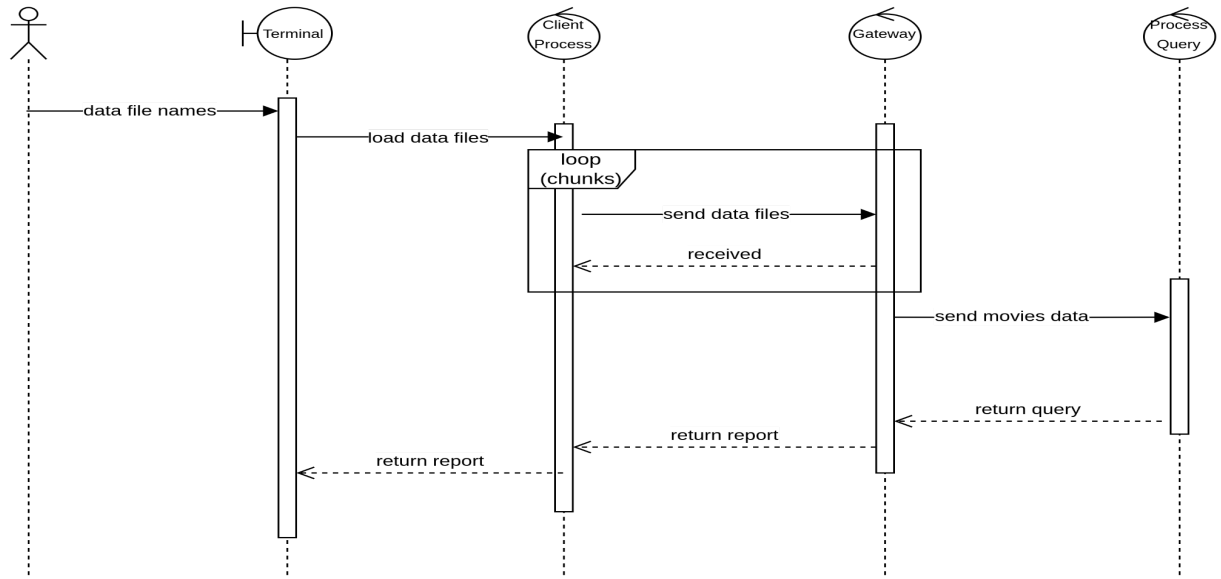
Se agregan los nuevos nodos Health Checkers que, si bien no se conectan con RabbitMQ, se conectan con todos los demás nodos a través de TCP para revisar constantemente si siguen 'vivos'. También se agrega el nodo Killer al sistema, el cual es el responsable de 'matar' aleatoriamente nodos del sistema. Finalmente se agrega el Client Spawner, el cual permite levantar un nuevo cliente mientras el flujo ya está en funcionamiento. Esto se hace a través de comandos en terminal.

Vista de Procesos

Esta vista trata los aspectos dinámicos del sistema: los procesos de sistema y cómo se comunican. Se enfoca en el comportamiento del sistema en ejecución. La vista considera aspectos de concurrencia, distribución, escalabilidad, etc.
Se incluyen dos secciones de diagramas: secuencia y actividad. En ambos casos, se decidió diagramar sólo los casos relevantes de explicación.

Diagrama de Secuencia

El diagrama de secuencia busca mostrar el flujo de mensajes en el sistema, entre los distintos nodos. Para esto se realizaron 2 gráficos. El primero, más general, muestra el flujo de mensajes para resolver cualquier Query. El cliente hace un pasamanos de información que recibe de la terminal y la envía al Gateway, a través de chunks. Luego el gateway le envía la información (procesada) a los distintos nodos que resuelven las queries.



El segundo gráfico muestra el flujo de mensajes de una Query específica, en este caso la Query 3. El Gateway persiste en el tiempo hasta que se realizan todas las operaciones. Finalmente el Gateway recibe la respuesta de la Query y se la vuelve a enviar al cliente para terminar su ciclo de vida.

Query 3

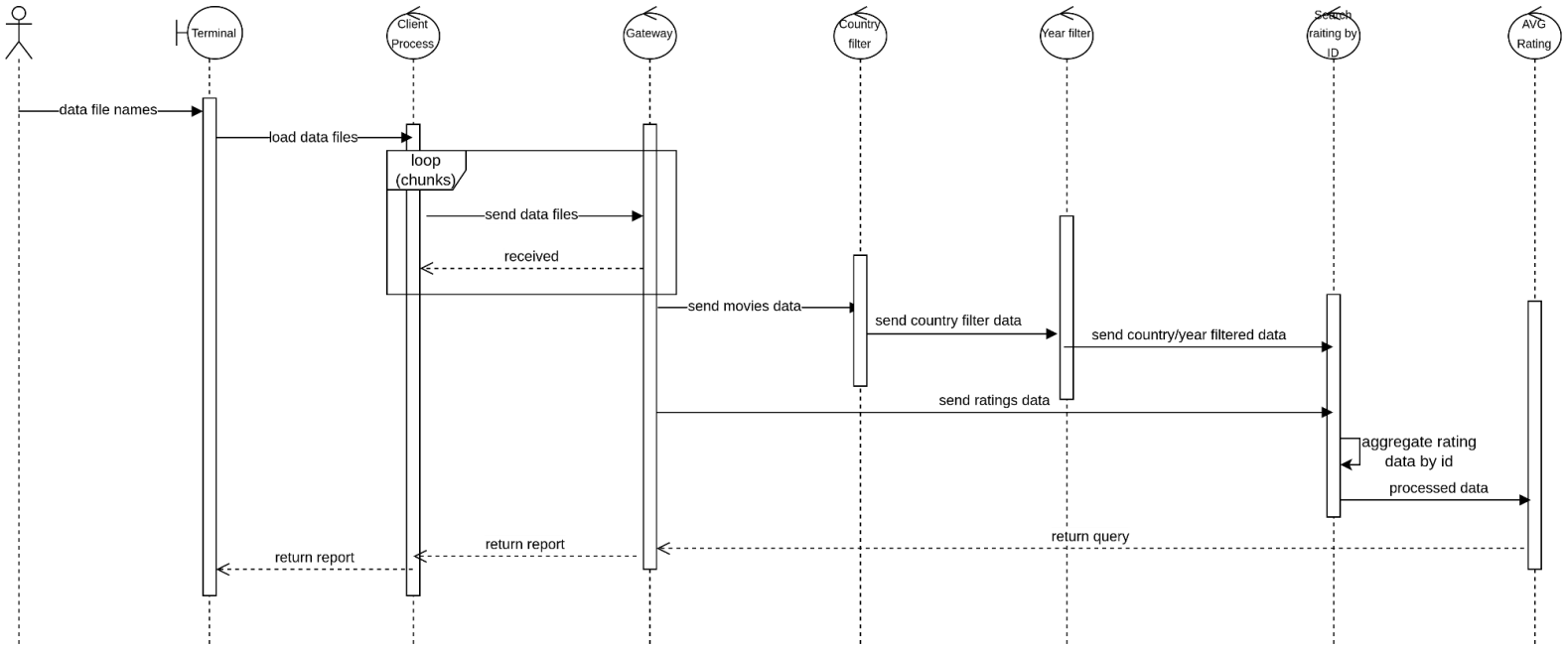
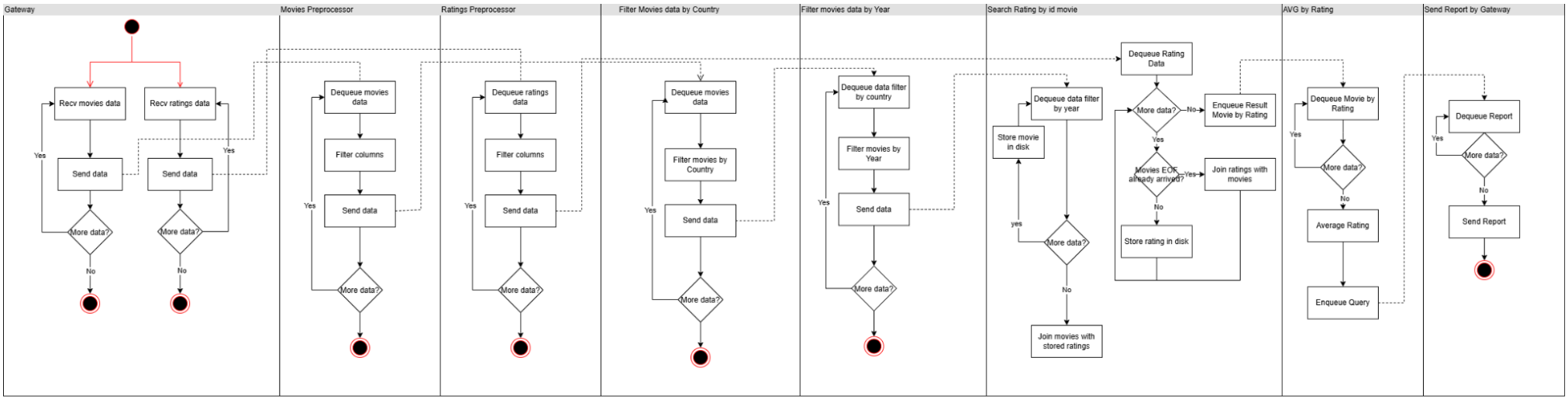
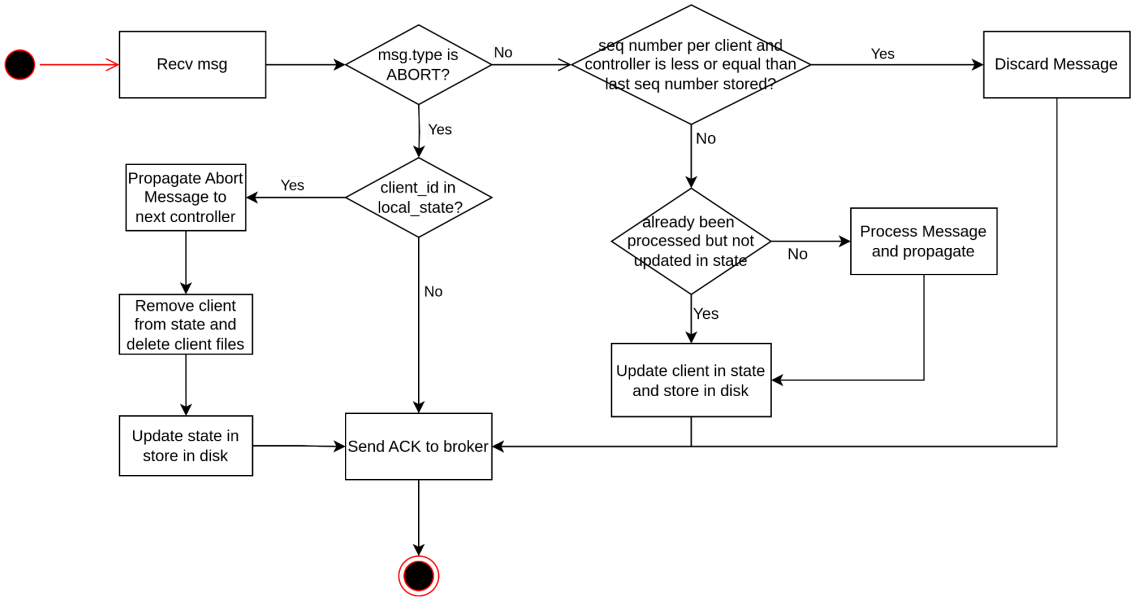


Diagrama de Actividad

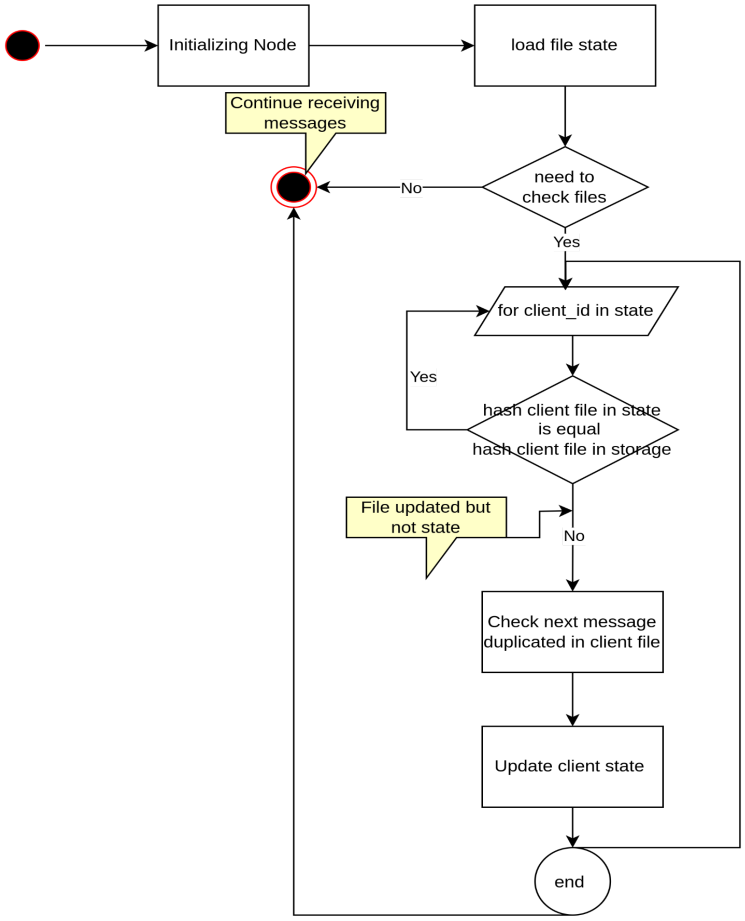
El caso representado es todo el proceso que se realiza para la Query 3, desde que se reciben los archivos en el gateway hasta la devolución del resultado de la query. Donde se ve cómo se procesa cada batch de información en cada uno de los nodos que están involucrados para la resolución de la query.


Gateway
Recv movies data
Recv ratings data
Send data
Send data
More data?
More data?
Yes
Yes
No
No
Movies Preprocessor
Dequeue movies data
Filter columns
Send data
More data?
Yes
Ratings Preprocessor
Dequeue ratings data
Filter columns
Send data
More data?
Yes
Filter Movies data by Country
Dequeue movies data
Filter movies by Country
Send data
More data?
Yes
Filter movies data by Year
Dequeue data filter by country
Filter movies by Year
Send data
More data?
Yes
Search Rating by id movie
Dequeue Rating Data
Dequeue data filter by year
Store movie in disk
More data?
yes
No
Join movies with stored ratings
More data?
No
Yes
Enqueue Result Movie by Rating
Movies EOF already arrived?
Yes
Join ratings with movies
No
Store rating in disk
AVG by Rating
Dequeue Movie by Rating
More data?
Yes
No
Average Rating
Enqueue Query
Send Report by Gateway
Dequeue Report
More data?
Yes
No
Send Report

Se generan también 2 nuevos diagramas de actividades. Uno es el flujo, recepción y propagación de mensajes para nuestro nodo más complejo, el Joiner, específicamente enfocado en visualizar cómo realiza la detección de duplicados:



El siguiente diagrama muestra el funcionamiento inicial de un nodo joiner después de una caída



En el gráfico se puede observar como al levantarse, lo primero que hace es cargar su estado, analizando por cada cliente si se cargó información en el archivo pero no en el que contiene estado del cliente. Esto lo hace comparando los hashes del archivo que recibió, con el hash guardado en el estado. Luego de restaurar el estado y archivos de todos los clientes, se prepara para recibir mensajes nuevamente.

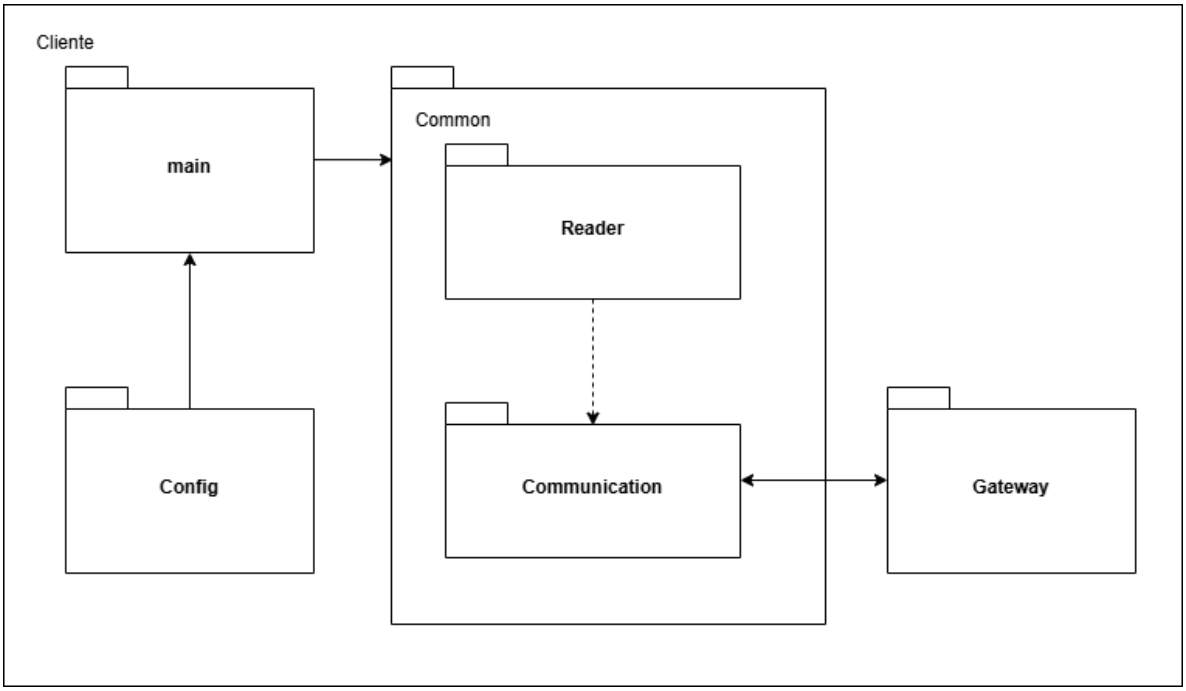
Vista de Desarrollo

La vista de desarrollo se centra en cómo está estructurado el sistema desde la perspectiva del programador y cómo se gestiona el software. Esta vista muestra cómo se divide el sistema en módulos, componentes y subsistemas. A nivel de implementación, se refleja cómo los desarrolladores organizan, construyen y mantienen el código, destacando aspectos como la estructura de carpetas, paquetes, bibliotecas y la relación entre los diferentes módulos del software.

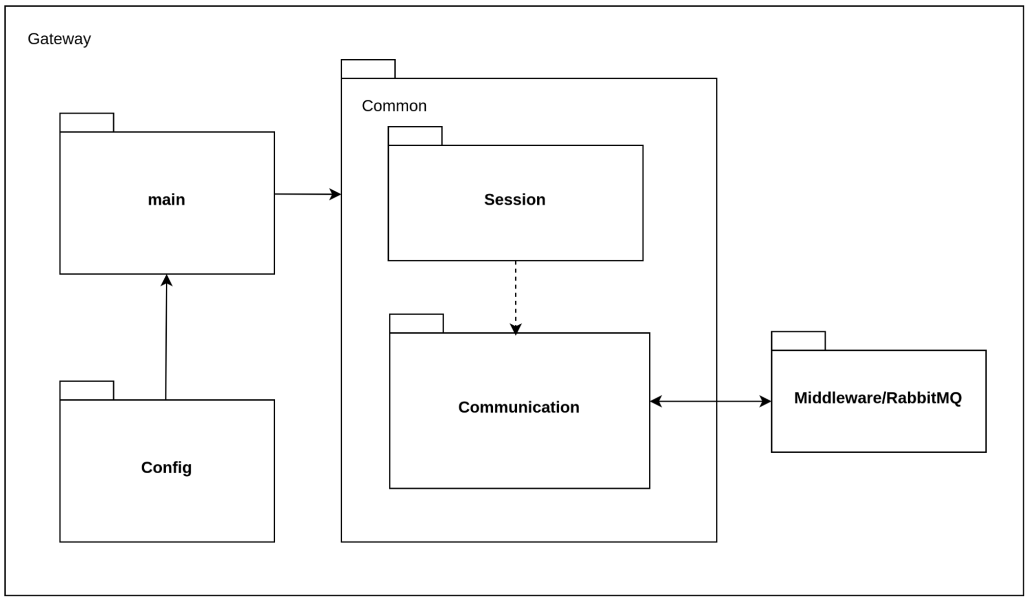
Diagrama de Paquetes

El diagrama de paquetes se encuentra dividido en distintos elementos lógicos en función de las responsabilidades de cada nodo.

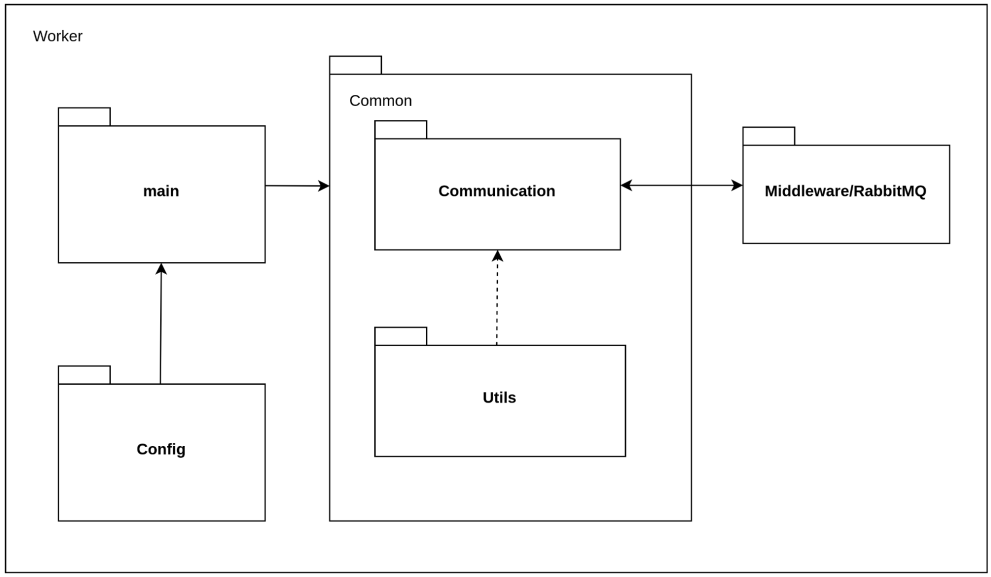
Cliente:
Es el responsable de leer los archivos a través del paquete reader y enviarlos al gateway a través de una conexión TCP haciendo uso del paquete Communication. A su vez, estos paquetes se encontrarán dentro del paquete common (Python).



Gateway:
El gateway del sistema posee dos paquetes. Por un lado el paquete Communication, se encarga de recibir nuevas conexiones y gestionarlas, por el otro, el paquete Session se encarga de la comunicación entre el gateway y el cliente, y entre el gateway y el middleware.



Worker:
Cada uno de los workers y filtros contarán con la misma estructura de paquetes la cual cuenta con un paquete de Utils para aquellas operaciones necesarias para el procesamiento del worker pero que pueden ser abstraídas. A su vez, el paquete Communication hará de interfaz entre el worker y el middleware.



Actualizaciones Tp Escalabilidad

Se actualizaron ciertas partes del diseño, todas descritas en sus respectivas partes del informe.

- Se modifica el orden entre el nodo Year filter y Country Filter. Se utiliza primero el nodo de Country Filter ya que va a generar una mayor reducción de datos para transferir a los demás nodos, lo que lo va a hacer más rápido. Se modifican todos los gráficos
- Se generan 3 nuevos nodos Data Processors. Estos filtran las columnas correspondientes de cada archivo movies, credits y ratings respectivamente. Transfieren la información a los demás nodos que resuelven las queries.
- Se optó usar RabbitMQ como middleware, haciendo eficiente y simple la transferencia de información entre distintos nodos, a través de las exchanges y queues que ofrece RabbitMQ

Protocolo

Para la comunicación entre el Cliente y el Gateway se desarrolló un protocolo de comunicación, el cual usa un DTO (Data Transfer Object), como medio para enviar la información a través de ambos.
Por otro lado la comunicación entre el Gateway y los controladores (Preprocessors, joiners, filters, aggregators y sinkers) se usa también un DTO.

A continuación se detallan estos DTO's:

- MessageProtocol: se usa para la comunicación entre Cliente y Gateway, consta de 3 atributos:
    - IdClient: Identificador del cliente que envía el mensaje
    - TypeMessage: Tipo de mensaje que se le envía la Gateway, se detalla una lista de los tipos de mensajes que se usan para la comunicación.
        - TYPE_QUERY: Indica al gateway que el mensaje contiene la query a realizar.
        - TYPE_FINISH_COMMUNICATION: Indica al gateway que la comunicación con el cliente terminó.
        - BATCH_MOVIES: Indica que el mensaje contiene el batch del archivo movies
        - BATCH_RATINGS: Indica que el mensaje contiene el batch del archivo ratings
        - BATCH_CREDITS: Indica que el mensaje contiene el batch del archivo credits
        - EOF_MOVIES: Indica el final de envio de batches del archivo movies
        - EOF_RATINGS: Indica el final de envio de batches del archivo ratings
        - EOF_CREDITS: Indica el final de envio de batches del archivo credits
        - RESULT_QUERY_<Number>: Indica que el mensaje contiene el resultado de la query <number>
        - EOF_QUERY_<Number>: Indica que ya se terminó de enviar el resultado de la query <number>
        - FINISH_SEND_FILES: Indica que se terminó de enviar todos los archivos al gateway
    - Payload: Contiene el mensaje a enviar

- MiddlewareMessage: se usa para la comunicación entre el Gateway y los controladores, consta de 4 atributos:
    - query_number: Indica el número de la query a realizar por el controlador.
    - client_id: Indica el cliente que realiza la query, también ayuda a identificar a quien se le debe responder el mensaje.
    - type: Indica el tipo de mensaje que se le envía al controlador. Estos pueden ser:
        - MOVIES_BATCH: Indica que el mensaje contiene el batch de movies a procesar
        - RATINGS_BATCH: Indica que el mensaje contiene el batch de ratings a procesar
        - CREDITS_BATCH: Indica que el mensaje contiene el batch de credits a procesar
        - EOF_MOVIES: Indica que se terminaron de procesar los batch de movies
        - EOF_RATINGS: Indica que se terminaron de procesar los batch de ratings
        - EOF_CREDITS: Indica que se terminaron de procesar los batch de credits
        - RESULT_Q{number}: Indica que el mensaje contiene el resultado de la query {number}
        - EOF_JOINER: Indica que los joiners finalizaron de enviar todos los mensajes
        - EOF_RESULT_Q{number}: Indica que el mensaje contiene el fin de la query {number}
    - payload: Contiene el mensaje a enviar.

Para el caso de la comunicación entre el Cliente y el Gateway, el protocolo calcula el tamaño de los bytes a enviar y envía esa cantidad al Gateway al antes de enviar el mensaje del DTO, el mensaje que indica el tamaño de dicho DTO es de 4 bytes.

Por otro lado los mensajes entre el Gateway y los Controladores se hace a través de RabbitMq donde el mensaje se convierte a String y se envía, dejando la forma de envío a los channels de RabbitMq

Actualizaciones Tp Multiples Clientes

Se actualizaron ciertas partes del diseño, todas descritas en sus respectivas partes del informe.

- Se agrega un nuevo nodo Group By Sentiment para la query 5, para agrupar cada batch por sentimiento NEGATIVO o POSITIVO, enviando al sinker el sentimiento, con la suma del cálculo budget/revenue, y la cantidad de películas que participaron para ese cálculo.
- Se agrega un nuevo nodo Group By Country para la query 2, para agrupar toda inversión de los países por cada batch. Luego se le envía al sinker el país con la suma total de inversión, dentro de ese batch.
- Se agrega el concepto de sharding y su funcionalidad, para agrupar las películas con sus respectivos ratings o créditos, en las queries 3 y 4. Para que una película sea asignada siempre en el mismo sharding, se utiliza módulo del id de la película, que va a ser el mismo en ambos archivos, y la cantidad de nodos Joiners que haya (id_movie % number_workers). De esta manera la data de una película específica, siempre va a ir al mismo joiner.

Actualizaciones TP Tolerancia a Fallos

Correcciones pendientes:

Primero se arreglaron correcciones previas del tp anterior.

- Para los nodos joiners, se guardaba toda la información de las movies y toda la información de los ratings o credits. Esto era un grave error ya que no solo se utilizaba espacio en memoria innecesariamente, sino que también el nodo debía esperar a que lleguen todos los batches de movies y todos los batches de ratings o créditos para poder empezar a procesar la información.

  Solución: Como se necesita obligatoriamente todos los datos de movies para empezar a procesar, se guardan todos los batches tanto de movies como de ratings/credits HASTA QUE llegue el End Of File de el archivo movies. Una vez que se obtienen todos los datos de movies, se puede empezar a procesar y joinear con los datos de ratings/credits, sin la necesidad de guardar información en memoria. También se ordenó de tal manera que se envía primero el archivo de movies, por lo que es muy probable que llegue el EOF antes de que llegue algún batch de ratings o de credits ya que el cliente envía los archivos de forma secuencial. En caso de que llegaran batches de credits/ratings antes del EOF de movies, ni bien llega el EOF de movies se procede a joinear primero contra esos batches de credits/ratings que habían llegado, se limpian esos archivos, y luego para los próximos batches se procede a joinear directamente sin persistir en disco los csv.

- Se arregla la asignación de ID del cliente. Antes el propio cliente enviaba su id al gateway. Ahora, el gateway al recibir una conexión, genera un ID aleatorio y único para el Cliente.

- Se mejora el guardado de archivos. Antes los archivos que usaban los nodos estaban en su contenedor, lo cual iba a ser un problema para el tp de tolerancia a fallos. Ahora se guardan los archivos por fuera del contenedor a través de volúmenes.

Cambio en la arquitectura del sistema:

En iteraciones anteriores, cada worker del sistema (excluyendo los joiners y los sinkers) consumía de una única queue compartida, por lo que se daban escenarios en los que múltiples nodos encolaban datos en una misma queue y múltiples nodos consumían datos de esa misma queue compitiendo entre sí para desencolar mensajes. Este tipo de arquitectura implicaba tener que implementar sincronización de los diversos nodos del sistema ya sea global o por nivel para tener un control de qué mensajes ya fueron procesados y cuáles no, para de este modo poder garantizar la inexistencia tanto de datos duplicados como de datos que se pierden en escenarios de tolerancia a fallos.

En su lugar se optó por una arquitectura en la que cada nodo consume de su propia queue y puede pushear en múltiples queues, seleccionando la queue a la que se va a pushear haciendo round robin usando el

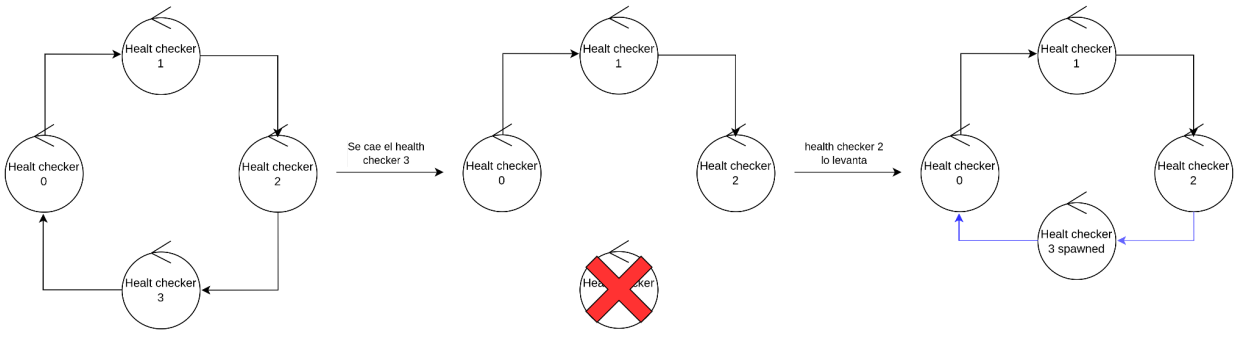
número de secuencia propio del controller para cada cliente y cada controlador del que se reciben mensajes, y la cantidad de workers que hay en el siguiente nivel (para esto se hace el cálculo de seq_num % cantidad de workers y de este modo se obtiene el id de la queue a la que vamos a pushear).

La comunicación de los procesos con el middleware RabbitMQ permanece en general tal como en la iteración anterior; el envío de ACKs hacia el mismo se realiza de forma manual y al final de todas las operaciones tras recibir un mensaje, para asegurar que se pudo enviar al siguiente nodo y guardar el estado actual del controlador antes de confirmar que se puede remover un mensaje de forma segura de una cola. Un detalle que fue agregado es la configuración de parámetros adicionales como confirm_delivery y el uso en conjunto de los flags prefetch_count y mandatory especificados en la documentación de RabbitMQ para tener garantías sobre cantidad de mensajes permitidos a consumir por cada proceso previo a realizar su correspondiente ACK (asegurando así que si un controlador no logra realizarlo antes de caerse, el mensaje permanezca encolado para consumo futuro); y sobre la publicación de mensajes a colas sin drop silencioso de mensajes (comportamiento por defecto del broker en casos borde de falla de envío). De este modo tenemos mayor control sobre cada mensaje, ya que una vez que es encolado en una queue, tenemos la garantía de que va a persistir en esa queue hasta que el nodo worker lo consuma y haga el ack manual.

Implementaciones:

1. Nodos Health checkers

   Encargados de solicitar periódicamente el estado de cada uno de los demás controladores del sistema. Para estos últimos se realiza adaptación para recibir dichas solicitudes por medio de herencia de la clase *ResilientNode*. Todos los nodos van a heredar de esta clase, la cual va a ser la responsable de manejar los estados y conexiones con los health checkers.
   Los health checkers son procesos stateless, por lo tanto son fácilmente replicables y no realizan persistencia. Cada controlador del sistema distribuido es monitoreado por un sólo health checker, con esto se evita race-conditions al intentar levantar algún controlador caído. Esto también aplica para los propios Health checkers, por lo que se diseñó una relación circular entre ellos, de tal forma que si se cae uno de ellos sólo es posible que sea detectado y levantado por el anterior (algoritmo de anillo). Tal cómo se puede observar en el siguiente gráfico:



De esta manera el sistema puede seguir funcionando siempre que haya un Health Checker vivo, ya que con esto se podrán ir levantando los demás junto con sus procesos monitoreados de forma circular. El monitoreo se realiza por medio de mensajes directos (utilizando la abstracción *SocketHandler* que encapsula una conexión tcp) desde los health checkers, que deben ser respondidos con un mensaje de tipo ALIVE. Los health checkers llevan a cabo esta revisión con un subproceso por nodo a monitorear, cada cierto intervalo de tiempo definido.

2. Killer

Sistema que da de baja a algún nodo del sistema, tiene dos modos de prueba: interactivo y random. A través de un comando de Makefile se puede acceder de cualquier modo, primero se debe asegurar que el sistema esté funcionando correctamente.

- Modo Interactivo:
  Este modo permite mucho más control a la hora de dar de baja a un contenedor, para esto envía una señal SIGKILL, con el objetivo de simular una caída sin control alguno por parte del sistema de algún nodo, como lo hace otro comando de que envía la señal SIGTERM, permitiendo que el nodo pueda cerrar los recursos usados.
  Una vez en modo interactivo (make killer-interactive):
  - list - Ver contenedores disponibles
  - kill <nombre_contenedor> - Matar contenedor específico
  - exit - Salir del modo interactivo
- Modo Random:
  Este modo no permite controlar los nodos que se dan de baja, eligiendo de forma aleatoria algún nodo, de todos los nodos levantados en el sistema. Este nodo también tiene un control sobre los health checkers que han sido dados de baja, dejando siempre un health checker funcionando para que el sistema se recupere nuevamente. Cuenta con una ventana de tiempo, esto permite que no pueda darle de baja a un mismo contenedor hasta que pase esa ventana de tiempo.

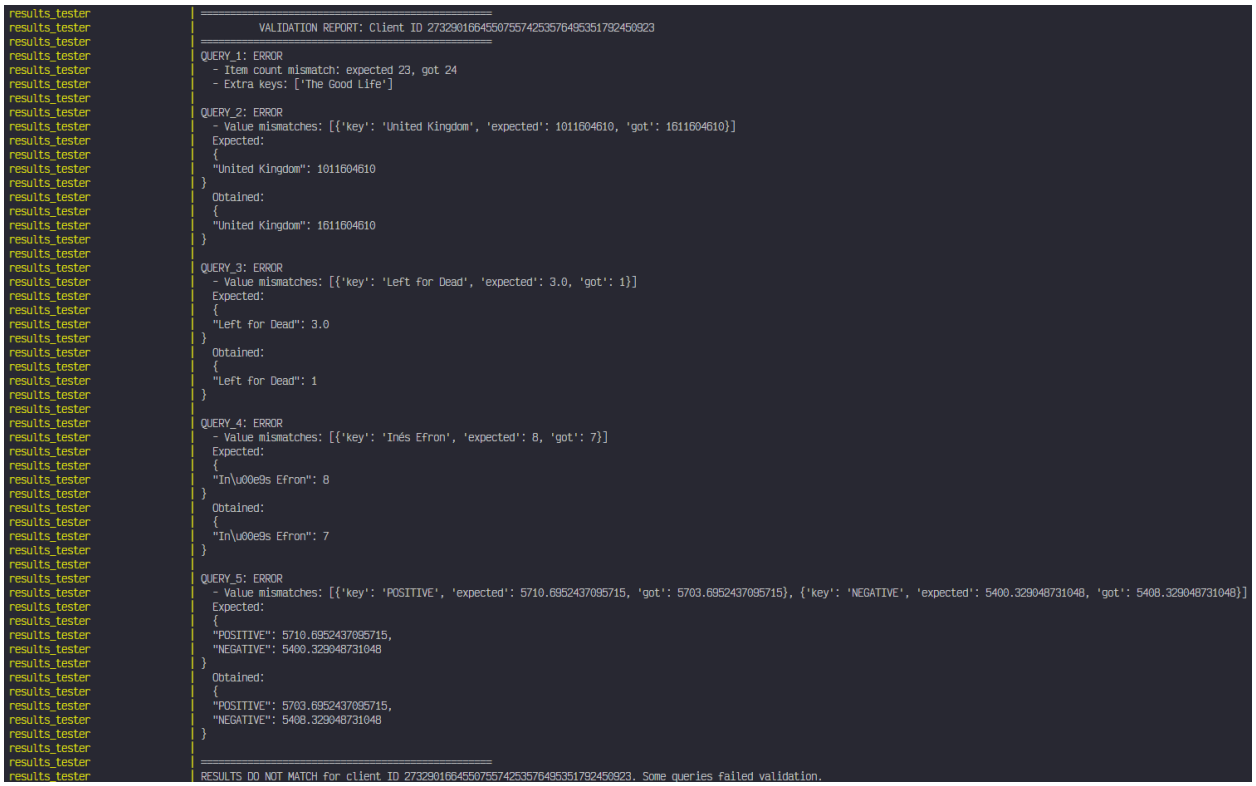
3. Sistema de comparación de resultados

Con el objetivo de validar la consistencia de los resultados obtenidos, se implementó un nodo adicional encargado de verificar las respuestas de las queries una vez que fueron entregadas al cliente.

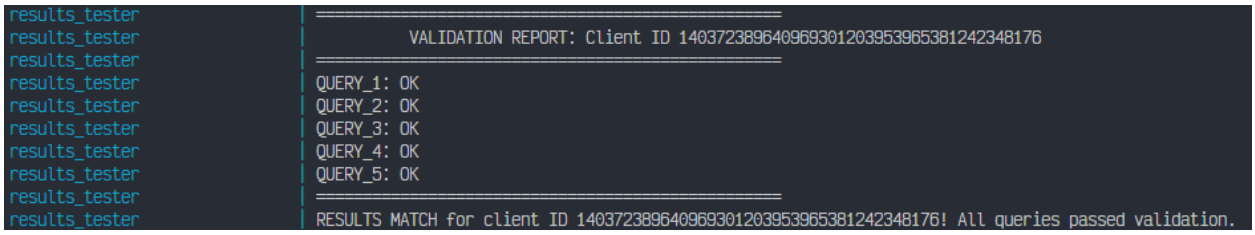
Este nodo actúa como un validador externo. Recibe por socket las respuestas correspondientes a las cinco queries ejecutadas por el sistema, y realiza un análisis comparativo para detectar posibles inconsistencias o errores en los resultados.

La verificación se realiza de forma automatizada y se reporta por consola, indicando claramente si una respuesta presenta diferencias con respecto al valor esperado, en qué parte se produce la discrepancia y, en caso contrario, si la respuesta es considerada válida. Esto permite auditar el comportamiento del sistema distribuido y facilita la detección de errores en los mecanismos de procesamiento o agregación de datos. Ejemplo de log por consola:

- Error en las respuestas de las consultas



- Éxito en las respuestas de las consultas



4. <u>Escritura de archivos atómicas</u>

Para evitar la corrupción de datos ante posibles fallos en medio del proceso de escritura —como la caída inesperada de un nodo—, se implementó una estrategia de escritura atómica mediante la clase FileManager.

El procedimiento asegura que el archivo de destino nunca quede en un estado inconsistente. La lógica consiste en los siguientes pasos:

1. Se crea un archivo temporal como copia del archivo original (si existe).
2. Se escribe la nueva información recibida (por ejemplo, un batch de datos) en este archivo temporal.

3. Una vez finalizada la escritura y asegurada su persistencia en disco (flush() y fsync()), se reemplaza el archivo original con el archivo temporal usando os.rename(), lo cual es una operación atómica.
4. Finalmente, se elimina cualquier archivo temporal obsoleto que haya quedado de ejecuciones anteriores.

Este mecanismo garantiza que nunca se escriba un archivo parcialmente actualizado, lo cual es fundamental para mantener la integridad del estado compartido entre nodos del sistema.

Además, la función que realiza esto retorna un hash determinístico (MD5) del archivo actualizado. Este valor puede ser utilizado por otros nodos, como los joiners, para verificar si un archivo fue modificado, permitiendo estrategias de sincronización eficientes. Este aspecto será desarrollado en detalle en secciones posteriores del informe.

5. Client spawner:

A diferencia del sistema principal donde se pueden crear clientes a través del script "generar-compose", esta nueva serie de comandos agregados al Makefile, permite crear a nuevos clientes que puedan conectarse al sistema simulando una conexión más real dado que las conexiones de los nuevos clientes serán en diferentes instancias de tiempo.
Los nuevos comandos agregados al Makefile que permite crear a estos nuevos clientes son los siguientes:

- make client-spawn: Pide el ingreso de la cantidad de clientes a ser spawneados (recomendado)
- make client-spawn-n: Pasar por línea de comando la cantidad de clientes
- make client-list: Listar todos los contenedores clientes spawneados
- make client-kill: Matar un contenedor (pide id del cliente)

Además, el sistema de creación de nuevos clientes también permite ver la salida de los logs por consola de los clientes, y así poder observar el comportamiento de los clientes

6. Sistema de detección de caída del cliente:

Este sistema permite al gateway detectar si el cliente se desconectó mediante el error del socket que se creó al conectarse el cliente, el cual permite enviar una señal a través de los canales de rabbitmq para abortar las operaciones que se realicen para ese cliente.

Los primeros nodos reciben esta señal y deciden si propagar o no el mensaje de abortar, dependiendo si el cliente terminó de ser procesado por el nodo mediante el EOF, que indica si todos los mensajes llegaron a ser entregados y procesados, por ese nodo.

Esto es así porque el último mensaje en las queues que son consumidas por los nodos serán o bien el EOF o el mensaje de ABORT, y solo quienes sigan procesando información sobre el cliente siempre recibirá como último mensaje algunos de los dos.

7. Restauración de estado de un nodo después de una caída

Cuando un nodo es finalizado abruptamente, ya sea por el killer o por una falla inesperada, el health checker encargado de su supervisión intentará comunicarse con él mediante un mensaje de verificación. Ante la falta de respuesta, este componente procederá a reiniciar el nodo utilizando la API de Docker.

¿Pero cómo se restaura su estado?
Cada nodo persiste su estado en un archivo JSON almacenado fuera del contenedor, tal como se explicó previamente. Esta decisión permite mantener la información crítica respaldada de manera persistente, independientemente del ciclo de vida del contenedor.

Al ser reiniciado, el nodo accede a su archivo de estado para reconstruir el contexto en el que se encontraba antes de la caída. Este estado incluye, entre otros datos, el último número de secuencia recibido y la cantidad de mensajes de fin de archivo (EOF) procesados. La detección y manejo de duplicados será abordada en la sección siguiente.

Además, los nodos stateful mantienen un archivo adicional en formato CSV, también almacenado por fuera del contenedor, desde el cual deben recuperar toda la información operativa previa. Esta estructura de almacenamiento externo permite una restauración simple y confiable del nodo.

Una vez cargados los archivos correspondientes, el nodo puede volver a operar desde el mismo punto en que fue interrumpido, garantizando la continuidad del sistema.

8. Sistema de detección de duplicación de mensajes

El correcto desarrollo y funcionamiento de la detección de mensajes duplicados fue nuestro principal desafío a la hora de realizar este trabajo práctico. Para poder implementar este sistema necesitábamos tener algunas funcionalidades antes de poner en funcionamiento este sistema, las cuales fueron: Health checker, para volver a poner en funcionamiento los nodos caídos, un killer que baje los nodos y sobre todo hacer que cada nodo se vuelva a poner en funcionamiento con el estado anterior al que se encontraba.
También, tenemos que hacer énfasis en los tipos de nodos que tenemos, dado que la mayoría de nodos son stateless y algunos pocos statefull, esto ocasiona que exista una variación adicional a la hora de detectar la duplicación de un mensaje.

En general, cada nodo realiza lo siguiente:

1. Recibir mensaje de batch

2. Procesar mensaje
3. Propagar el resultado
4. Actualizar seq number
5. Actualizar state en disco
6. Mandar ACK

Al realizar esta secuencia de pasos, en una eventual caída por parte del nodo, se pueden ocasionar casos de duplicación de mensajes, los cuales son:

- Se recibe mensaje batch y se realiza la caída del nodo antes de propagar el mensaje, el broker no va a recibir nunca el ACK y por lo tanto el mensaje va a seguir encolado y eventualmente va a ser consumido por el nodo cuando se restaure.
  Siguiendo el orden que realiza un nodo, para este caso el resultado nunca se propagó y por lo tanto el estado nunca se actualizó, dando como resultado que el mensaje vuelva a ser procesado por el nodo. En este caso no hay duplicación de mensaje.
- Recibir mensaje batch, procesar el mensaje, propagar la respuesta, en ese instante se produce la caída del nodo sin llegar a enviar ACK: En este caso se duplica un mensaje ya que a pesar de que el nodo va a volver a procesar el mensaje y volverlo a propagar como en el punto anterior, el siguiente nodo va a recibir el mismo mensaje otra vez, siendo aquí donde el siguiente nodo va a descartar el mensaje por que va a detectar el duplicado a través del sistema que emplea el nombre del controlador que envía el mensaje así como el sequence number, toda esta información está adherida al estado local del nodo que lo guarda por cliente algo como lo siguiente:

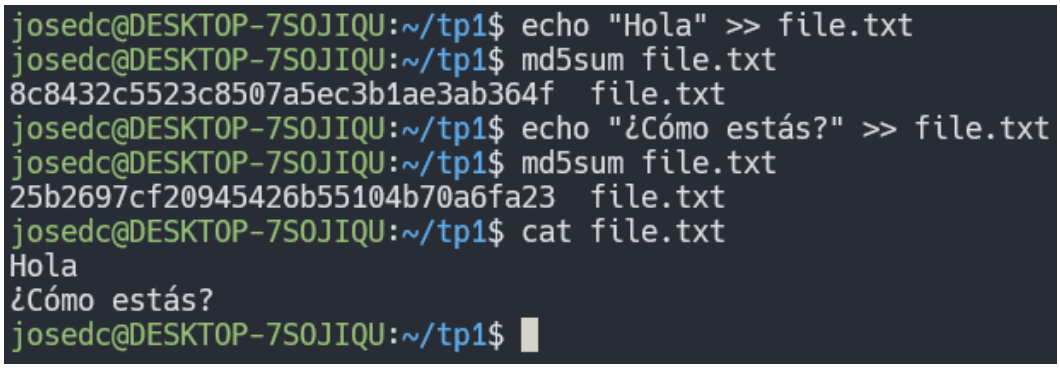
  “client_id”: {
  “last_seq_number: 12,
  “<controller_name_1>”: <seq_number_controller_1>
  “<controller_name_2>”: <seq_number_controller_2>
  }

  - last_seq_number, el número de secuencia propio del nodo para ese cliente.
  - <controller_name>: nombre del controlador que envía el mensaje
  - <seq_number_controller>: número de secuencia que envía el controlador

  Siendo que por el *controller_name* y el *seq_number_controller* se detecta el duplicado del mensaje.

Ahora la variación que existe es con los nodos que son statefull, dado que estos a parte de controlar la duplicación de mensajes provenientes de otros nodos, tiene que controlar que la información que guarda en sus archivos no esté duplicada. Para esto se emplea un sistema que realiza un hash determinístico a los archivos guardados y se guarda el hash determinístico obtenido, de manera que se pueda detectar cambios en los archivos, esto pasa por que al agregar información nueva en el archivo hace que el hash cambie.
Entonces, al caer el nodo, si al volver a funcionar el hash del archivo es diferente al guardado significa que el archivo contiene información nueva, permitiendo saber que el siguiente mensaje batch no está actualizado en el estado del nodo pero si en el archivo (Siempre teniendo en cuenta

que la actualización del archivo es atómica). Esto sucede gracias a la configuración de RabbitMQ que solo puede ir procesando un mensaje a la vez y procesar el siguiente mensaje sólo si se recibe el ACK del mensaje anterior.



Ejemplo de hash de un archivo con mensajes nuevos

Además que estos tipos de nodos no propagan su resultado hasta tener completa toda la información de nodos anteriores,por lo tanto los nodos siguientes no van a tener que detectar un mensaje duplicado.

La secuencia de estos nodos quedaría así:

1. Recibir mensaje de batch
2. Procesar mensaje
3. Propagar el resultado si tengo toda la información
4. Actualizar seq number
5. Actualizar state en disco
6. Mandar ACK

Plan de actividades

Actividad	Responsable
Armar datasets de prueba (5% de datos).	Todos
Armar resultados usando la notebook de Gabriel para tenerlos como referencia.	Todos
Protocolo Cliente - Gateway (communication)	Todos
Código Cliente: lectura de archivos y envío al gateway	José
Código Cliente: lectura de reportes y escritura en archivos	José
Gateway: config inicial y aceptar nuevas conexiones	Mauricio
Gateway: parseo de mensajes y uso del middleware (rabbitMQ)	Mauricio
Gateway: envío de reportes	Mauricio
Filtros del primer nivel (año, país, solo 1 país inversor)	Mateo
Q5 -> NLP Processing + Budget / Revenue Aggregator	Mateo
Top 5 países y Top 10 de actores + Películas y Géneros	Mateo
Average Budget/Revenue + Average Rating	Mauricio
Joiners (rating y credits)	Jose
Soportar Multi Clients	Todos
Refactor de arquitectura	Todos
Health Checkers	Todos
Resilient Node	Todos
Client Spawner	Todos
Node killer	Todos