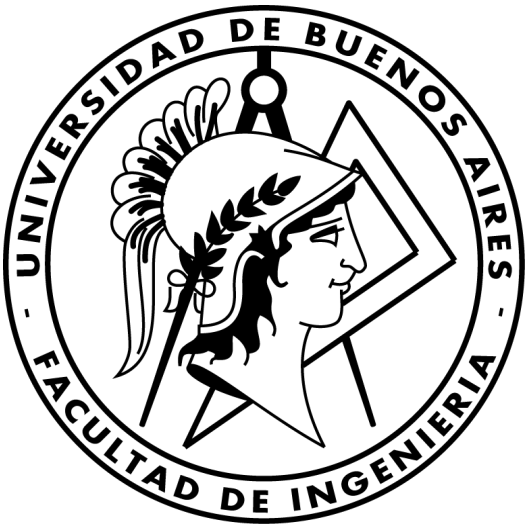
FACULTAD DE INGENIERÍA,
UNIVERSIDAD DE BUENOS AIRES
2024



Amazon Reviews

Sistemas Distribuidos (75.74)

Padrón	Alumno	Correo electrónico
108183	Juan Manuel Diaz	gbedoya@fi.uba.ar
107602	Gabriel Bedoya	gbedoya@fi.uba.ar

Ayudante: Gabriel Robles

Índice

Introducción

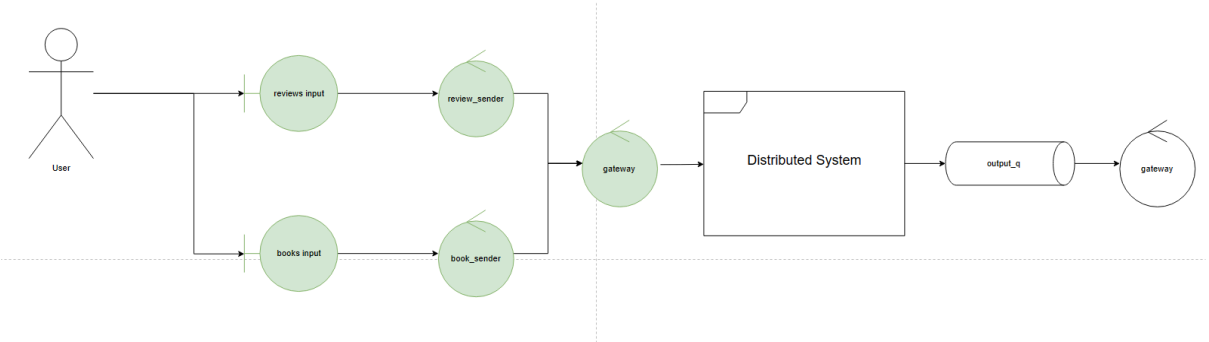
El objetivo de este trabajo es crear un sistema distribuidos que analice las reseñas de libros en el sitio de Amazon, dichas reseñas poseen título del libro, texto del comentario y rating. Por cada título de libro, se conoce categoría, fecha de publicación y autores.

El sistema debe permitir obtener:

- Título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan 'distributed' en su título.
- Autores con títulos publicados en al menos 10 décadas distintas ○ Títulos y autores de libros publicados en los 90' con al menos 500 reseñas.
- 10 libros con mejor rating promedio entre aquellos publicados en los 90’ con al menos 500 reseñas.
- Títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto.

Propuesta

Se construyó un sistema distribuido que utiliza como broker interno de comunicación el message oriented middleware RabbitMQ, y externamente se comunica mediante un gateway con el cliente a través de un protocolo de comunicación. Dicha comunicación es transparente para el usuario al no conectarse con el broker interno, y permitir enviar tanto el input como recibir el output por dicho protocolo, un diagrama que permite ejemplificar la arquitectura es:



Ejecución

Para la ejecución del sistema, se provee un Makefile que soporta los siguientes comandos:

- make docker-compose-up: permite levantar el sistema con el docker-compose-dev.yaml provisto en el repositorio
- make docker-compose-client: Permite ejecutar un cliente y conectarse al sistema mediante el gateway. El archivo yaml que utiliza es docker-compose-client.yaml

- make docker-compose-down-client: Finaliza la ejecución del cliente y elimina los containers creados.
- make docker-compose-down-all: Finaliza la ejecución tanto del sistema como del cliente y todos los containers asociados.

Para poder ejecutar el sistema correctamente, el cliente debe poseer el archivo de books y reviews provistos por amazon, los cuales se pueden obtener en: Amazon Books Reviews
Se mencionó anteriormente los archivos yaml tanto del cliente como del sistema, ambos archivos permiten configurar el entorno de ejecución, para el cliente lo más importante es dónde se encuentran los archivos a enviarse al gateway (los paths correspondientes), y el directorio donde se guardarán los outputs en respuesta del sistema a través del gateway. Además se provee un parámetro de configuración que es la cantidad de mensajes a enviar durante la transferencia de los archivos.
En el caso del sistema, el archivo yaml es mucho más extenso y se permite configurar aspectos más ligados a la arquitectura y por ejemplo queues/exchanges internos que maneja el sistema.

Vista de Escenarios - Casos de Uso

- Dado un usuario que posee los datasets book details y reviews, cuando el usuario ejecuta el sistema podrá consultar el título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan 'distributed' en su título, entonces obtendrá un archivo con la salida de la consulta.

- Dado un usuario que posee los datasets book details y reviews, cuando el usuario ejecuta el sistema podrá consultar los autores con títulos publicados en al menos 10 décadas distintas, entonces obtendrá un archivo con la salida de la consulta.

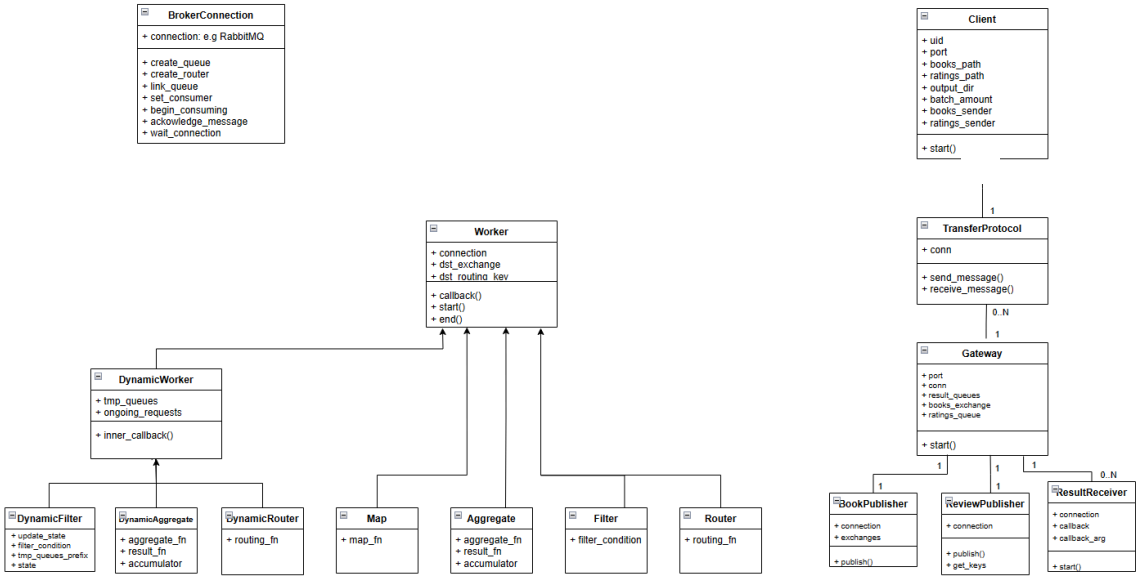
- Dado un usuario que posee los datasets book details y reviews, cuando el usuario ejecuta el sistema podrá consultar los títulos y autores de libros publicados en los 90' con al menos 500 reseñas, entonces obtendrá un archivo con la salida de la consulta.

- Dado un usuario que posee los datasets book details y reviews, cuando el usuario ejecuta el sistema podrá consultar los 10 libros con mejor rating promedio entre aquellos publicados en los 90’ con al menos 500 reseñas, entonces obtendrá un archivo con la salida de la consulta.

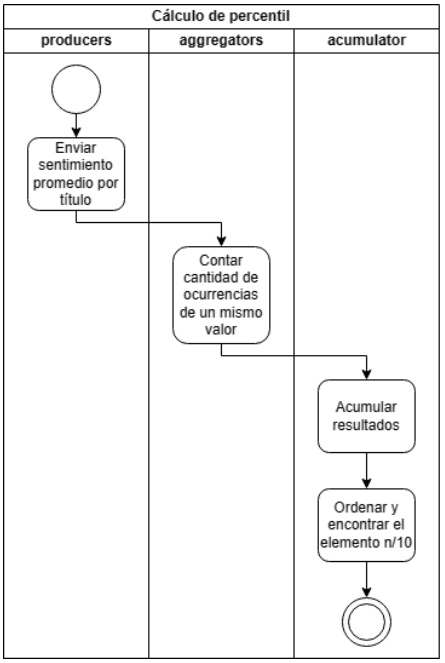
- Dado un usuario que posee los datasets book details y reviews, cuando el usuario ejecuta el sistema podrá consultar los títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto, entonces obtendrá un archivo con la salida de la consulta.

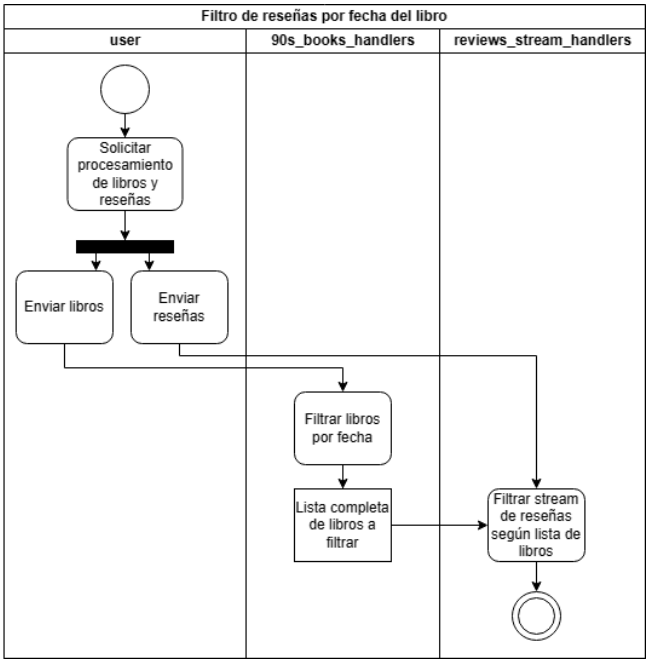
Vista Lógica - Diagrama de Clases

El siguiente diagrama muestra las relaciones entre las clases del sistema, las flechas indican herencia de la clase a la que apunta la fecha, además se agrega la cardinalidad de cada clase empleada así como los métodos y atributos más relevantes de cada clase.



Vista de Procesos - Diagramas de Actividades

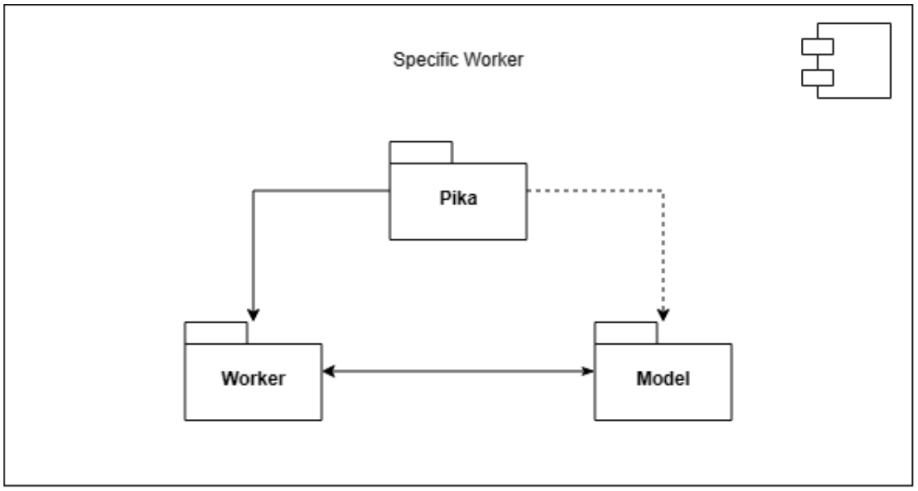


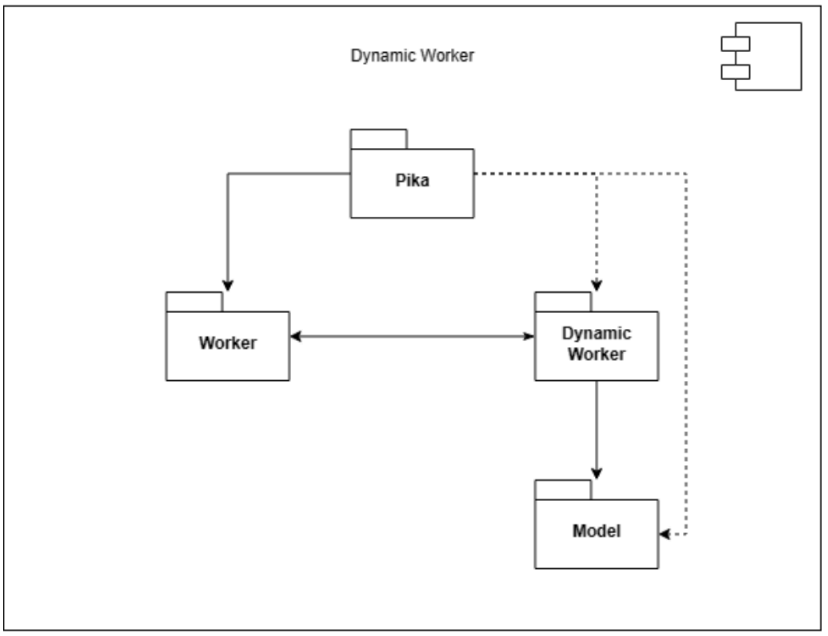


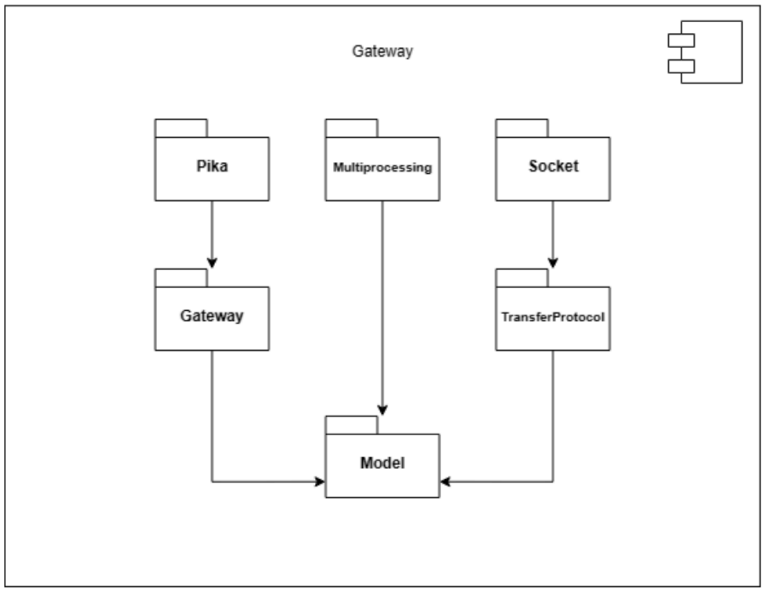
El flujo de percentiles está basado en el sentimiento promedio de títulos de libros, se inicia enviando el sentimiento promedio (utilizando la librería nltk) de las reseñas asociadas a cada título de libro.

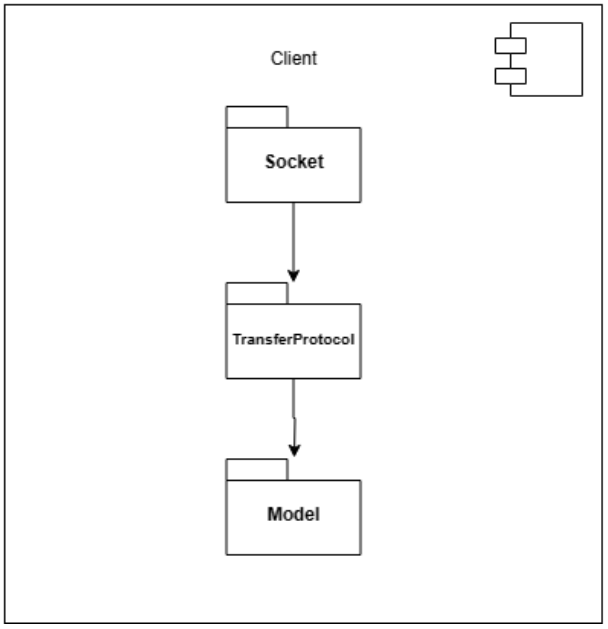
Dichos valores se acumulan hasta recibir el mensaje EOF del nodo anterior que envía, indicando que no se recibirán más resultados para dicho cliente, se ordenan los resultados y se procede a encontrar el valor correspondiente al percentil deseado.
El caso del otro diagrama es similar, sin embargo al emplear dos inputs para el resultado final, se maneja una barrera que espera hasta obtener la lista completa de los libros a filtrar.

Vista de Desarrollo - Diagrama de Paquetes





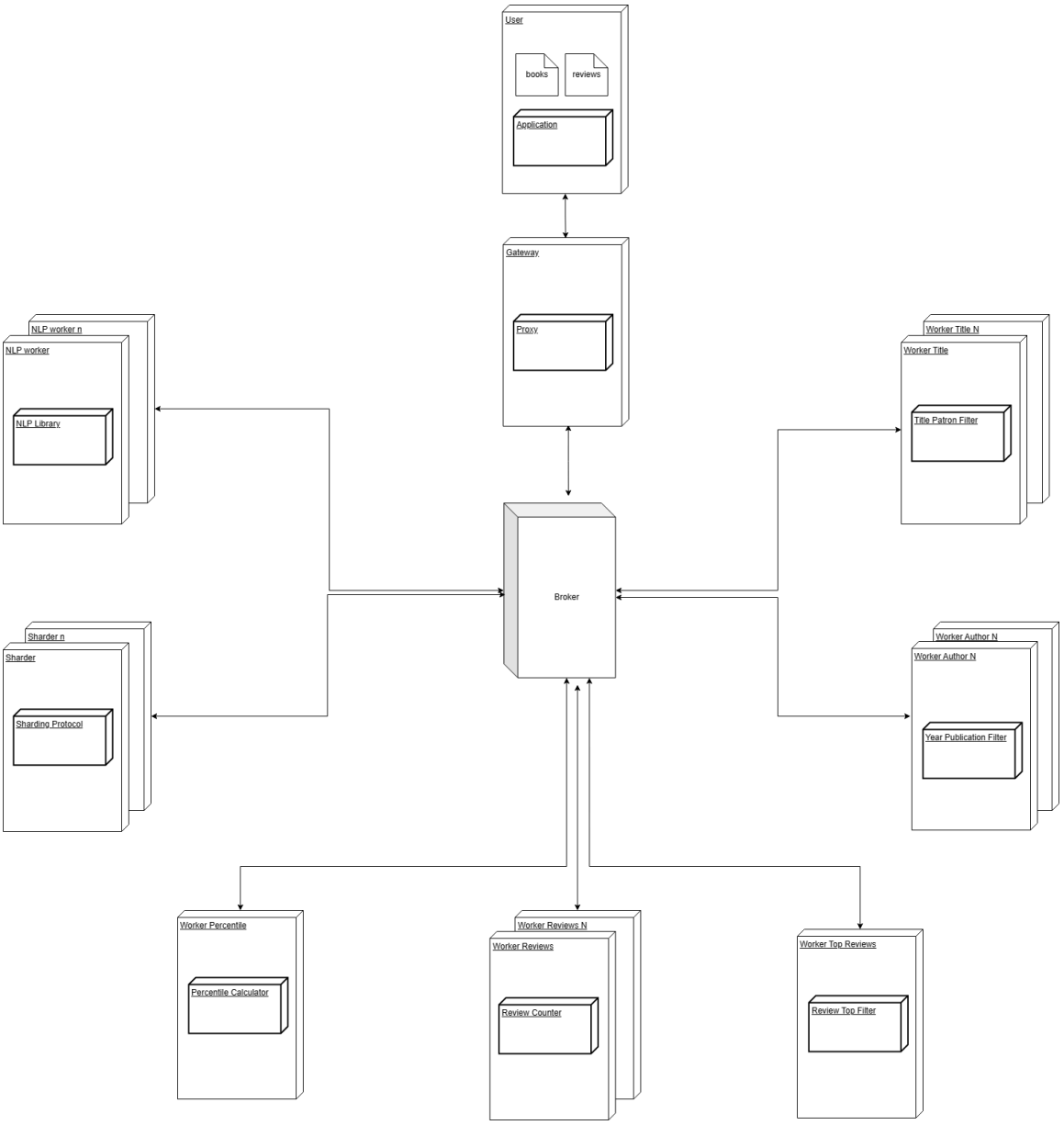

Gateway
Pika
Multiprocessing
Socket
Gateway
TransferProtocol
Model


Client
Socket
TransferProtocol
Model

Vista Física - Diagrama de Despliegue

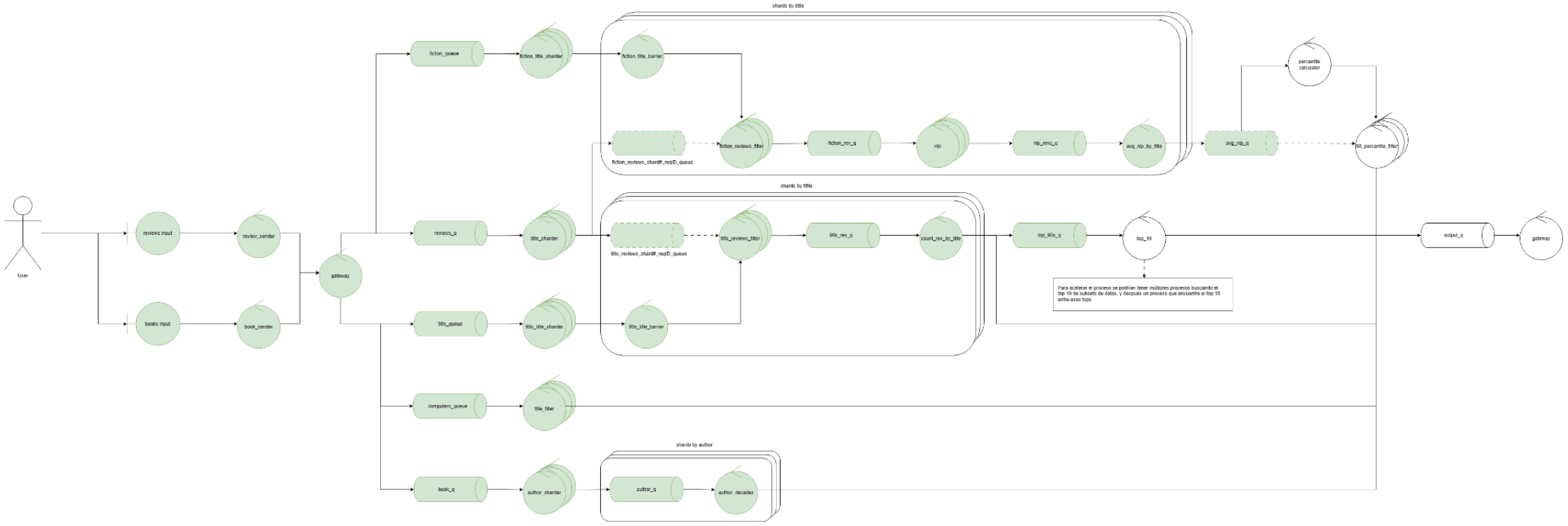
En el siguiente diagrama se muestra una topología similar a una estrella, esto es debido a que todos los nodos internos se comunican a través del broker (en este caso RabittMQ), exceptuando Cliente-Gateway que se comunican vía tcp.

Además todos los nodos poseen flechas en ambos sentidos, esto es porque incluso los nodos internos del sistema envían mensajes a través del broker a la siguiente layer, y recibe como input el output de la layer anterior.



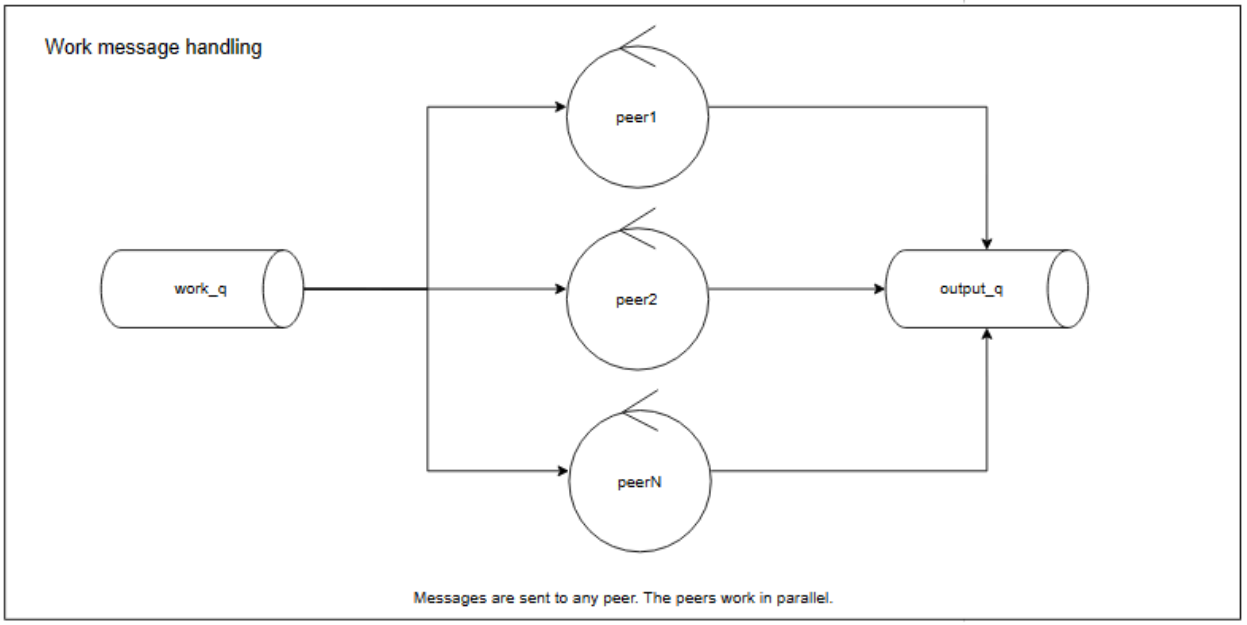
Vista Física - Diagrama de Robustez

El siguiente diagrama provee un esquema general del sistema completo (ampliación del diagrama mostrado al comienzo del documento), como observación existen nodos que poseen duplicados, esto hace referencia a que pueden escalar según la necesidad y recursos disponibles.

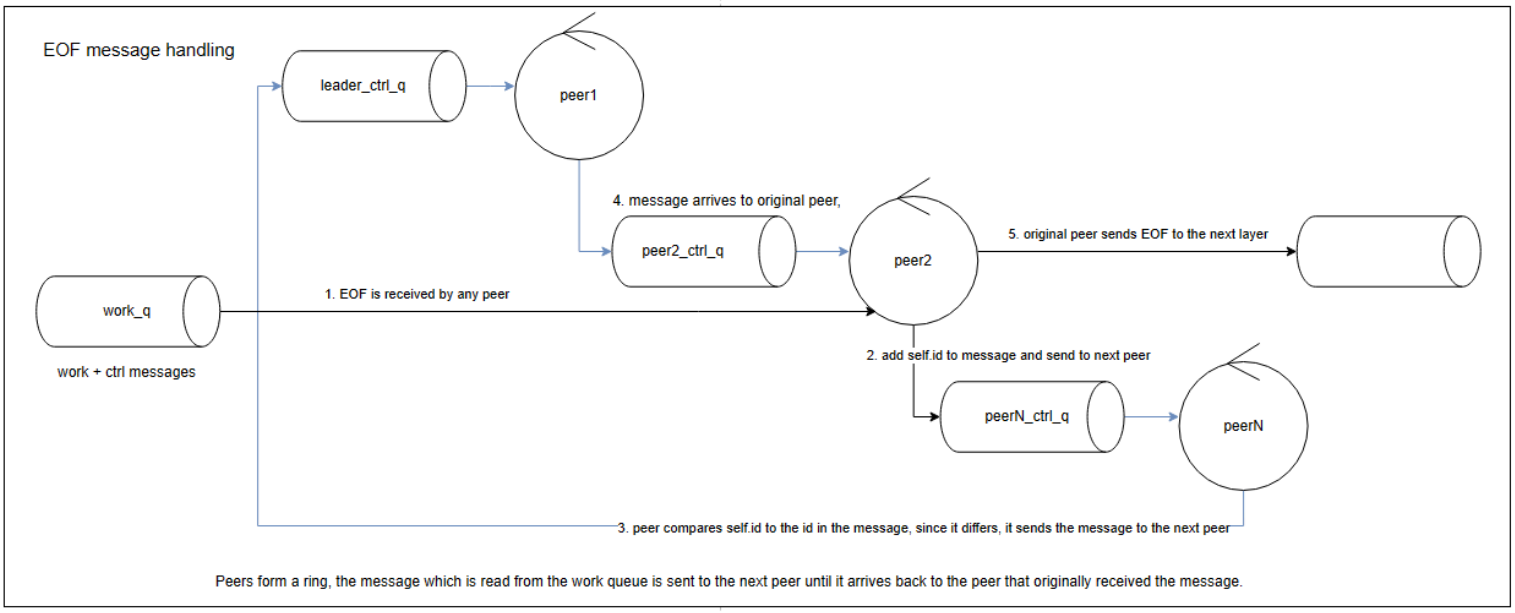


Clusters

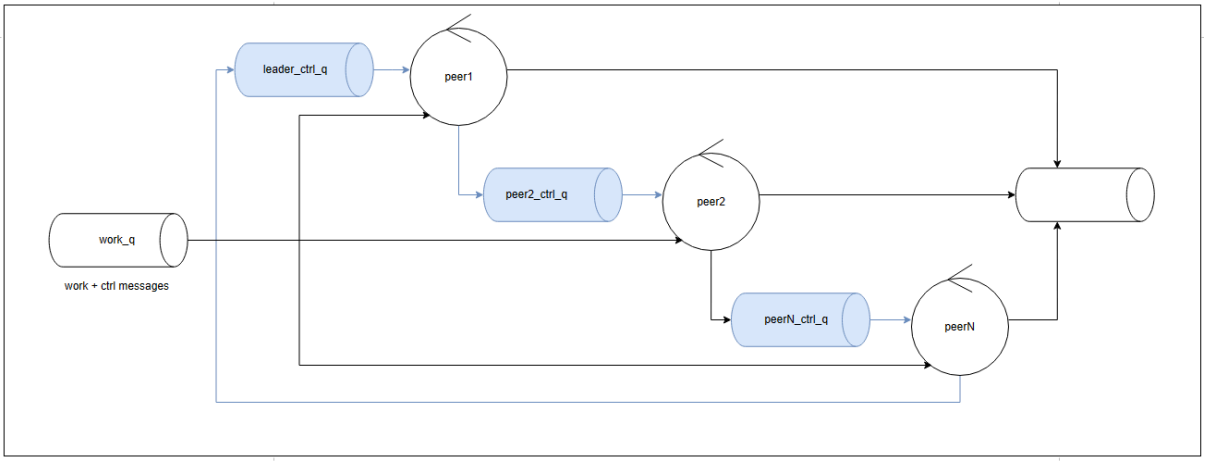
A continuación se provee una visión general de cómo se conforman los clusters de workers que se escalan:



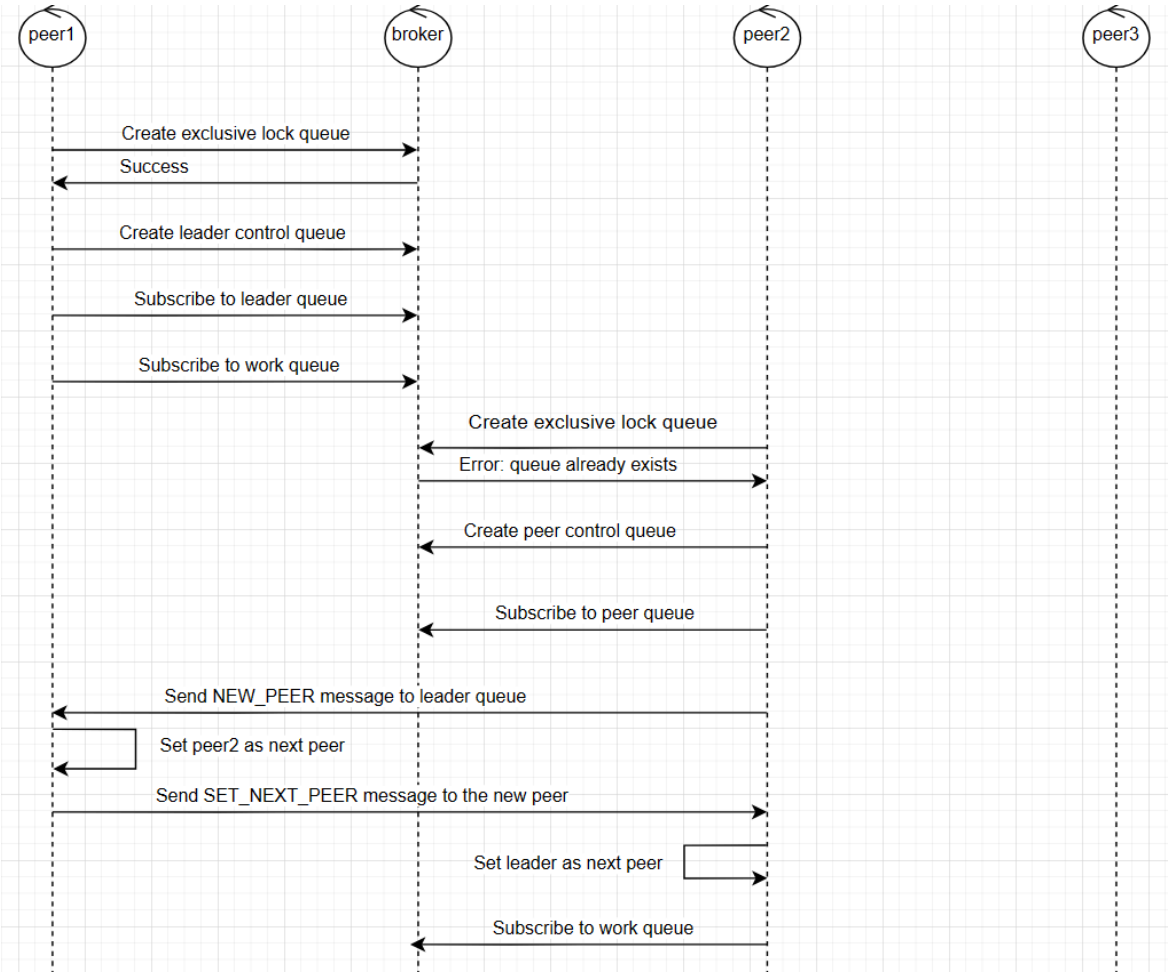
Sin embargo para el manejo de EOFs, al tener peers que trabajan en paralelo surge un problema de sincronización, al tener que comunicar a los demás peers que se recibió el EOF, para esto se dispone de una arquitectura de anillo de la siguiente manera:

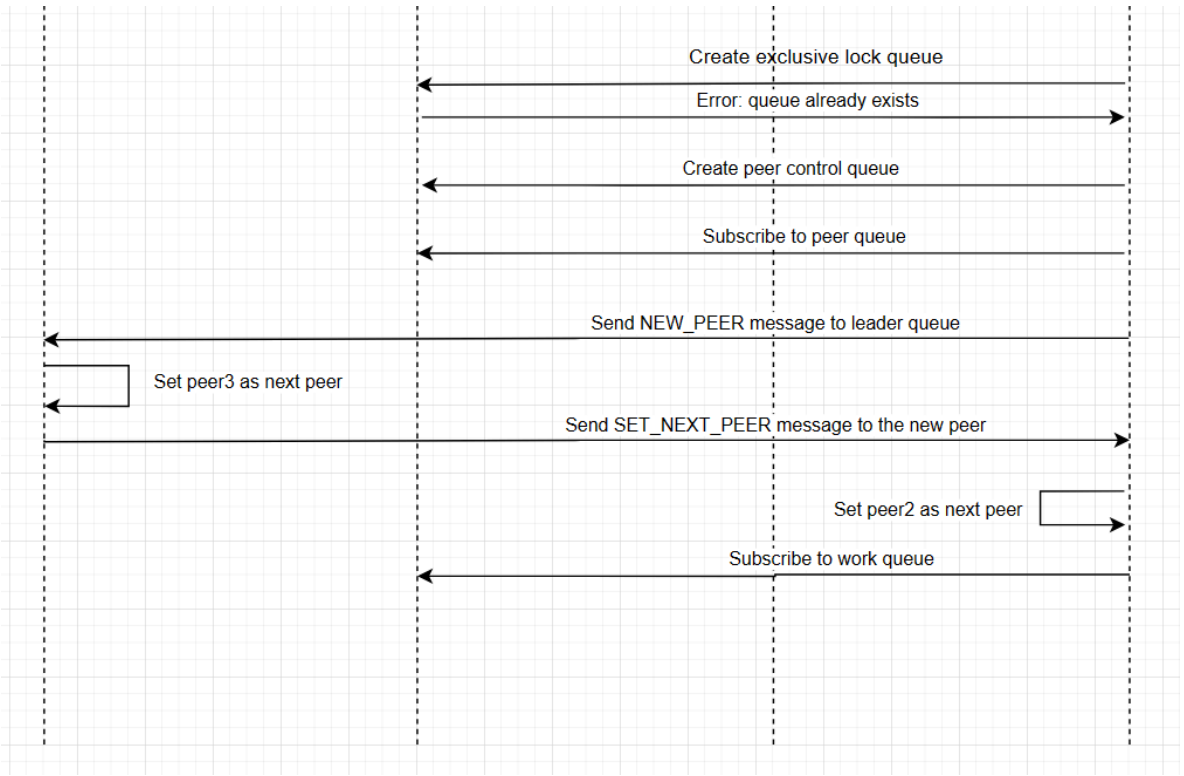


En el diagrama anterior se observa la disposición de la arquitectura en forma de anillo para el caso mencionado, en general se obtiene:



Desde una otro punto de vista, la creación del anillo y el peer discovery se puede analizar con un diagrama de flujo, en el cuál se muestra cómo un primer peer obtiene un lock mediante la conexión de una queue seleccionándolo como líder, mientras que los demás al no poder, no lo serán:





Tolerancia a fallas:

Además de lo anteriormente mencionado, se requiere que el sistema sea tolerante a fallas e interrupciones que puedan surgir durante el procesamiento de las querys, para este propósito se adoptaron diversas medidas, como la persistencia del estado de algunos peers, filtrado de mensajes duplicados (problema que puede surgir al desplegar un nodo que se cayó previamente y reprocesa la información), monitoreo del estado de los nodos mediante healtchecks con un healthchecker y replicación de este último en un cluster para asegurar su correcto funcionamiento ante la caída de un nodo healthchecker.
A continuación se provee una descripción más detallada de los métodos implementados para este objetivo

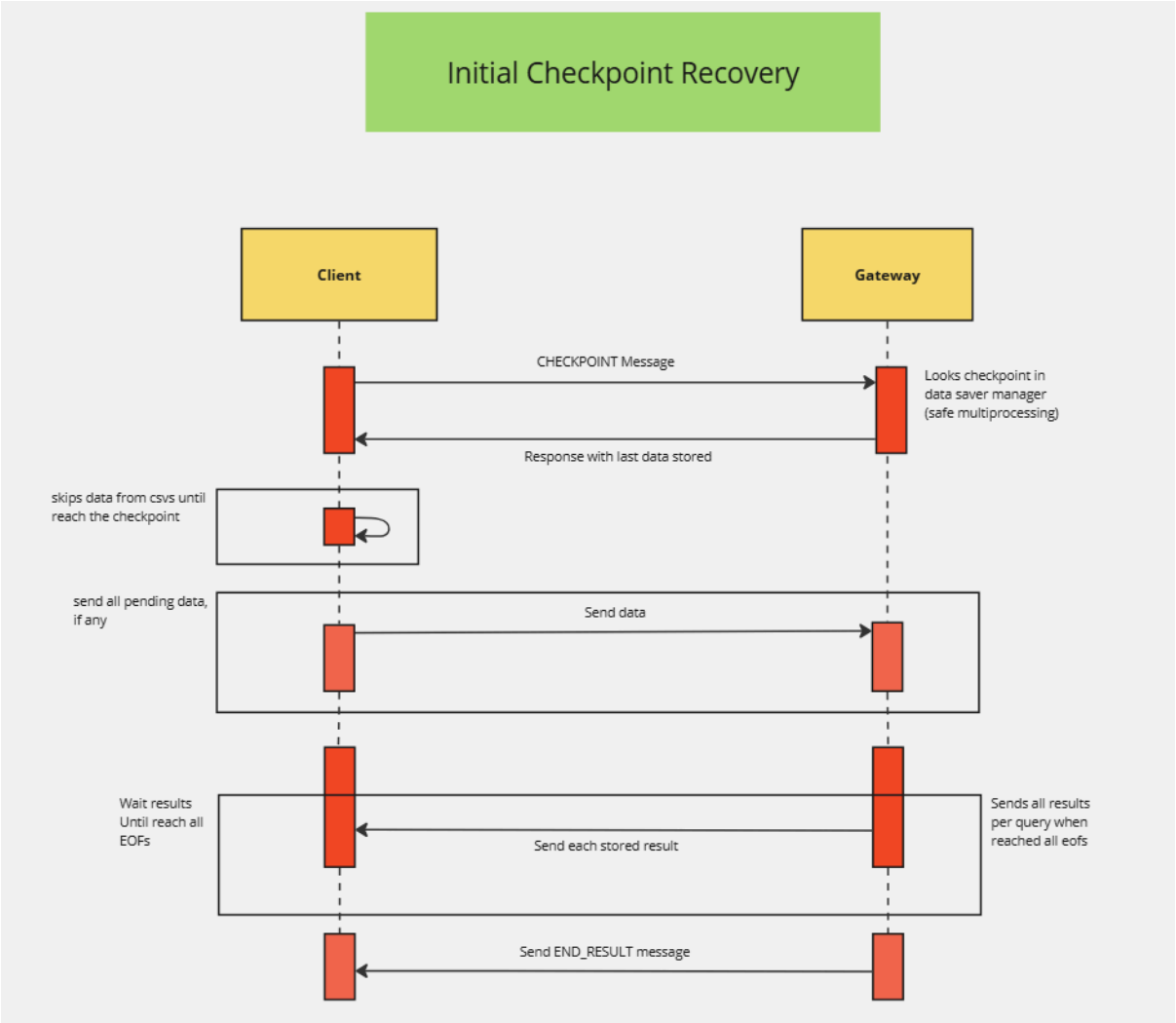
Gateway - Cliente

En el caso de la comunicación entre el cliente y el gateway (vía TCP), si alguno de estos se cae se reinicia el protocolo de comunicación que consiste en solicitar al gateway el estado en el que se encuentra el checkpoint. Para esto el gateway persiste información relevante para la comunicación.

El gateway almacena el último id procesado (publicarlo a los workers) del cliente y adicionalmente si el mensaje es de tipo EOF, esto permite al responder el checkpoint decidir hasta qué línea del csv se debe ignorar debido a que ya fue anteriormente enviada. Si el checkpoint recibido del gateway es un eof, directamente ignora el archivo.

Durante la fase de espera de los resultados de las queries, el gateway almacena todos los mensajes recibidos por las queues de result y una vez recibidos todos los EOFs de cada query se las manda al cliente.

La persistencia de resultados queda asegurada por parte del gateway ante una caída del mismo y posterior restauración.



Healhchecks

Para determinar si un nodo del sistema está caído o no, se realizan healthchecks periódicamente mediante TCP, todos los nodos escuchan los pedidos de healthcheck y envían 1 byte que representa el estado de salud del nodo en ese momento.

Si un nodo no responde un healthcheck (el nodo se detuvo inesperadamente) o su estado de salud es “broken”, se ejecuta un comando de docker restart por parte del healthchecker conectándose al docker daemon (similar a docker in docker).

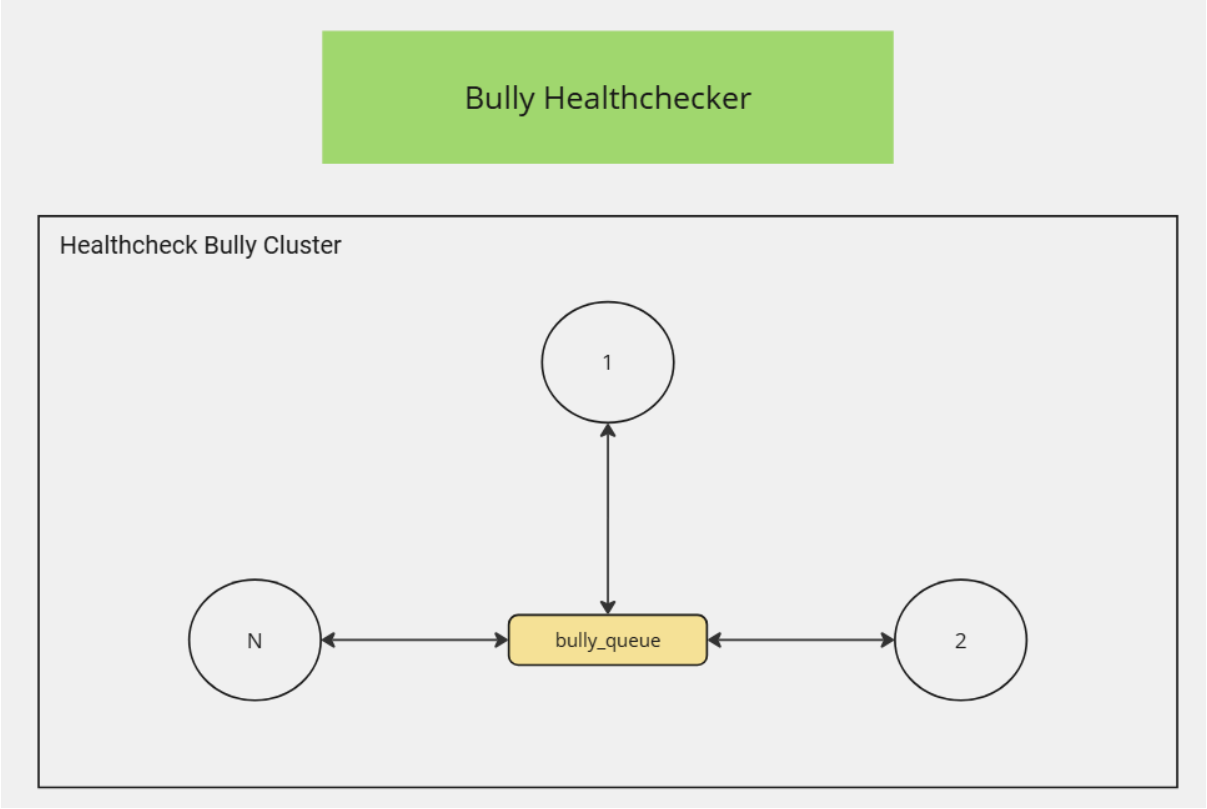
Sin embargo esto genera un nuevo punto de fallo, ¿qué pasa si se cae el healthchecker?

Si tuvieramos un sólo healthchecker y se cae, provocaría que nadie lo reinicie y consecuentemente todos los nodos que se caigan luego de este, no se son reiniciados, desencadenando una posible falla masiva por este single point of failure.

Es por esto que se provee un clúster de healthcheckers, la réplica de estos nodos permite amortiguar la probabilidad de un error catastrófico y que si hay por lo menos uno, los otros serán reiniciados por este.

Las ventajas de la replicación de nodos es una excelente solución para este problema, sin embargo da lugar a un nuevo conflicto, la comunicación y coordinación entre réplicas dado el grave problema que es tener varios nodos que lancen instancias en simultáneo.

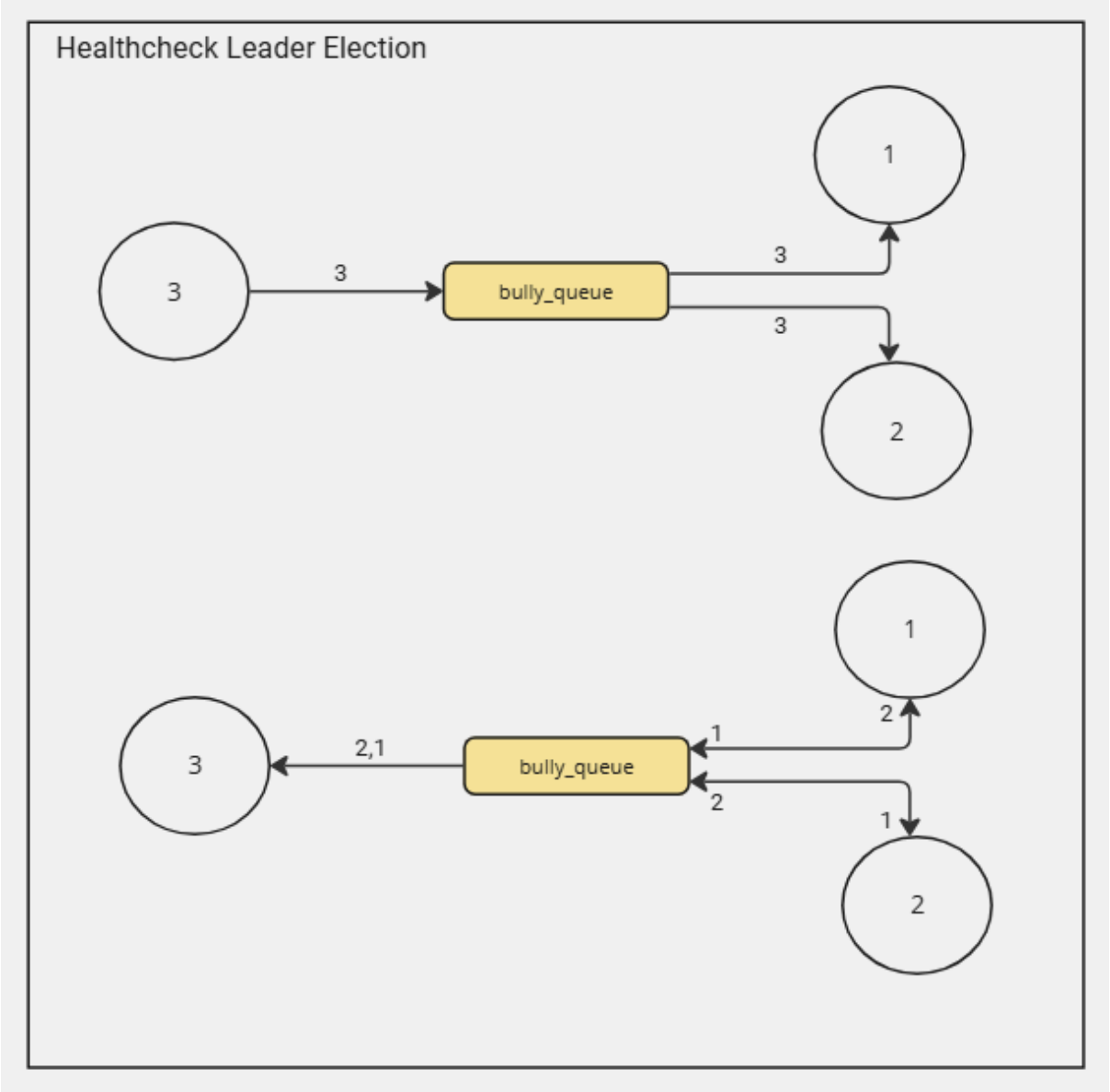
Es por esto que se implementó un algoritmo distribuido de bully para el clúster de healthcheckers:



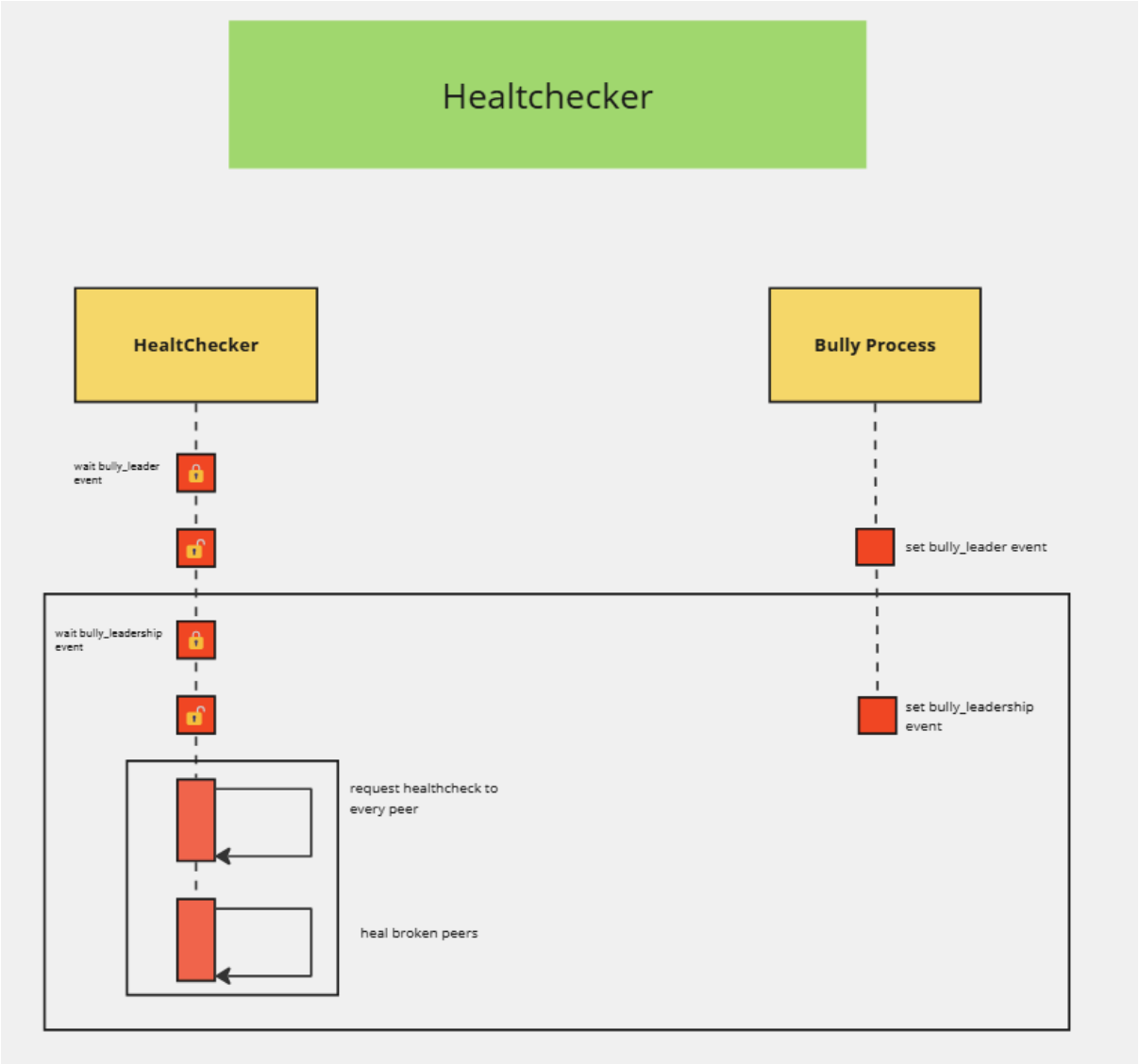
El mismo consiste en realizar votaciones períodicas entre los nodos para determinar quién es el líder actualmente, para ello hay un intervalo de tiempo en el que se reciben los mensajes de election y posteriormente se hace la votación.

La propagación del id se realiza por dos motivos:

- Mediante un clock recurrente que da inicio al envío y posterior recolección de ids.
- En respuesta de un id recibido, entendiendo que hay un peer que realiza un election.



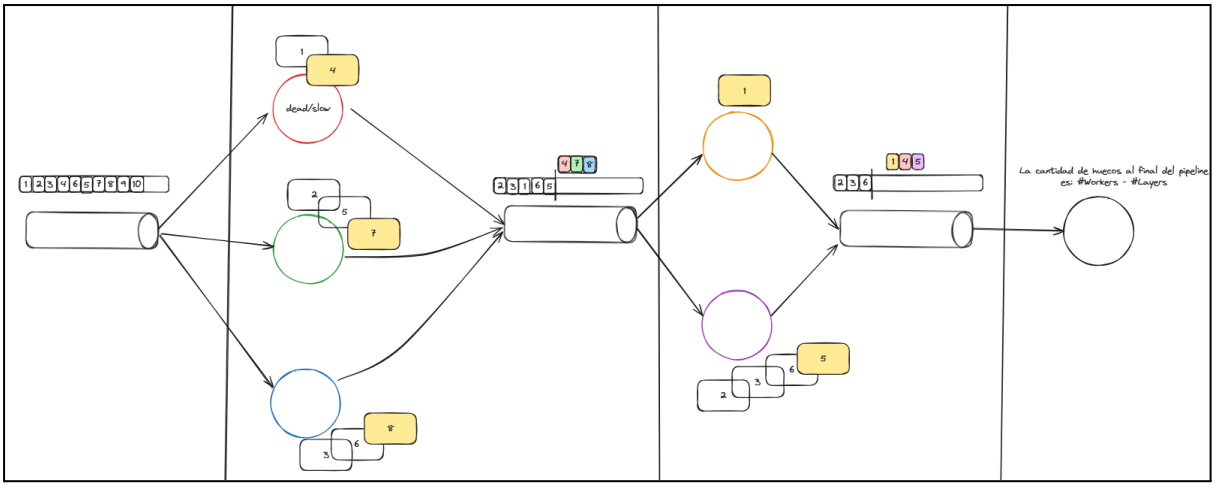
Adicionalmente, se provee un mecanismo de notificación para el escenario en el que un máster se cae, un slave lo reinicia asumiendo el máster, y luego el máster anterior gana un election por tener el id más alto; este escenario provoca que hayan dos másters.



Para esto al ganar un election se envía una notificación de sí sigue siendo máster, si no lo es, se bloquea y no realiza healthchecks.

Mensajes repetidos

Otro de los problemas provenientes de una caída de un nodo, es volver a procesar la información, generando de esta forma un mensaje duplicado, para ello una forma de resolverlo se puede ejemplificar con el siguiente patrón:



Para N workers, en cualquier momento dado la queue puede tener máximo n-1 "huecos" o IDs faltantes para tener todos los mensajes desde 0 hasta el máximo actual.

En la siguiente layer, hay que considerar los huecos de layers anteriores, que se manifiestan como IDs desordenados. La cantidad máxima de huecos es n-1 + m-1.

La cantidad de huecos al final del pipeline es: #Workers - #Layers