.UBAfiuba
FACULTAD DE INGENIERÍA

Amazon Books Analyzer

Escalabilidad, Middleware y Coordinación de Procesos

Primer cuatrimestre 2024

Apellido y Nombre	Padrón	Email
Buzzone, Mauricio	103783	mbuzzone@fi.uba.ar
De Angelis Riva, Lukas Nahuel	103784	ldeangelis@fi.uba.ar

Índice

1. Alcance

Se pide crear un sistema distribuido que analice las reseñas a libros en el sitio de Amazon y que permita obtener resultado de las siguientes 5 consultas:

1. Título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan "distributed" en su título.
2. Autores con títulos publicados en al menos 10 décadas distintas.
3. Títulos y autores de libros publicados en los 90' con al menos 500 reseñas.
4. 10 libros con mejor rating promedio entre aquellos publicados en los 90' con al menos 500 reseñas.
5. Títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto.

1.1. Cálculos auxiliares

Mostramos cálculos auxiliares que ayudarán a tomar decisiones respecto a los elementos a guardar en memoria.

- Hay cerca de 210.000 libros y 3.000.000 de reseñas a trabajar.
- En promedio los títulos de los libros tienen 50 caracteres.
- En promedio hay 1,30 autores por libro y el nombre de los autores mide en promedio 15 caracteres.
- El 20 % de los libros son de los 90' y el 16 % son de categoría ficción.

2. Vista de Escenarios

2.1. Diagrama de casos de uso

El sistema contempla tres casos de uso principales, marcados en el siguiente diagrama.

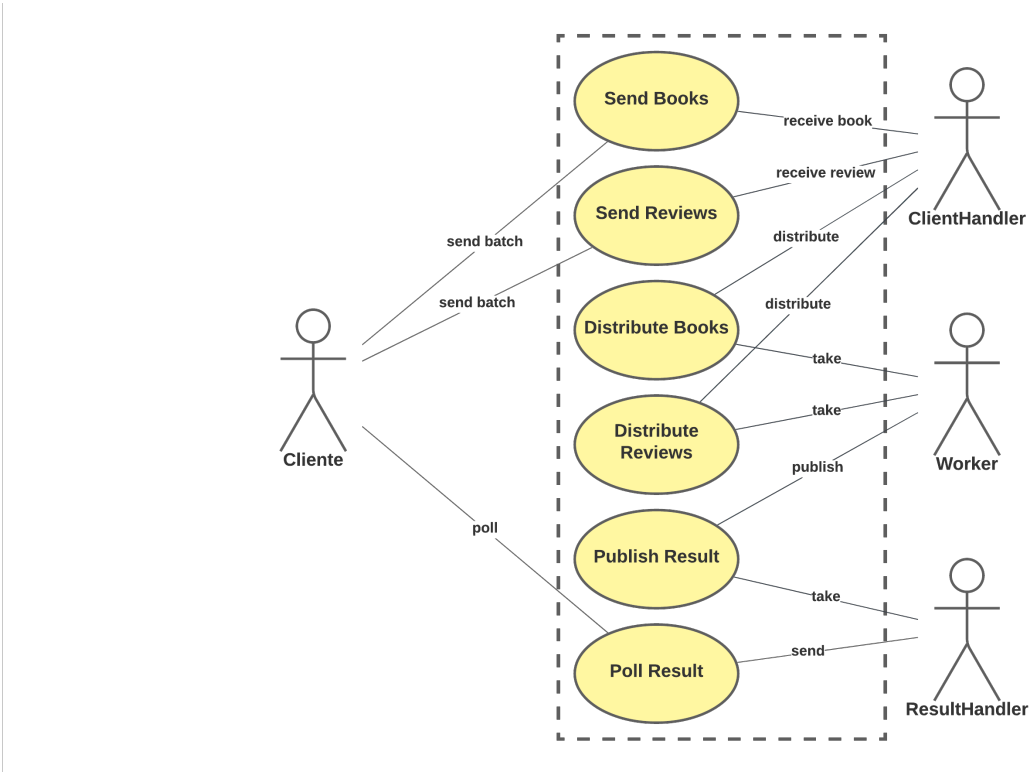

Figura 1: Diagrama de casos de uso

El *cliente* enviará *batches* de libros y reseñas al sistema. Estos serán tomados y distribuidos por el *ClientHandler*. Los workers del sistema tomarán los datos, los procesarán y publicaran los resultados correspondientes.

Dichos resultados serán guardados por *ResultHandler* que se encargará de responderle al cliente cuando este realice un pedido de Poll.

3. Vista Lógica

3.1. DAG

Se presenta el siguiente diagrama DAG mostrando las relaciones entre las distintas actividades. Se puede observar como el procesamiento de las consultas están divididas en etapas con tareas simples e individualmente escalables.

Figura 2: DAG- Amazon Books Analyzer

3.2. Diagramas de Clases

Mas cercano a la implementación se muestran los diagramas de clases de las partes mas importantes del sistema.

3.2.1. Middleware

La clase **Middleware** presenta una interfaz mínima para separar Pika de nuestro sistema. Luego las distintas clases hijas se encargan de declarar de donde y hacia donde se enviarán los mensajes.

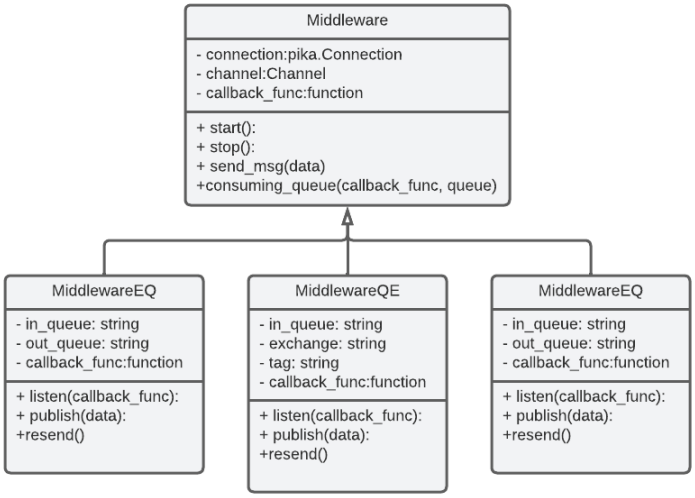

Figura 3: Diagrama de clases Middleware

Donde:

- **MiddlewareQQ**: se trata de un middleware que comunica, tanto por input como por output, con el patrón producer/consumer. **QQ** hace referencia a Queue-to-Queue.
- **MiddlewareEQ**: se trata de un middleware que se comunica, por input utilizando el patrón publisher/subscriber y por output utilizando producer/consumer. En este sentido **EQ** hace referencia a Exchange-to-Queue.
- **MiddlewareQE**: se trata de un middleware que se comunica, por input utilizando el patrón producer/consumer y por output utilizando el patrón publisher/subscriber. En este sentido **QE** hace referencia a Queue-to-Exchange.

Unificando la interfaz de comunicación de los 3 posibles Middlewares en las siguientes sencillas funciones:

- `listen`: escuchar mensajes.
- `publish`: publicar mensajes.
- `resend`: enviar mensajes al input.

3.2.2. Workers y Handlers

Todos los handlers tiene una misma lógica principal donde lo que realmente cambia es lo que hacen con los mensajes que reciben a través del middleware y como se envían hacia el siguiente paso del pipeline, de modo que existe una clase **Worker** que se encarga de encapsular toda la lógica. Luego, los distintos handlers sólo deben implementar una función para trabajar los datos para cada caso particular.

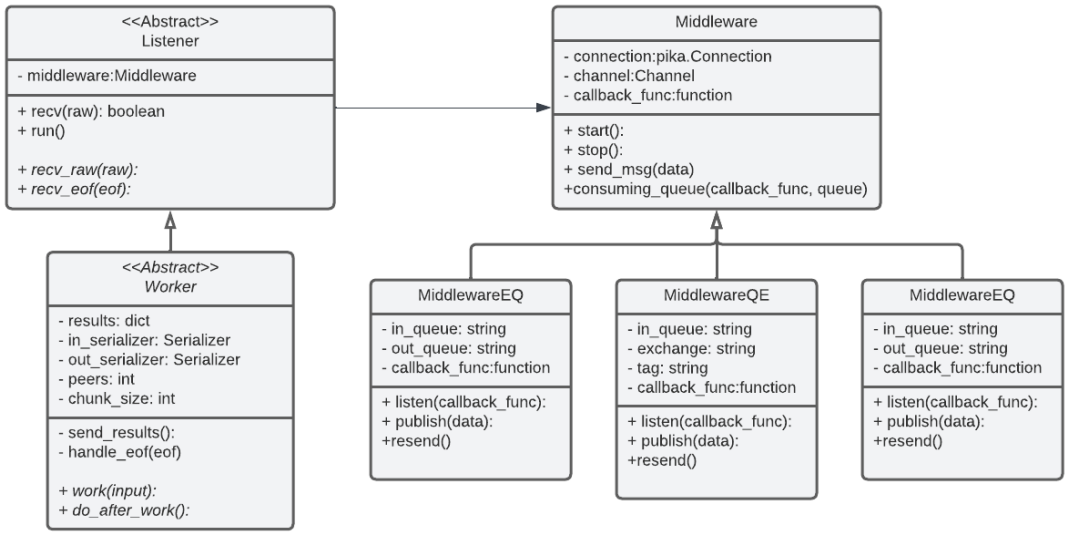

Figura 4: Diagrama de clases Worker

En el diagrama entonces podemos observar dos clases abstractas:

- **Listener**: abstrae la interfaz de comunicación contar el middleware que recibe al instanciarse. Es necesario definir las siguientes funciones:
  - `recv_raw`: recepción de un chunk de datos crudos.
  - `recv_eof`: recepción de un mensaje de finalización
- **Worker**: abstrae la lógica de recepción de un chunk de datos y envío de resultados. Los datos recibidos son deserializados utilizando `in_serializer` y los resultados a enviar son serializados utilizando `out_serializer`. Es necesario definir las siguientes funciones:
  - `work`: el trabajo a realizar sobre un único dato del chunk
  - `do_after_work`: define qué se hace luego de haber realizado `work` sobre todo un chunk

4. Vista de Procesos

A continuación se presentan distintos escenarios y como se maneja el flujo de mensajes a través del sistema.

4.1. Flujo de la consulta 1

En el siguiente diagrama se presenta como un batch de libros es enviado para las consulta 1. Se puede observar cómo todos los libros seleccionados por tener las características solicitadas son enviados al **ResultHandler**.

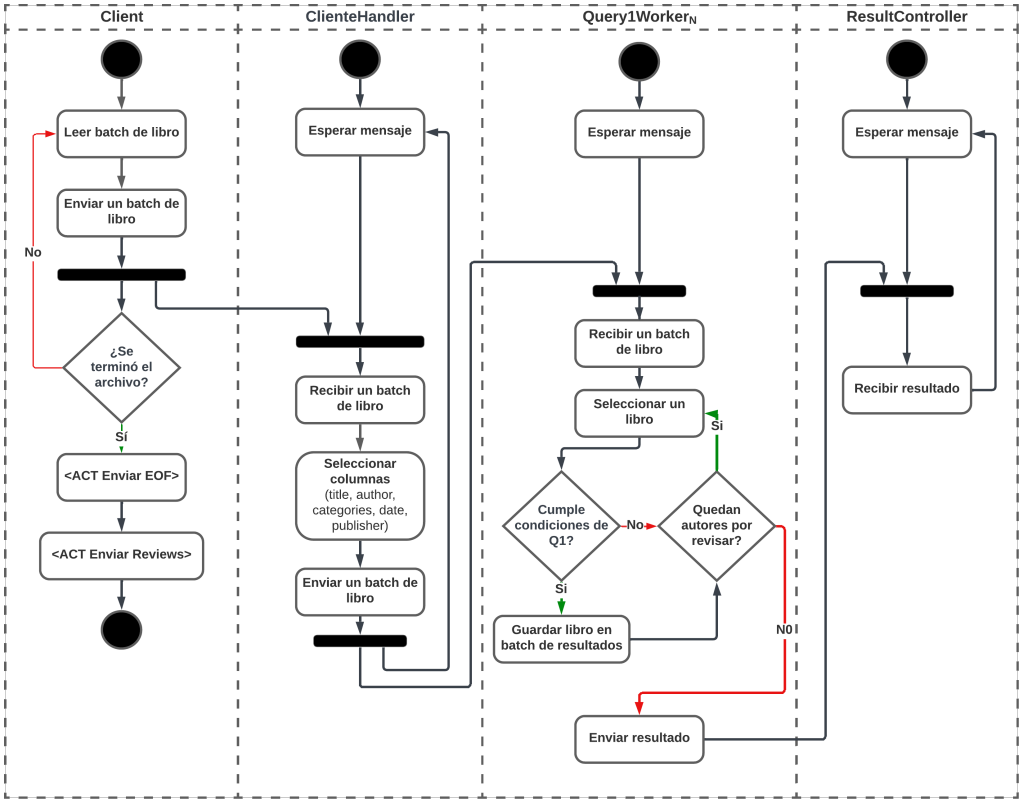

Figura 5: Flujo de mensajes para la Consulta 1

4.2. Flujo para la Consulta 2

En el siguiente diagrama se ve como como funciona el flujo de mensajes para la consulta 2.

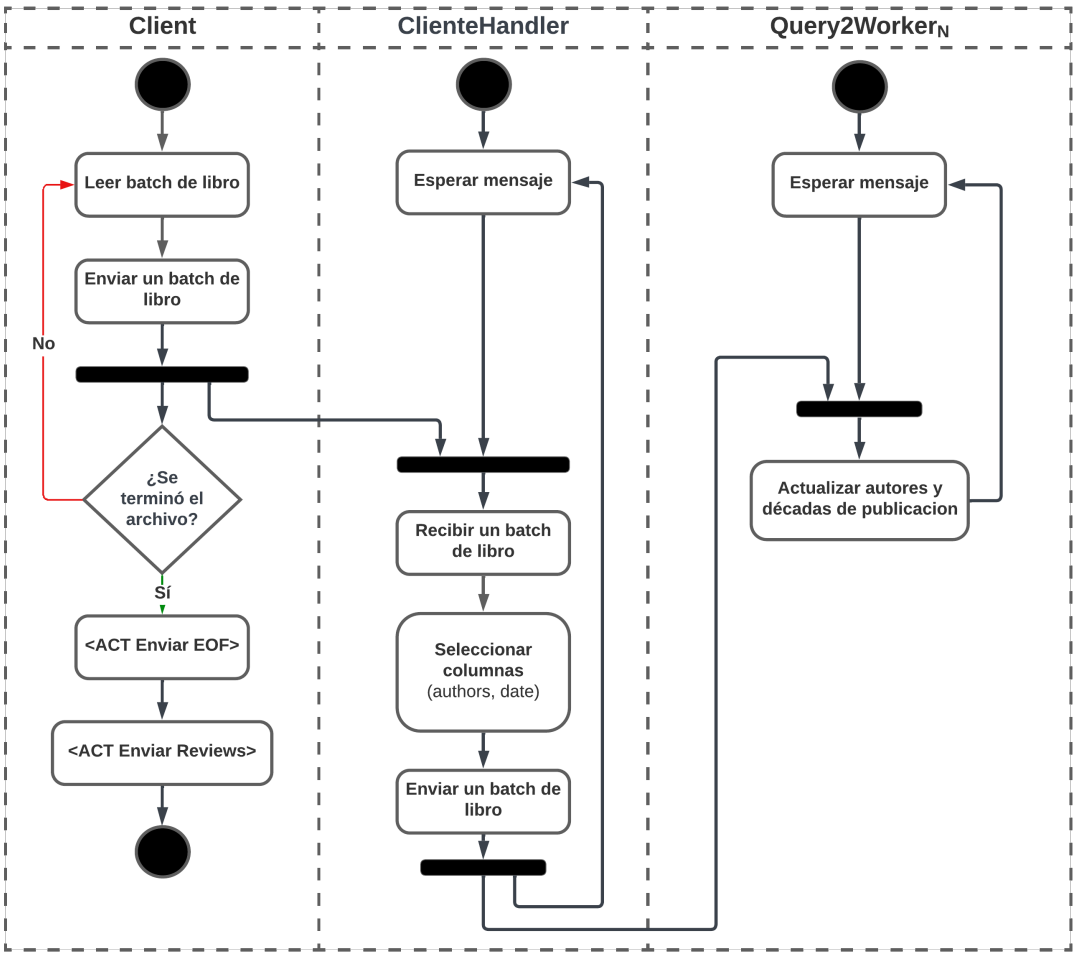

Figura 6: Flujo de mensajes con libros para la Consulta 2

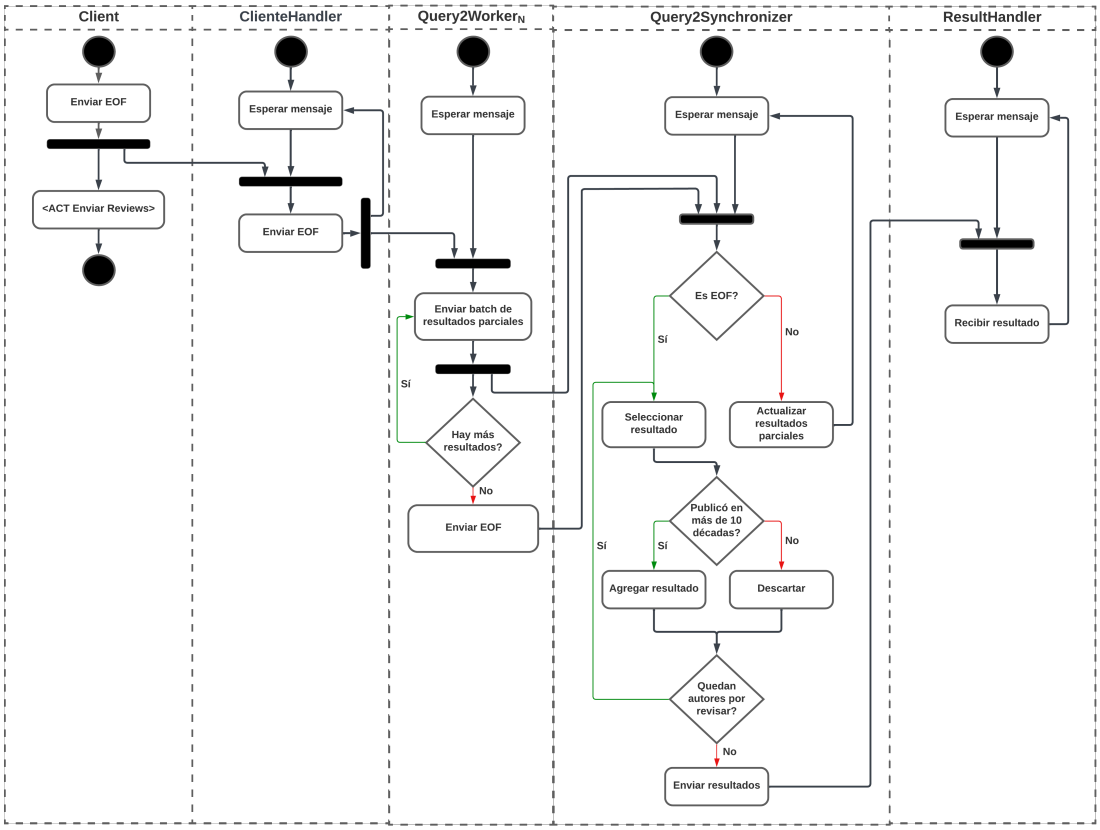

Figura 7: Flujo de mensaje EOF para la Consulta 2

Nota: Las consultas 3, 4 y 5 también hacen uso de N Workers y 1 Synchronizer y por lo tanto los flujos para los mensajes de review serán muy similares al realizado para la consulta 2

4.3. Flujo de mensaje EOF

Además del diagrama de actividad para el mensaje EOF de la Consulta 2, se quiere exponer en más detalle el mecanismo por el cual todos los Workers que están escuchando una misma cola producer/consumer son notificados de que ya no habrán más mensajes.

En este escenario, en el *Chunk* enviado por el cliente hay solo 2 libros. Luego de enviarlo el cliente envía un EOF, notificando que ya no enviará más libros. El mensaje de EOF contiene un parámetro especial (*peers*), que representa la cantidad de workers que recibieron dicho mensaje. Este parámetro es inicializado en cero por el ***ClienteHandler*** antes de ser enviado al middleware.

Cuando un ***Worker*** recibe dicho mensaje pueden ocurrir dos cosas:

1. Si aún quedan peers por ser informados, envía el EOF habiendo sumado 1 al contador.
2. Si él es el último, entonces informa al ***Synchronizer*** que ya no habrán más resultados a juntar.

En este caso ***Worker 1*** toma el *(EOF, 0)*. Como tiene información parcial de los resultados, primero envía dicha información al sincronizador y luego envía *(EOF, 1)* a ***Worker 2***. Como también tiene información parcial, la envía al sincronizador y por último al observar que es el último worker, envía (EOF, 0) al sincronizador para informar que ya no habrán más resultados parciales a juntar.

Por último, como se ve en la figura, cuando ***Synchronizer*** obtiene resultados parciales hace *"merge"* de dichos resultados con los que ya almacenó y cuando recibe un *EOF* publica dichos resultados con la tag *Qx*.

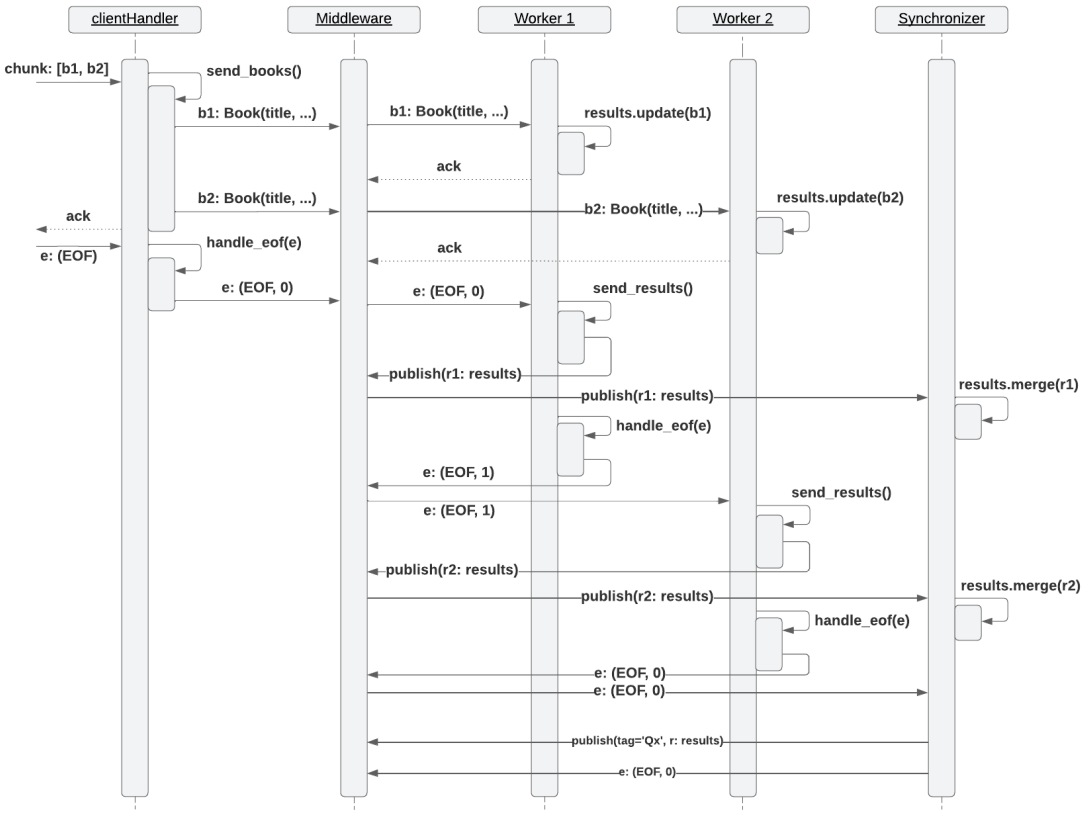

Figura 8: Diagrama de secuencia EOF

5. Vista de Desarrollo

5.1. Diagrama de paquetes

Inicialmente se presenta un diagrama de paquetes para observar la estructura del repositorio.

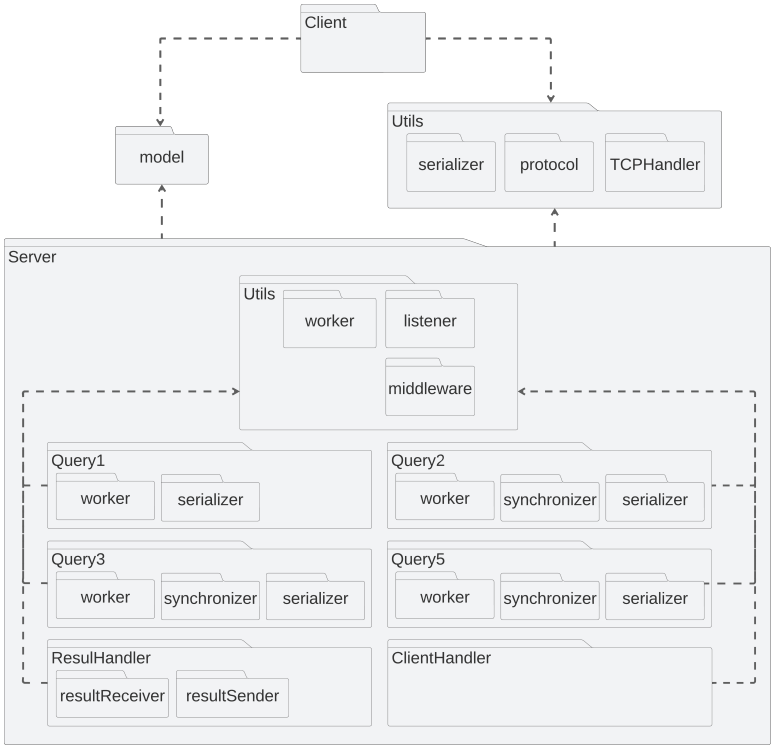

Figura 9: Diagrama de paquetes

5.2. Decisiones de diseño

5.2.1. Protocolo de comunicación

Tanto para la comunicación entre el sistema y el cliente, como para la comunicación dentro del sistema. Desarrollamos nuestro propio protocolo siguiendo una estructura TLV para estructurar los mensajes.

5.2.2. Patrones de diseño

Para implementar la consulta 2, 3, 4 y 5, aplicamos un mismo patrón. Donde los libros o reseñas llegan a un único punto de entrada, y este se encargar de distribuir la carga entre los distintos trabajadores. Luego todos estos resultados parciales se juntan en un único punto, que sera el encargado de enviar los resultados a la parte del sistema que se encarga de guardarlos.

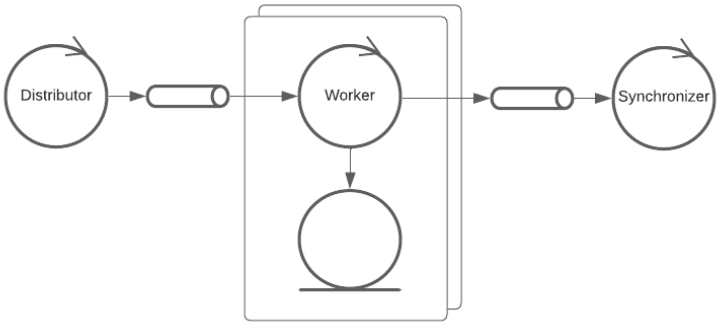

Figura 10: Patrón Distributor-Worker-Synchronizer

6. Vista de Física

6.1. Diagrama de Robustez

A continuación se muestra el diagrama de robustez divido para los distintos flujos dentro del sistema.

Consulta 1

En primer lugar se presenta la sección del sistema que incumbe a las consulta 1 (Título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan "distributed" en su título). Como se puede observar, un **Query1Worker** se encarga de recibir libros que envia el **clienteHandler** y solo envia al **ResultHandler** aquellos que cumplen con la condición.

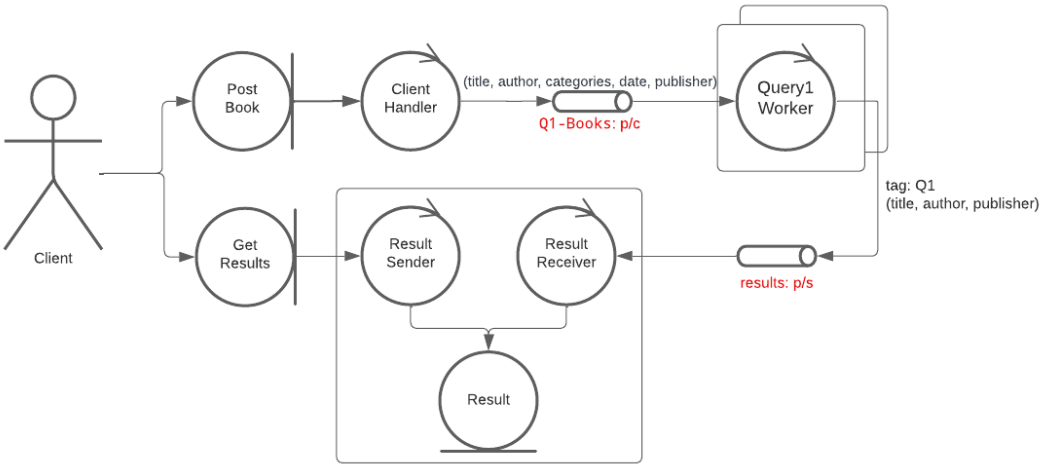

Figura 11: Diagrama de Robustez Q1.

Consulta 2

El segundo caso que presentamos es el de la consulta 2 (Autores con títulos publicados en al menos 10 décadas distintas). En esta ocasión, el procesamiento se realiza en dos partes. Inicialmente se distribuye el trabajo de recolectar todos los autores y las décadas en las que publicaron algún libro entre distintos **Query2Worker**. Luego, se envían estos resultados parciales a un sincronizado

que se encarga de juntar la información y enviar como resultados aquellos que tengan libros publicados en mas de 10 décadas distintas.

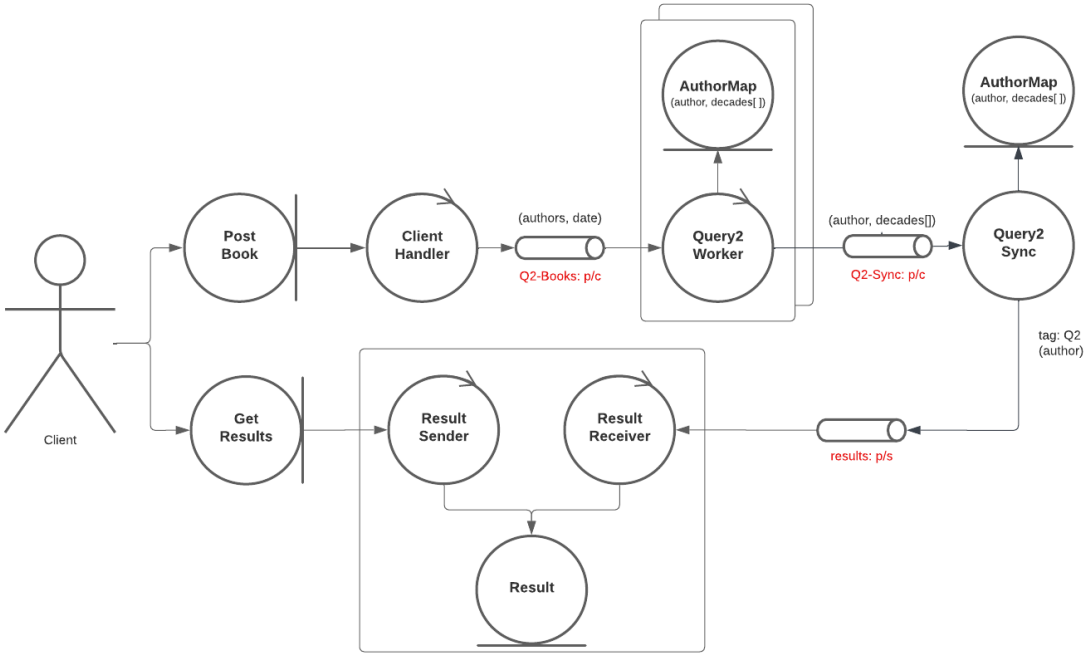

Figura 12: Diagrama de Robustez Q2.

Nota: Se decidió guardar el mapa ***AuthorMap*** en memoria ya que, asumiendo que los autores escriben durante 5 décadas diferentes habrá un total de 273.000 entradas ($210000 \cdot 1{,}30$) de en promedio 35 bytes cada una ($15b + 4b \cdot 5$). Por lo que la cota será de 10Mb

Consultas 3 y 4

A continuación se presenta las consultas 3 (Títulos y autores de libros publicados en los 90' con al menos 500 reseñas) y 4 (Libros con mejor rating promedio entre aquellos publicados en los 90' con al menos 500 reseñas). Para ambos consultas es necesario utilizar tanto la tabla de reseñas, como la de libros. Inicialmente se envían a todos los **Query3Handler** todos los libros por una cola publisher/subcriber. Los libros publicados en los 90' son almacenados y el resto son descartados. Una vez se enviaron todos los libros, se comenzaran a enviar todas las reseñas y cada worker se encargara de actualizar la cantidad de reseñas y el rating para cada libro. Por ultimo, se envían estos resultados parciales a un sincronizado que se encarga de juntar la información y enviarla.

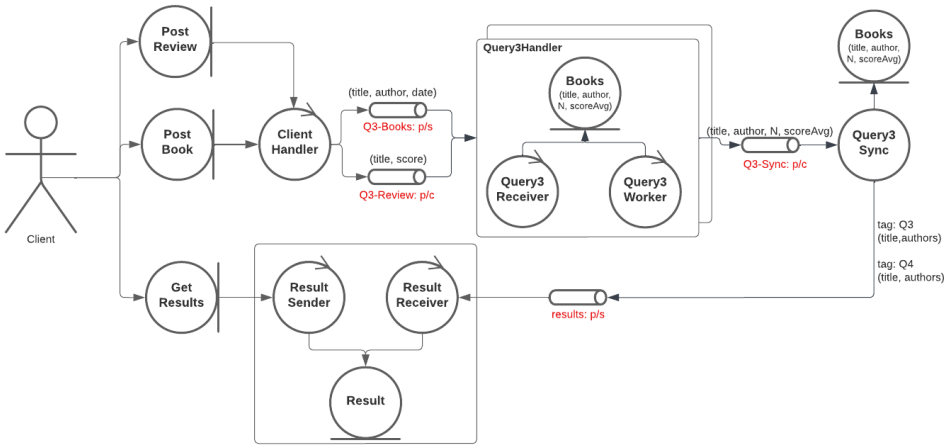

Figura 13: Diagrama de Robustez Q3 y Q4.

Nota: Se decidió guardar el mapa ***Books*** en memoria ya que tendrá un total de 210000 entradas de en promedio 77,5 bytes cada una ($50b + 1{,}3 \cdot 15b + 4b + 4b$). Por lo que la cota será de 15,5Mb

Consulta 5

Por ultimo, se presenta la consulta 5 (Títulos en categoría "Fiction" cuyo sentimiento de reseña promedio esté en el percentil 90 más alto). Al igual que en las consultas anteriores el procesamiento debe realizarse en dos partes. Inicialmente los **Query5Handler** recibe todos los libros y almacena solo los que son del genero ficción. Luego para cada reseña de alguno de los libros de ficción, se calcula el sentimiento promedio. Por ultimo se envían estos resultados parciales a un sincronizado que se encarga de juntar la información y obtener el 90 percentil mas alto, para luego enviar los resultados.

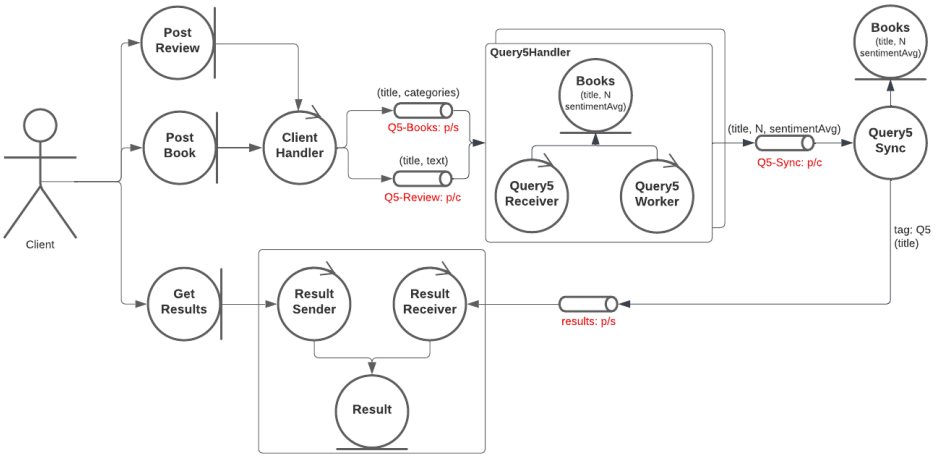

Figura 14: Diagrama de Robustez Q5.

Nota: Se decidió guardar el mapa ***Books*** en memoria ya que tendrá un total de 210000 entradas de en promedio 58 bytes cada una ($50b+4b+4b$).

Por lo que la cota será de 11,6Mb

6.2. Diagrama de Despliegue

A continuación se muestran las aplicaciones que se despliegan en el sistema. Como podemos observar, todas ellas se comunican a través del middlerware. Esto quiere decir que el middlerware es un punto único de falla, pero que los distintos componentes no depende entre si.

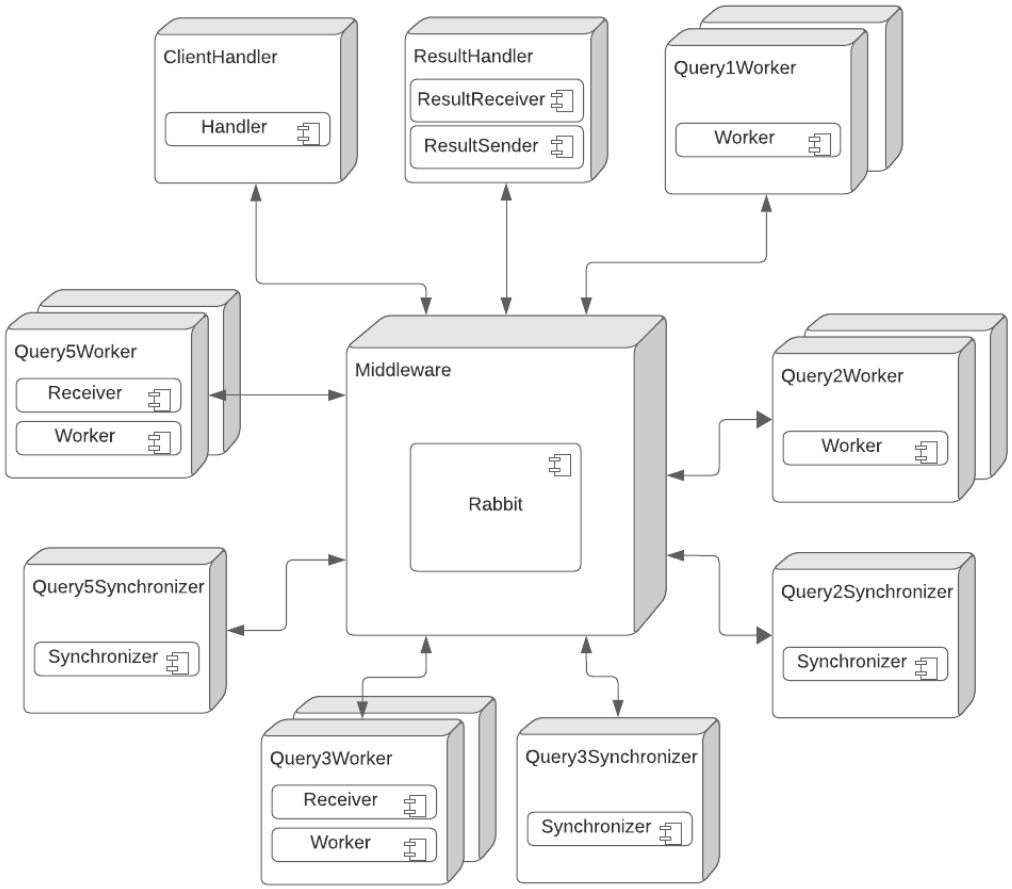

Figura 15: Diagrama de despliegue