Trabajo Práctico - Escalabilidad

Integrantes:
Erick Martinez - 103745
Miguel Angel Vasquez Jimenez - 107378

Sistemas Distribuidos
75.74

1C 2024

Introducción

Se presenta a continuación la documentación que cubre la solución del trabajo práctico de Escalabilidad para la materia Sistemas Distribuidos.

El mismo consiste en el desarrollo de Distributed Books Analyzer, un sistema distribuido escalable que permite realizar procesamiento de una extensa cantidad de datos, provenientes de datasets de libros y reseñas publicados en la plataforma de Amazon, con fines de análisis y aprovechamiento de multicomputing para la realización de consultas al sistema, con el objetivo de proponer campañas de marketing según los resultados obtenidos.

El sistema responde a las siguientes consultas:

- Título, autores y editoriales de los libros de categoría "Computers" entre 2000 y 2023 que contengan 'distributed' en su título
- Autores que tengan títulos publicados en al menos 10 décadas distintas
- Títulos y autores de los libros publicados en la década del 90' con al menos 500 calificaciones
- Top 10 libros con mejor promedio de calificación entre aquellos publicados en la década del 90’ con al menos 500 calificaciones
- Título de libros del género "Ficción" cuyo sentimiento de reseña promedio está en el percentil 90 más alto

Hipótesis / Supuestos

Se toman en cuenta los siguientes:

- El proceso servidor es el punto de entrada y salida del sistema distribuido.
- La ingesta de datos se simula por medio de un proceso denominado cliente, al cual se le da acceso a los set de datos en formato csv. Dicho proceso solo se encarga de leer y enviar los datos al sistema para su procesamiento, y posteriormente esperar los resultados de sus consultas, completando así su ciclo. Como el mismo cumple el rol de fuente de scraping para el servidor, el mismo no pierde la conexión durante el ciclo mencionado.
- El sistema no permite integrar nodos nuevos una vez que ya está en marcha dado que requiere un debido ordenamiento sincronizado para su inicialización.
- El middleware RabbitMQ no termina su ejecución ni corta su conexión con los nodos, exceptuando cuando se da de baja el sistema distribuido.
- El cliente puede volver a ejecutarse de forma externa al sistema distribuido, pero solo cuando ya hubiese terminado todo un ciclo de envío de consultas y recepción de resultados.

Ejecución

Se brinda un Makefile donde se definen los targets principales para la ejecución y monitoreo del sistema. A su vez se provee de un script `create-compose.sh` que parametriza la cantidad de workers para los procesos que pueden ser ejecutados en múltiples instancias independientes, el cual sobreescribe el archivo `docker-compose.yaml` apropiadamente.
Se pueden encontrar los targets relevantes en el archivo README.md del repositorio.

Comunicaciones

A continuación se detallan decisiones relacionadas a la comunicación entre los procesos del sistema distribuido.

Se decidió utilizar un protocolo de texto siguiendo un formato similar a los mismos archivos csv. Esto aplica tanto para la comunicación cliente-servidor como para la comunicación general entre procesos que utilizan el middleware.

Los mensajes pueden llegar en dos formatos, según contengan datos a procesar representados como líneas/filas de valores separados por comas (con ciertas restricciones y transformaciones aplicadas durante el preprocesado), o el mensaje de finalización (EOF).

En cuanto a los mensajes con datos a procesar, los mismos pueden venir en formato de batch o de forma independiente como un solo registro. Para hacer que esto fuese consistente se decidió que sólo los procesos encargados de despachar grandes cantidades de registros de la forma más rápida posible sean los que reciban y envíen exclusivamente batches de datos. Estos procesos son: sanitizadores, preprocesadores (los cuales reciben todos los datos iniciales, siendo los que mayor cantidad de datos tienen que despachar para que el resto del sistema distribuido pueda tener aprovechamiento de sus recursos), y los nodos de merging entre reseñas y libros.

De esta manera los procesos que requieren hacer una evaluación más fina con datos reducidos pueden encargarse exclusivamente de procesar registros individuales sin importar la cantidad de mensajes que reciban por ser ligeros.

Ahora, en cuanto a las restricciones y transformaciones aplicadas durante el preprocesado, se hace una ligera modificación sobre los datos, particularmente sobre los que deben ser interpretados como strings y listas. El formato por defecto aplicado en los archivos de csv para listas define la posibilidad de intercalar caracteres especiales de encapsulamiento de strings (comillas dobles y simples) entre los datos en sí. Esto no supone un problema, pero para hacer que todos los mensajes enviados en nuestro sistema sean consistentes con el formato que definimos se hacen leves modificaciones sobre los datos durante el preprocesado y sanitización. De esta forma los procesos subsecuentes pueden adherirse al protocolo definido para el sistema.

Luego, enfocándose en la comunicación general de los procesos con el middleware RabbitMQ, el envío de ACKs hacia el mismo se realiza de forma manual para asegurar que se pudo hacer el procesado y envío al siguiente nodo antes de confirmar que se puede remover un mensaje de forma segura de una cola.

Decisiones de Implementación de la Capa de Middleware

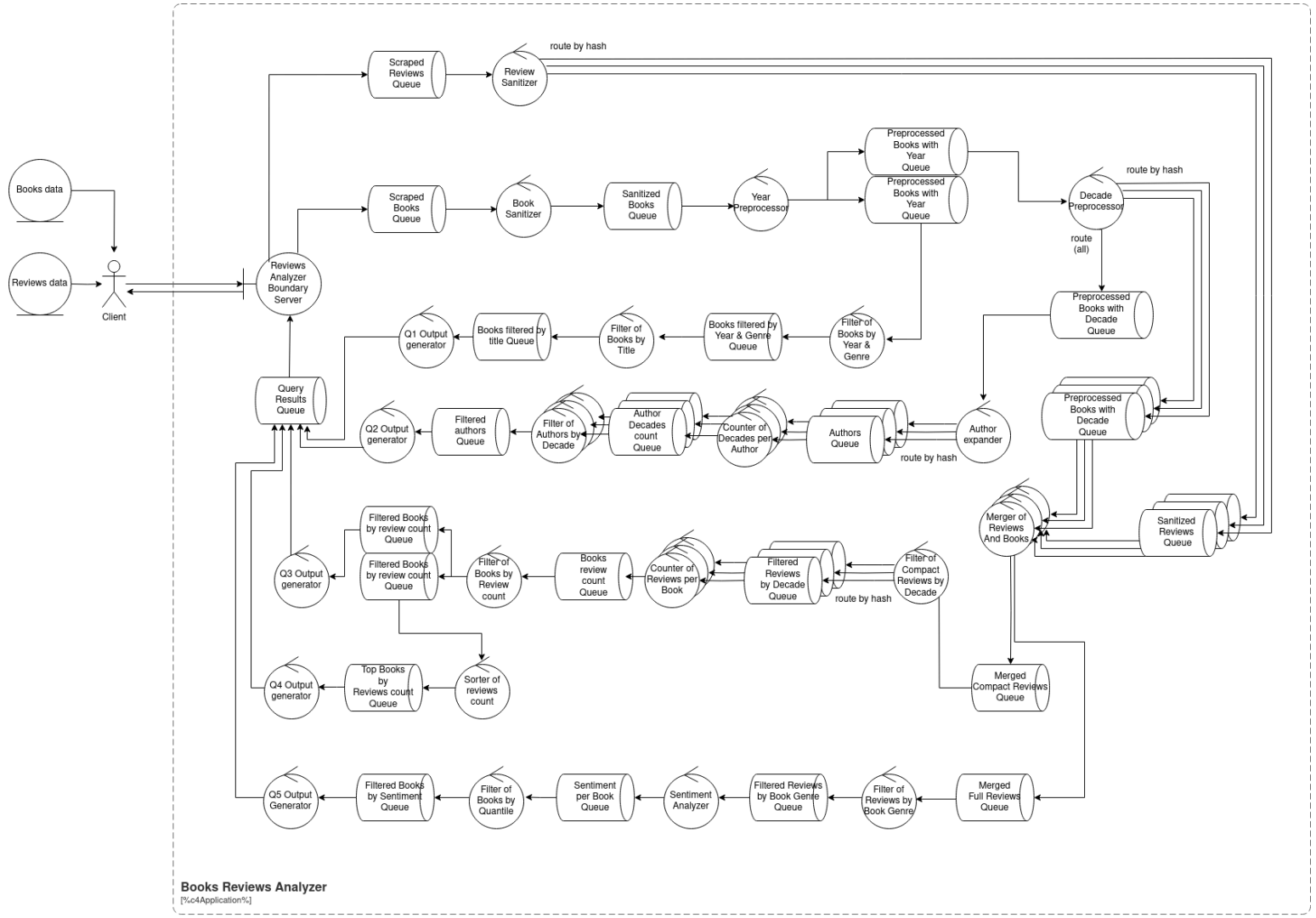
Se implementó de forma común la capa encargada de inicialización de conexiones, envío y recepción de mensajes del middleware. Esta capa provee una interfaz para abstraer a los distintos procesos sobre la declaración de colas y exchanges, soportando múltiples topologías de las mismas dependiendo de lo requerido por cada proceso.
Para hacer consistentes dichas definiciones se decidió declarar todo exchange como de tipo directo (ya que no imposibilita el uso como fan-out para el caso que lo requiriese). Como cada uno de los procesos se comporta (en su gran mayoría) tanto como consumidor como productor de mensajes, se tomó la decisión de declarar los bindings de colas desde el lado del proceso que toma el rol de productor de otro. De esta manera se evita cualquier pérdida de mensajes que puede ocurrir de no hacerse de esta forma, dada la naturaleza concurrente del sistema al inicializar el mismo.

Vistas

Nota: Todos los diagramas se pueden observar con mejor claridad en el directorio **misc/** del repositorio.

Vista Física

- Diagrama de Robustez



Se pueden observar los 23 procesos y los flujos principales que componen el sistema.
En particular los controladores Filter of Author per Decade; Counter of Decades per Author: Counter of Reviews per Book y Merger of Reviews and Books son escalables ya que sus procesamientos son independientes y ayudan a reducir considerablemente la carga de mensajes entre ellos.

Se podría escalar de igual manera el sentiment analyzer y enviar los mensajes hasheando por título del libro al igual que alguno de los filtros que se pueden observar en el diagrama, pero se decidió no agregar complejidad extra debido a que la cantidad de reviews es lo más que conlleva más carga al sistema en comparación a la cantidad de libros. Pero de quererse escalar se podría hacer simplemente incluyendo el envío de las reviews merengadas por hash, manejando los múltiples “EOF” y levantando múltiples procesos en el docker-compse.

De igual manera el “Filter of Books by Title”, “Filter of Compact Reviews by Decade”, “Filter of Reviews by Book Genre” se decidieron implementar con un único proceso, pero igualmente podrían ser escalados con las consideraciones anteriores y de igual forma que se escaló el “Filter of Authors by Decade”. Se decidió esto ya que son procesos simples y muy rápidos, que solo reciben mensajes y aplican un filtro simple que es enviado al siguiente proceso.

Observaciones generales y posibles mejoras

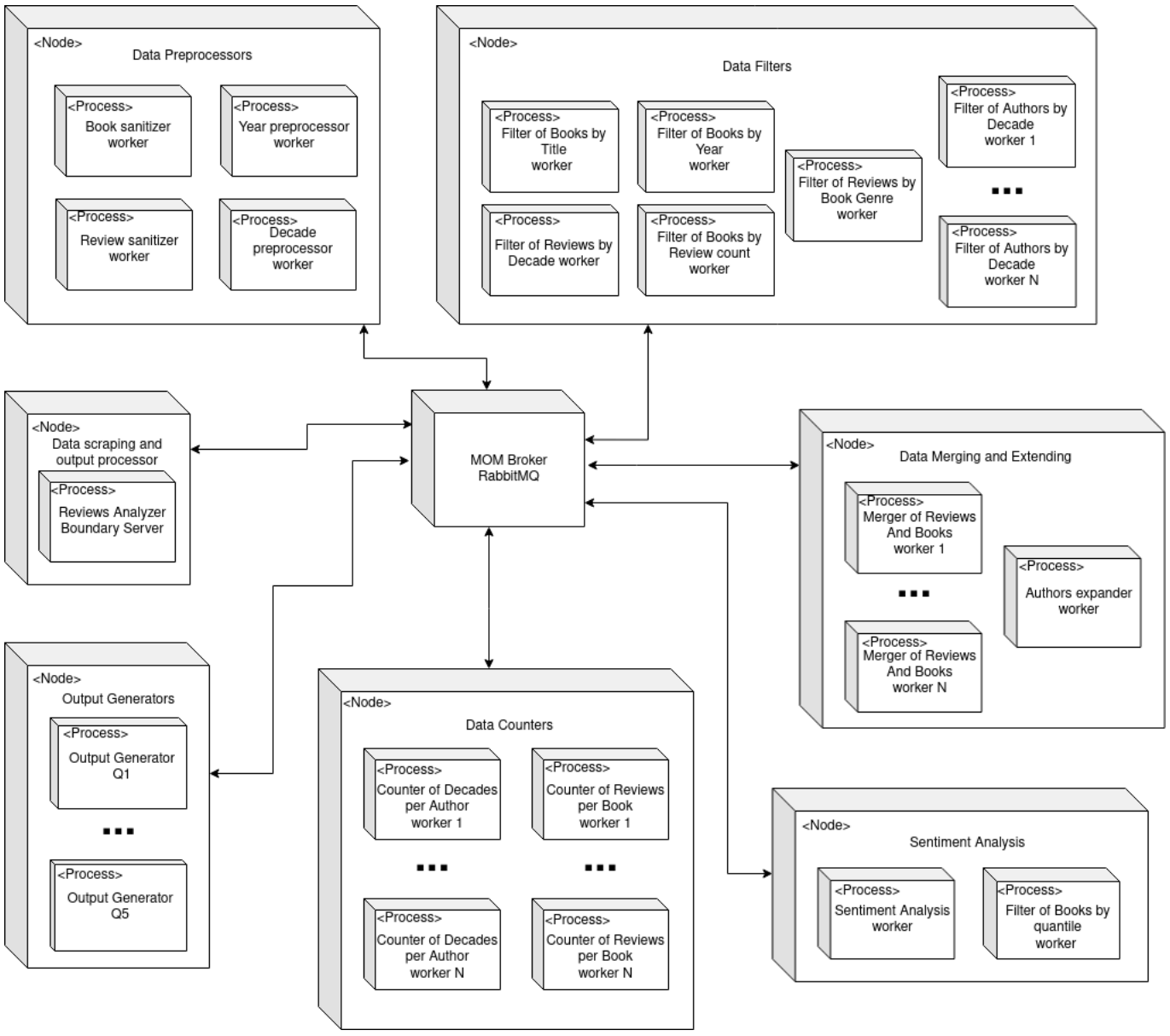
Existen algunos procesos que si bien no lo son actualmente, pueden perfectamente ser modificados para que sean escalables dada su naturaleza independiente. Este es el caso del Sentiment Analyzer y su predecesor Filter of Reviews by Book Genre, asi como el Filter of Compact Reviews by Decade; donde el primero en particular se puede ver más afectado dado al cómputo que tiene que realizar al calcular polaridades en contraste a la alta tasa de mensajes que tiene que recibir como un solo proceso. Algo no tan evidente pasa tambien en el proceso Author expander, que a su vez se podría llegar a hacer escalable para reducir un poco su carga.

Para aplicar estos cambios a cualquiera de los ejemplos mencionados, se tendría que adaptar el script generador de compose para que añada dinámicamente colas adicionales y workers de la misma forma que como los procesos que ya son escalables. Los productores de dichos procesos tendrían que hashear por título de libro para seleccionar la cola de destino, efectuando asi la independencia entre procesos.

Para el resto de procesos del diagrama no se considera mucho la posibilidad de adaptarlos a escalables, bien sea porque dependen del uso de todos los datos finales en conjunto (Query Result Generators y Filters finales que recopilan datos) o porque realmente no tendrían mucho beneficio al ser escalables (como los preprocessors, que si bien tienen que handlear mayor parte de la data, actúan como un filtro simple y no se observa ninguna congestión en las colas, se tiene

buen balance entre rate de consumo y rate de envío postprocesado. Sumado a que no se podría garantizar tan facil la consistencia del hashing por la incerteza de los datos puros sin sanitizar)

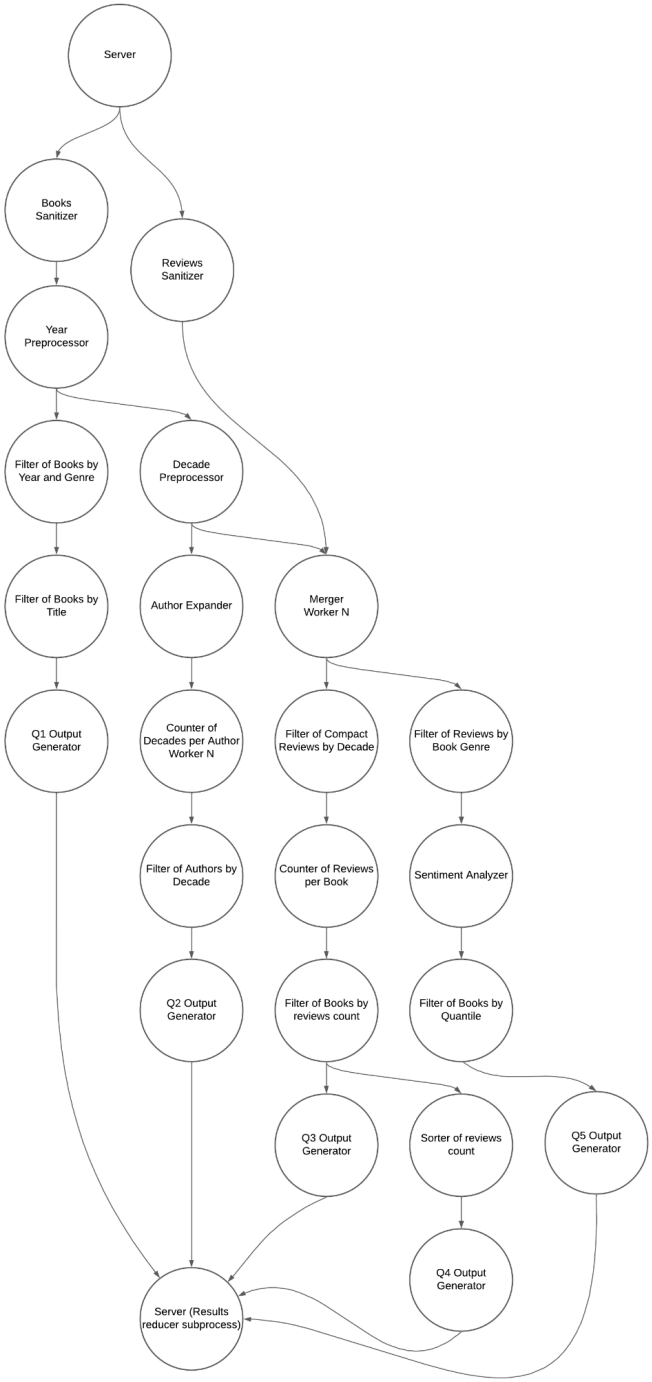
- Diagrama de Despliegue



En el diagrama se observa la relación entre los distintos nodos que contendrían a los procesos, cada uno en su contenedor. Cabe destacar como los procesos que realizan un uso intensivo de cómputo se definieron por separado, así como son los procesos relacionados a análisis de sentimiento. También se destaca el nodo

de Data Merging and Expanding en este mismo aspecto, por la alta cantidad de datos que tienen que procesar.

- DAG

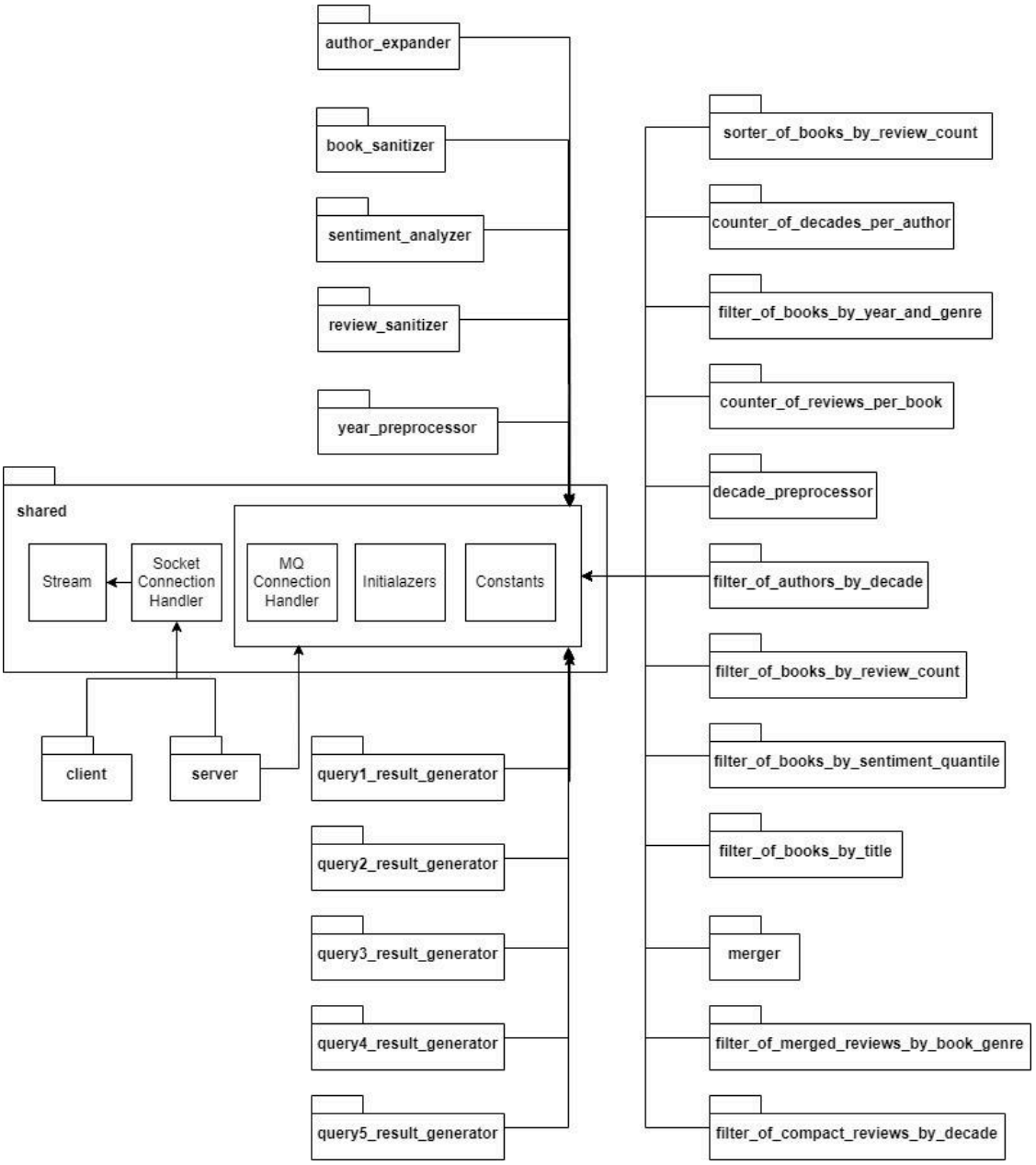


Se puede visualizar el grafo acíclico dirigido del sistema, representando los flujos de datos que se dan a lo largo del mismo. Particularmente se destaca el comportamiento de:

- [Proceso] Worker N: Son procesos que son independientes entre sí y por lo tanto cada uno realiza el mismo flujo del diagrama.
- Server: Es representado como dos nodos distintos dado que efectivamente el mismo tiene el proceso principal de donde consume los datos del cliente, y un proceso hijo que se encarga de recibir concurrentemente los resultados que pueda ir recibiendo mientras reenvía los datos del cliente hacia el sistema.
- Merger: Es uno de los nodos más importantes dado que se encarga de realizar join de reviews con todos los libros que posea. Esto implica que, como va recibiendo de ambos tipos de mensajes simultáneamente, tiene la capacidad de almacenar temporalmente reviews de las que no tenga a disposición un libro registrado inmediatamente al recibirlas, para ser debidamente procesadas cuando posea más registros de libros.

Vista de Desarrollo

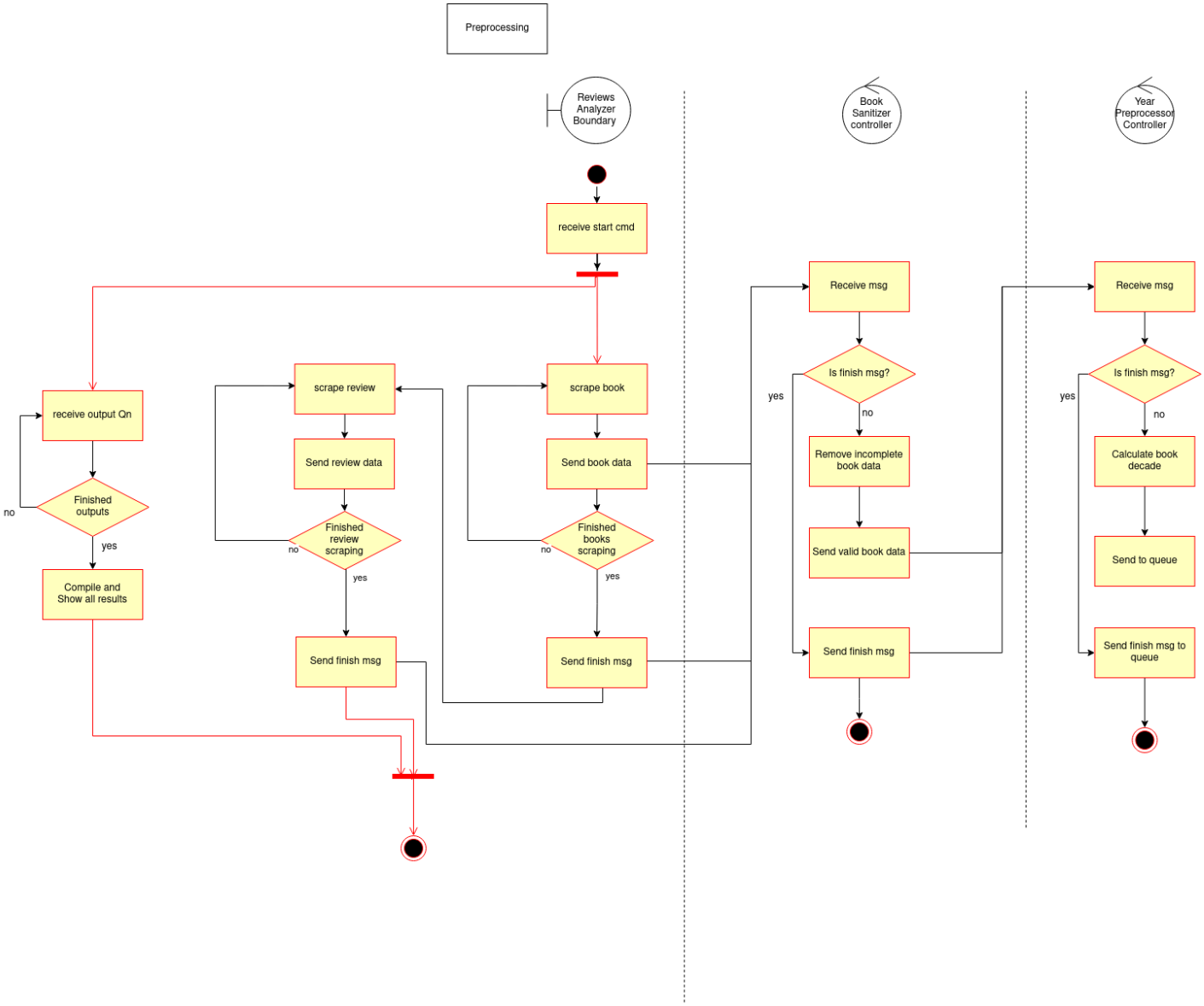
- Diagrama de paquetes



Podemos observar en el diagrama de paquetes que tanto cliente como servidor se comunican a través de sockets utilizando ambos el “Socket connection Handler” Y todos los demás procesos del servidor utilizan el “MQ Connection Handler” para la comunicación a través de las colas de RabbitMQ, siendo esta la capa de middleware común a cada uno de los componentes del servidor distribuido. Además queda compartido la inicialización de configuraciones y logs para todos los procesos.

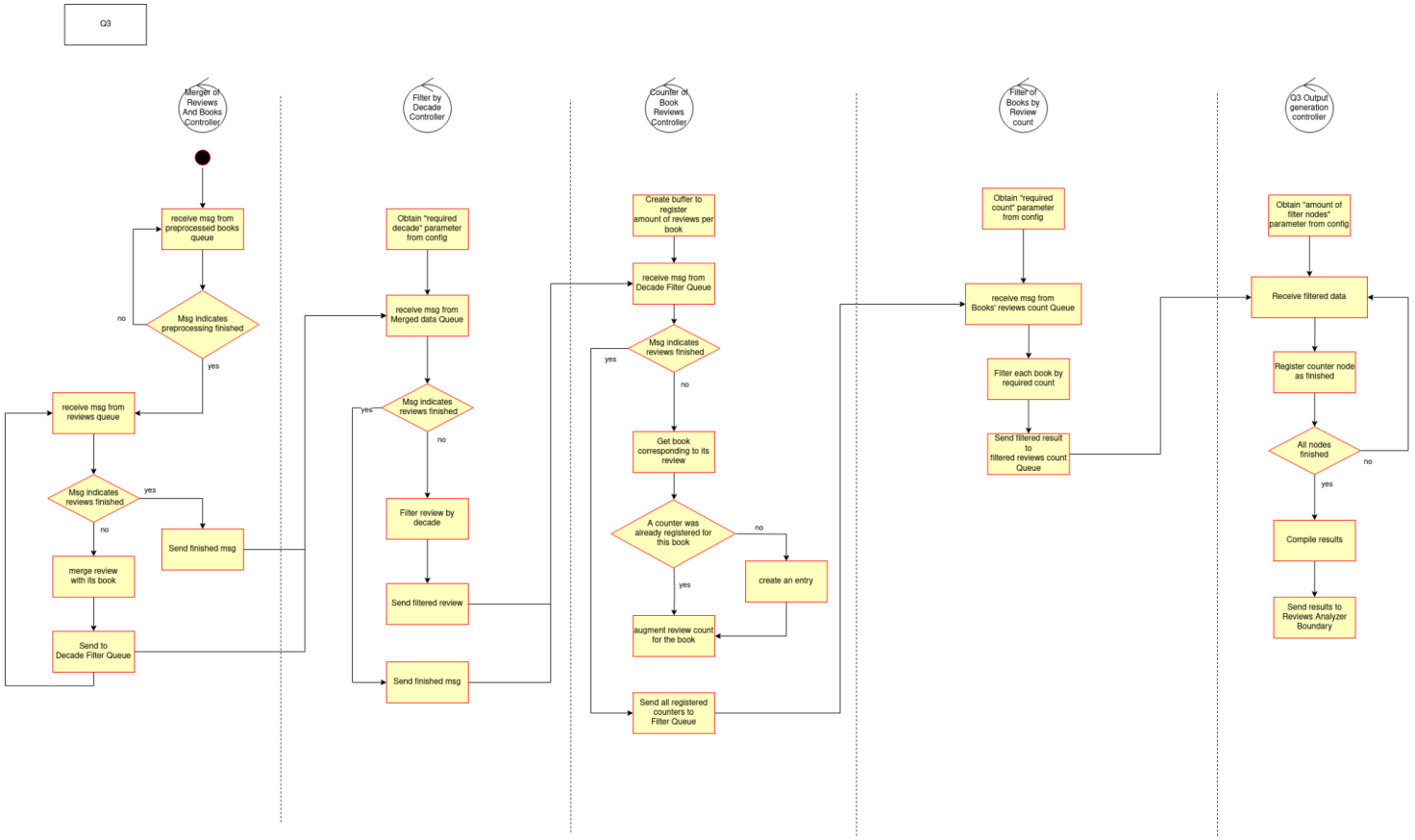
Vista de Procesos

- Diagrama de actividades previas a procesado de datos



Se puede observar el inicio de el preprocesamiento de los datos, en el que el servidor a partir de los datos recibidos por el cliente, estos son reenviados al sanitizador correspondiente, en el que se efectúan limpiezas del formato de los datos, además de un filtrado inicial de las líneas con columnas incompletas indispensables para el procesado correspondiente, además del cálculo de la década de publicación de los libros a partir de las fecha recibidas.
Se puede observar que a partir de este momento se comienzan a cascadear los “EOFs”, necesarios para que los siguientes procesos puedan saber cuando termina su procesamiento respectivo y manejarse según sea el caso de cada uno.

- Diagrama de actividades consulta Q3



Decidimos ejemplificar el procesamiento de la Q3 ya que es la consulta independiente con la mayor cantidad de procesos, a excepción de la Q4, que utiliza output de la Q3 y realiza solo un paso adicional.

Con esto ejemplificamos el procesamiento general de cada consulta, las cuales parten de obtener los datos preprocesados de libros (y reviews a partir de la Q3). En este caso actúa el proceso merger que se encarga de hacer un join simple por título de los datos de cada review con su libro correspondiente. Cuando se completa dicho merge (se recibe el EOF de reviews) se pasan en cascada los EOFs a las colas de los procesos siguientes. Luego tenemos un filtro inicial por década en este caso, para filtrar la mayor cantidad de datos posibles antes de pasar al siguiente proceso, el contador de reviews y finalmente otro filtro para llegar al “Output generator” que se encarga de armar el mensaje que será enviado finalmente al cliente. Este último proceso debe esperar los EOFs correspondientes para poder concluir que se terminó de procesar toda la data y que ya tiene toda la información correcta para responder a la Query.