Trabajo Práctico N°1: Flights Optimizer
Sistemas Distribuidos I (75.74)

Alumno	Padrón	Mail
Axel Kelman	103479	akelman@fi.uba.ar
Juan Cruz Caserío	104927	jcaserio@fi.uba.ar

Introducción

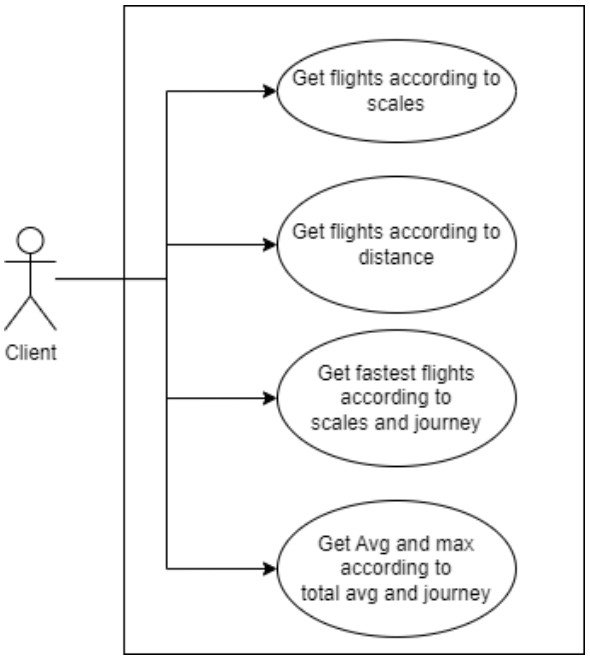
El presente documento tiene como objetivo presentar la arquitectura propuesta para la resolución del Trabajo Práctico N°1: Flights Optimizer. El problema a resolver consiste en diseñar e implementar un sistema distribuido que analice registros de precio de vuelos de avión con el fin de mejorar la oferta a los clientes.

Requisitos funcionales del sistema:

- Se solicita un sistema distribuido que analice 6 meses de registros de precios de vuelos de avión
- Los registros poseen trayectos (aeropuertos origen-destino), tarifa total, distancia total, duración, cada segmento con escalas y aerolíneas.
- Los registros se ingresan progresivamente al ser escaneados de internet.
- ID, trayecto, precio y escalas de vuelos de 3 escalas o más.
- ID, trayecto y distancia total de vuelos cuya distancia total sea mayor a cuatro veces la distancia directa entre puntos origen-destino.
- ID, trayecto, escalas y duración de los 2 vuelos más rápidos para cada trayecto entre todos los vuelos de 3 escalas o más.
- El precio avg y max por trayecto de los vuelos con precio mayor a la media general de precios.

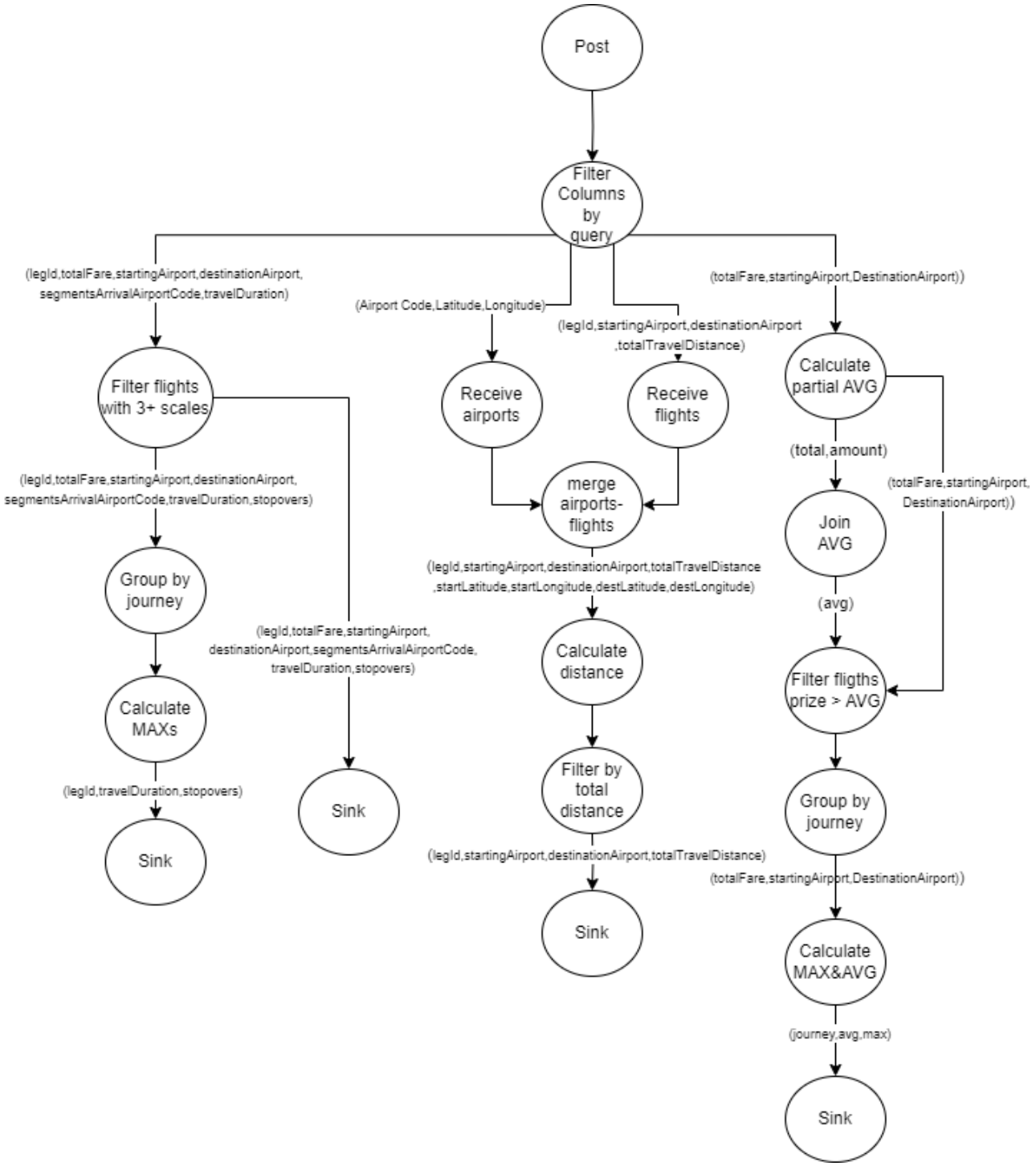
Casos de Uso:

Se espera que el sistema pueda responder a las siguientes casuísticas:



Vista Lógica

DAG:



Vista Física

Diagrama de Robustez:

En este diagrama se pueden visualizar los distintos procesos involucrados y su respectiva comunicación por medio de queues de RabbitMQ (para esto se utilizó la librería Pika de Python).

Para la resolución de las consultas planteamos un patrón de comunicación de tipo pipeline worker por filter, siendo específico según la consulta a resolver. Las distintas etapas del pipeline se resolverán en diferentes nodos, donde algunos tendrán la posibilidad de escalar con el objetivo de lograr el aumento del volumen de datos a procesar.

El cliente enviará batchs de vuelos o aeropuertos por medio de un socket al Post Handler que filtra los campos necesarios para cada query. A partir de este punto se dividen los flujos según corresponda para cada query.

Finalmente, para la salida de la información el sistema escribirá en distintos archivos expuestos al cliente los resultados de cada consulta.

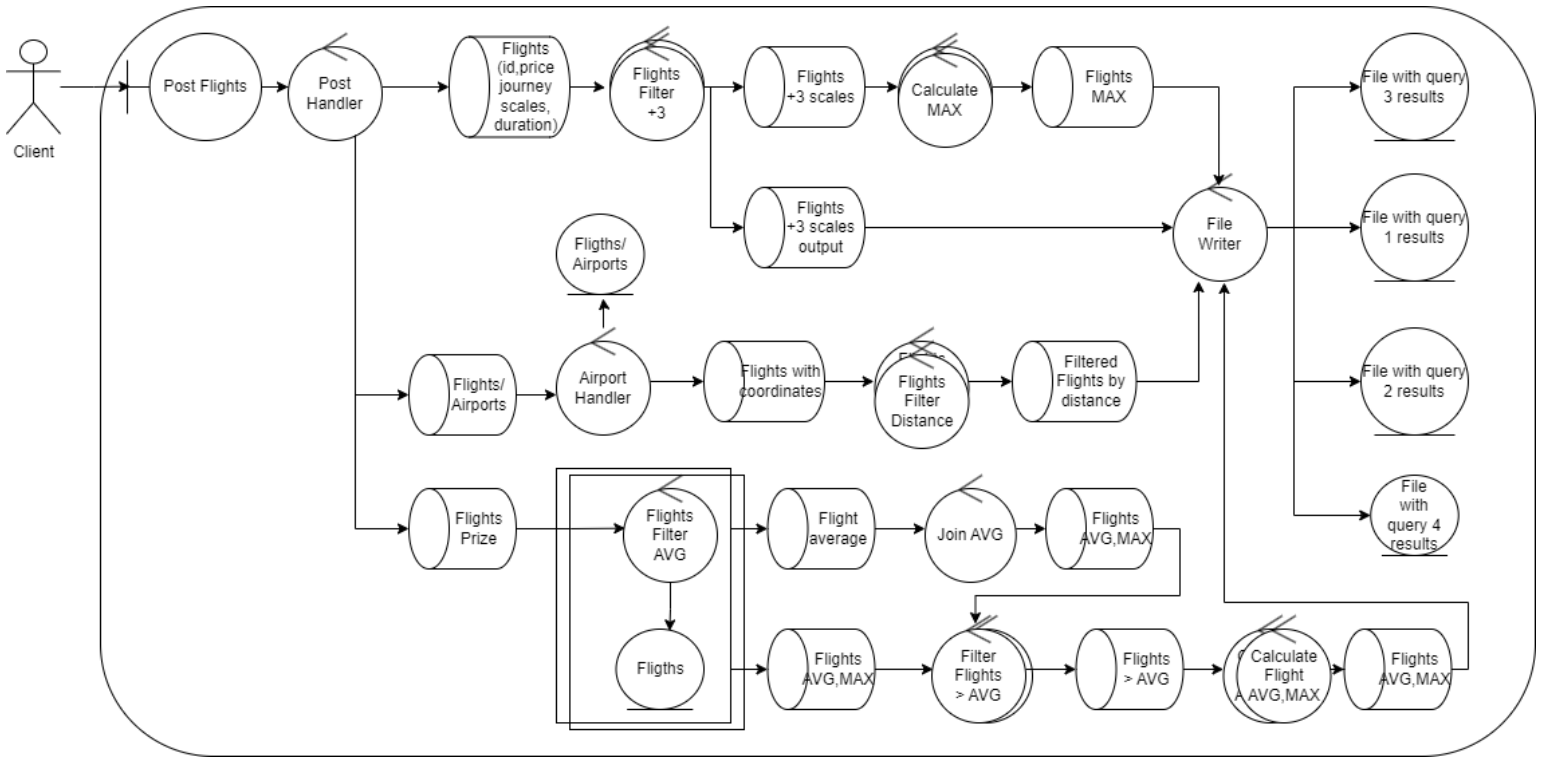
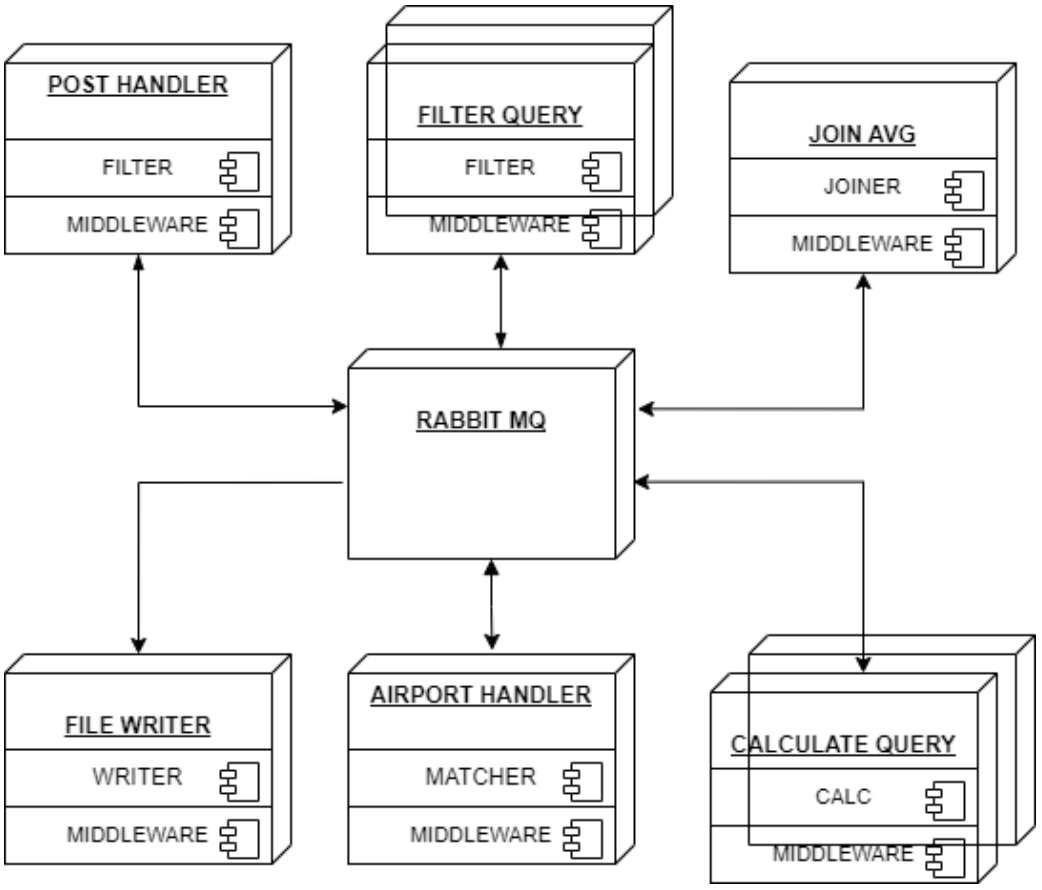


Diagrama de Despliegue:

A continuación se presenta el diagrama de despliegue. Para el mismo se optó por agrupar nodos cuya funcionalidad sea similar para evitar redundancias donde destacan los nodos que filtran y los nodos que realizan cálculos.



Vista de Procesos

Diagrama de Actividad

Se muestra continuación el flujo de todas las queries en diagramas de actividad

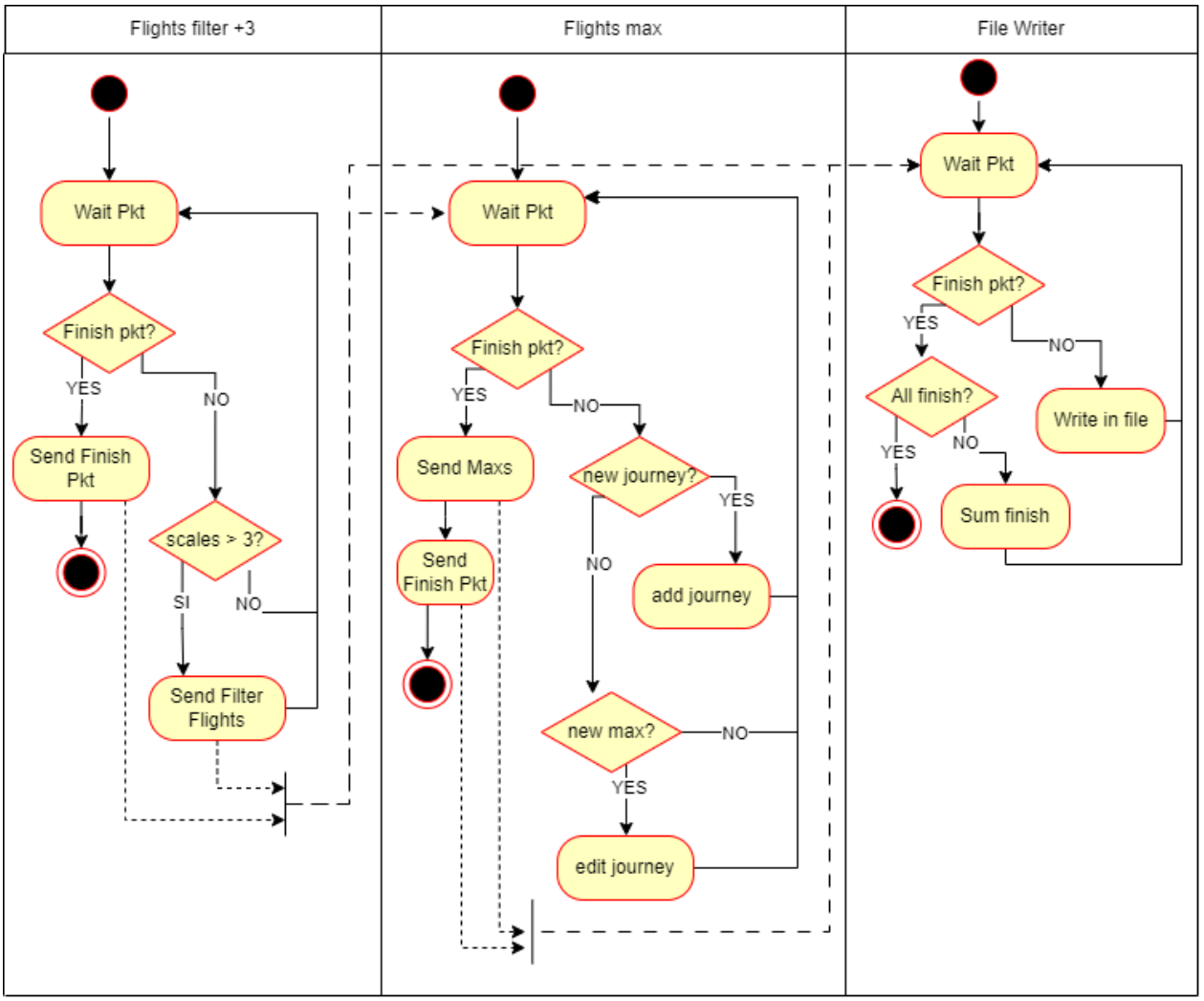


Diagrama 1: Representación del funcionamiento de la query1 y query3

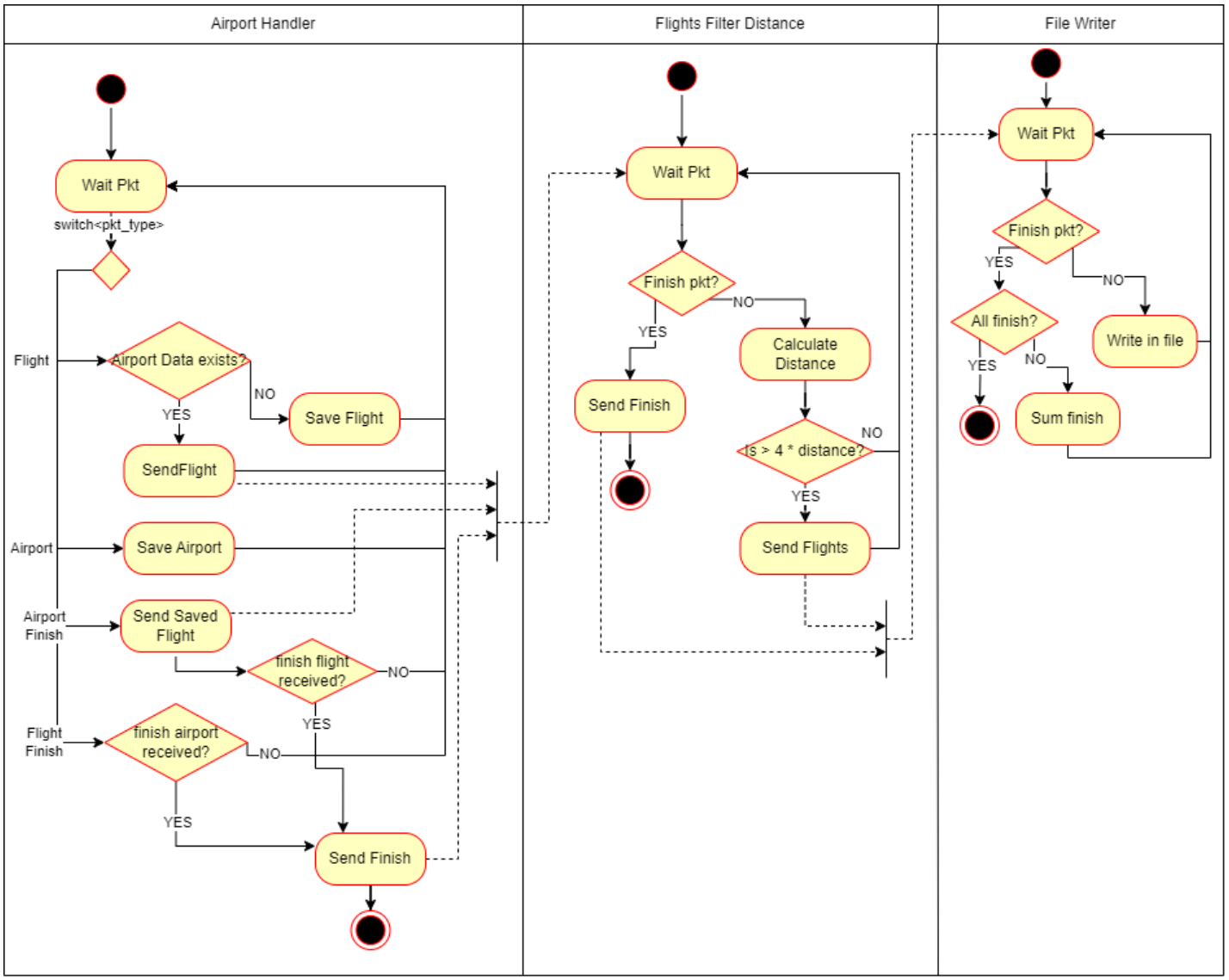


Diagrama 2: Representación del funcionamiento de la query2

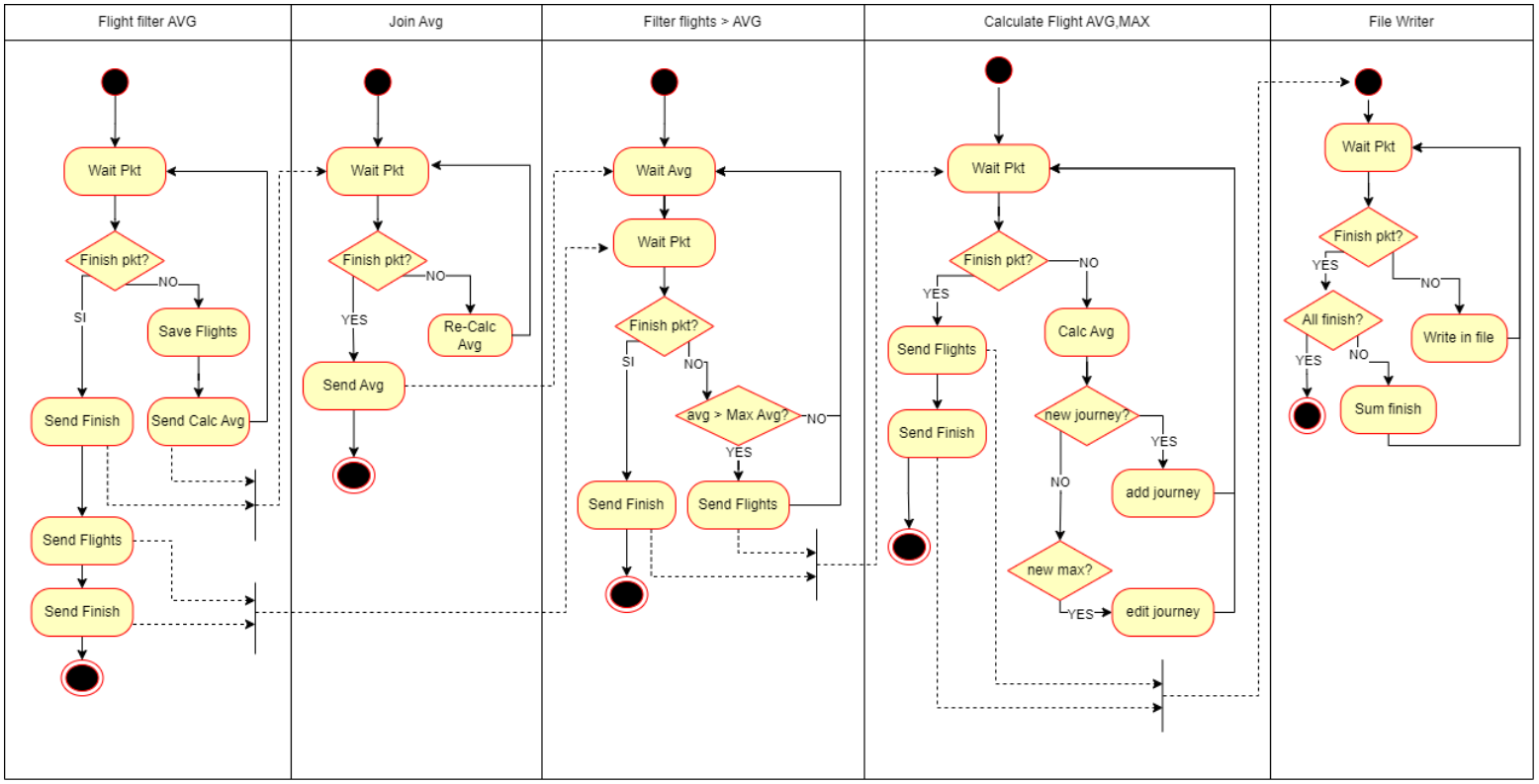


Diagrama 4: Representación del funcionamiento de la query4

Diagramas de Secuencia

Se muestran ahora los diagramas de secuencia con el fin de ejemplificar el funcionamiento al recibir un EOF el sistema. Se ejemplifican los escenarios de finalización del pipeline de la query 2 y 4. En el caso de la finalización de la query 2 se asume que ya se habían recibido previamente todos los aeropuertos necesarios.

Por el lado, de la query 4 se muestra en un primer diagrama la finalización de la primera parte del pipeline, es decir, el momento en que se ha calculado la media general de precios y se puede comenzar a realizar el filtrado y agrupamiento finales. Finalmente, se muestra la finalización de la query 4, una vez que se enviaron todos los vuelos almacenados en memoria.

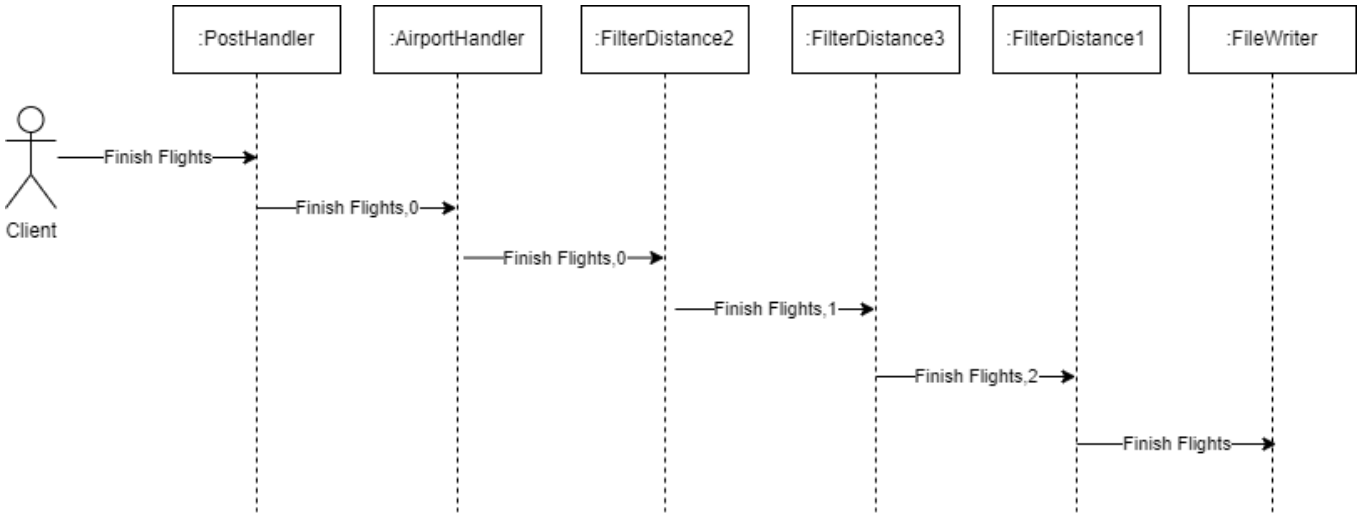


Diagrama 1: Escenario de finalización del pipeline de la query 2

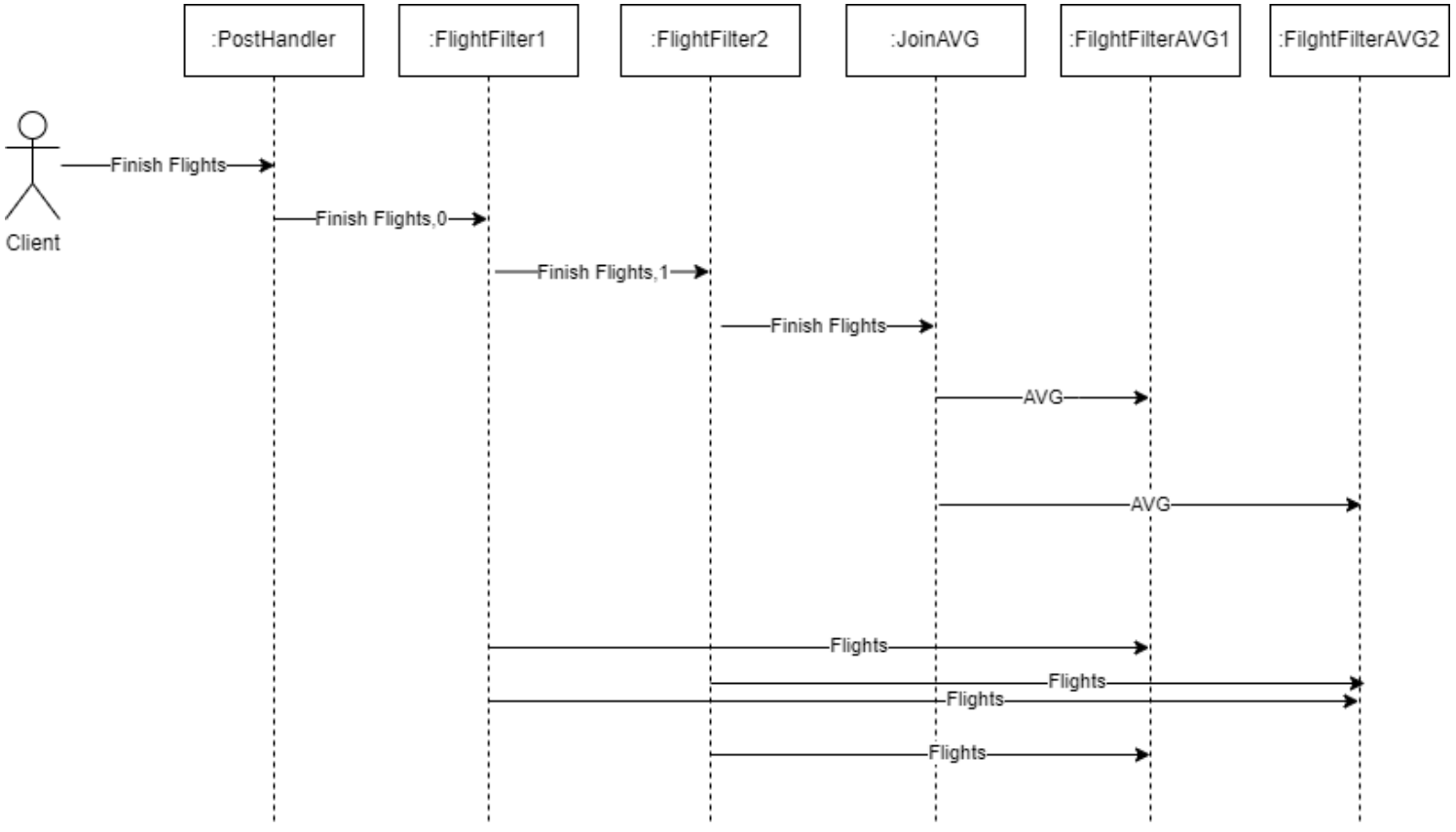


Diagrama 2: Escenario de finalización de primera parte de query 4 y comienzo de segunda parte

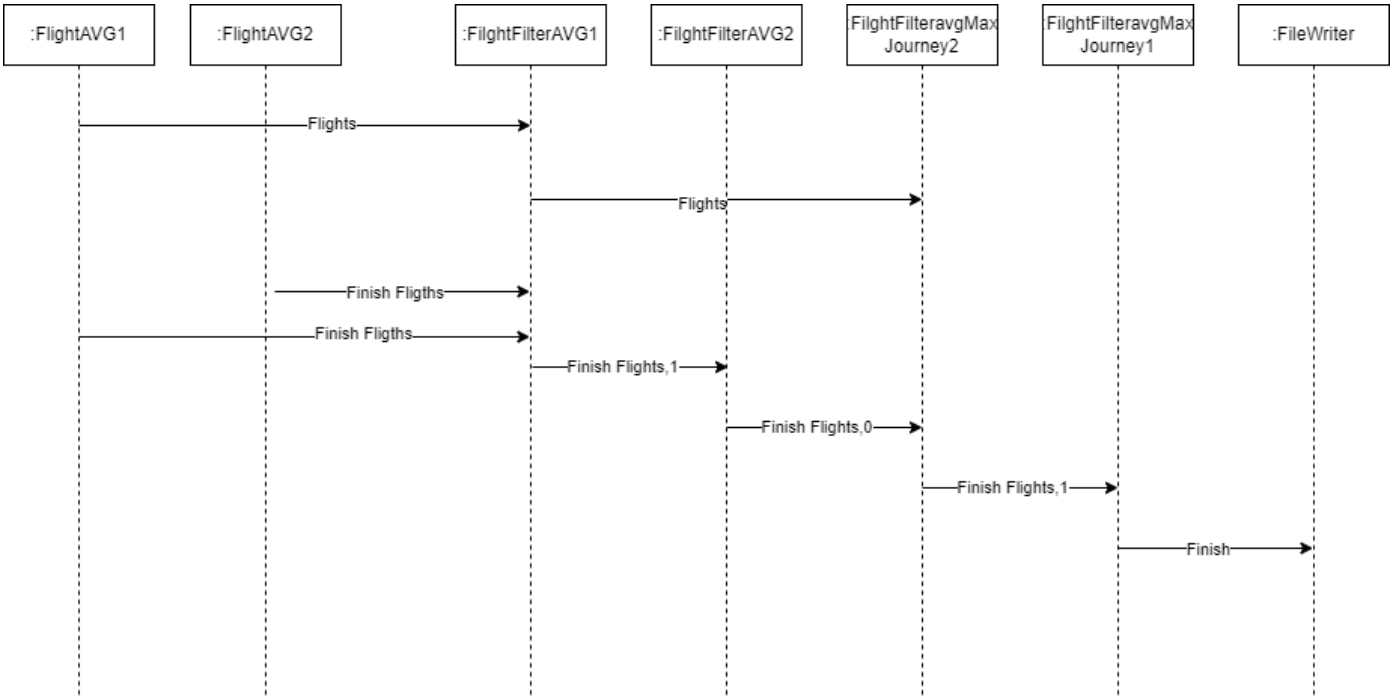
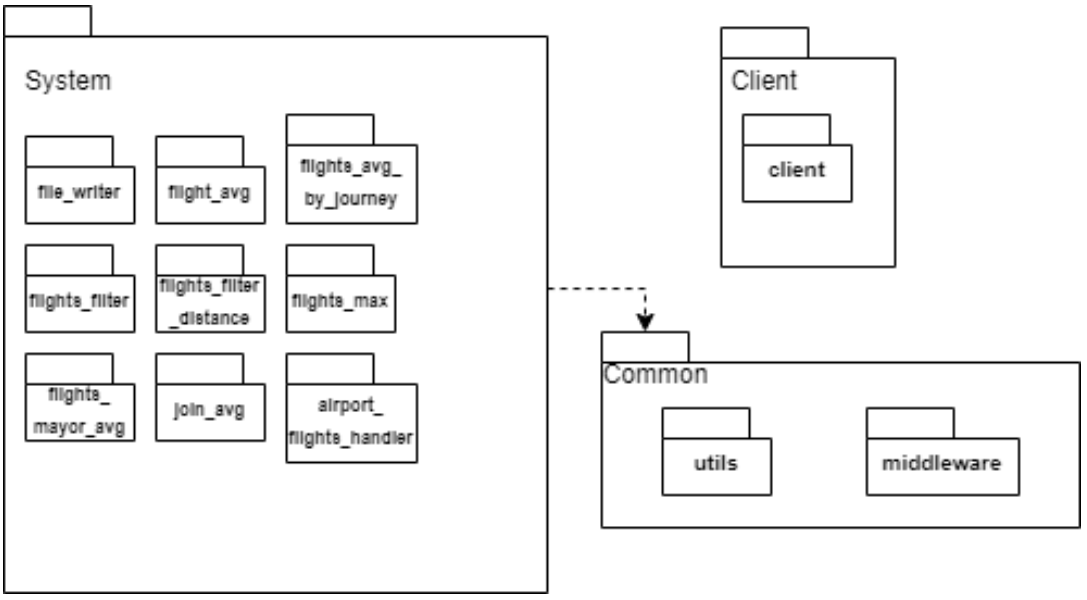


Diagrama 3: Escenario de finalización segunda parte de query 4

Vista de Desarrollo

Diagrama de Paquetes

En este diagrama se muestra la organización final del código. Se definieron carpetas utils y middleware comunes a todos los nodos del sistema para favorecer la reutilización de código y algunas constantes del protocolo implementado.



Tipos de mensajes:

Todos los mensajes se estructuran con un encabezado común de 3 bytes y un payload que respeta la siguiente estructura:

PKT_TYPE[1 byte]	PKT_SIZE[2 byte]
PAYLOAD	

Donde el pkt_type indica el tipo de paquete, se reservan dos bytes para el tamaño de paquete encodeado en big endian y a continuación se coloca el payload correspondiente

Los tipos de paquete quedan definidos de la siguiente manera:

Tipo de paquete	Número
FLIGHTS_PKT	0
HEADERS_FLIGHTS_PKT	1
AIRPORT_PKT	2
FLIGHTS_FINISHED_PKT	3
AIRPORT_FINISHED_PKT	4
HEADERS_AIRPORT_PKT	5

Trabajo futuro:

A futuro, la implementación de la escucha de dos queues distintas en el componente “flights_mayor_avg” es un punto que se podría mejorar, dado que en la queue de flights se utilizó una del tipo publish-subscribe seteando diferentes routing keys para cada nodo cuando se podría haber buscado la manera (aunque se intentó pero no se llegó a buen puerto) de usar un modelo product-consumer.

Por otro lado, el handler de airports y flights de la query2 podría optimizarse ligeramente para separar algunas responsabilidades permitiendo escalar el mismo.

Otro punto pendiente es el de realizar pruebas exhaustivas para encontrar la mejor configuración de nodos, lanzando diferentes pruebas de rendimiento aunque esto escapaba en cierta medida a los alcances del presente trabajo.