Facultad de Ingeniería
Universidad de Buenos Aires

Sistemas Distribuidos I (75.74)

Flights Optimizer

TP Escalabilidad: Middleware y Coordinación de Procesos

Documento de arquitectura

Alumnos

- Guido Bergman (104030)
- Luis Waldman (79279)

Índice

Alcance del sistema

El sistema *flights optimizer* procesa registros de vuelos de avión y a partir de ellos permite conocer:

- Los vuelos con 3 escalas o más
- Los vuelos cuya distancia total es mayor a cuatro veces la distancia directa entre el origen y el destino
- Los 2 vuelos más rápidos por trayecto, es decir origen y destino del vuelo, considerando solo los vuelos con 3 escalas o más
- El precio promedio y máximo para cada trayecto, considerando solamente los vuelos cuyo precio esté por encima del promedio de todos los precios

Además, el sistema soporta un número configurable de conexiones concurrentemente y es tolerante a fallos.

Arquitectura de software

El sistema estará desplegado en un entorno multi-computadora, en cual habrá uno o mas nodo para el cliente, mientras que el servidor se separará en:

- **Server**: El servidor administra la sesión de cada cliente y garantiza que no se conecten en simultáneo cierto número de clientes. A medida que va recibiendo los datos se los manda a los filtros y en caso de que se pierda la conexión con el cliente también se lo informa a los filtros enviando FLUSH.
- **Filtro Escalas**: A medida que recibe vuelos, cuenta las escalas y a los que tienen más de 3, los manda como resultado y además le manda el vuelo al Filtro Velocidad.
- **Filtro Velocidad:** A medida que recibe vuelos, va guardando los dos más rápido por trayecto. Cuando recibe el fin de vuelos manda como resultado lo que tenía guardado.
- **Filtro Distancia**: Primero recibe los Aeropuertos y los almacena en memoria. Luego, a medida que recibe vuelos, manda como resultados aquellos para los que la distancia total sea mayor a cuatro veces la distancia directa entre puntos origen-destino.
- **Filtro Precios**: A medida que recibe vuelos, va guardando los precios de todos los vuelos agrupados por trayecto y contando la cantidad y el promedio por cliente. Cuando llega el fin de vuelo, envía estos últimos datos al calculador de promedio. Cuando recibe de éste el promedio general, recorre el archivo de vuelos para calcular el promedio y máximo para cada trayecto de los vuelos más caros que el promedio.
- **Calculador promedio:** Recibe los promedios del filtro de precios y cuando recibió todos, manda el promedio general.
- **Watchdog:** Es el encargado de verificar que todos los otros elementos del sistema están funcionando y, en caso de que haya alguno caído revivirlo.

Casos de uso

En el siguiente diagrama se muestran los distintos casos de uso, que corresponden a los distintos tipos de resultados que puede recibir el cliente por parte del servidor.

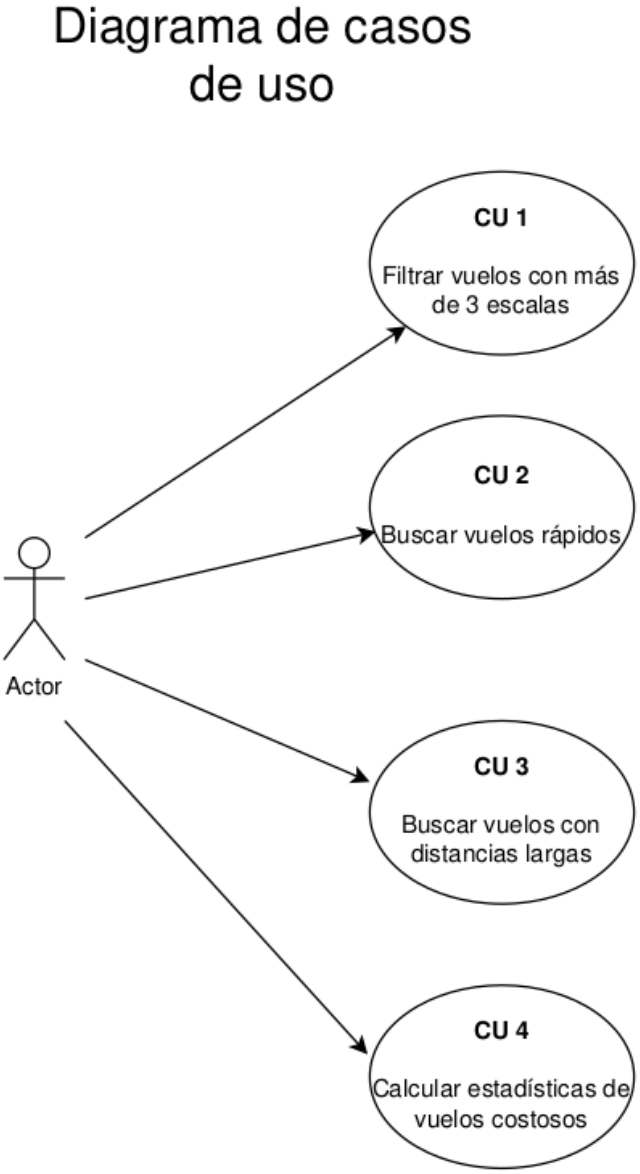

Fig 1. Diagrama de casos de uso

Vista Física

Diagrama de Robustez

En la figura 2 se pueden ver los distintos *controllers* y *entities* que habrá el sistema, así como las colas que se usarán para la comunicación.

Documento de robustez

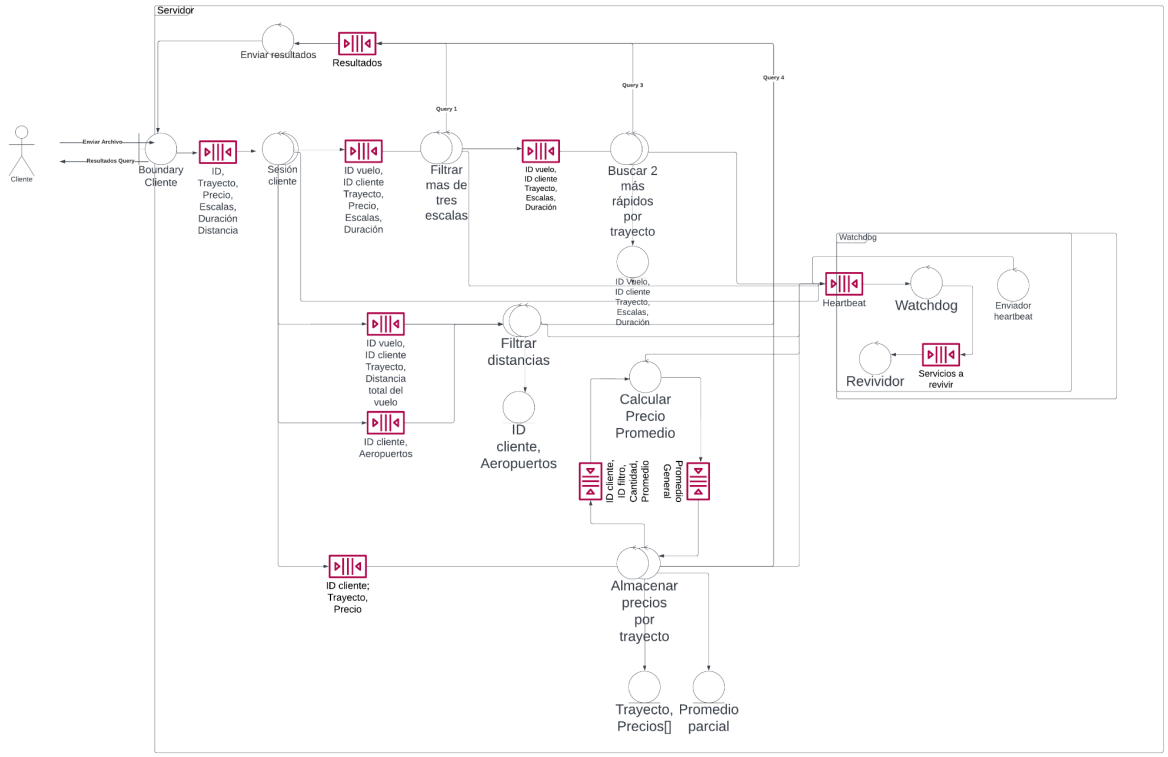

Fig 2. Diagrama de robustez

Diagrama de Despliegue

En la figura 3 se puede observar cómo estarán desplegados los *controllers* en los distintos nodos.

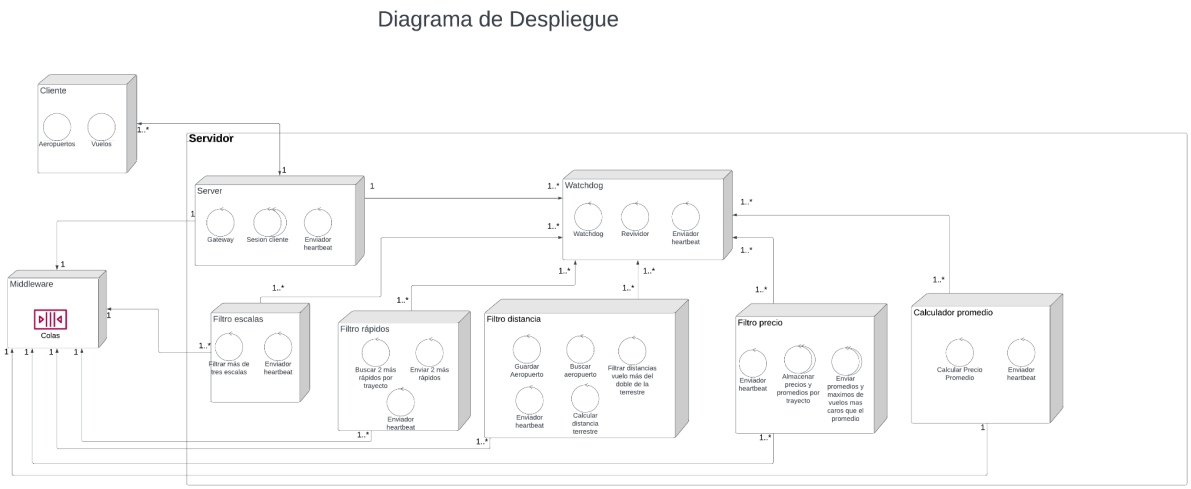

Fig 3. Diagrama de despliegue

Vista Desarrollo

Diagrama de paquetes

En la figura 4 se muestran los paquetes en los cuales estará separado el código.

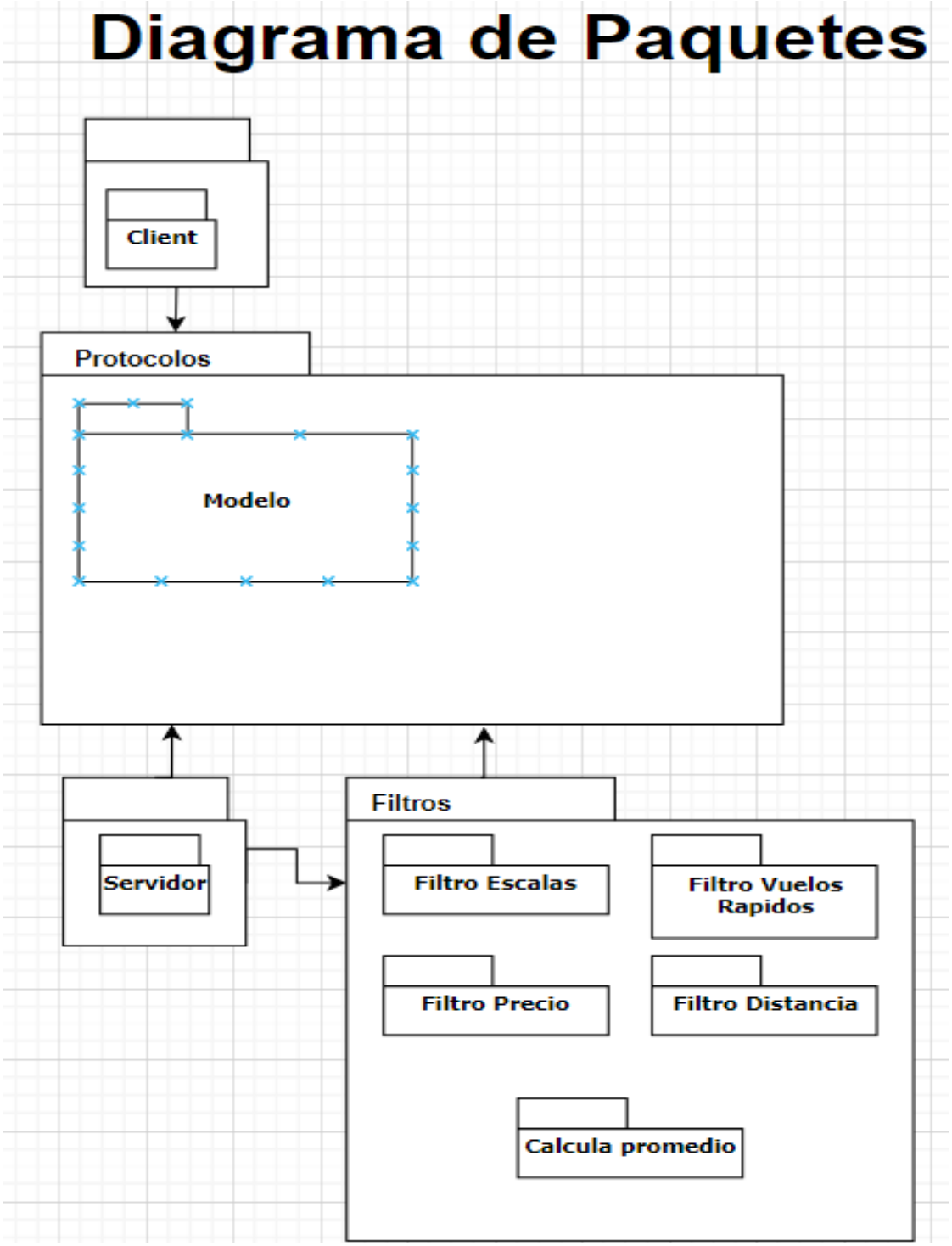

Fig 4. Diagrama de paquetes

Vista Procesos

DAG

En el siguiente DAG se podrán ver las distintas tareas que se realizan para obtener los resultados finales.

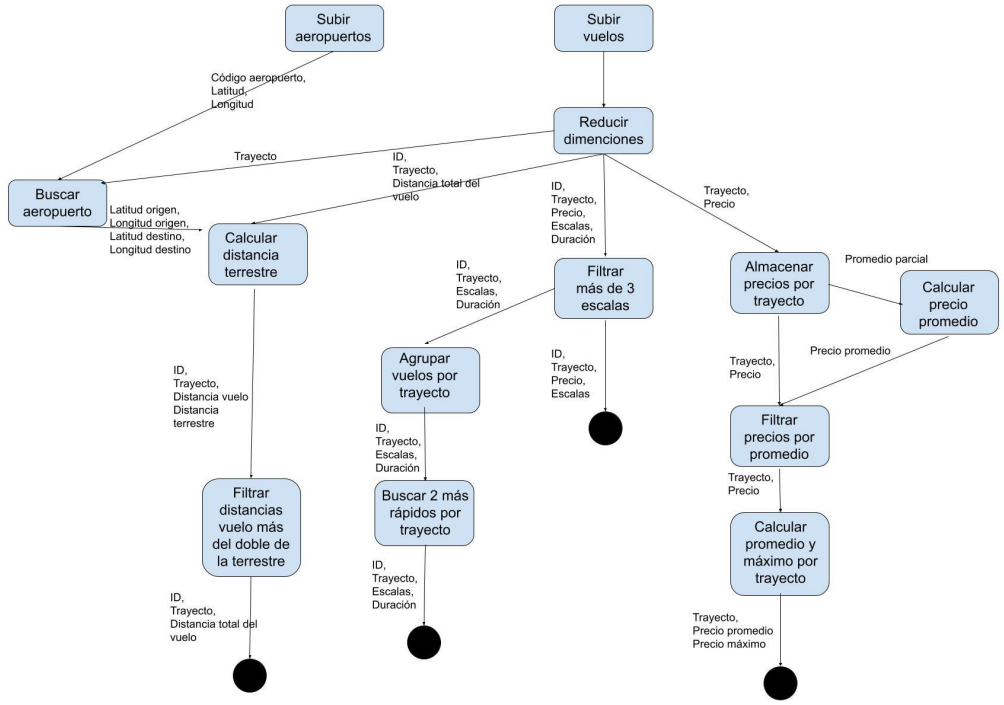

Fig 5. DAG

Diagramas de Actividad

En la figura 6 se muestra el flujo de trabajo que seguirá el sistema para devolverle al cliente los resultados solicitados.

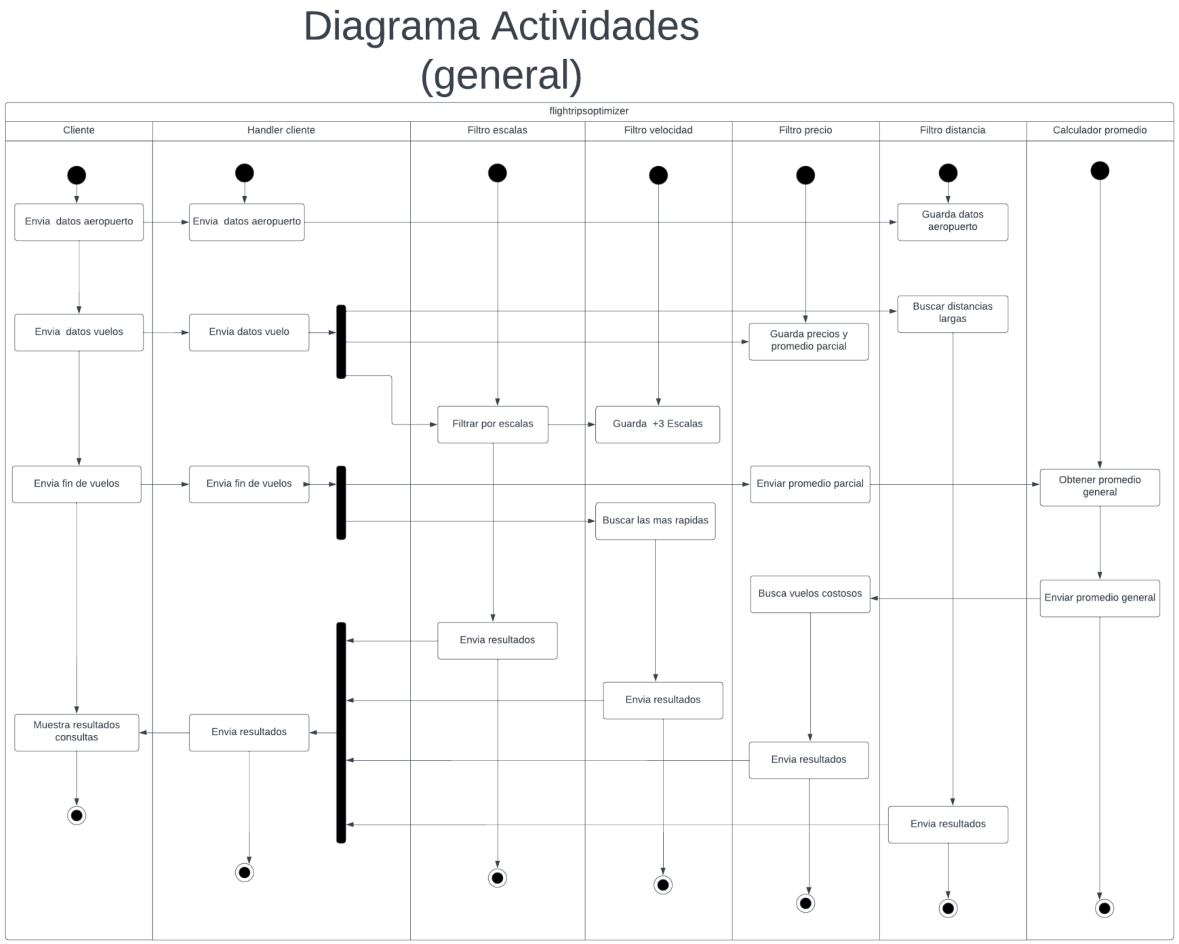

Fig 6. Diagrama de actividades general

En la figura 7, se pueden ver las distintas actividades que realizan el filtro de escalas y el de velocidad, tanto para procesar un vuelo como para procesar el fin de vuelos.

Como se puede ver en la figura, los filtros de escala envían los vuelos con 3 o más escalas a los filtros de velocidad. Para esto, se utiliza una estrategia de *sharding*. Para implementarla, se utiliza una cola con tópicos. Para determinar el tópico correspondiente a un vuelo se calcula un *hash* a partir del trayecto del vuelo. Esto permite que todos los vuelos de un mismo trayecto los reciba un mismo filtro de velocidad.

Al recibir el fin de vuelos, cada uno de los filtros de escalas enviará un mensaje de fin de vuelos a cada uno de los filtros de velocidades. Esto lo hace enviando el mensaje a cada uno de los tópicos correspondientes a un filtro de velocidad.

También se puede observar, que el filtro de velocidad no envía los resultados al recibir el primer mensaje de fin de vuelos, sino que espera a haber recibido los mensajes de fin de vuelos de todos los filtros de escalas. Esto permite que haya varios filtros de escala enviando vuelos y que solo se empiecen a enviar los resultados cuando se hayan terminado de recibir todos los vuelos.

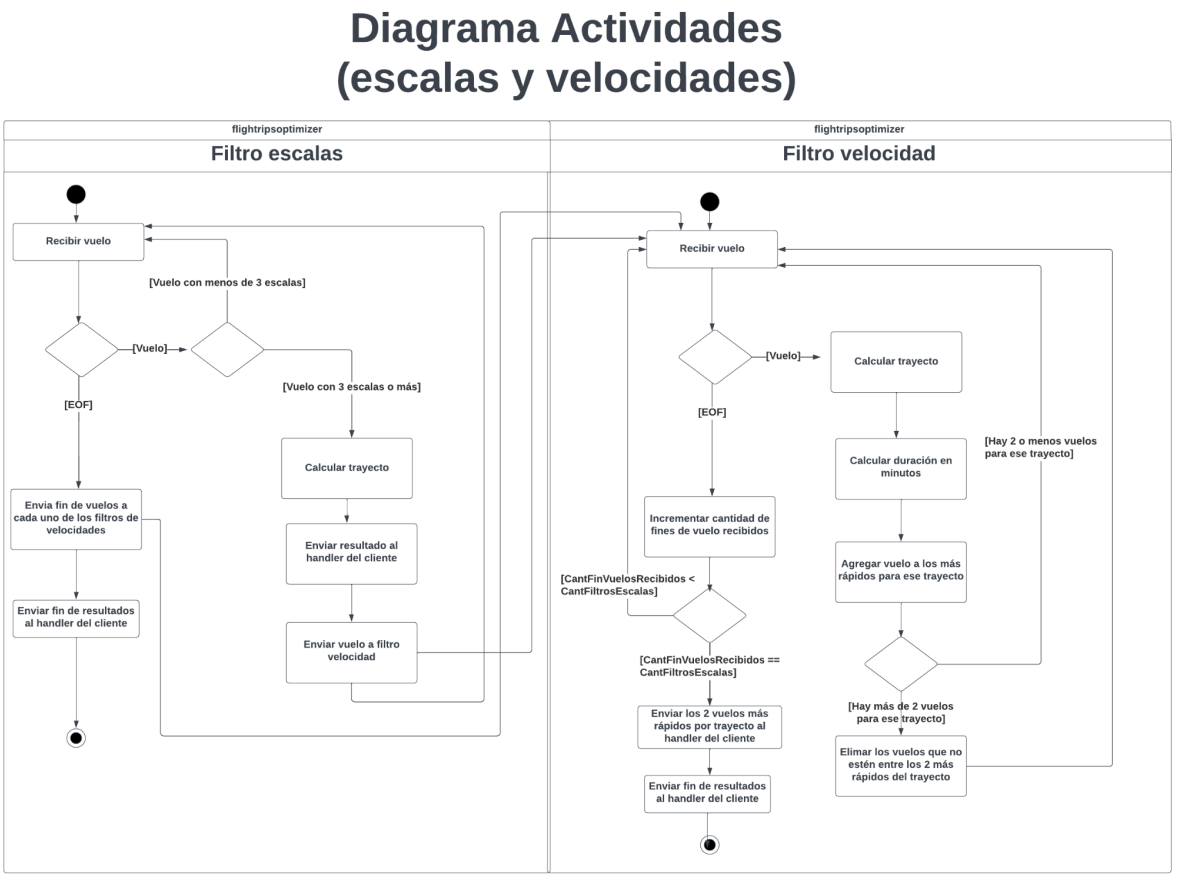

Fig 7. Diagrama de actividades filtros escalas y velocidad

En la figura 8 se pueden ver las distintas actividades que realizan el filtro de distancia y el handler del cliente, para recibir los aeropuertos, procesar los vuelos y el fin de vuelos. En el caso de los aeropuertos, dado que todos los filtros de distancias deben recibir todos los aeropuertos, se utiliza un exchange de tipo ‘fanout’.

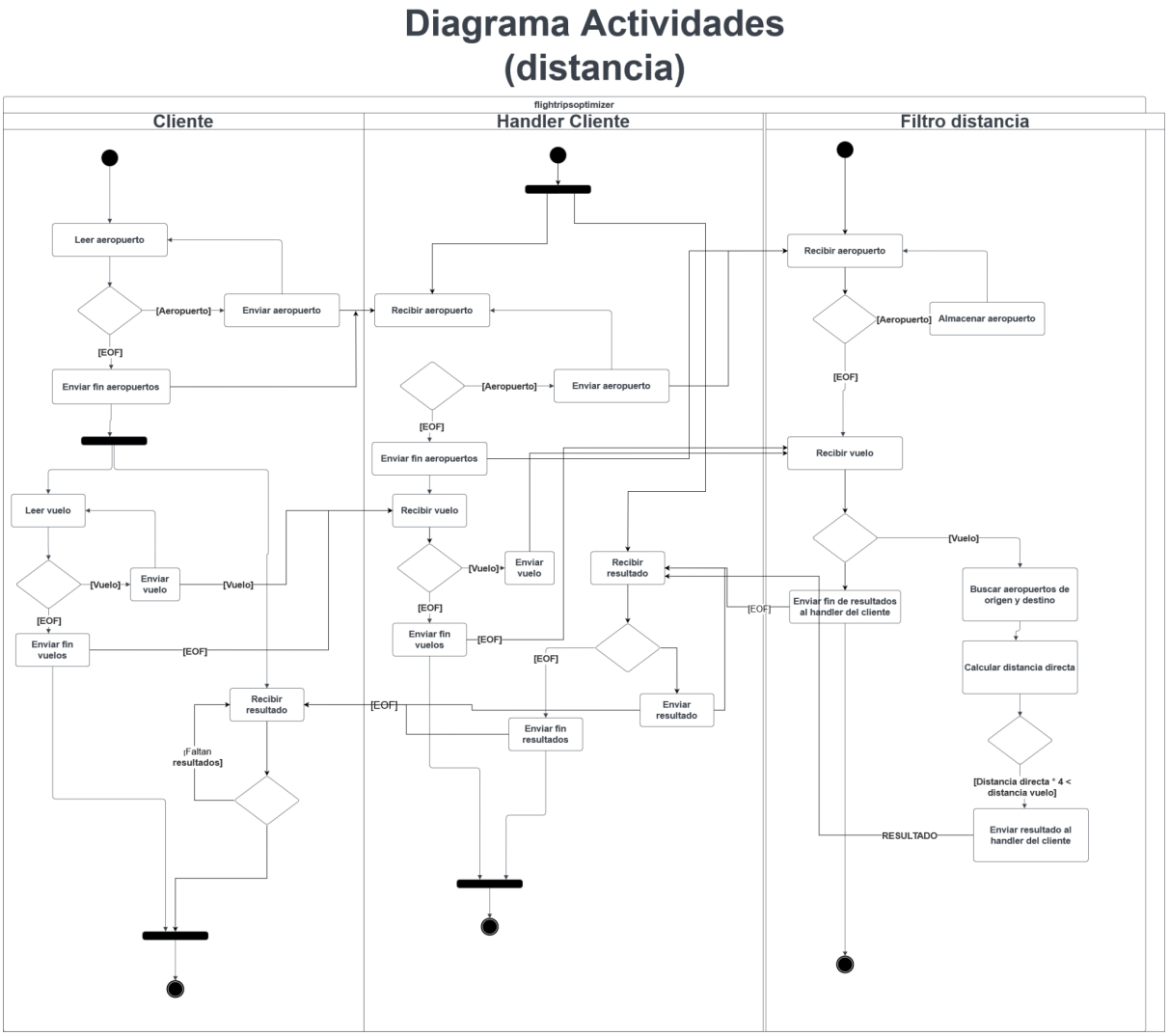

Fig 8. Diagrama de actividades filtro distancia

En la figura 9 se pueden ver las actividades del filtro de precios y el calculador de promedios. Cabe destacar que el filtro de precios utiliza la misma estrategia de *sharding* que el filtro de vuelos rápidos, es decir colas por tópico basadas en un *hash* del trayecto.

Diagrama Actividades (precio)

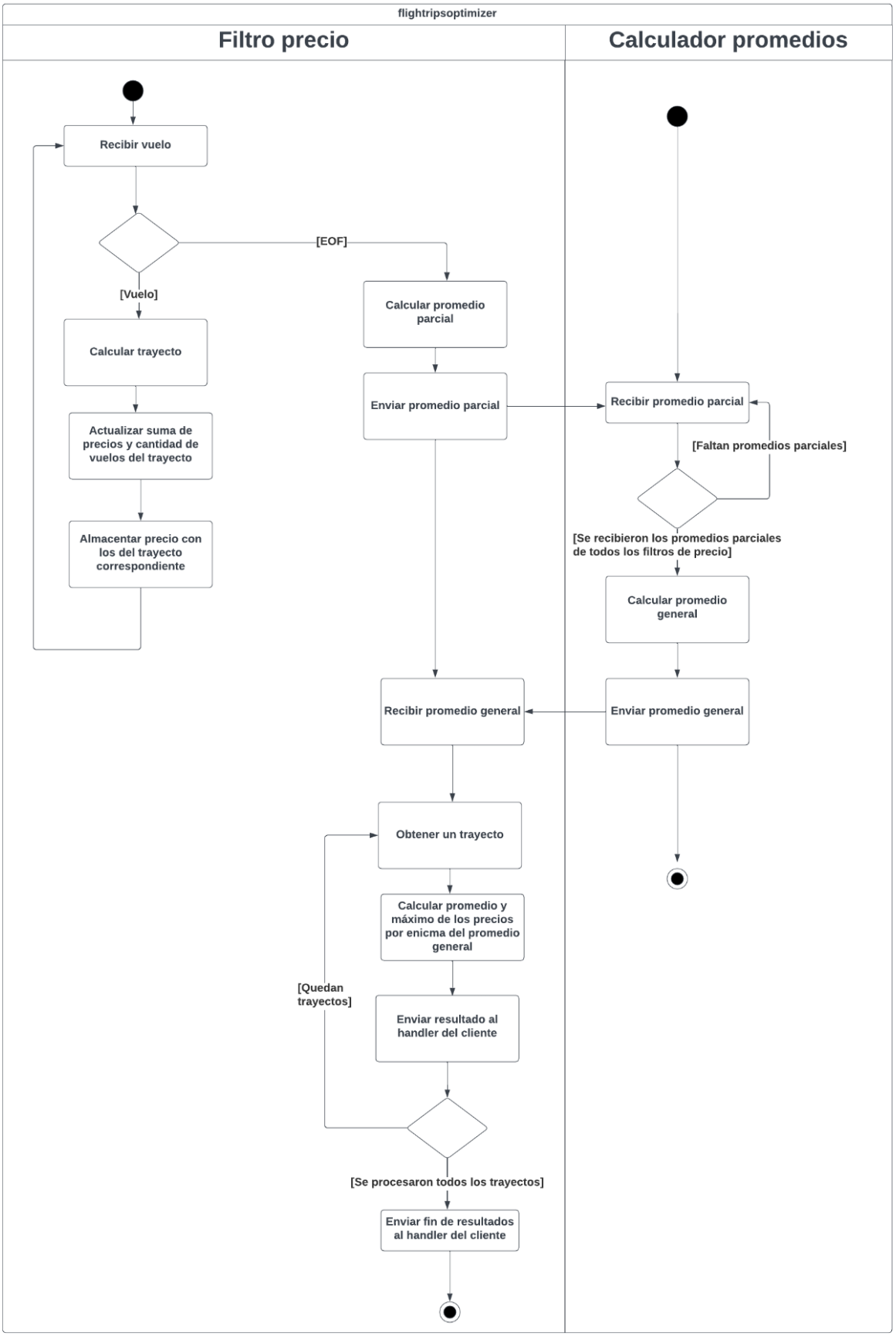

Fig 9. Diagrama de actividades filtro precio

Soporte para múltiples clientes

Para admitir múltiples clientes en el servidor, desarrollamos la clase SesionCliente. que estado de un cliente.
Cuando al servidor le llega una solicitud de conexión, en caso de estar disponible, se crea un objeto SesionCliente que acepta la solicitud y crea un archivo con un ID único de cliente. Durante todo el proceso esta clase va a manejar el flujo de datos entre el cliente y los filtros. Crea una cola para recibir los resultados y a cada dato que recibe lo envía a los filtros. Al mismo tiempo, escucha en la cola de resultados y a medida que van llegando se los manda al cliente.

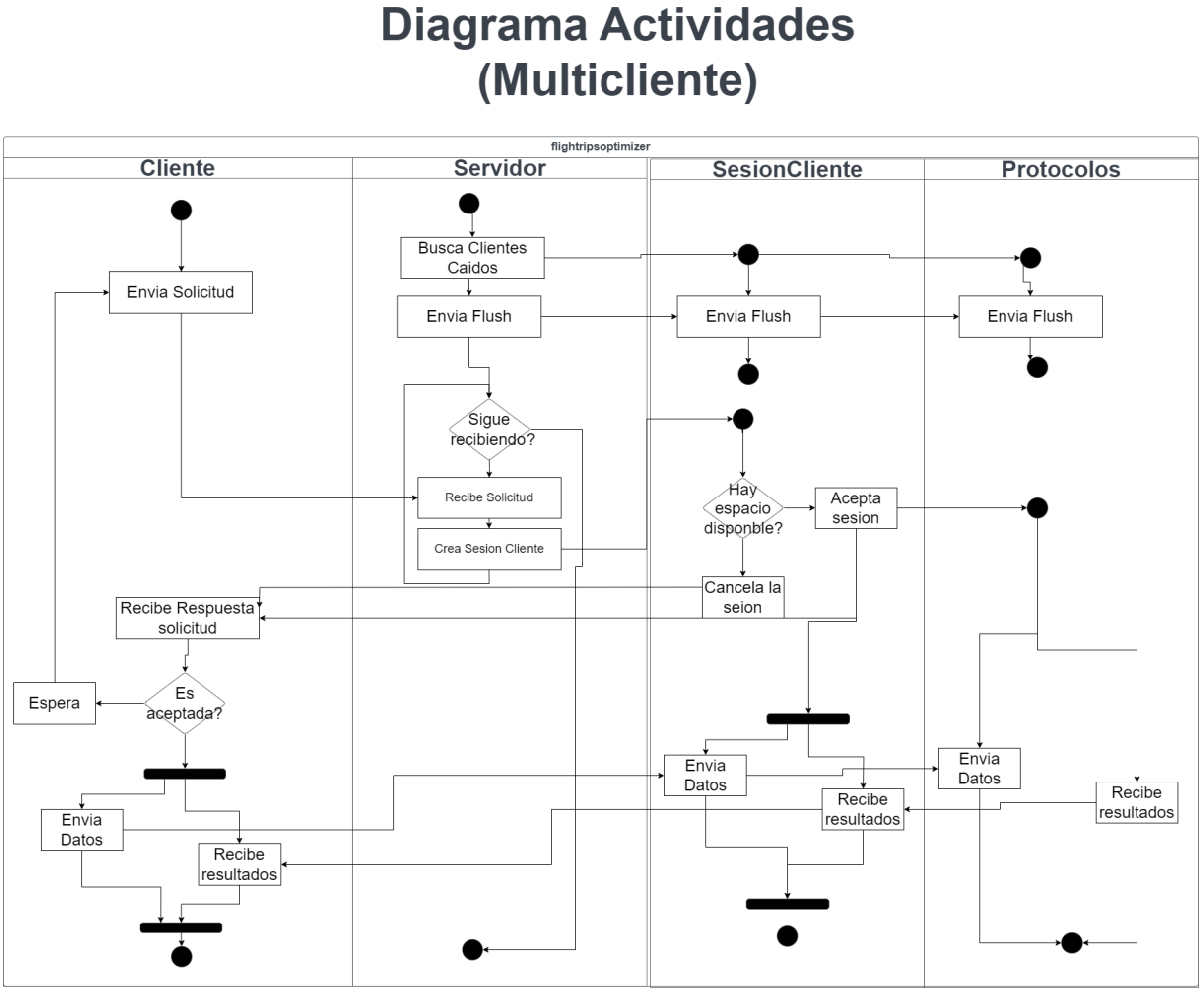

Fig 10. Diagrama de actividades multicliente

Al finalizar el proceso, el archivo es borrado. Cuando el servidor se inicia, se fija que archivos quedaron sin borrar: así idéntica a los clientes que quedaron inconclusos, entonces borra el archivo y manda FLUSH a los distintos filtros para que hagan lo mismo.

Diagrama de secuencia

Cálculo de precios

En el siguiente diagrama se muestra la secuencia que sigue el filtro de precios para calcular las estadísticas de los precios

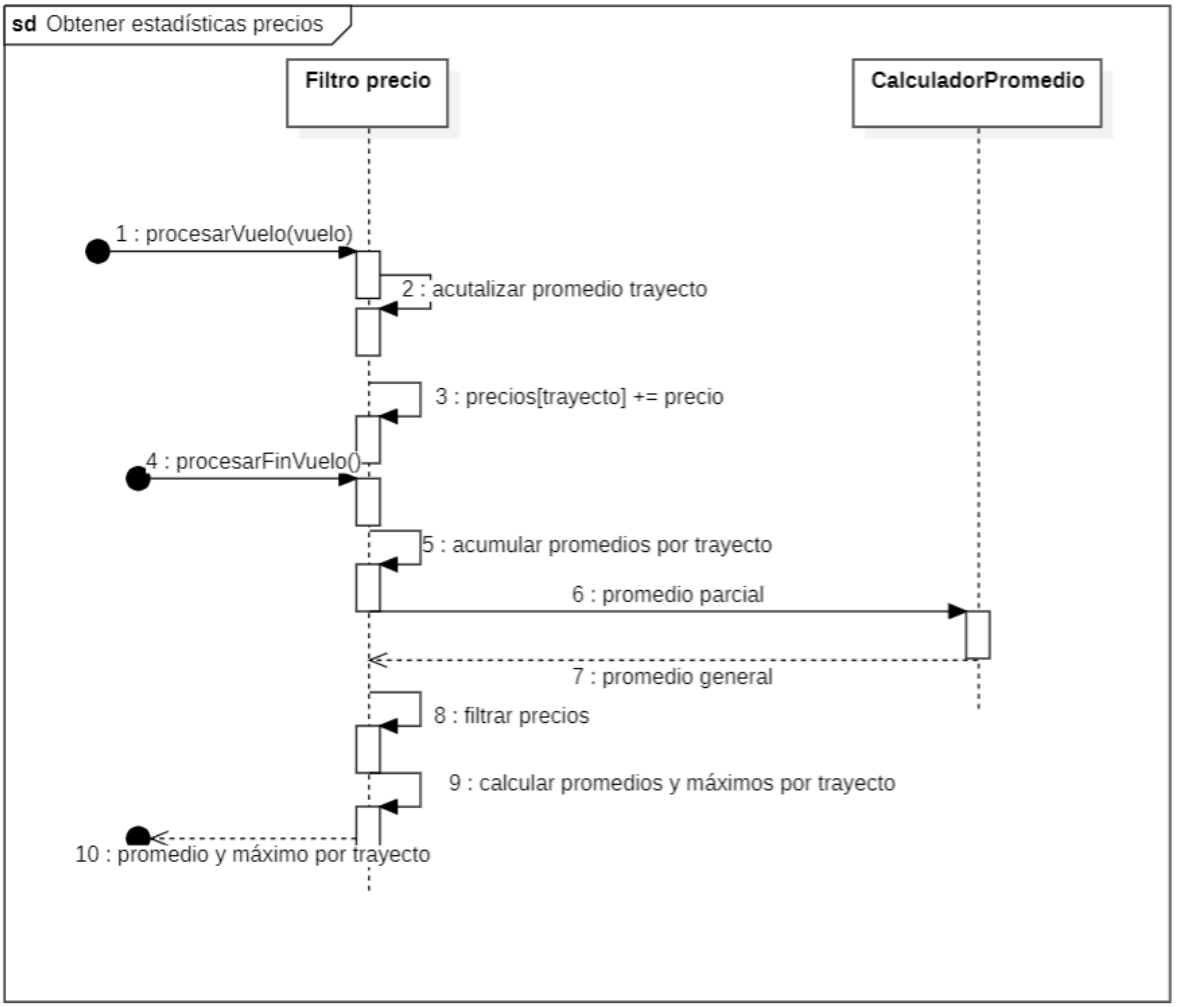

Fig 11. Diagrama de secuencia - Calcular estadísticas precios

Tolerancia a Fallos

Para hacer el sistema tolerante a fallos abordamos dos ejes: primero necesitamos que cuando se de una falla en un filtro, este vuelva a levantarse. Además de levantarse, debe recuperar el estado de lo que ya

Persistencia

Cada vez que algún filtro saca algún mensaje de la cola, se genera un contenido que se persiste junto con un hash del mensaje, que se utiliza para filtrar duplicados, y antes de mandar el ACK. De esta manera nos aseguramos que no queden archivos persistidos de mensajes que fallaron en una instancia posterior. Si, por ejemplo, la aplicación fallará luego de guardar el archivo y antes de mandar el ACK; cuando se reinicia recupera los mensajes guardados y cuando lo reciba de nuevo, lo reconocerá como duplicado.
Para asegurarnos que las escrituras sean atómicas, debemos escribir menos de 4096 bytes[1]. Esto nos limita a la hora de escribir resultados y tenemos que ajustar el total de elementos que mandamos en cada mensaje para ajustarnos a eso. El texto guardado es recorrido cuando se inicia una instancia para actualizar el estado. El filtro de precios, por ejemplo, guarda los precios de los vuelos que recorrió ordenados por trayecto. De está forma, el esquema propuesto permite garantizar la atomicidad en las escrituras y que el filtro de duplicados se recupere junto con el contenido de los mensajes, lo cual evita inconsistencias. Sin embargo, tiene la desventaja de que será necesario leer todo el archivo de disco para recuperar el estado. Es por esto que los filtros buscan ajustar el contenido que persiste para que sea requerido el menor procesamiento posible. Por ejemplo, el filtro de velocidad solo persiste de su estado, el hash de los mensajes y cuando ocurren cambios en los vuelos más rápido de un trayecto.

La secuencia mencionada se muestra en la siguiente figura. Cabe destacar que ProtocoloBase, es la clase padre de la cual heredan muchos de los otros protocolos, lo cual permite evitar la duplicación de código.

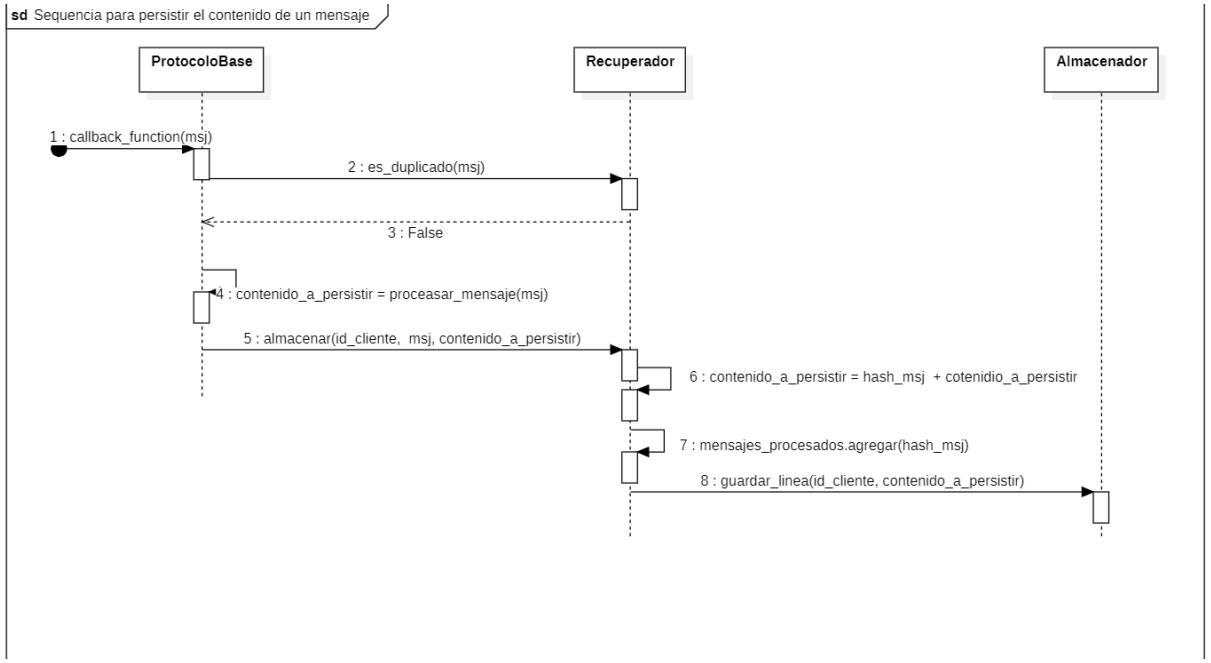

Fig 12. Diagrama de secuencia - Persistencia

En el caso del filtro de precios, el funcionamiento normal es así:

[1] https://www.notthewizard.com/2014/06/17/are-files-appends-really-atomic/

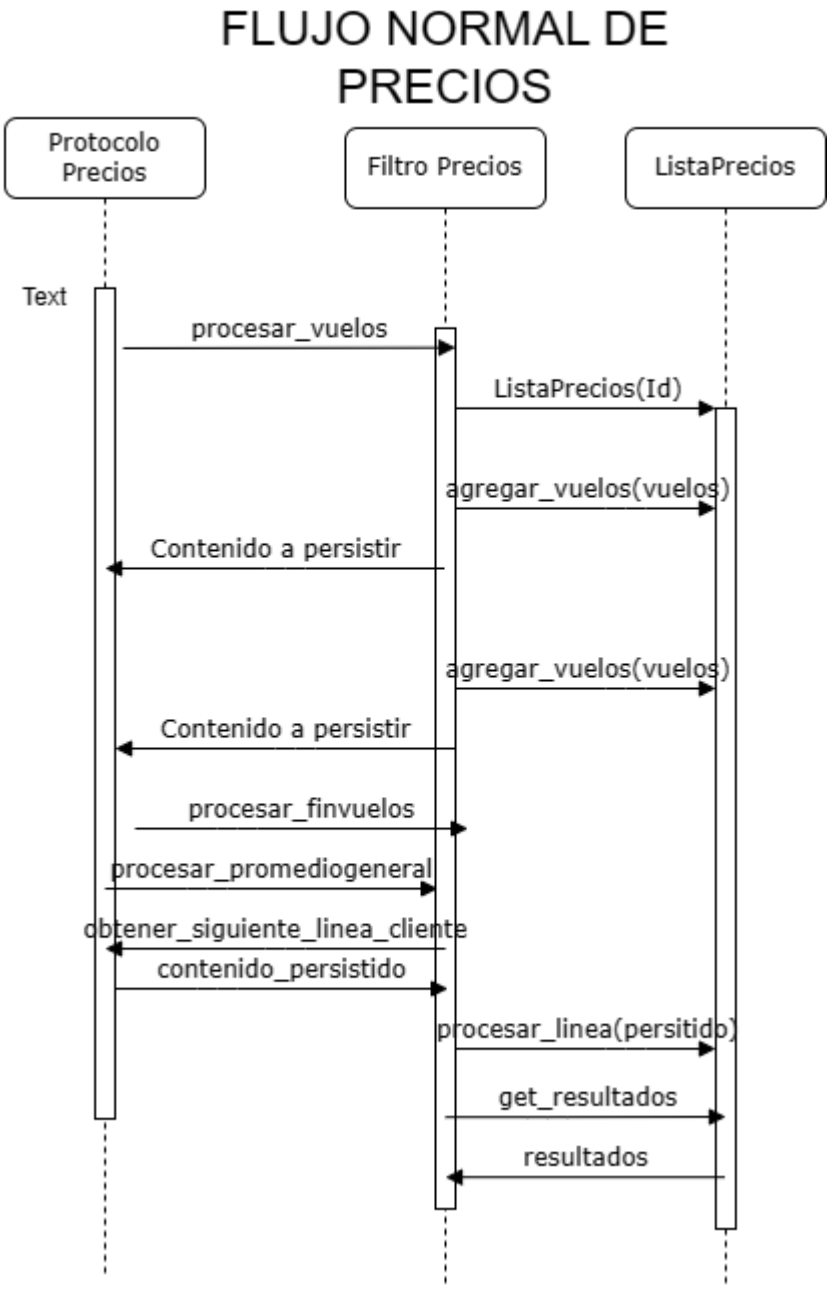

Fig 13. Diagrama de secuencia - Recuperacion de promedios

Y se recupera con ayuda de los datos que ya persistio:

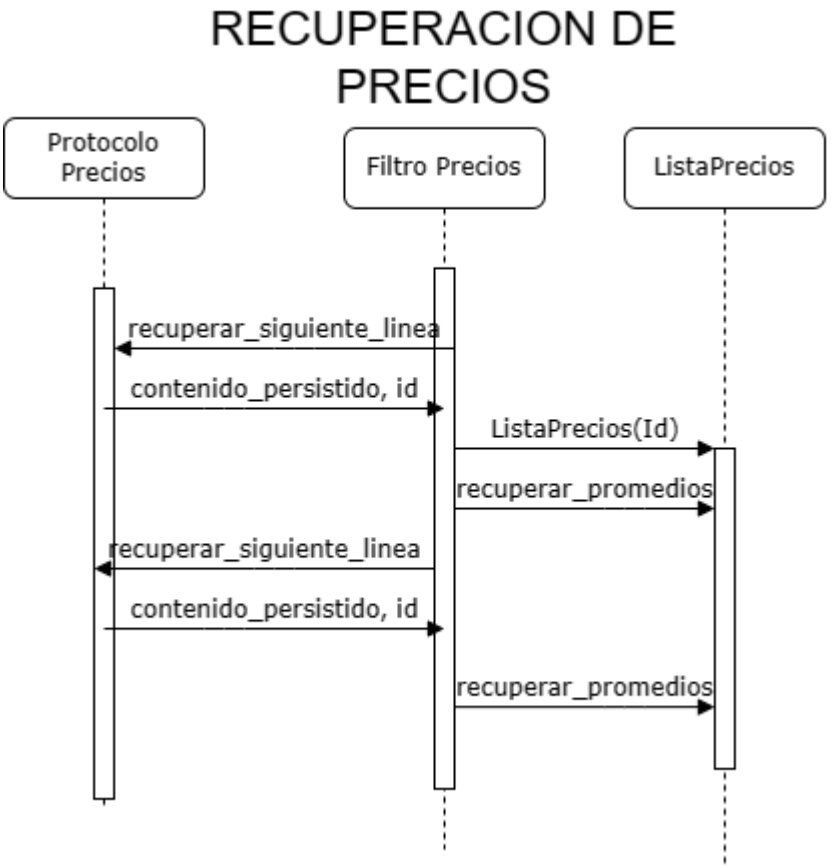

Fig 13. Diagrama de secuencia - Recuperacion de promedios

Recuperación

Para volver a restablecer los servicios caídos, se crearon watchdogs. Cada watchdog recibe cada 1 segundo el heatbeat del servidor, todos los filtros, el calculador de promedios y los otros watchdogs. Los watchdogs reciben los heartbeats utilizando un socket que además tiene un timeout para poder revivir a los otros servicios, en caso de que el único componente vivo del sistema sea un watchdog. Cuando se recibe un heartbeat el timeout del socket, el watchdog recorrerá la lista de todos los servicios del sistema y verifica si alguno está muerto. Para verificar esto resta el timestamp del último heartbeat recibido de cada servicio con el timestamp actual. Si la diferencia es mayor a un timeout, se considerará que el servicio está muerto. En este caso, el watchdog verificará si le corresponde a el revivirlo, y en caso afirmativo lo revivirá. Esto se puede hacer de dos maneras:

- Si el otro servicio no es un watchdog, se calculara un hash a partir de su nombre, que se usará para determinar el id del watchdog al que le corresponde revivirlo
- Si el servicio es un watchdog, solo lo revivirá en caso de que el otro watchdog tenga el id siguiente al suyo. Esto permite evitar que un watchdog se tenga que revivir a sí mismo.

De está forma, el sistema soporta caídas de todos los componentes, exceptuando el caso de que se caigan todos los watchdogs en simultáneo. La principal ventaja de está implementación es su simplicidad: no requiere elecciones de líder, por lo que se reduce el

overhead que produce. Una desventaja de este esquema es que no tolera particionamiento de red.

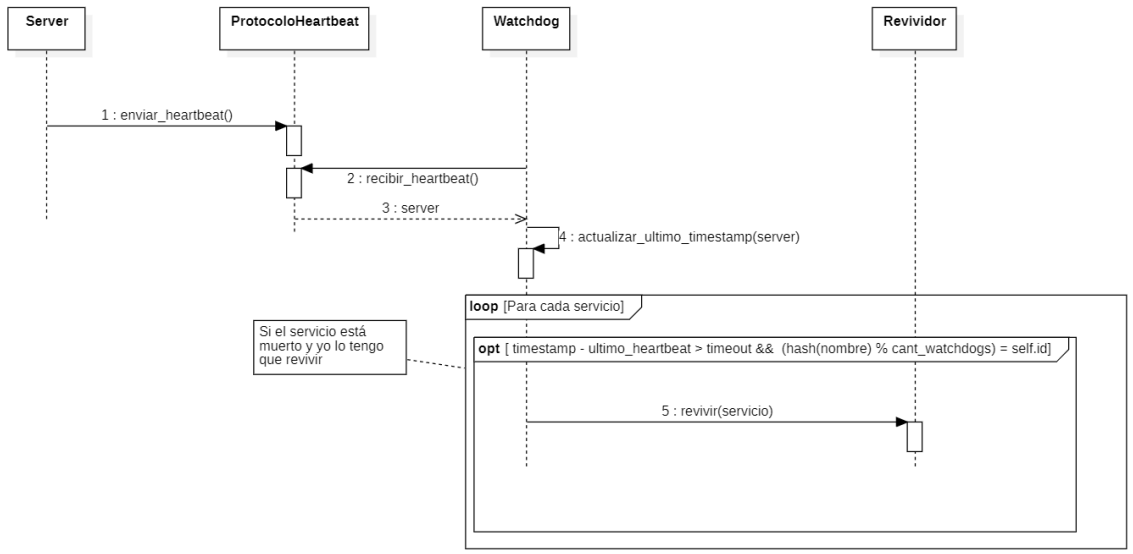

Fig 13. Diagrama de secuencia - Recuperación

Vista Lógica

Diagrama de clases

En la figura 14, se puede observar las clases del cliente, así como los protocolos que usará para comunicarse con el servidor.

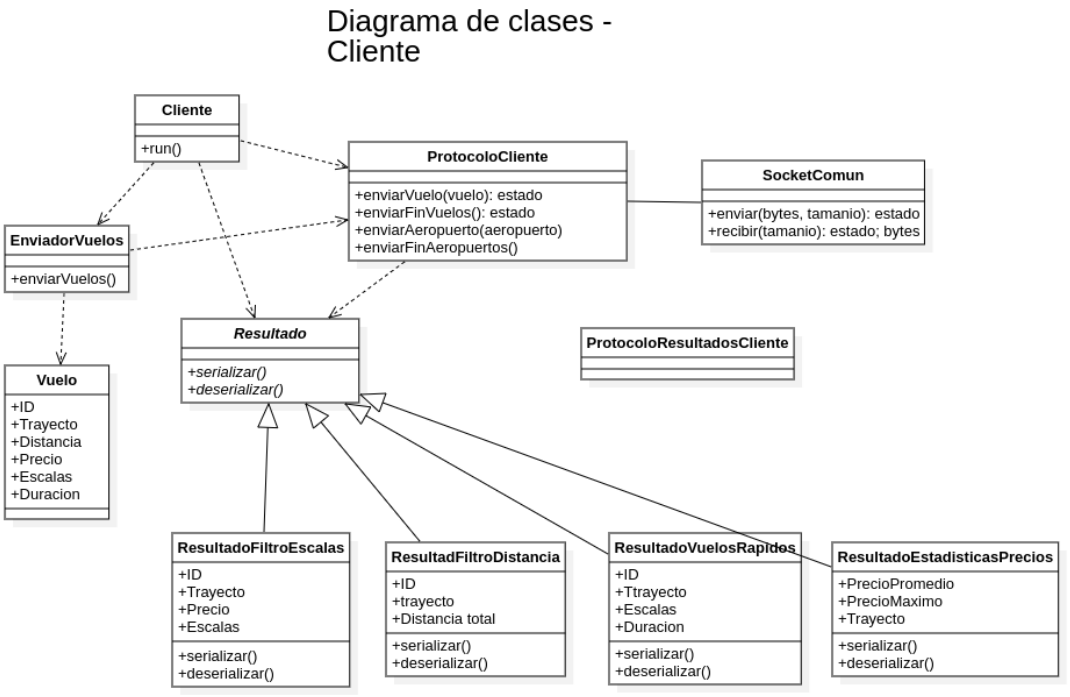

Fig 14. Diagrama de clases - Cliente

En la siguiente figura se muestran las clases en el servidor para el caso de uso correspondiente al filtro de los vuelos con más de 3 escalas. Los protocolos que se pueden observar en el diagrama corresponden al protocolo que usa para comunicarse con el cliente, ProtocoloCliente, y algunos de los protocolos que usarán los distintos nodos del servidor para comunicarse entre sí, que son ProtocoloResultados, ProtocoloFiltroEscalas y ProtocoloVuelosRapidos. Estos últimos se comunicaran usando un Message Oriented Middleware de colas.

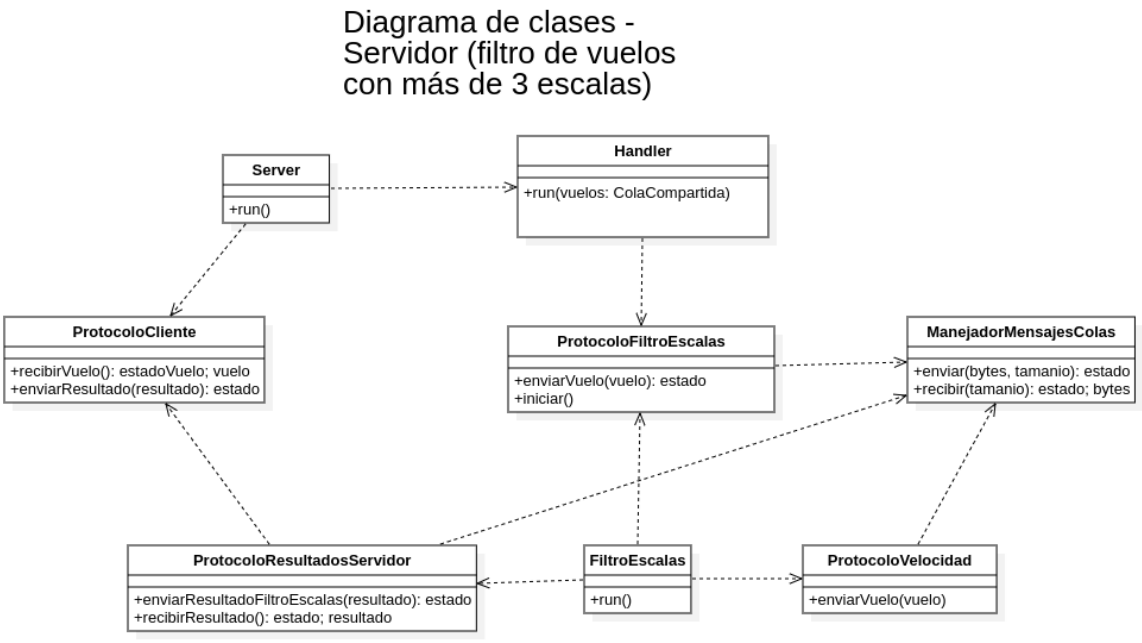

Fig 15. Diagrama de clases - Servidor (filtro de vuelos con más de 3 escalas)

Vista Protocolos

En esta sección se explican los distintos protocolos desarrollados para la comunicación entre nodos. Todos ellos son protocolos binarios.

Protocolos de comunicación entre el cliente y el servidor

Inicio de sesion

El mensaje para solicitar sesion es un caracter con la letra ‘S’
Si hay disponibilidad, el servidor acepta la secion respondiendo ‘T’
Si no, la canciela respondiendo ‘C’

Vuelos

Los mensajes para enviar un batch de vuelos inician con:

- Identificador de mensaje para batch de vuelos: un byte con un carácter que será la letra ‘V’’
- Tamaño del batch: *unsigned short* de 2 *bytes*

Luego se envía, para cada vuelo, su información en el siguiente formato:

- ID vuelo: *string* 32 bytes con *encoding* UTF-8
- Origen: *string* 3 bytes con *encoding* UTF-8
- Destino: *string* 3 bytes con *encoding* UTF-8
- Precio: *float* de 4 bytes
- Longitud string escalas: *unsigned short* de 2 *bytes*
- Escalas: *string* de 50 bytes. Para las escalas se usa el *encoding* UTF-8 y luego se agrega un *padding* de ceros hasta completar los 50 bytes
- Duración: *string* de 8 bytes. Se utiliza UTF-8 para el *encoding* del string de la duración, por ejemplo ‘PT11H54M’, y se completa con ceros al final, en caso de que la duración tenga menos de 8 bytes.
- Distancia: *short* de 2 bytes. Se completa con -1 en caso de que el vuelo no tenga distancia

El fin del envío de vuelos se indica enviando un mensaje con el identificador del fin de vuelos, que es un byte con la letra ‘F’.

Aeropuertos

El formato de los mensajes para enviar los aeropuertos consiste en:

- Identificador de mensaje aeropuerto: un byte con un carácter que será la letra ‘A’’
- Tamaño del batch: *unsigned short* de 2 *bytes*
- *Lista de Aeropuertos:*
    - ID aeropuerto: *string* 3 bytes con *encoding* UTF-8

    - Latitud: *float* de 4 bytes
    - Longitud: *float* de 4 bytes

Para indicar el fin del envío de aeropuertos, se envía un mensaje con el identificador del fin de aeropuertos, que consiste en un byte con la letra ‘F’.

Vuelos y Aeropuertos en colas de Rabbit

Los mensajes de vuelos y aeropuertos cambian en esta version cuando estan del lado del servidor. Ahora, agregamos el ID_CLIENTe que se genera para cada sesion.

Entonces el formato de los mensajes para enviar los aeropuertos queda:

- Identificador de mensaje aeropuerto: un byte con un carácter que será la letra ‘A’’
- ID_CLIENTE: *string* 32 bytes con *encoding* UTF-8
- Tamaño del batch: *unsigned short* de 2 *bytes*
- *Lista de Aeropuertos:*
    - ID aeropuerto: *string* 3 bytes con *encoding* UTF-8
    - Latitud: *float* de 4 bytes
    - Longitud: *float* de 4 bytes

Mientras que la cabecera del mensaje de vuelos quedaría:

Los mensajes para enviar un batch de vuelos inician con:

- Identificador de mensaje para batch de vuelos: un byte con un carácter que será la letra ‘V’’
- ID_CLIENTE: *string* 32 bytes con *encoding* UTF-8
- Tamaño del batch: *unsigned short* de 2 *bytes*

Resultados

Resultados escalas

Los mensajes para enviar los vuelos con 3 o más escalas consisten en:

- Identificador de resultado escalas: es un byte con la letra ‘E’ (mayúscula)
- Id vuelo: *string* 32 bytes con *encoding* UTF-8
- Trayecto: *string* 7 bytes con *encoding* UTF-8, que contiene el Origen y el Destino del vuelo. El formato es ‘<origen>-<destino>’, por ejemplo ‘BUE-MAD’.
- Precio: *float* de 4 bytes
- Longitud string escalas: *unsigned short* de 2 *bytes*
- Escalas: *string* de 50 bytes. Para las escalas se usa el *encoding* UTF-8 y luego se agrega un *padding* de ceros hasta completar los 50 bytes

El fin del envío de estos resultados se indica enviando al cliente un mensaje con byte con la letra ‘e’ (en minúscula).

Resultados filtro distancia

El formato de los mensajes los vuelos cuya distancia total sea mayor a cuatro veces la distancia directa entre puntos origen-destino es:

- Identificador de resultado distancia: es un byte con la letra ‘D’ (mayúscula)
- Id vuelo: *string* 32 bytes con *encoding* UTF-8
- Trayecto: *string* 7 bytes con *encoding* UTF-8, que contiene el Origen y el Destino del vuelo. El formato es ‘<origen>-<destino>’, por ejemplo ‘BUE-MAD’.
- Distancia: *unsigned short* de 2 bytes.

El fin del envío de estos resultados se indica enviando al cliente un mensaje con byte con la letra ‘d’ (en minúscula).

Resultados filtro distancia

El formato de los mensajes los vuelos cuya distancia total sea mayor a cuatro veces la distancia directa entre puntos origen-destino es:

- Identificador de resultado distancia: es un byte con la letra ‘D’ (mayúscula)
- Id vuelo: *string* 32 bytes con *encoding* UTF-8
- Trayecto: *string* 7 bytes con *encoding* UTF-8, que contiene el Origen y el Destino del vuelo. El formato es ‘<origen>-<destino>’, por ejemplo ‘BUE-MAD’.
- Distancia: *unsigned short* de 2 bytes.

El fin del envío de estos resultados se indica enviando al cliente un mensaje con byte con la letra ‘d’ (en minúscula).

Resultados vuelos rápidos

Los mensajes para enviar los 2 vuelos más rápidos por trayecto, entre todos los vuelos de 3 escalas o más son:

- Identificador de resultado vuelos rápidos: es un byte con la letra ‘R’ (mayúscula)
- Id vuelo: *string* 32 bytes con *encoding* UTF-8
- Trayecto: *string* 7 bytes con *encoding* UTF-8, que contiene el Origen y el Destino del vuelo. El formato es ‘<origen>-<destino>’, por ejemplo ‘BUE-MAD’.
- Longitud string escalas: *unsigned short* de 2 *bytes*
- Escalas: *string* de 50 bytes. Para las escalas se usa el *encoding* UTF-8 y luego se agrega un *padding* de ceros hasta completar los 50 bytes

- Duración: *string* de 8 bytes. Se utiliza UTF-8 para el *encoding* del string de la duración. Se completa con ceros al final, en caso de que la duración tenga menos de 8 bytes.

El fin del envío de estos resultados se indica enviando al cliente un mensaje con byte con la letra ‘r’ (en minúscula).

Resultados estadísticas precios

Los mensajes para para enviar el precio promedio y máximo por trayecto de los vuelos con precio mayor a la media general de precios consisten en:

- Identificador de resultado precios: es un byte con la letra ‘P’ (mayúscula)
- Trayecto: *string* 7 bytes con *encoding* UTF-8, que contiene el Origen y el Destino del vuelo. El formato es ‘<origen>-<destino>’..
- Precio promedio: *float* de 4 bytes
- Precio máximo: *float* de 4 bytes

El fin del envío de estos resultados se indica enviando al cliente un mensaje con byte con la letra ‘p’ (en minúscula).

Protocolos de comunicación entre el handler del cliente y los filtros

Envío de vuelos al filtro de escalas

Los mensajes que se usan para enviar los vuelos al filtro de escalas contienen:

- Identificador de mensaje para vuelo: un byte con un carácter que será la letra ‘V’
- Tamaño del batch: *unsigned short* de 2 *bytes*
- ID vuelo: *string* 32 bytes con *encoding* UTF-8
- Origen: *string* 3 bytes con *encoding* UTF-8
- Destino: *string* 3 bytes con *encoding* UTF-8
- Escalas: *string* de 50 bytes. Para las escalas se usa el *encoding* UTF-8 y luego se agrega un *padding* de ceros hasta completar los 50 bytes
- Duración: *string* de 8 bytes. Se utiliza UTF-8 para el *encoding* del string de la duración, y se completa con ceros al final, en caso de que la duración tenga menos de 8 bytes.

El fin del envío de los vuelos al filtro de escalas se indica con un mensaje con byte con la letra ‘F’.

Envío de aeropuertos al filtro de distancias

Los mensajes que se usan para enviar los aeropuertos, y el fin de los mismos, al filtro de distancias tienen el mismo formato que los que usa el cliente para enviarlos al servidor.

Envío de vuelos al filtro de distancias

Los mensajes que se usan para enviar los vuelos al filtro de distancias contienen:

- Identificador de mensaje para vuelo: un byte con un carácter que será la letra ‘V’
- Tamaño del batch: *unsigned short* de 2 *bytes*
- ID vuelo: *string* 32 bytes con *encoding* UTF-8
- Origen: *string* 3 bytes con *encoding* UTF-8
- Destino: *string* 3 bytes con *encoding* UTF-8
- Distancia: *short* de 2 bytes. Se completa con -1 en caso de que el vuelo no tenga distancia

El fin del envío de los vuelos al filtro se indica con un mensaje con byte con la letra ‘F’.

Envío de vuelos al filtro de vuelos rápidos

Los mensajes que se usan para enviar los vuelos al filtro de vuelos rápidos contienen:

- Identificador de mensaje para vuelo: un byte con un carácter que será la letra ‘V’
- Tamaño del batch: *unsigned short* de 2 *bytes*
- ID vuelo: *string* 32 bytes con *encoding* UTF-8
- Origen: *string* 3 bytes con *encoding* UTF-8
- Destino: *string* 3 bytes con *encoding* UTF-8
- Duración: *string* de 8 bytes. Se utiliza UTF-8 para el *encoding* del string de la duración, por ejemplo ‘PT11H54M’, y se completa con ceros al final, en caso de que la duración tenga menos de 8 bytes.

El fin del envío de los vuelos al filtro de vuelos rápidos se indica con un mensaje con byte con la letra ‘F’.

Envío de vuelos al filtro de estadísticas de precios

Los mensajes que se usan para enviar los vuelos al filtro de estadísticas de precios:

- Identificador de mensaje para vuelo: un byte con un carácter que será la letra ‘V’
- Tamaño del batch: *unsigned short* de 2 *bytes*
- ID vuelo: *string* 32 bytes con *encoding* UTF-8
- Origen: *string* 3 bytes con *encoding* UTF-8
- Destino: *string* 3 bytes con *encoding* UTF-8
- Precio: *float* de 4 bytes

El fin del envío de los vuelos al filtro se indica con un mensaje con byte con la letra ‘F’.

Resultados

Los mensajes para enviar los resultados, y el fin de los mismos, entre el servidor y los filtros tienen el mismo formato que los que se usan para enviarlos del servidor al cliente.

Protocolo de comunicación entre el filtro de estadísticas de precios y el calculador de promedios

Cada uno de los filtros de estadísticas de precios, enviará su promedio parcial en un mensaje con el siguiente formato:

- Promedio parcial: *float* de 4 bytes

- Cantidad de vuelos: *int* de 4 bytes

Una vez que hayan recibido todos los promedios parciales, el calculador de promedios le enviará, a cada uno de los filtros de precios, el promedio general en un *float* de 4 bytes

Listado de tareas a ejecutar y división entre integrantes

Las tareas a ejecutar para la configuración del sistema son:

Nombre tarea	Integrante que la desarrollará
Desarrollo Middleware	Guido Bergman
Configuración Docker y Docker-Compose	Guido Bergman
Implementación protocolo	Guido Bergman y Luis Waldman

La división de las tareas para las otras partes del sistema será:

Parte del sistema	Integrante que lo implementará
Cliente	Guido Bergman
Hander Cliente	Guido Bergman
Buscador de vuelos con distancias largas	Guido Bergman
Filtro escalas	Luis Waldman
Buscador vuelos rápidos	Luis Waldman
Calculador estadísticas vuelos costosos	Luis Waldman