Trabajo Práctico N°1

Ciclismo en América Del Norte

Curso: Sistemas Distribuidos I (75.74)

Alumno: Ignacio Iragui

Padrón: 105110

Cuatrimestre: 2023 1C

Índice

Scope

El sistema a desarrollar desea implementar un modelo distribuido que analiza los datos de viajes en bicicleta entre estaciones de distintas ciudades enviados por un cliente y que permite a partir de estas contestar a 3 consultas. Las consultas a contestar son:

- La duración promedio de viajes que iniciaron en días con precipitaciones mayores a 30mm.
- Los nombres de estaciones que al menos duplicaron la cantidad de viajes iniciados en ellas entre 2016 y el 2017.
- Los nombres de estaciones de Montreal para la que el promedio de los ciclistas recorren más de 6km en llegar a ellas.

La idea es poder realizar un sistema que sea capaz de escalar a futuro para estar preparado ante un eventual caso en el cual se aumente el flujo de entradas y clientes.

Vista Lógica

Primero debemos entender que disponemos de 3 estructuras “fundamentales”, los viajes, las estaciones y los reportes del clima. Estos serán mandados por parte del cliente como se puede observar en el diagrama de casos de uso de manera entera ya que no debe porque saber que información será utilizada. Pero desde el lado de los servidores los filtradores se encargan de filtrar solo la información requerida para ser almacenados ocupando el menor espacio posible.

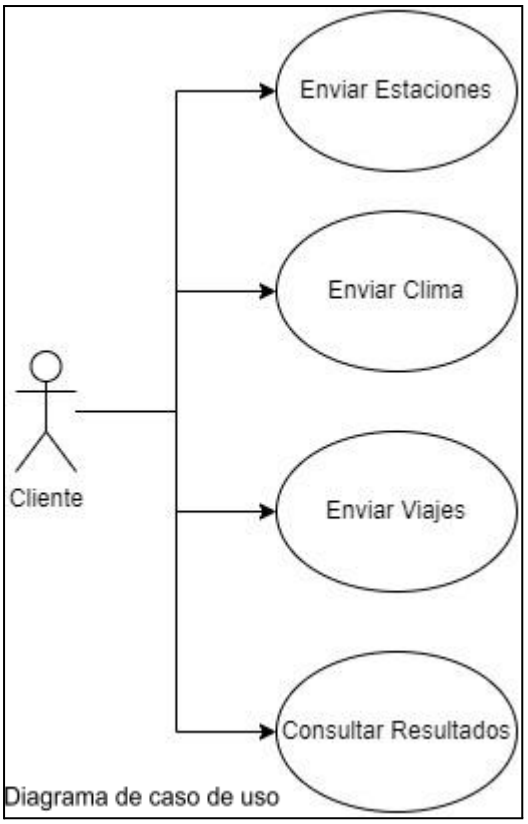


Diagrama de caso de uso

Para las estaciones nos quedamos con el código y el nombre ya que de las estaciones solo nos interesa devolver los nombres en lugar de los códigos en las consultas 2 y 3. Por otro lado también necesitamos la latitud y la longitud ya que para la consulta 3 debemos poder calcular la distancia entre estaciones.

Para los reportes del clima sólo conservamos la fecha para poder identificarlas y las precipitaciones totales para poder realizar la consulta 1.

En el caso de los viajes, estos datos no serán almacenados por lo cual no se cambia la clase. Simplemente se recibe un viaje entero y los filters envían a los joiners los números necesarios para que este actualice los resultados parciales.

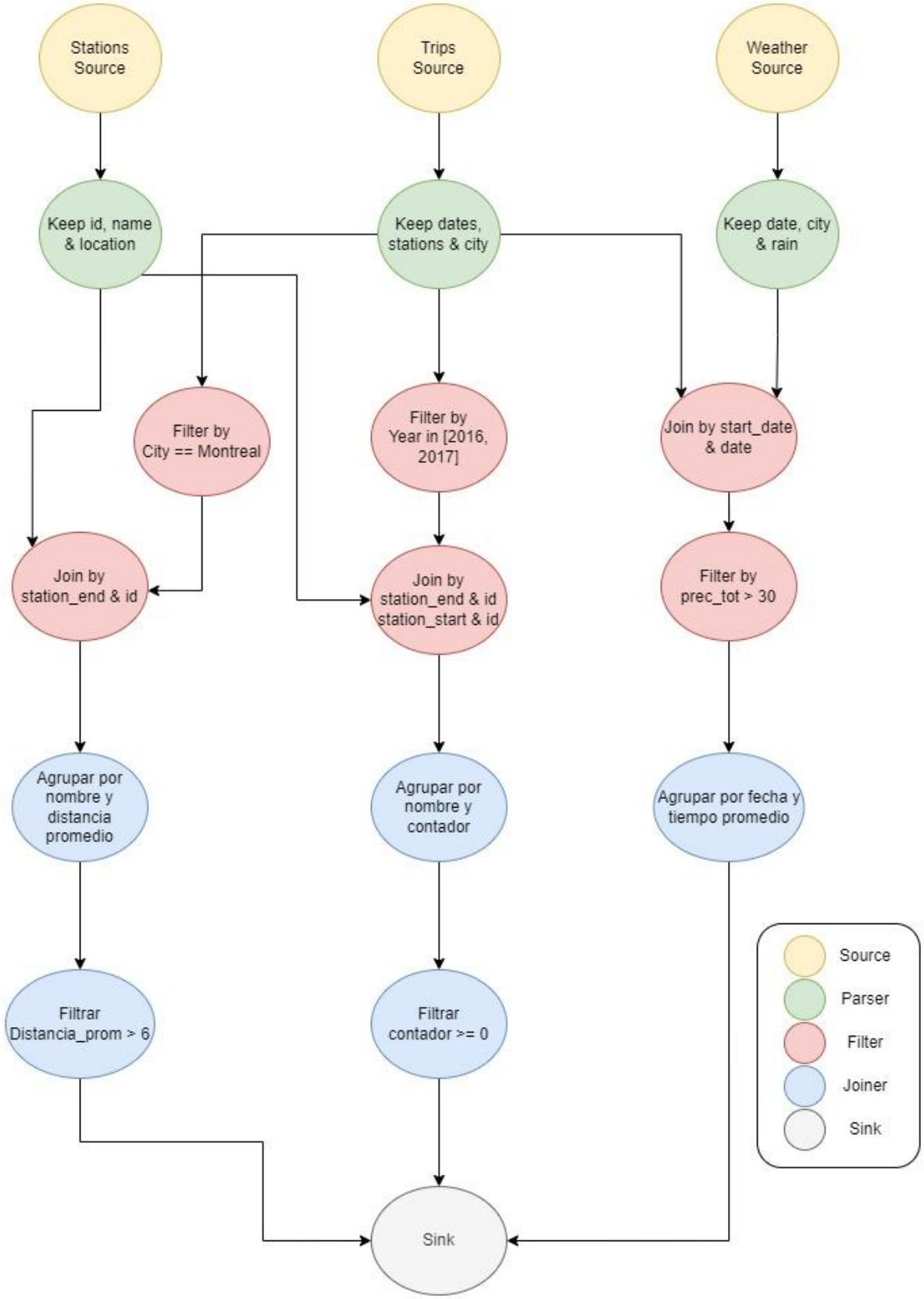
Para almacenar los resultados parciales el joiner requiere almacenar datos distintos para cada consulta.

Para la consulta 1, al ser un promedio se debe guardar el promedio actual y la cantidad de datos cargados así al recibir uno nuevo se realiza un promedio ponderado y se actualiza el total cada uno de estos datos en un diccionario con clave las fechas.

Para la consulta 2, se debe guardar un diccionario que almacene un valor parcial que sirve para saber si se duplicó o no todavía. Este valor se forma arrancando en 0 y restando 2 cada vez que llegue un viaje de 2016 y sumando 1 para un viaje de 2017. De esta forma si al finalizar el valor es mayor o igual a 0 la cantidad de viajes de 2017 debe haber sido como mínimo el doble de los de 2016.

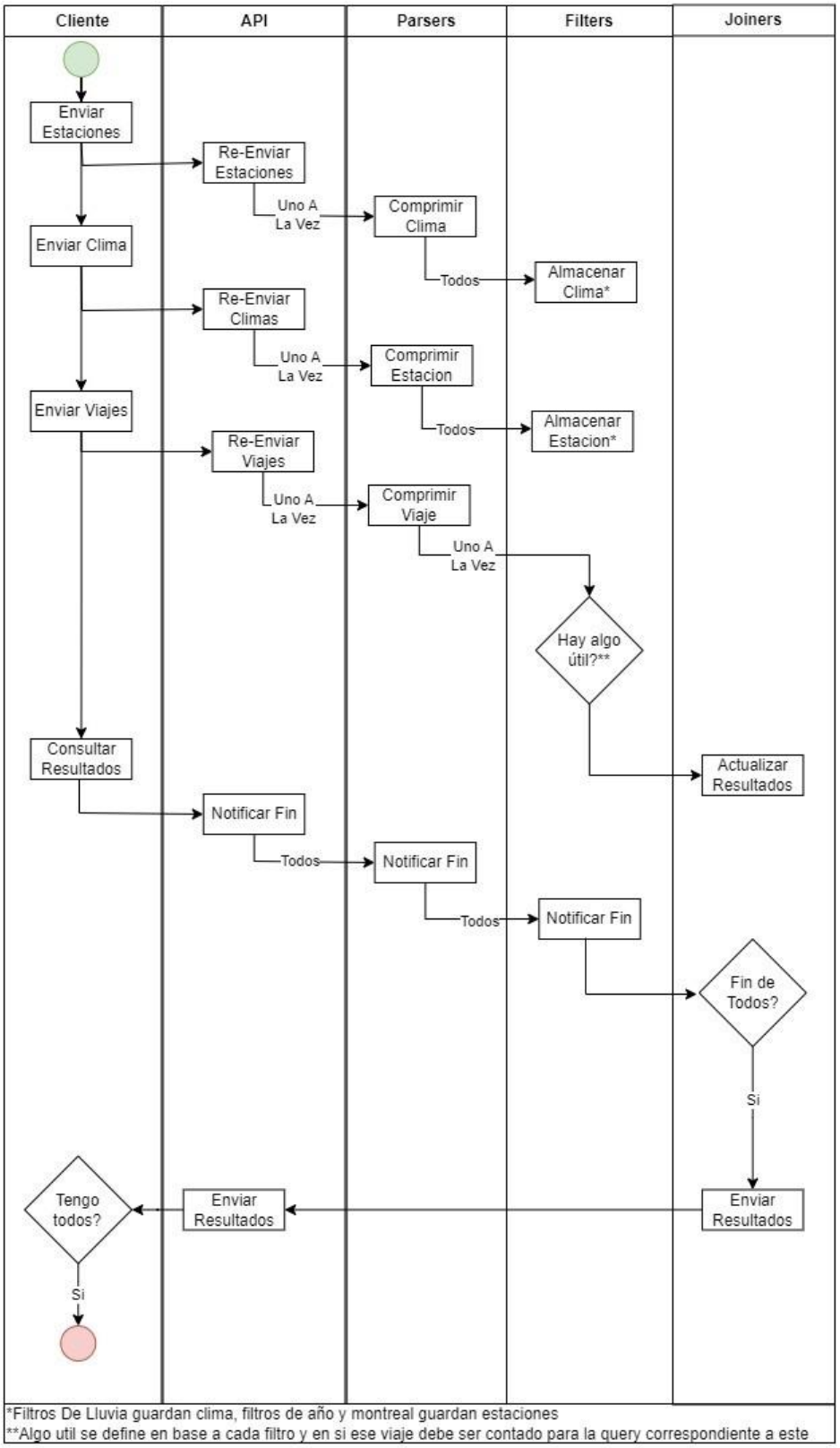
Para la consulta 3, necesitamos un diccionario para distinguir entre estaciones y luego realizamos el promedio como en la consulta 1.

Para poder visualizar todo esto en un unico gráfico, se incorpora un DAG que lo presenta.



Vista de Procesos

Para entender los procesos que ocurren primero debemos observar el diagrama de actividades que explica el sistema.



Vemos que tenemos 5 entidades todas que actúan por separado, el cliente, la API que actúa de middleware, los parsers, los filters y los joiners. Entre estos miembros debemos destacar que los filters y los parsers pueden ser n en paralelo.

Todas las entidades podrían estar localizadas en distintas máquinas por lo que no comparten ningún recurso en memoria, pero igual debemos asegurarnos la sincronización. Esto significa planificar la comunicación para evitar 2 inconvenientes.

1. Doble filtrado de viajes: Para esto al recibir un batch de viajes el parser mediante la utilización de rabbitMQ enviará el grupo a registrar solamente a UN filter.
2. Devolución de resultados sin estar todos filtrados: Para esto el joiner debe esperar a recibir el fin por parte de todos los filters correspondientes

Ahora debemos entender los mensajes compartidos entre los distintos procesos.

El cliente al middleware le pasará toda la información que lea del archivo línea a línea aclarando previamente a que ciudad corresponde lo que está enviando. Para poder completar las query a medida que va enviando los viajes, el cliente deberá primero enviar las estaciones y el clima para que se pueda realizar la comparación posteriormente.

El middleware funciona como un comunicador entre ambas partes por lo cual a los workers no le modificará el mensaje, solo le agrega información para que los parsers sepan el largo y sepan si se trata de una línea de archivo o del aviso de fin. Para mejorar el rendimiento se debería implementar un envío mediante batch de líneas.

Los parsers una vez leída la información reducen la información a aquella que sea necesaria para el proceso de filtrado posterior, en base a lo ya dicho en la vista lógica y se la envían a los filtros. Los filtros solo reciben aquella información que ellos necesiten, es decir, el filtrador por lluvia no le interesa la información de las estaciones por la cual nunca la recibe.

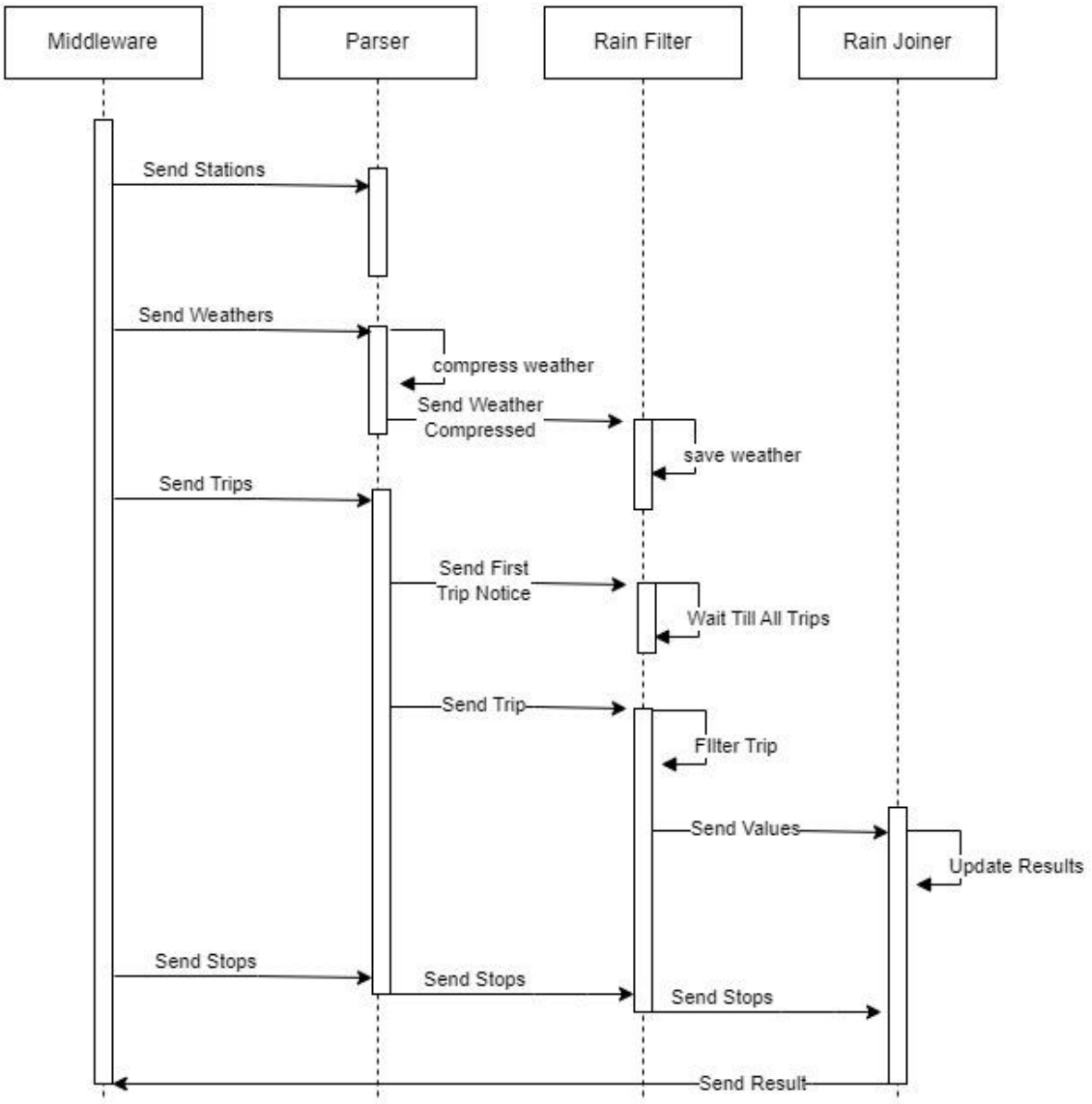
A su vez debe informar el fin de la información para que los filtros puedan saber cuando comenzar a procesar viajes y el fin de los viajes para que se le pueda informar a los joiners, para esto se debe utilizar un sistema de identificación de anuncios de fin para que el filter puede estar seguro que todos los parsers concluyeron su trabajo.

Luego entre los filter y los joiners debemos minimizar la información compartida para agilizar el proceso, por lo que en caso de que el filter encuentre que el viaje es “util” solo le pasar las claves y los valores correspondientes para actualizar el resultado

Caso	Clave	Valor
Rain Query	Fecha	Duración
Year Query	Estación y Nombre	-2 si es 2016 y 1 si es 2017
Montreal Query	Estación y Nombre	Distancia

Por último para enviar las query a cada uno de los joiners se las envían al server mediante RabbitMQ y este se las envía directo al cliente hasta enviar las 3.

Esto se puede visualizar también en un diagrama de secuencia como el presente aquí abajo, este caso representa el proceso de un rain filter.



Vista De Desarrollo

Desde un punto de vista de implementación podemos ver que requerimientos funcionales tiene cada entidad del sistema. Vamos a ir en el orden de cómo se van enviando los mensajes que también implican un nivel de complejidad de menor a mayor.

En primer lugar el cliente, entendemos que este debe ser un cliente sencillo con poco entendimiento del sistema por lo cual solo deberá manejar sockets para comunicarse con el middleware y un lector de archivos para poder enviar los datos.

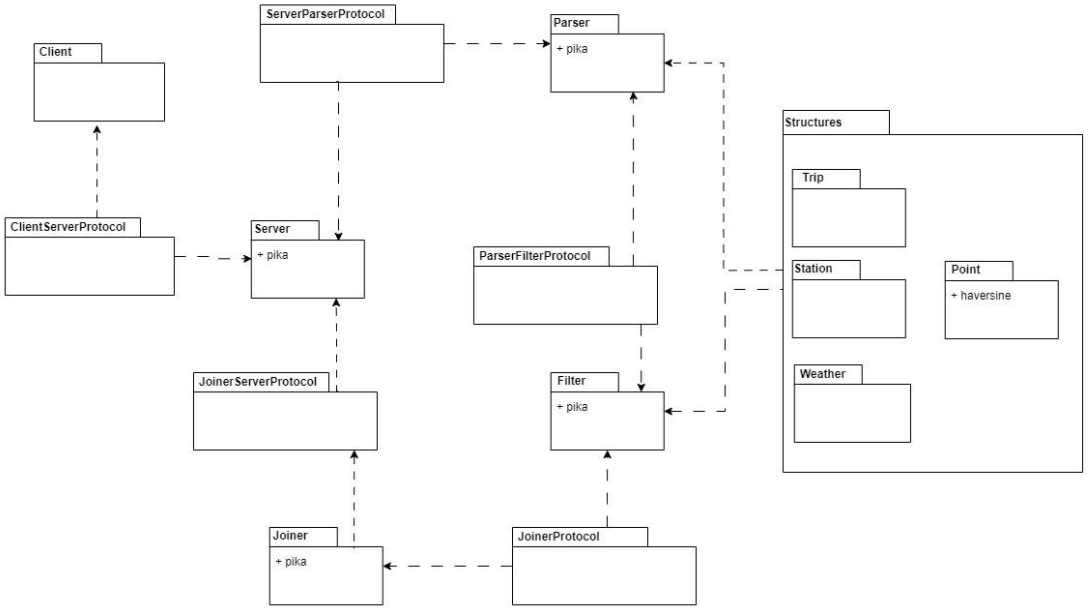
Luego el middleware debe tener un socket para leer del lado del cliente y con la cola que derivara a los parsers y cuando necesite leer el resultado final de los joiners.

Los parsers por su parte simplemente deben tener acceso a la librería de pika para poder comunicarse con el middleware para recibir elementos y con los filters para enviarlos-

Los filters por su parte al igual que los parsers necesitan de pika para comunicarse, pero también necesitaran tener acceso a la librería de haversine para calcular distancia y a la clase creada propia de filtro.

Por último el joiner tiene un caso muy similar a los filters solo que en este caso no necesitamos de la librería para calcular distancias. Simplemente pika y la librería propia para actualizar resultados parciales paso a paso.

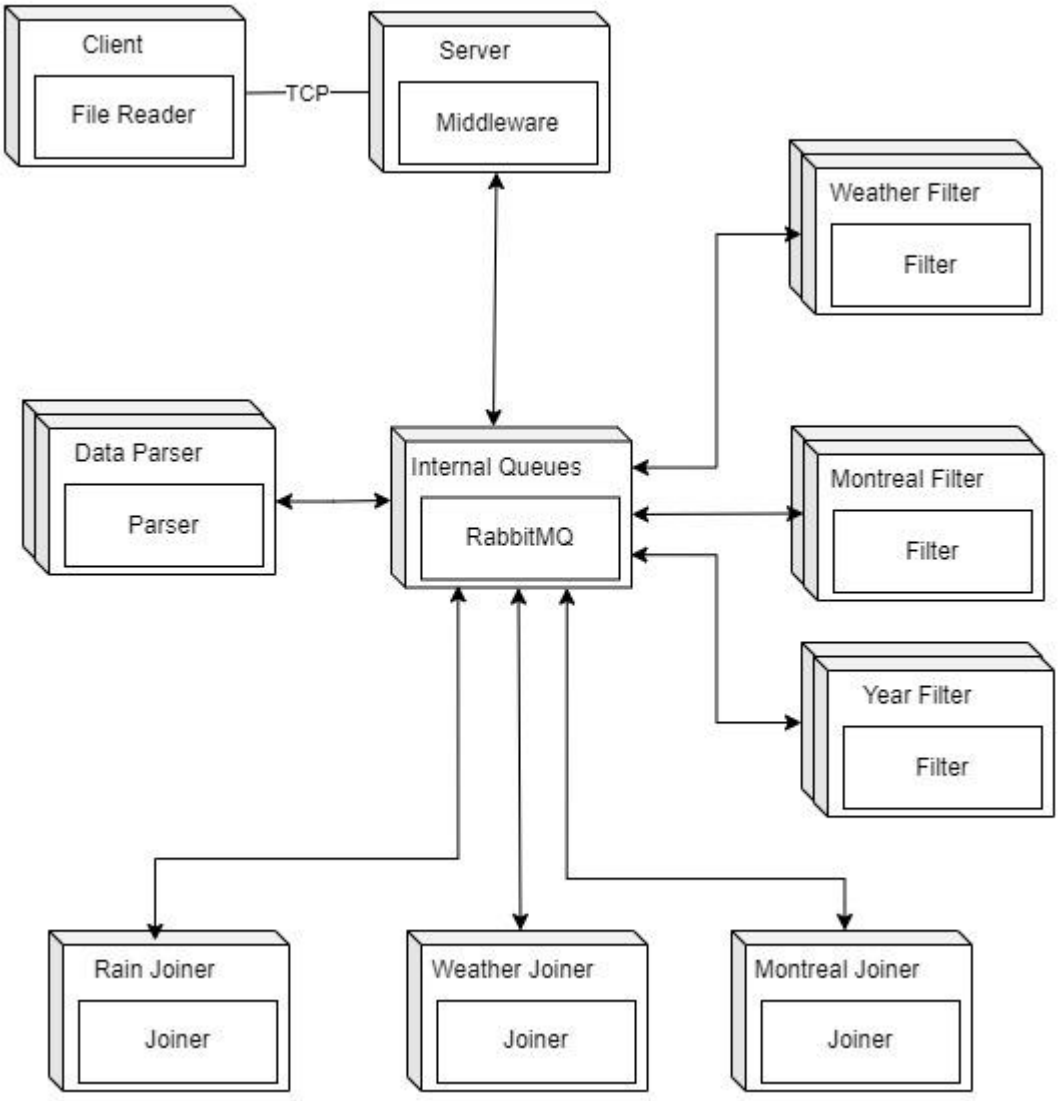
Todo esto se puede visualizar en el diagrama de paquetes presente en la parte inferior.



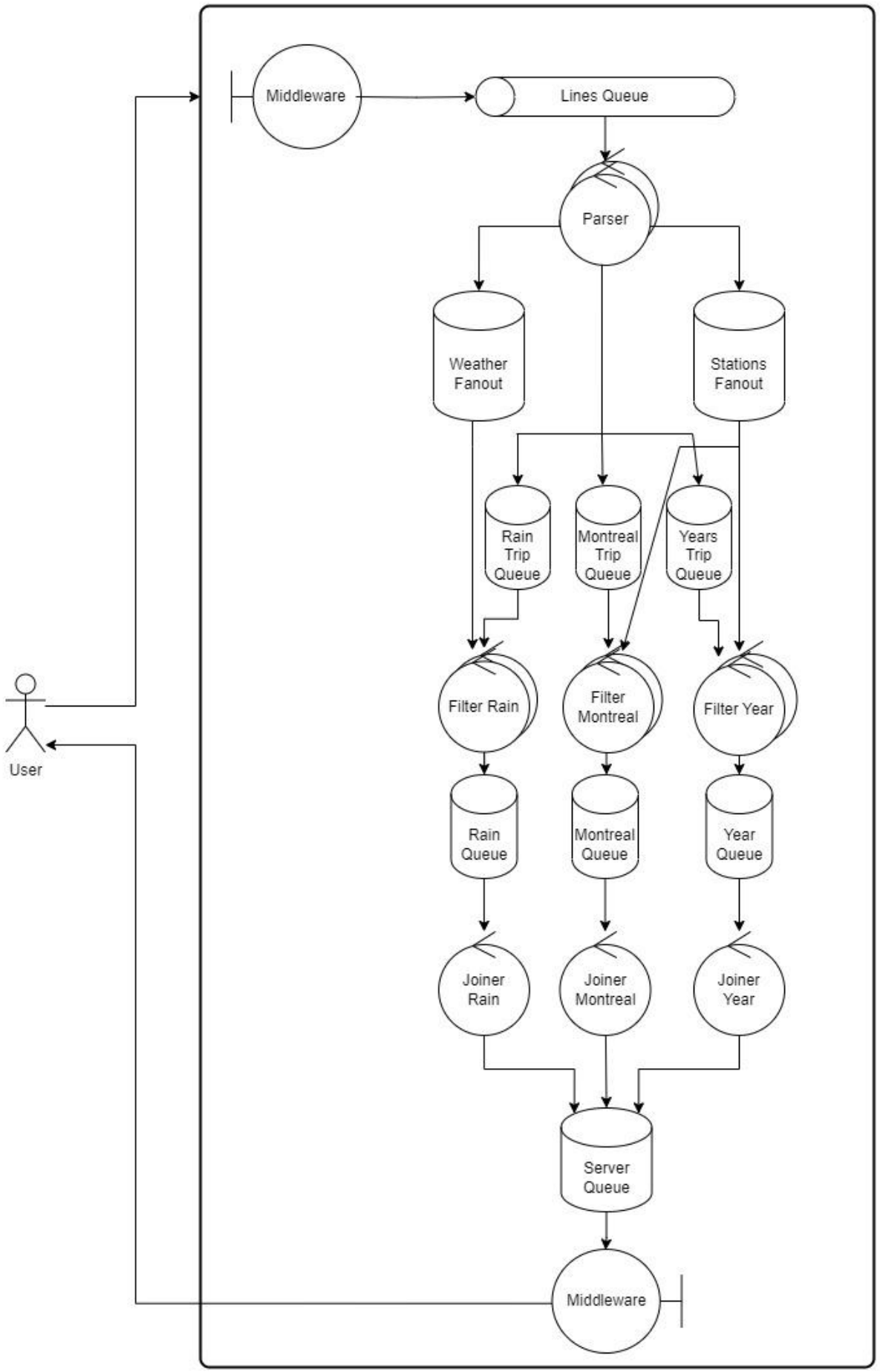
Vista Física

Para poder visualizar la vista física se crearon dos diagramas, un diagrama de despliegue y otro de robustez.

En primer lugar podemos ver el diagrama de despliegue que nos muestra donde va a estar corriendo cada parte de nuestro sistema.



En segundo lugar debemos observar el diagrama de robustez que nos sirve para poder visualizar dónde está la escalabilidad de este sistema. En este caso podemos ver como nuestra escalabilidad viene por parte de los filters y los parsers.


Middleware
Lines Queue
Parser
Weather Fanout
Stations Fanout
Rain Trip Queue
Montreal Trip Queue
Years Trip Queue
Filter Rain
Filter Montreal
Filter Year
Rain Queue
Montreal Queue
Year Queue
Joiner Rain
Joiner Montreal
Joiner Year
Server Queue
Middleware
User

A su vez en el diagrama de robustez podemos ver como ninguna información es almacenada en archivos y todo transcurre en memoria ya que los únicos datos “pesados” son los de los viajes pero estos nunca deben ser almacenados, mientras que las estaciones y el clima pueden ser almacenados en memoria ya que no tienen tanto volumen y al estar en memoria simplificará el acceso aumentando la velocidad. De tener un volumen mayor se deberían ir creando archivos a la par a medida que se reciben los datos y luego leer de estos archivos.