FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

[75.74] SISTEMAS DISTRIBUIDOS I

1ER CUATRIMESTRE 2023

TP Individual - Middleware y Coordinación de procesos

Índice

1. Introducción

En el siguiente documento se presenta la arquitectura de Bike Rides Analyzer, un sistema distribuido de análisis de viajes en bicicleta de las ciudades de Montreal, Toronto y Washington.

El sistema esta compuesto de por una aplicación cliente, un broker de RabbitMQ y un servidor con múltiples servicios destinados a operar y transformar los datos para otorgar métricas agregadas al cliente.

2. Vista Física

2.1. Robustez

Los diagramas de robustez están divididos de manera tal de ilustrar los diferentes nodos que intervienen en la resolución de cada una de las consultas.

Para todos los casos, los datos son inicialmente ingestados al sistema por el cliente. Un patrón que notaremos tanto para los filtros, los consumers, los joiners, aggregators y appliers es que podrán ser escalados para soportar un mayor volumen de carga. Esto es así debido a que los mismos no comparten ningun tipo de estado entre ellos, de manera tal que se puede aprovechar la distribución de carga para generar paralelismo en el sistema.

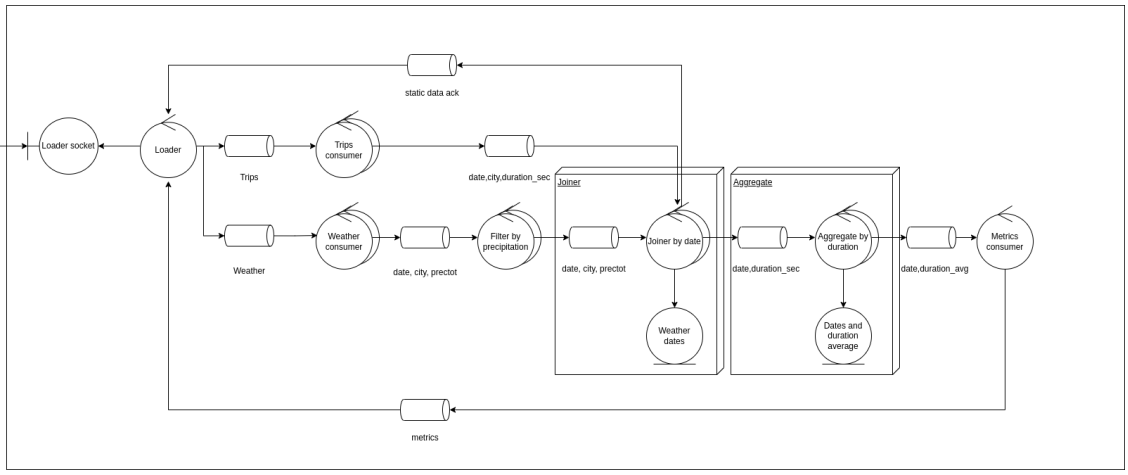

Figura 2.1: Diagrama de robustez que incluye a los nodos relevantes en el calculo del promedio de duración de viajes en dias con altas precipitaciones

Para esta consulta, cada worker de tipo Weather Consumer se encargará de extraer los datos del clima relevantes para la consulta y enviarlos al grupo de filtros Filter By Precipitation. Luego de ser procesada esta data estática, se guardará en el Joiner. Los joiners podrán ser replicados debido a que la carga estática será broadcasteada a ellos. Al llegar la data de trips, cada uno hará la operación de join de manera paralela. Los nodos Aggregate by Duration recibiran de manera independiente datos de la etapa de join para poder agregarlos (en este caso calcularán un rolling average). Estos a su vez irán guardando los datos agregados en

memoria. Finalmente, al recibir el mensaje de End Of Stream, los datos agregados serán enviados al Metrics Consumer, que será el encargado de combinar la salida de cada Aggregator y devolversela al Loader.

Se tiene a continuación el diagrama de robustez con los nodos que intervienen en la resolución de la segunda consulta.

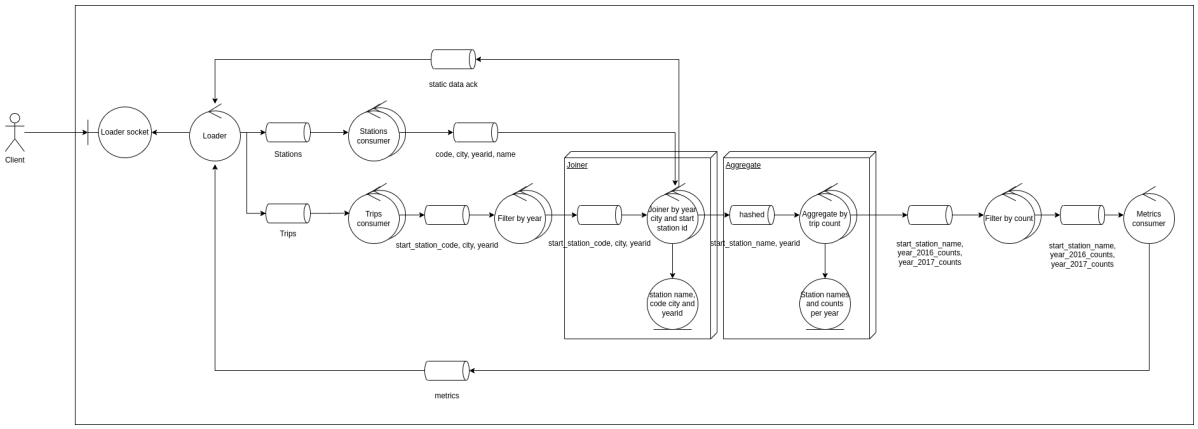

Figura 2.2: Diagrama de robustez que incluye a los nodos relevantes en el calculo de las estaciones que duplican las salidas entre 2016 y 2017

Como en la consulta anterior, los grupos de nodos que se presentan tienen la misma responsabilidad, agregar, agrupar y filtrar. Estos son desplegables de manera independiente y escalables, siguiendo la misma lógica.

Por último, se muestra el diagrama de robustez relevante a la resolución de la tercera consulta.

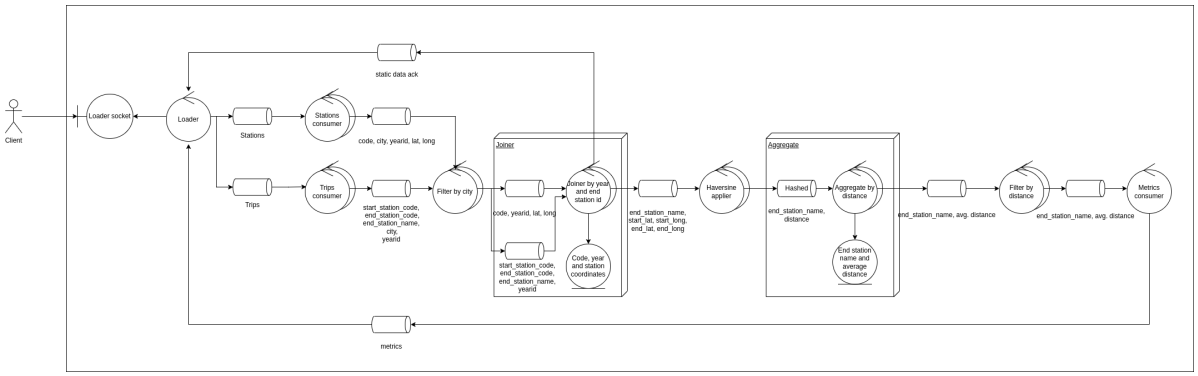

Figura 2.3: Diagrama de robustez que incluye a los nodos relevantes en el calculo de las estaciones con media de viaje mayor a 6km en Montreal

En este caso, se agrega otro grupo de nodos encargados de calcular distancias, Haversine Applier. Nuevamente replicables debido a que no comparten ningun tipo de estado. En este caso vemos además como se pueden reutilizar los nodos para filtrar datos sin importar el origen de los mismos, como es el caso del grupo de Filter By City. Si se quisiera ahora agregar otra ciudad a la consulta, bastaría con cambiar la configuración de estos filtros para que tome en cuenta a esas nuevas ciudades.

2.2. Despliegue

A continuación se muestra el diagrama de despliegue del sistema.

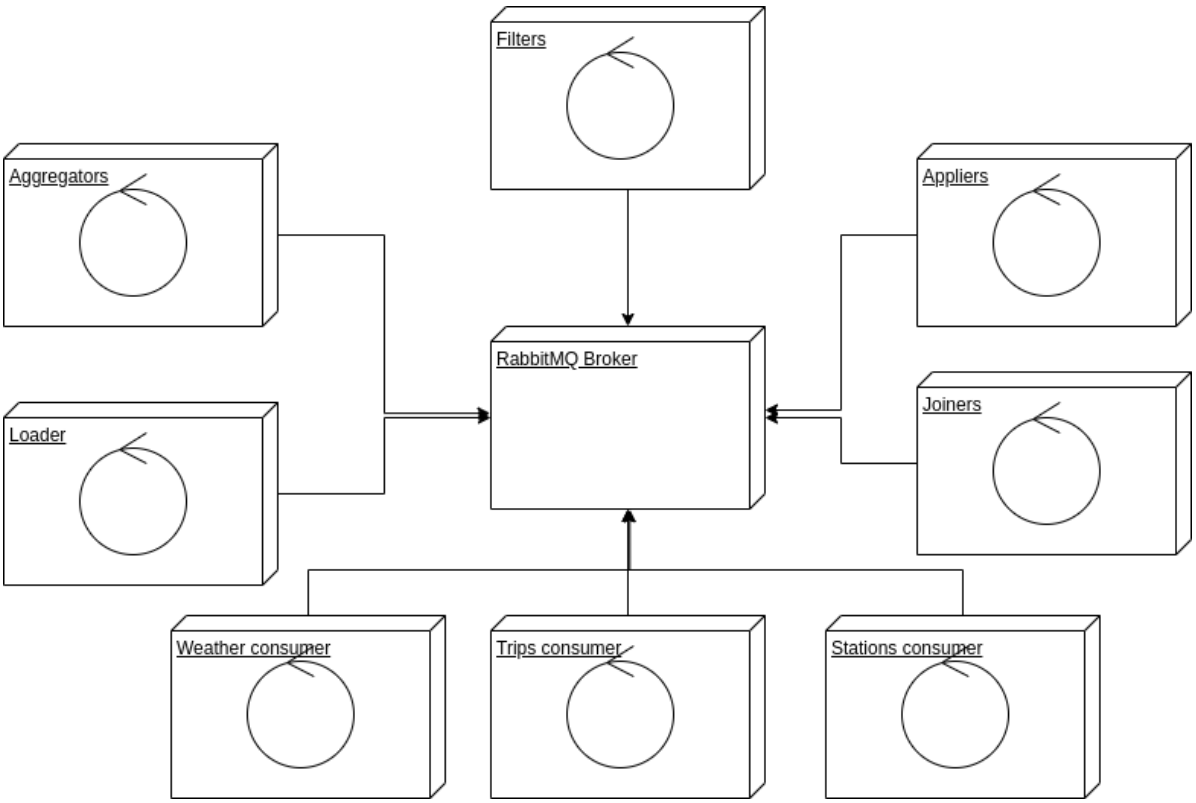

Figura 2.4: Despliegue de Bike Rides Analyzer

Se puede apreciar una fuerte dependencia del sistema al broker de RabbitMQ. Esto es asi ya que todos los servicios utilizan al broker para poder comunicarse entre si. El Loader, además, funciona como servicio de frontera con el cliente. Para comunicarse utiliza un socket TCP/IP, de tal manera que se aisla al cliente del modelo de comunicación interno del sistema. Los nodos de tipo Filter, Aggregator y Joiner se agrupan en un único componente genérico simplemente para no cargar la figura, pero cada uno de ellos podrá ser desplegado en un nodo independiente.

3. Vista de Desarrollo

3.1. Diagrama de Paquetes

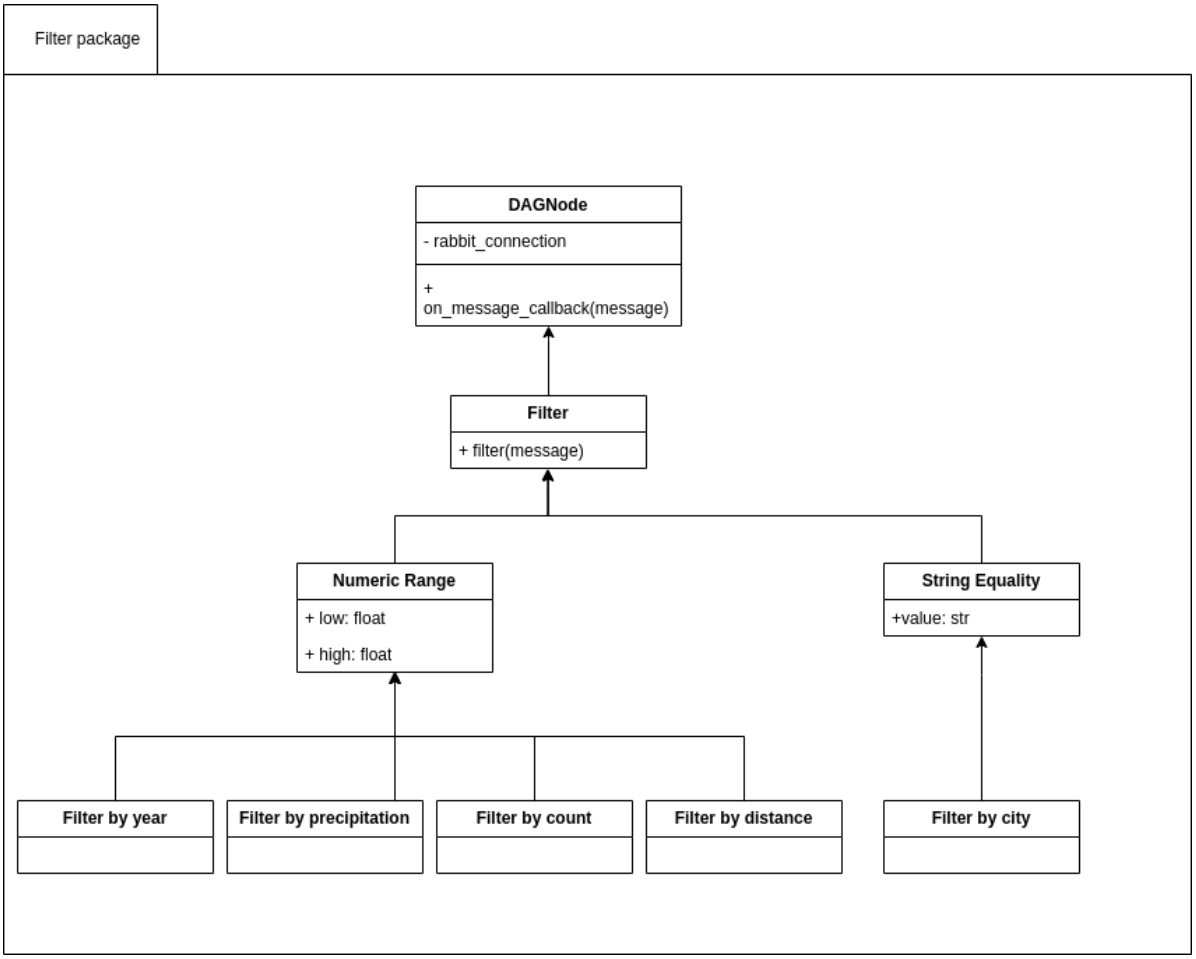

Figura 3.1: Diagrama de paquetes para los Filters

En la figura se puede ver a los componentes que forman al paquete de Filters. Cada filter heredará de un DAGNode, responsable de tener una conexión con Rabbit. A su vez se distinguirán en dos tipos de filtros, aquellos que checkeen igualdad contra un dato de tipo string y aquellos que filtren en un rango numérico.

4. Vista de Procesos

4.1. Diagrama de Actividades

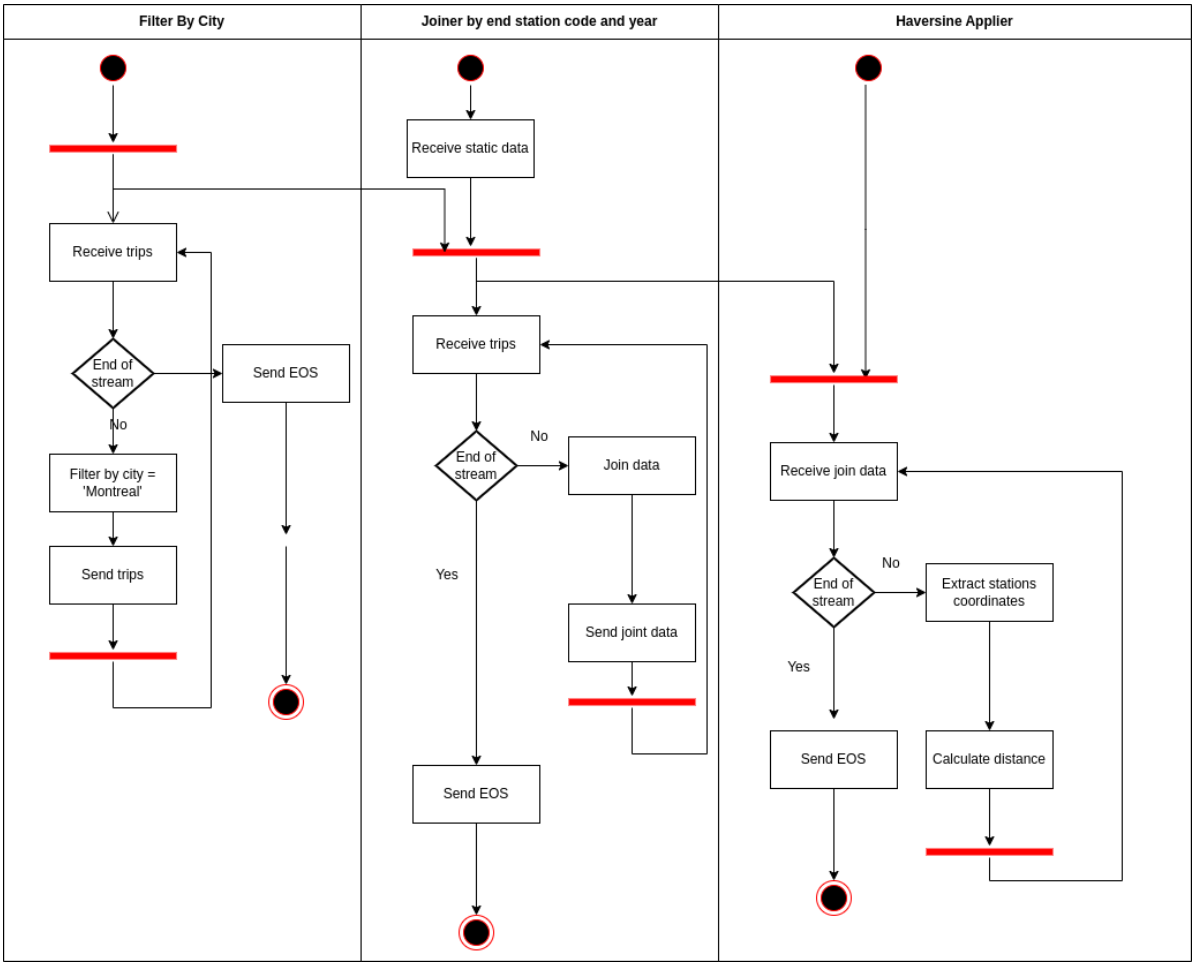

Figura 4.1: Diagrama de actividades, filtro, junta y calculo de distancias

En el siguiente diagrama de actividades, se muestra parte de la resolución de la tercera consulta. Vemos que las diferentes actividades, filtro, junta y calculo de distancias, son realizadas en paralelo. A medida que el filtro selecciona aquellos viajes de la ciudad de Montreal, estos son enviados al Joiner, que los recibe y hace la junta con sus datos estáticos de estaciones, y al realizar la junta envía los datos al Haversine Applier, que se encargará de recibir los datos de la junta, extraer las coordenadas de estaciones de inicio y final y calcular las distancias. Una vez llegado el end of stream, se procederá a enviar la información de una etapa en otra, de esta manera sincronizando a los procesos sobre el final del stream.

4.2. Diagrama de Secuencia

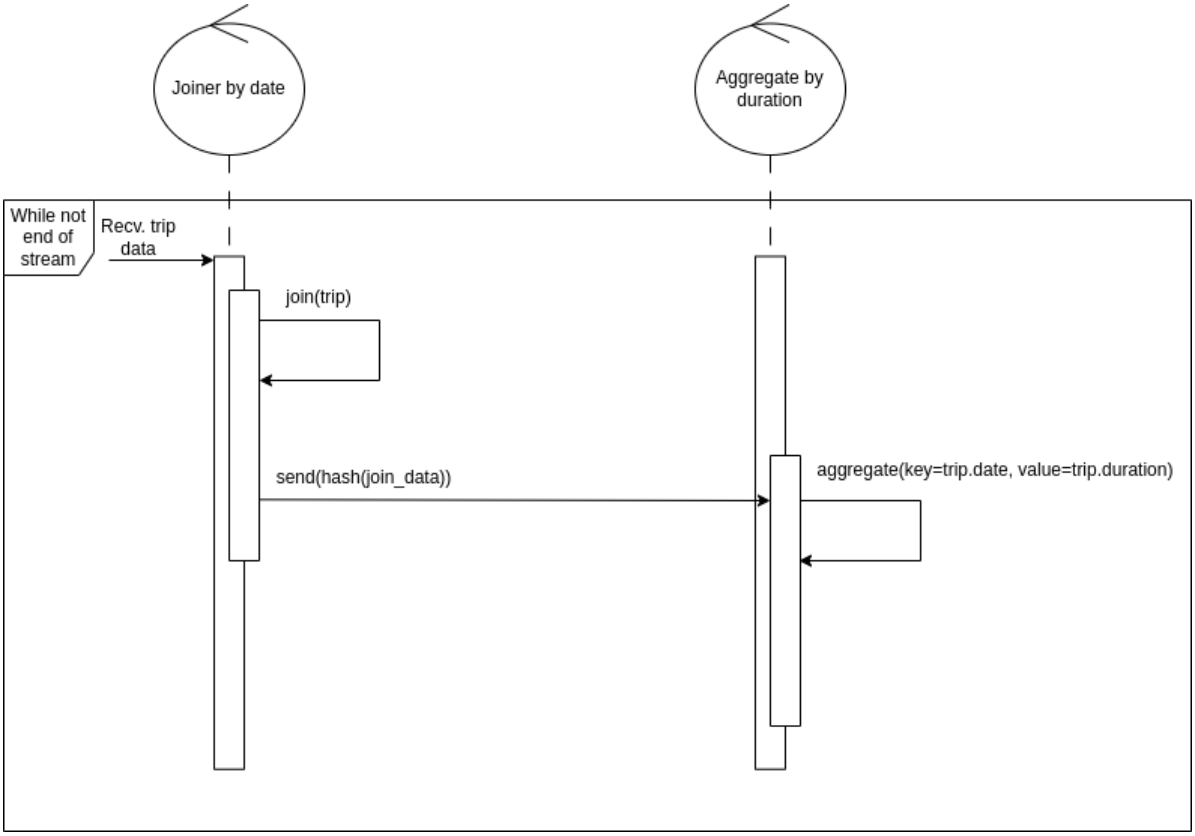

Figura 4.2: Diagrama de secuencia, hashing de claves para agregadores

Una de las caracteristicas del sistema es que se separa la etapa de junta de la etapa de agregación. Esto fue pensado asi de manera tal de evitar que la etapa de join sea un cuello de botella. Ahora, como en particular la etapa de agregación puede ser replicada, para estar agregando las particiones de datos bien, se necesita algún método que permita agregar estas particiones de manera tal de no generar datos inconsistentes entre corridas. Esto es posible gracias a que al momento de enviar la data joineada, los Joiners se encargan de hashear las claves para que los Aggregators reciban según ese hash. Esto permitirá además distribuir la carga de manera equitativa, contando con que el hash es consistente y tiene un output pseudo-random. Vemos el caso de la primera consulta, en donde el Joiner routea datos según la clave de fecha. Los datos que sean de fechas iguales terminarán siendo agregados por el mismo Aggregator.

5. DAG

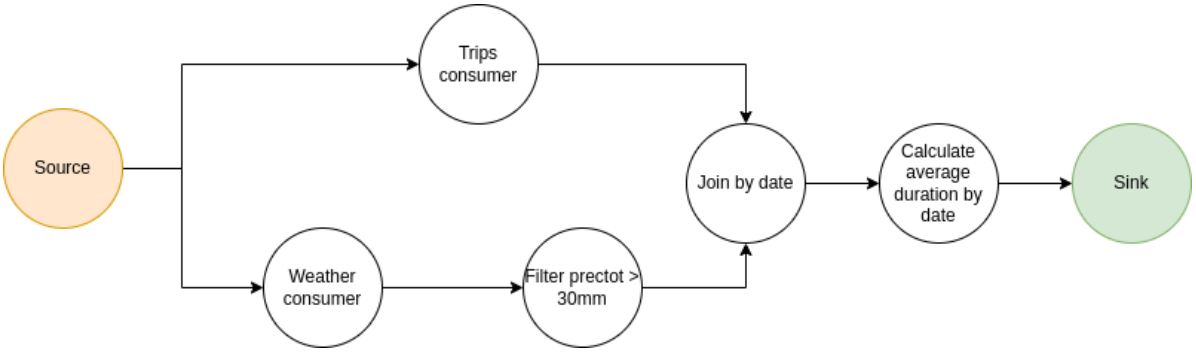

Figura 5.1: DAG, consulta de duración de viaje en dias lluviosos

En este DAG se puede apreciar como se da el flujo de información relevante a la primera consulta. Tanto la información de viajes como de clima fluye desde el source. Los datos del clima serán filtrados por su campo de precipitación (dando lugar a un Filter por ese campo como nodo) y llegarán a una etapa de Join, en donde se agruparán con los datos de viajes por fecha. A medida que se vayan agrupando, se calculará un rolling average de dicha duración de viaje, y con ese dato tendremos resuelta la consulta.

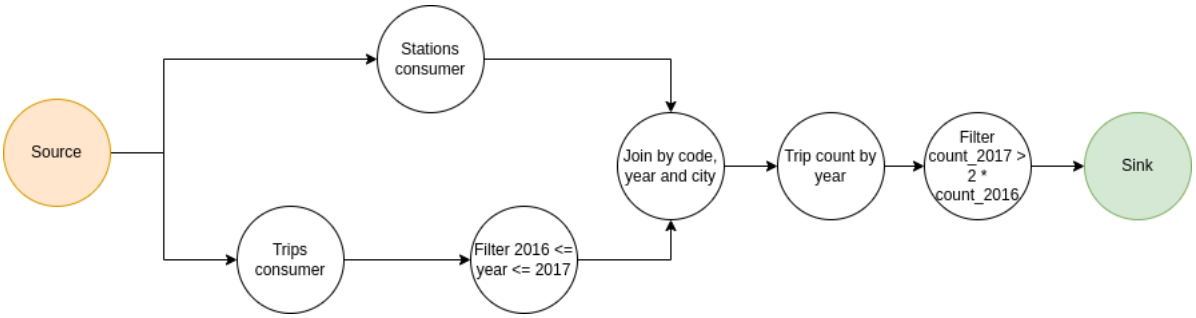

Figura 5.2: DAG que muestra el flujo de información para la consulta de viajes por año entre 2016 y 2017

En este caso, se consumiran datos tanto de estaciones como de viajes. Los datos de viajes serán filtrados por año, para luego ser agrupados con los datos de estaciones por el código de estacion de inicio, el año y la ciudad. A medida que los resultados de la junta se materialicen, serán delegados a la etapa de agregación, que en este caso tendrá un contador para 2016 como para 2017, y podrá entonces acumular esas cantidades para cada estacion de salida. Una vez agregados los datos, se filtrarán aquellos que hayan duplicado la cantidad de viajes en 2017, descartando aquellos que no hayan tenido algun viaje en 2016.

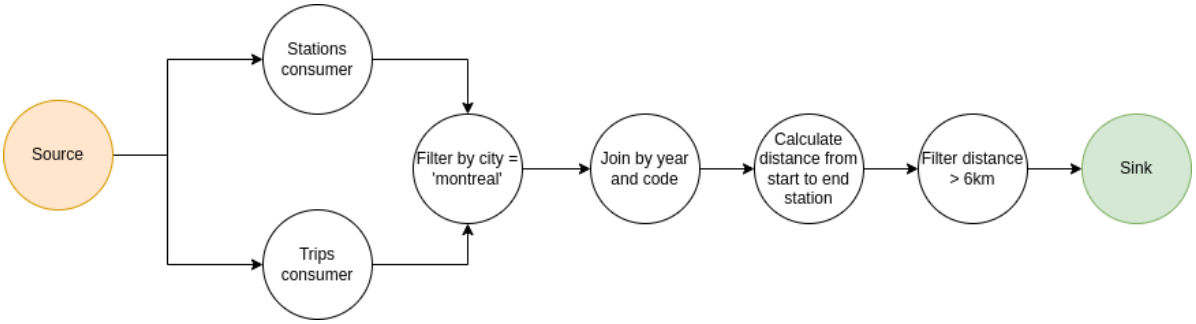

Figura 5.3: DAG que muestra el flujo de información para la consulta de estaciones de llegada con promedio de distancia recorrida mayor a 6km, para la ciudad de Montreal

De la misma manera que en el DAG anterior, se consumiran datos de viajes como de estaciones, que serán primero filtrados por ciudad (en este caso Montreal). Una vez filtrados, se agruparán por los códigos de estaciones y año (debe estar por la naturaleza de los datos) para entonces obtener coordenadas de origen y destino, además del nombre de la estación. Como paso siguiente, se calculará la distancia utilizando Haversine, para luego filtrar y quedarnos con aquellas estaciones cuya distancia promedio para llegar desde cualquier otra sea mayor a 6km.