Documento de arquitectura: Bike Rides Analyzer

Introducción

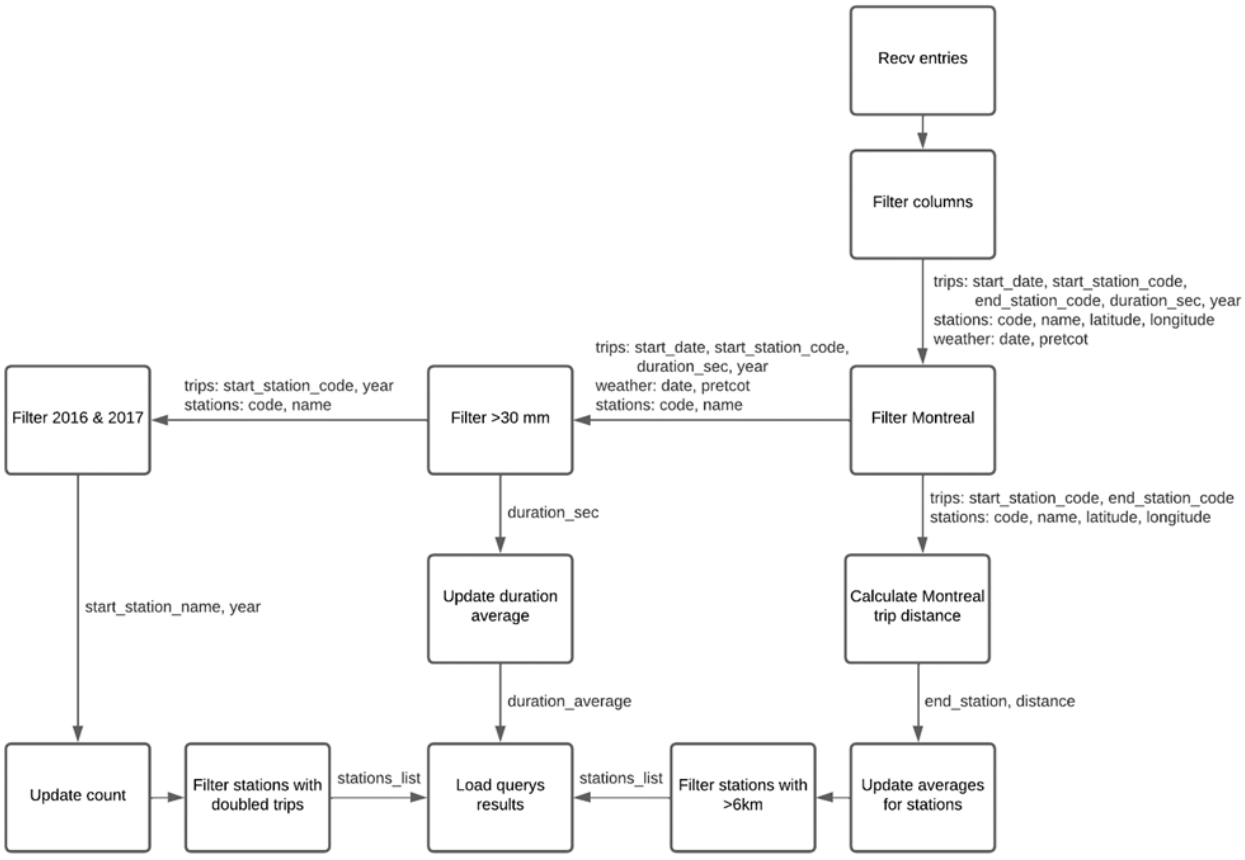
El presente documento describe la arquitectura de un sistema distribuido para el procesamiento y análisis de un conjunto masivo de datos de viajes en bicicleta en ciudades como Toronto, Montreal y Washington. El sistema está diseñado para manejar grandes volúmenes de datos, compuestos por tres tablas principales: la tabla de viajes, que contiene información detallada sobre los viajes en bicicleta realizados; la tabla del tiempo, que contiene datos meteorológicos diarios; y la tabla de estaciones, que contiene información sobre las estaciones de inicio y llegada de los viajes.

La arquitectura del sistema se basa en una arquitectura distribuida y utiliza tecnologías de contenedores de Docker y colas de RabbitMQ para la gestión eficiente de los datos. El sistema se compone de varios procesos, cada uno ejecutándose en contenedores de Docker, que trabajan en conjunto para procesar y analizar los datos de manera escalable y eficiente. El sistema consta de un cliente que envía los datos a diferentes colas de RabbitMQ, y otros procesos que se encargan de filtrar la información y ejecutar diferentes análisis sobre los datos filtrados, como el cálculo de la media de tiempo de los viajes en días de lluvia con precipitaciones superiores a 30 mm, entre otros posibles análisis.

El objetivo de este documento es proporcionar una comprensión clara de la arquitectura del sistema distribuido para el procesamiento y análisis de datos de viajes en bicicleta, así como brindar una guía para el desarrollo, despliegue y mantenimiento del sistema. A través de este documento, se pretende facilitar la comprensión de la estructura y funcionamiento del sistema, así como su capacidad para manejar grandes volúmenes de datos y ejecutar análisis complejos de manera eficiente.

DAG

Modelamos el problema de procesamiento de datos en el siguiente DAG. Los nodos indican tareas y las aristas flujos de datos. Podemos ver en las aristas los datos que son transmitidos entre tareas del sistema para su procesamiento distribuido.



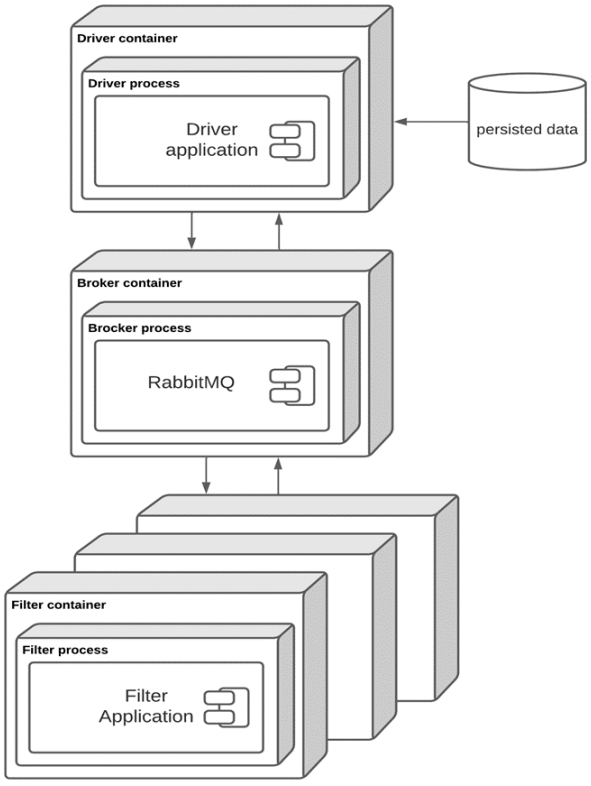
Vista Física

El sistema distribuido será implementado en un entorno multicomputadora utilizando Docker, una plataforma de contenedores que permite la creación, despliegue y ejecución de aplicaciones de forma eficiente y portátil. Docker Compose será utilizado para lograr un despliegue rápido y sencillo en diferentes contenedores que servirán a los diversos procesos del sistema distribuido.

En este entorno, se utilizará un proceso llamado "driver" que se encargará de cargar los datos al clúster de procesamiento y recibir los resultados de las consultas procesadas. Además, se implementará un servicio de cola de mensajes Rabbit MQ con el objetivo de desacoplar la comunicación entre las entidades del sistema y mantener adecuadamente aisladas las capas del sistema. Esta arquitectura de colas de mensajes permitirá una comunicación asincrónica y eficiente entre los diferentes componentes del sistema, mejorando la escalabilidad y la robustez del sistema distribuido.

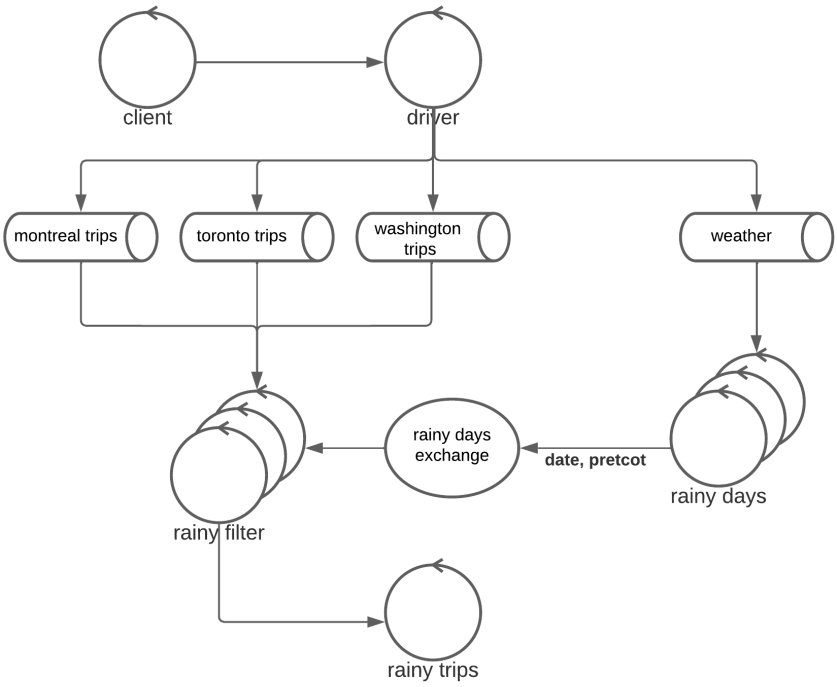
Además, se implementarán una serie de procesos de filtro que se encargarán de ejecutar el cómputo distribuido en función de los datos recibidos. Estos procesos de filtro serán

replicables y escalables, lo que permitirá aumentar la capacidad de procesamiento del sistema a medida que sea necesario para manejar mayores volúmenes de datos.
En resumen, el sistema distribuido se desplegará en un entorno multicomputadora utilizando Docker y Docker Compose para un despliegue rápido y sencillo en contenedores. Se utilizará un proceso “driver” para cargar los datos y recibir los resultados de las consultas, y se implementará un servicio de cola de mensajes Rabbit MQ para mejorar la comunicación entre las entidades del sistema. Además, se utilizarán procesos de filtro replicables y escalables para ejecutar el cómputo distribuido. Esta arquitectura proporcionará una solución eficiente, escalable y robusta para el procesamiento de datos en el sistema distribuido.

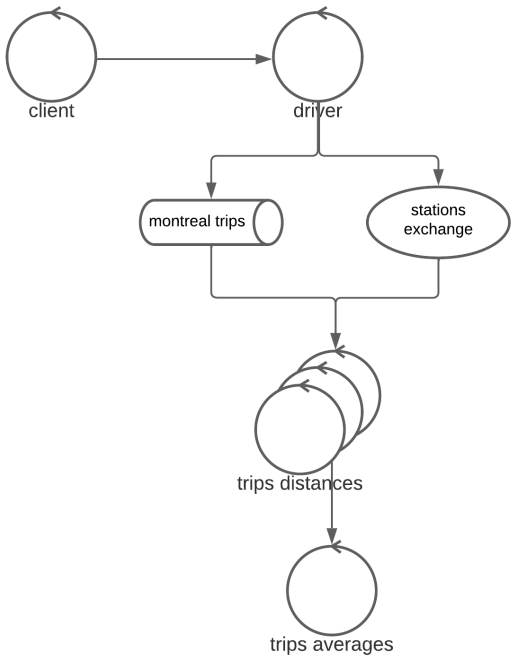


En el diagrama de robustez de nuestro sistema distribuido, se pueden observar varios procesos que se ejecutan en paralelo, lo que permite un procesamiento eficiente de los datos. Estos procesos están diseñados para realizar diferentes tareas específicas. Por ejemplo, algunos procesos se encargan exclusivamente de filtrar campos y registros de los datos, garantizando que solo la información relevante sea procesada. Otros procesos se dedican a realizar cálculos y análisis en función de los datos que reciben de los filtros, generando resultados valiosos para el sistema.
Una de las ventajas clave de esta configuración es su alta escalabilidad. Es posible replicar los filtros de manera horizontal, lo que significa que se pueden agregar más instancias de filtros a medida que se requiera para aumentar la capacidad de procesamiento del sistema. Esto permite adaptar el sistema a las demandas cambiantes de procesamiento de datos, lo que lo hace altamente flexible y capaz de manejar grandes volúmenes de información de manera eficiente.

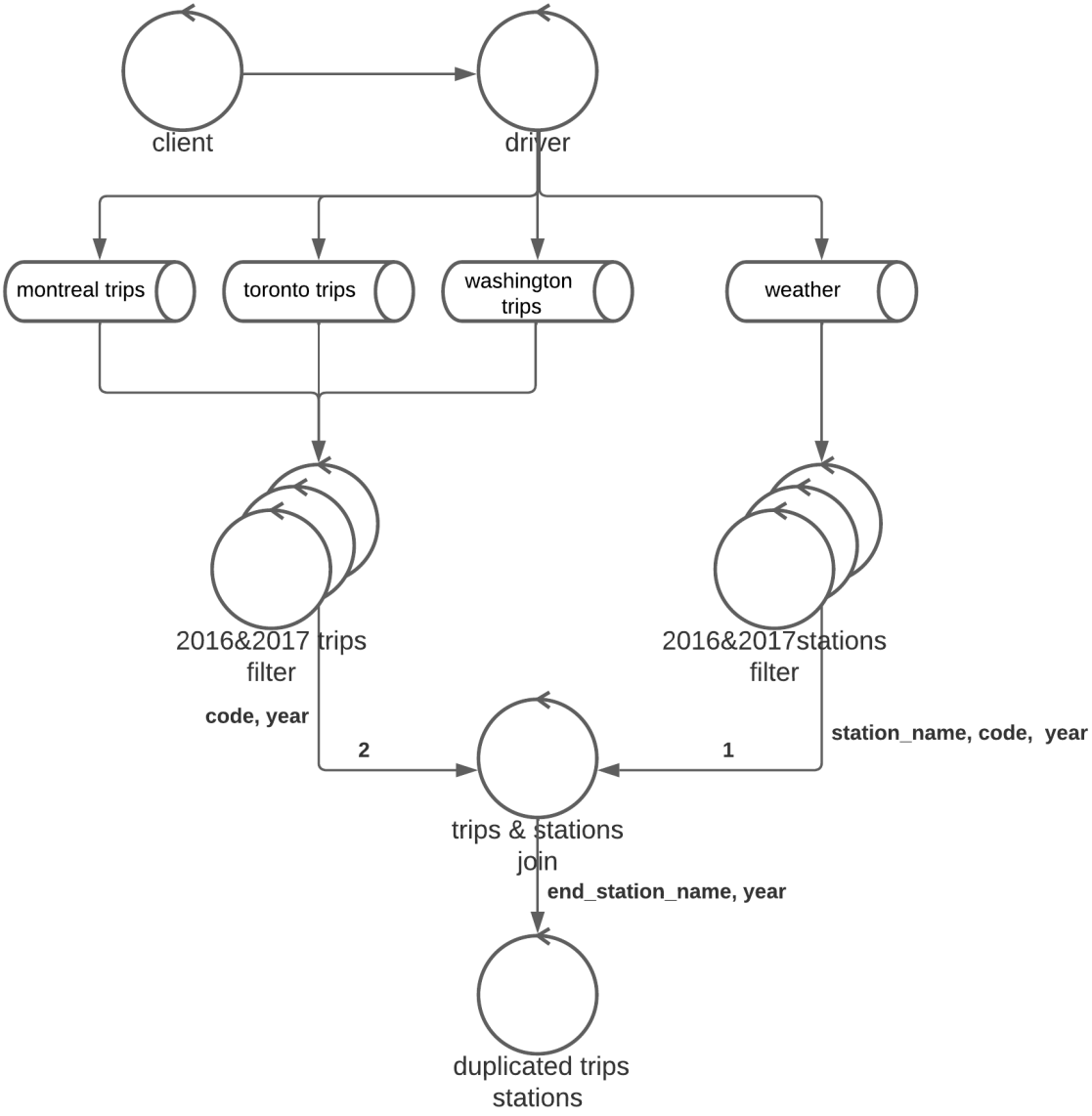
Además, el diagrama de robustez también muestra cómo los procesos se ejecutan de forma paralela, lo que implica que pueden trabajar en paralelo en diferentes nodos o servidores del sistema distribuido.



Se filtra únicamente registros de viajes en los que en el día del viaje fue mayor a 30 milímetros. Filtros ejecutandose en paralelo reciben todos los viajes desde colas de Rabbit siguiendo el patron de Work-Queues. Estos filtran viajes de dias con lluvia. Viajes en dias de lluvia son descartados. Los viajes son pasados a una ultima entidad que lleva el conteo de las duraciones de los viajes para cada dia. Al recibir EOF ejecuta el computo y obtiene los promedios para cada dia.



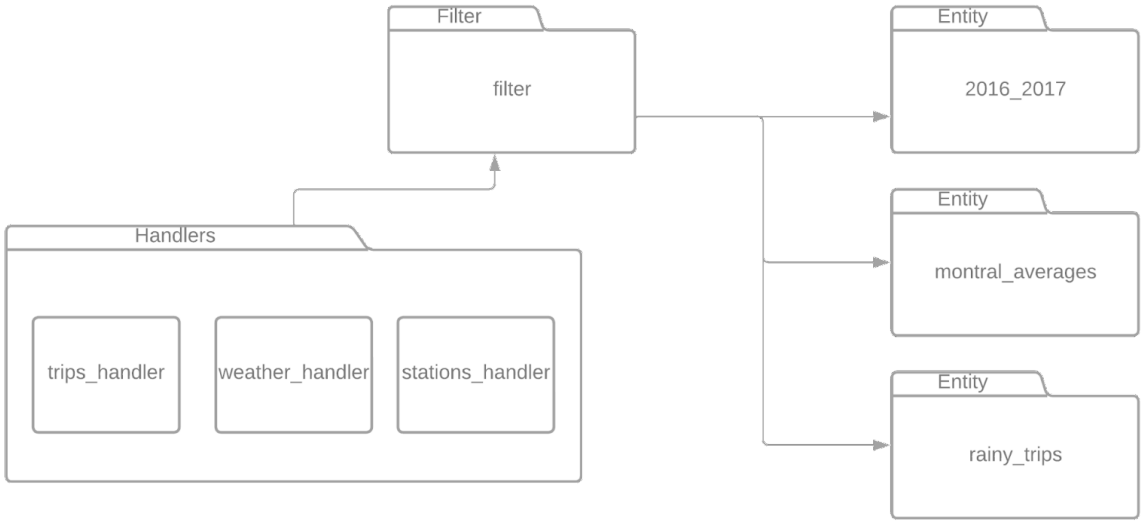
Se filtra únicamente registros de la ciudad de Montreal. Calcula la distancia entre estación de inicio y estación de fin. Una ultima entidad mantiene el promedio para cada estacion. Finalmente se filtra por estaciones con un promedio mayor a 6.



Los filtros reciben todas los datos provenientes del cliente (pasando por el driver) para ser filtrados. Solo pasan los registros del 2016 o del 2017, otros registros son descartados. Se realiza la junta entre viaje y estacion para luego filtrarlo a una ultima entidad que se encarga de mantener la sumatoria de viajes para cada anio para cada estacion. Al recibir EOF este ejecuta el calculo y obtiene las estaciones que duplicaron viajes.

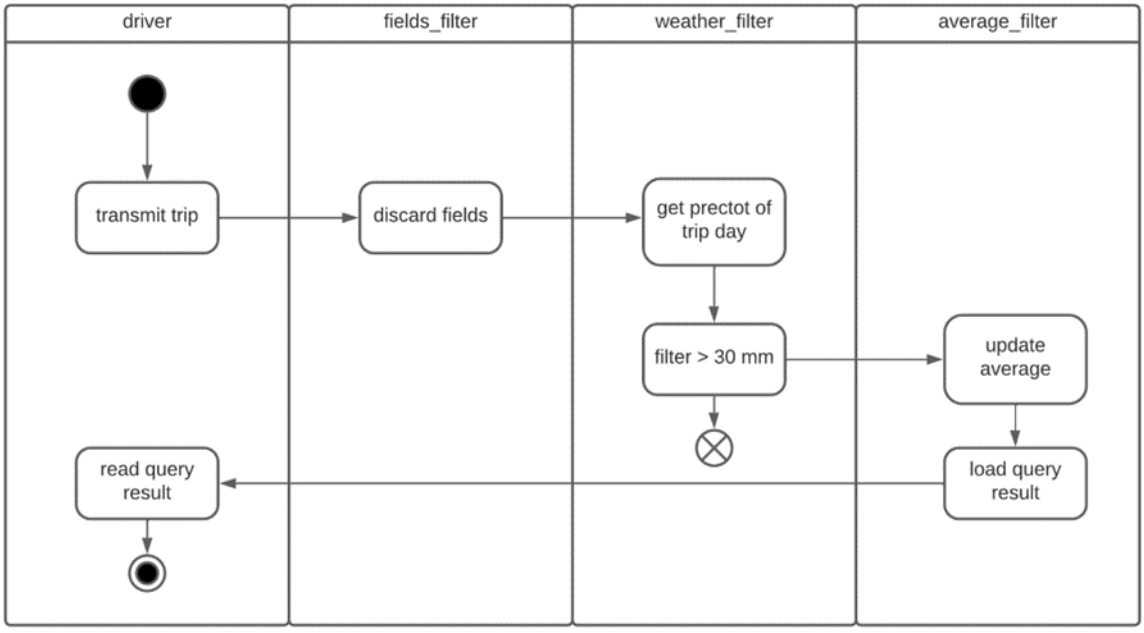
Vista de desarrollo

Hay componentes reutilizados en el codigo de la aplicacion.Un modulo de comunicacion abstrae la configuracion y el pasaje de mensajes a traves del middleware, desacoplando la capa de mensajeria de la capa de procesamiento.
El sistema sale gracefully al recibir SIGTERM gracias al manejo de signals provisto por la libreria estandar de Python, y por un objeto que encapsula todos los componentes que deben ser cerrados de manera correcta por la aplicacion ante la necesidad de terminar el programa de manera imprevista. Al recibir un SIGTERM se llama a una funcion que procede a cerrar todo elemento necesario (por ejemplo los canales al middleware.

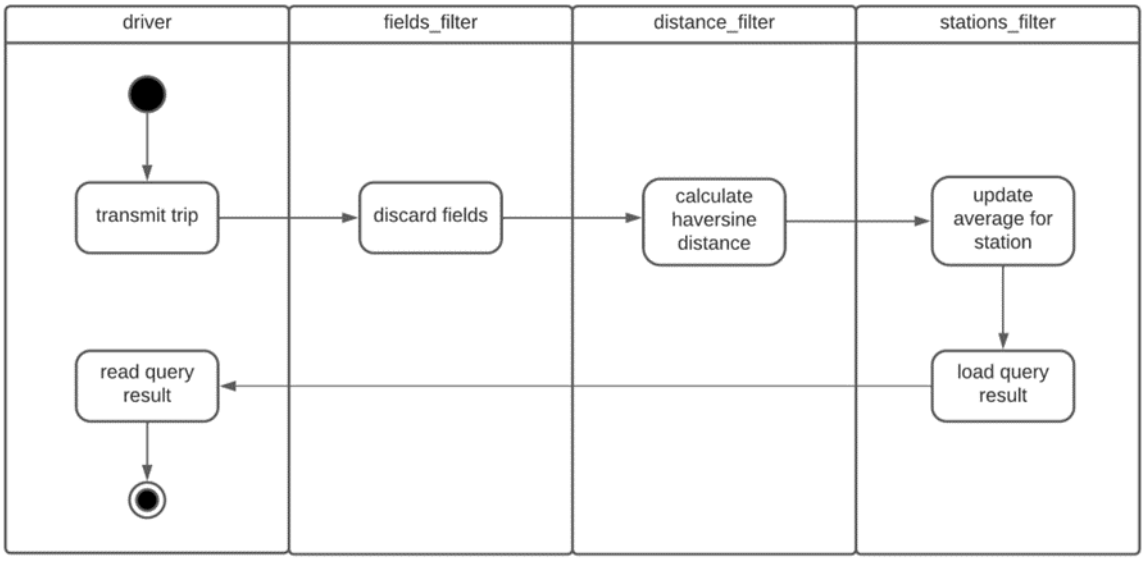


Vista de Procesos

Para la siguiente vista se muestran dos escenarios bastante típicos del sistema que representan una buena porción de la funcionalidad del mismo. En estos diagramas se asume que las “side tables” ya han sido completamente cargadas.



El primer diagrama nos muestra la secuencia para calcular el promedio de duración de los viajes en los que la lluvia fue mayor a 30 milímetros. El proceso consiste en la transmisión de un viaje por parte del proceso “driver” de la aplicación. Se calcula la precipitación del día, y se vuelve a filtrar en caso de no corresponder con la condición de la consulta. Finalmente se actualiza el valor de la media y se lo carga para que pueda ser consultado nuevamente por el proceso driver.



En el segundo diagrama podemos ver como se vuelve a aplicar una secuencia de filtros y lógicas para encontrar las estaciones las cuales son destino de viajes de distancias

mayores a 6 km. A los viajes se les realiza el cálculo de la distancia entre estación de salida y estación de llegada, aplicando una función haversine. Con este cómputo realizado se puede pasar el dato al siguiente proceso que mantiene los promedios de las estaciones actualizados y continuar informando al “driver” los resultados de la consulta.

El cliente publica un mensaje que es recibido en el middleware. El middleware maneja el paquete recibido y lo redirige a las colas de las entidades.