UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

Sistemas Distribuidos I

(75.29)

Trabajo Práctico N.º 1

1º Cuatrimestre de 2023

Armando Tomás Civini

104350

Fecha de entrega: 2/05/2023

Scope

Bike rides analyzer es un sistema distribuido y escalable que permite procesar datos de viajes en bicicleta de cientos de ciudades para obtener estadísticas sobre ellos.

Requisitos funcionales

- Se solicita un sistema distribuido que analice los registros de viajes realizados con bicicletas de la red pública provista por grandes ciudades.
- Los registros cuentan con el tiempo de duración del viaje, estación de inicio y de fin. Se posee también lat., long. y nombre de las estaciones así como la cantidad de precipitaciones del día del viaje.
- Los registros se ingresan progresivamente, al recibirse de cada ciudad.
- Se debe obtener:
    - La duración promedio de viajes que iniciaron en días con precipitaciones >30 mm
    - 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.

Requisitos no funcionales

- Para construir una simulación realista se define la serie de datos:
    - https://www.kaggle.com/datasets/jeanmidev/public-bike-sharing-in-north-america
- Considerar el uso de la librería haversine para calcular distancias.
- El sistema debe estar optimizado para entornos multicomputadoras
- Se debe soportar el incremento de los elementos de cómputo para escalar los volúmenes de información a procesar
- De ser necesaria una comunicación basada en grupos, se requiere la definición de un middleware
- Se debe soportar una única ejecución del procesamiento y proveer graceful quit frente a señales SIGTERM.

Arquitectura

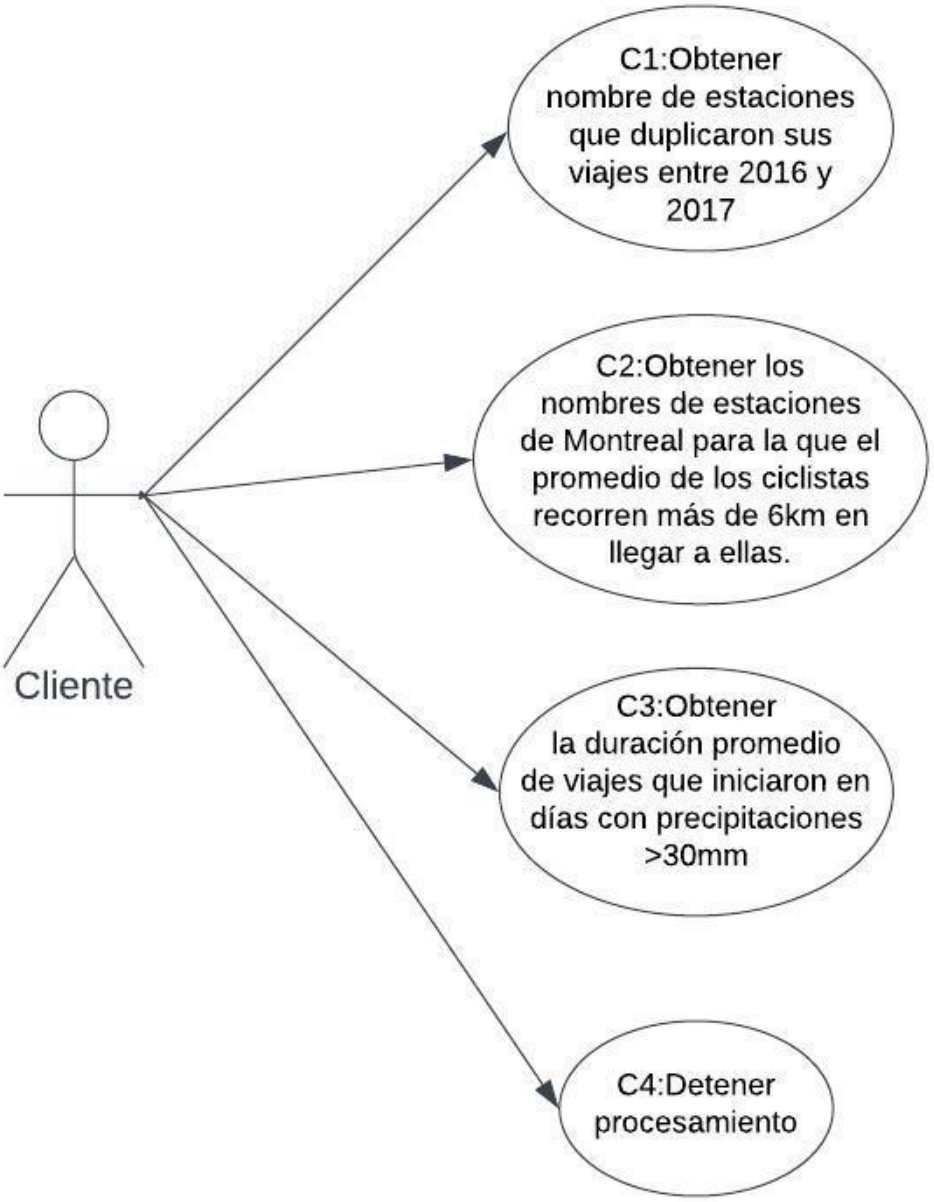
Bike Ride Analyzer utiliza una arquitectura distribuida para procesar un gran volumen de datos, de manera escalable, sobre los viajes en distintas ciudades y así obtener distintas métricas sobre éstos.

El sistema se divide en nodos que realizan distintas etapas del procesamiento y se comunican mediante colas. Los nodos están implementados en containers de docker y las colas las provee una instancia de RabbitMQ centralizada.

El cliente el cual quiere conseguir las distintas métricas especificadas, se conectará a el endpoint encontrado en el nodo servidor, enviará primero los datos correspondientes al

preprocesamiento de los viaje y luego enviará los datos que corresponden a los viajes. Finalmente éste esperará el resultado por parte de este mismo endpoint.

Casos de uso



En el diagrama se pueden observar las distintas capacidades del sistema. Los casos de uso del 1 al 3 muestran los 3 resultados posibles del procesamiento.

El primero cuenta para todas las estaciones, todos los viajes que hubieron en los años 2016 y 2017 y devuelve aquellas estaciones que doblaron la cantidad de viajes entre estos años.

El segundo caso de uso calcula para cada estación de Montreal la distancia promedio que recorren los ciclistas en llegar a ellas y devuelve aquellas en donde la distancia promedio sea superior a 6 kilómetros.

El tercer caso de uso calcula la duración de los viajes en los días donde llovió más de 30mm.

Finalmente hay un último caso de uso que incluye el detener la ejecución del algoritmo.

Vista Lógica

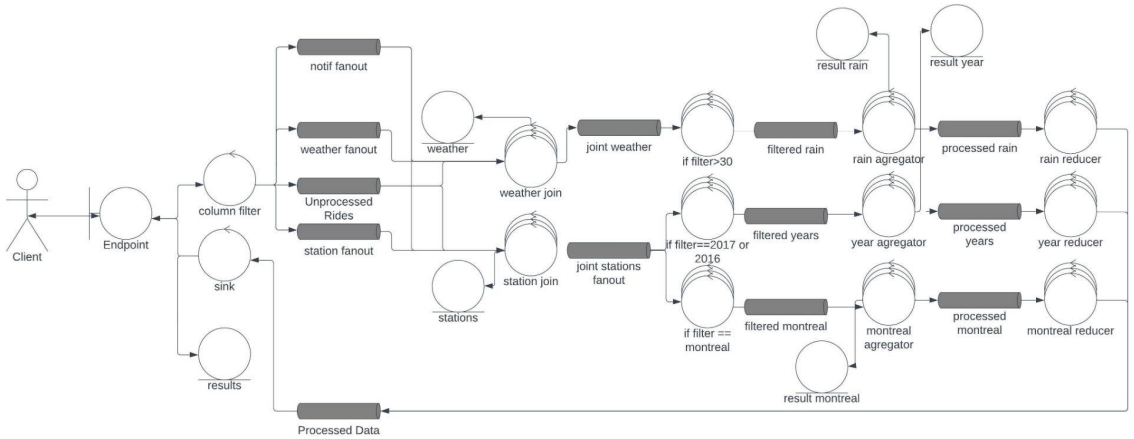
DAG

En éste diagrama se representa el flujo de los datos a través del sistema. Tiene dos partes principales, la primera siendo el flujo de los datos de preprocesamiento. Estos son los datos que tienen que estar completos, previo a enviar y procesar los viajes. Por un lado tenemos los datos de las estaciones, los cuales se filtran y se envían al nodo join stations. Esto también se hace para los datos del weather como se puede ver en la parte izquierda del diagrama.

La otra parte de este diagrama es ver el flujo de los viajes a través del sistema. Podemos ver como la información filtrada de los viajes se duplican entre los nodos que procesarán las consignas de stations y los nodos de weather. En estos nodos se combinará la información de cada viaje con los datos de preprocesamiento correspondientes para así poder ir aplicando los cálculos y reducciones necesarias para obtener el resultado deseado.

Vista Física

Diagrama de Robustez

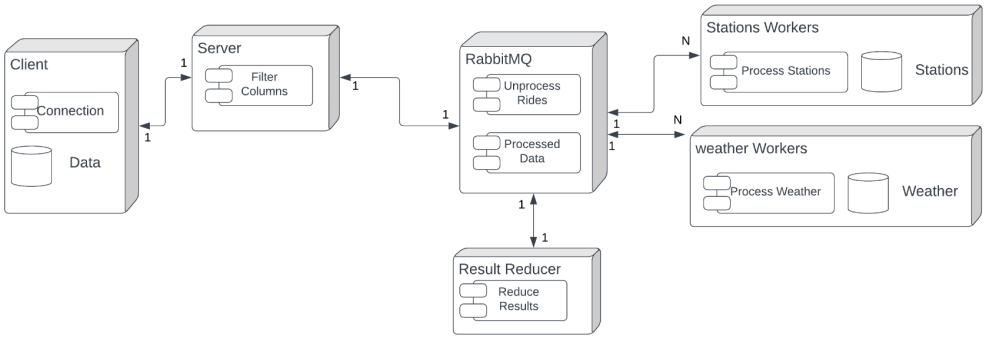


En el diagrama de robustez se puede observar la composición física del sistema. En este diagrama se observa cómo entran en juego las colas para el pasaje de datos entre procesos. Hay dos tipos de colas. Aquellas que dicen fanout enviarán el dato producido a todos los consumidores escuchando, mientras que las que no lo dicen, solo se lo enviarán a uno de los consumidores.

Otro componente importante para aclarar es que para la cola *processed rides,* ésta duplica los viajes entre los station joins y los weather joins pero, al tener estos nodos multiplicidad, los viajes sólo le llegarán a uno de cada tipo.

finalmente es importante remarcar el funcionamiento de la cola llamada *notif fanout,* la cual sirve para avisar a los nodos de que el envio de viajes a terminado.

Diagrama de Despliegue



En este diagrama se observa la división en containers del sistema y las relaciones entre ellos.

Vista de Procesos

Diagrama de Actividades

En esta vista se intenta ejemplificar los pasos que lleva a cabo el sistema para obtener los resultados. Se dividen estos pasos en dos partes.

Diagrama de actividades, Upload de estaciones

La primera etapa es la de enviar los datos de preprocesamiento que deben tener los nodos para llevar a cabo los cálculos. En este se ejemplifica para el ejemplo de las estaciones pero la secuencia con los datos weather es idéntica.

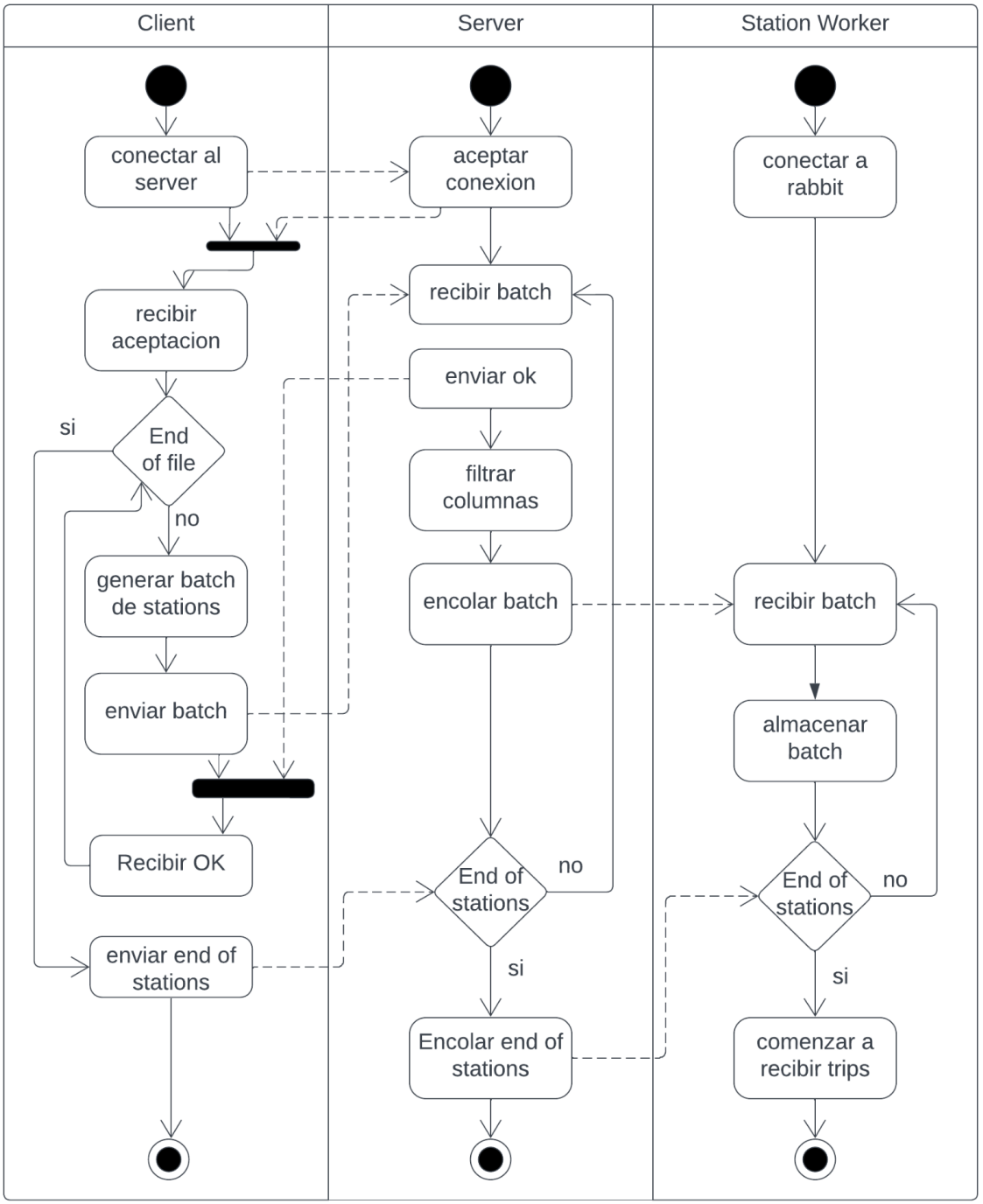
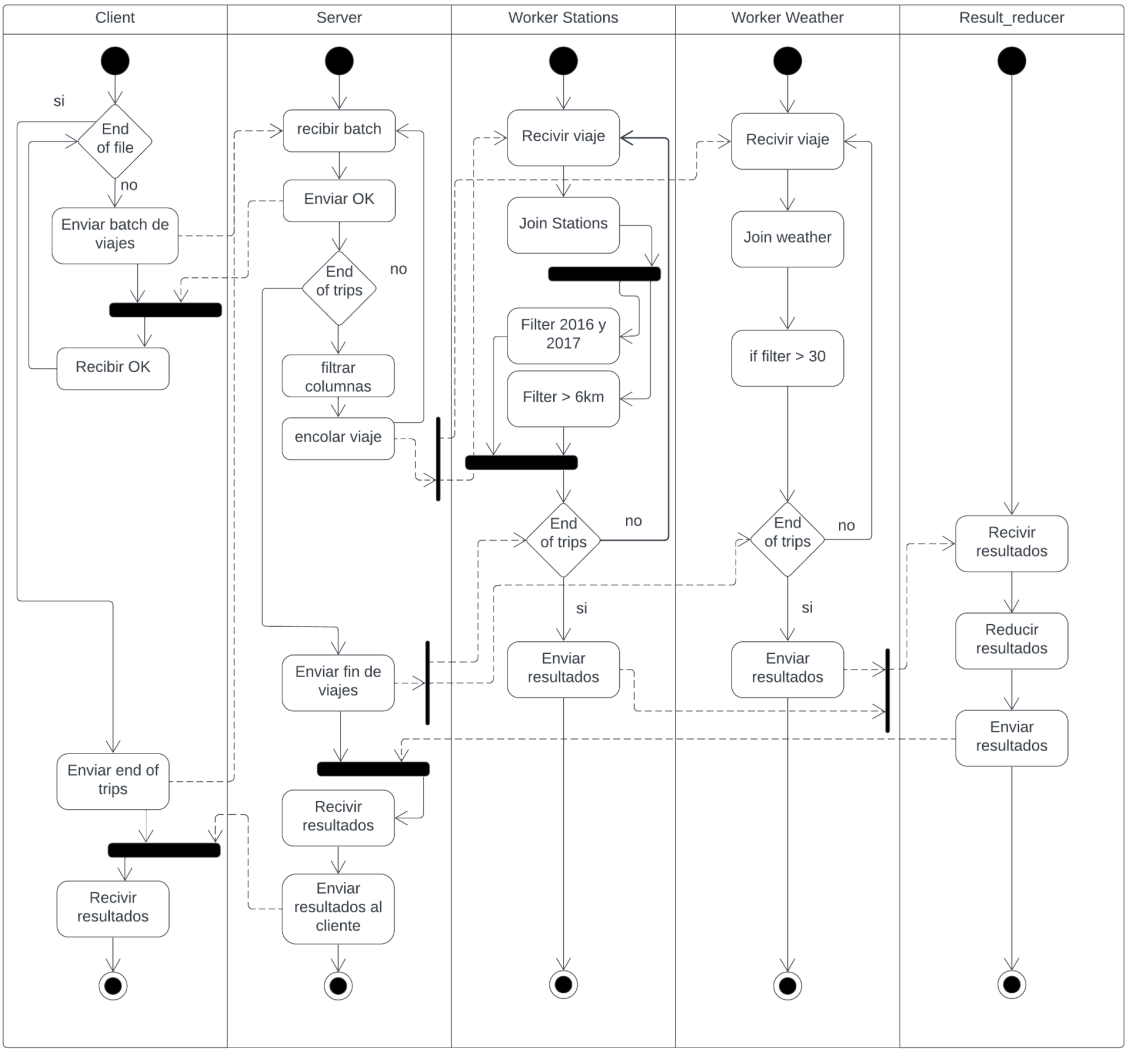


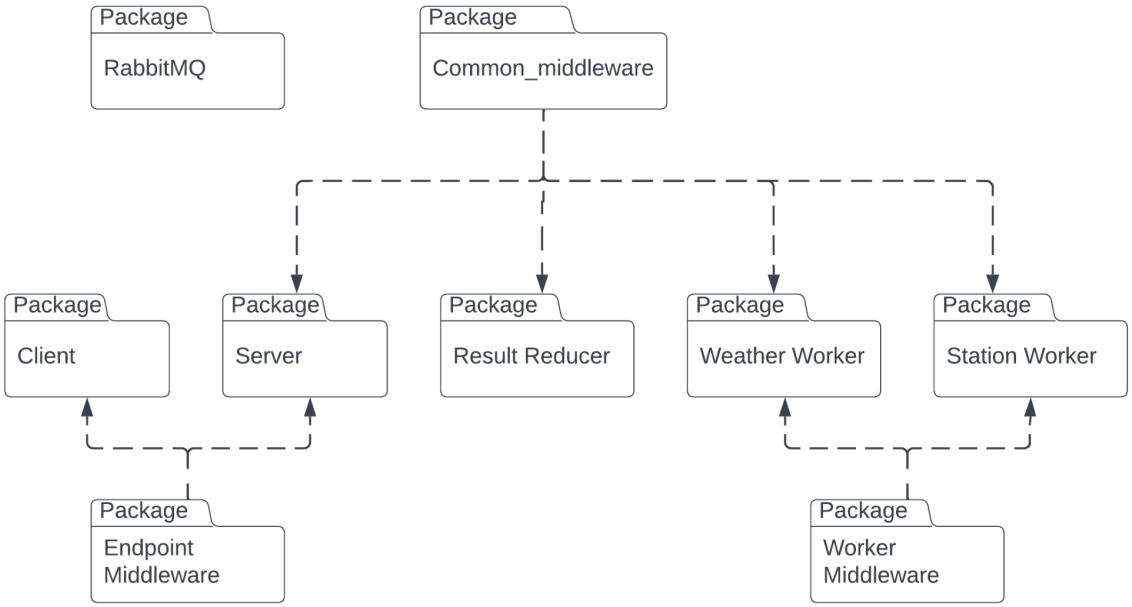
Diagrama de Actividades, procesamiento de viajes

Esta etapa es la del procesamiento de los viajes, en donde se puede ver a ambos tipos de workers realizando sus operaciones en los datos y como se combinan.


Client
Server
Worker Stations
Worker Weather
Result_reducer
si
End of file
no
Enviar batch de viajes
Recibir OK
Enviar end of trips
Recivir resultados
recibir batch
Enviar OK
End of trips
no
filtrar columnas
encolar viaje
Enviar fin de viajes
Recivir resultados
Enviar resultados al cliente
Recivir viaje
Join Stations
Filter 2016 y 2017
Filter > 6km
End of trips
no
si
Enviar resultados
Recivir viaje
Join weather
if filter > 30
End of trips
no
si
Enviar resultados
Recivir resultados
Reducir resultados
Enviar resultados

Vista de Desarrollo

Diagrama de Paquetes



En esta vista se muestra cómo se estructura el código del sistema. La división es similar a los nodos del sistema. Una gran ventaja de esta estructura es el paquete *Worker Middleware*. Él cual contiene una generalización de los nodos station y weather por lo cual es muy simple agregar nuevas operaciones y estadísticas al sistema a través de nuevos nodos. En el paquete *Endpoint Middleware* se encuentra definido el protocolo de comunicación entre el cliente y el servidor. Finalmente en el paquete *Common Middleware* hay clases útiles para todo el sistema como las que se encargan de la comunicación con RabbitMQ.

Limitaciones

Fallas

El sistema ante cualquier falla en alguno de los nodos no podrá realizar la tarea correspondientemente, ni podrá recuperar o reutilizar el procesamiento ya invertido.

Cálculo parcial

El sistema no puede devolver resultados parciales ni tener un flujo de datos ilimitado ya que éste necesita terminar de procesar los viajes para terminar el cálculo.

Múltiples clientes

El sistema no acepta múltiples clientes subiendo información. Aunque modificarlo para esto es simple.

Velocidad bajo cargas bajas

El sistema al ser distribuido tiene muchas ventajas como la escalabilidad. Pero esto puede tornarse una desventaja ya que para ciertos volúmenes de datos bajos los mensajes a través del sistema agregan complejidad y una solución centralizada podría ser preferible.