Sistemas distribuidos (75.74)

TP-2

COVID19 Statistics

Apellido y Nombre	Padrón	Correo electrónico
Funes Federico	98372	fede.funes96@gmail.com

Índice

1. Alcance

Se propone diseñar una arquitectura para analizar una masiva cantidad de información respecto a casos de coronavirus en ciudades distribuidas por Italia, y, documentar y mostrar a través de distintas vistas el diseño de la arquitectura planteada para el problema.

2. Requerimientos funcionales

- Se solicita un sistema distribuido que procese estadísticas de datos individuales sobre casos positivos y decesos con info geo-espacial.
- La información es relevada in-situ y luego ingresada al sistema por lotes; indicando día, latitud y longitud de la muestra.
- Se conoce la ubicación de los polos poblacionales más importantes (ciudades o cabezas de provincia) con los que se quiere vincular cada muestra individual.

Se debe reportar:

- Totales de nuevos casos positivos y decesos por día.
- Listado de las 3 regiones con más casos positivos.
- Porcentaje de decesos respecto de cantidad de casos positivos detectados.

3. Requerimientos no funcionales

- Dada la ausencia de plataformas GIS, se define la pertenencia de una muestra a cierta región como aquella que minimice la distancia en Km

al polo poblacional.

- 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
- Coordinar varios procesos para realizar un análisis por lote de datos y no obtener resultados parciales.
- Distribuir bien las responsabilidades de cada nodo y ser flexible.

Sin embargo, se poseen las siguientes restricciones que afectan directamente la arquitectura:

- Ciertas operaciones requieren realizarse en un único nodo (Esto serían los agrupamientos por ciudad y por fecha por ejemplo)
- Para evitar mostrar resultados parciales, es necesario tener algún nodo o nodos que controlen como manejar cuando se termina un lote de datos.

4. Escenarios

En el siguiente diagrama de casos de uso se puede ver como se utiliza el sistema:

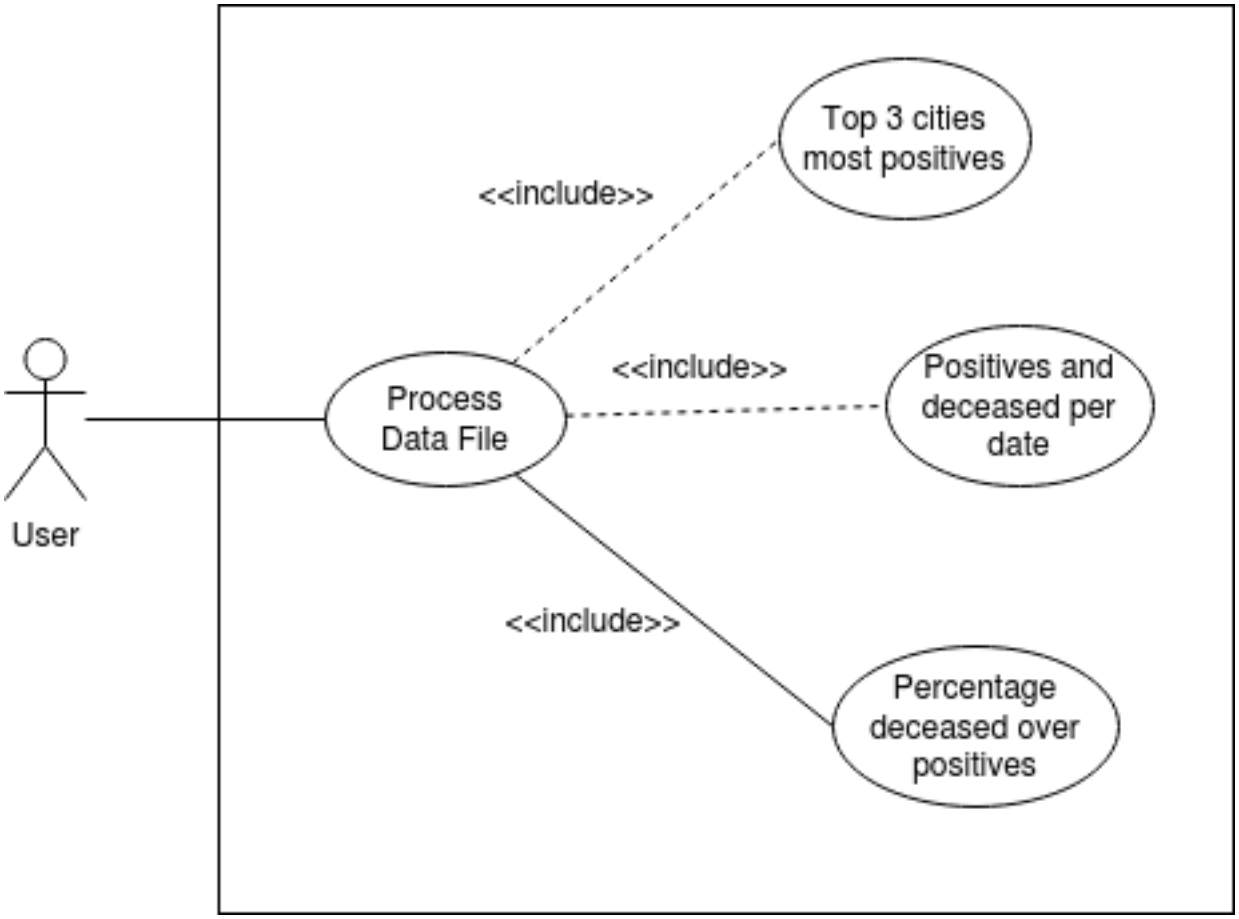

Figura 1: Casos de uso

Caso de uso	Process Data File
Dependencia	- Top 3 cities most positives: Para el cálculo de Top 3 con más casos positivos. - Positives and deceases per date: Para saber cantidad de decesos y positivos por fecha. - Percentage deceased over positives: Para el cálculo del porcentaje de decesos respecto a los positivos
Precondición	—
Flujo básico	1- El cliente solicita el procesar un archivo de muestras en conjunto con un archivo de lugares correspondientes. 2- Se ejecuta el caso de uso: Top 3 Cities most positives 3- Se ejecuta el caso de uso: Positives and deceases per date 4- Se ejecuta el caso de uso: Percentage deceased over positives 5- El sistema adjunta la información y la guarda para próximos usos
Flujos alternativos	—
Postcondición	El sistema genera un archivo con la información de los cálculos realizados-

Caso de uso	Top 3 cities most positives
Dependencia	—
Precondición	—
Flujo básico	1- El sistema recibe dos archivos, uno de muestras y uno de lugares para procesar. 2- El sistema calcula el top 3 de ciudades con más casos positivos.
Flujos alternativos	—
Postcondición	—

Caso de uso	Positives and deceases per date
Dependencia	—
Precondición	—
Flujo básico	1- El sistema recibe un archivo de muestras. 2- El sistema procesa la cantidad de positivos y decesos por fecha.
Flujos alternativos	—
Postcondición	—

Caso de uso	Percentage deceased over positives.
Dependencia	—
Precondición	—
Flujo básico	1- El sistema recibe un archivo de muestras. 2- El sistema calcula el porcentaje de decesos con respecto a positivos.
Flujos alternativos	—
Postcondición	—

5. Interpretación del problema

En el siguiente DAG (Directed Acyclic Graph) podemos ver como interpretar las operaciones, el orden entre ellas, cual depende de cual y ver caminos críticos:

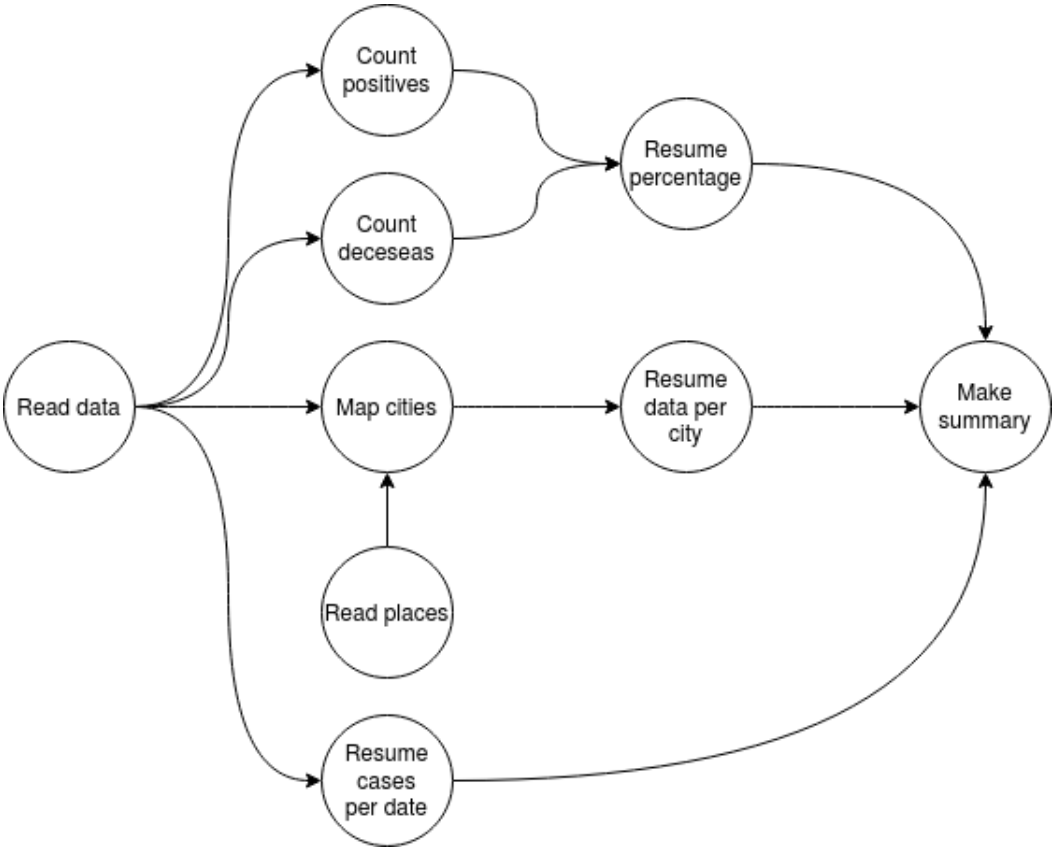

Figura 2: Directed Acyclic Graph (DAG) de operaciones

Las operaciones se consideraron a grandes rasgos en general pero se puede ver como la mayoría solo dependen de la información de entrada, con lo que se podrían paralelizar un montón de procesos.

6. Vista física

Se dispone a mostrar la arquitectura del sistema (Robustez) y la distribución en nodos físicos (despliegue) mostrando ventajas y desventajas de la arquitectura definida.

6.1. Diagrama de robustez

Se tiene una arquitectura definida de la siguiente manera:

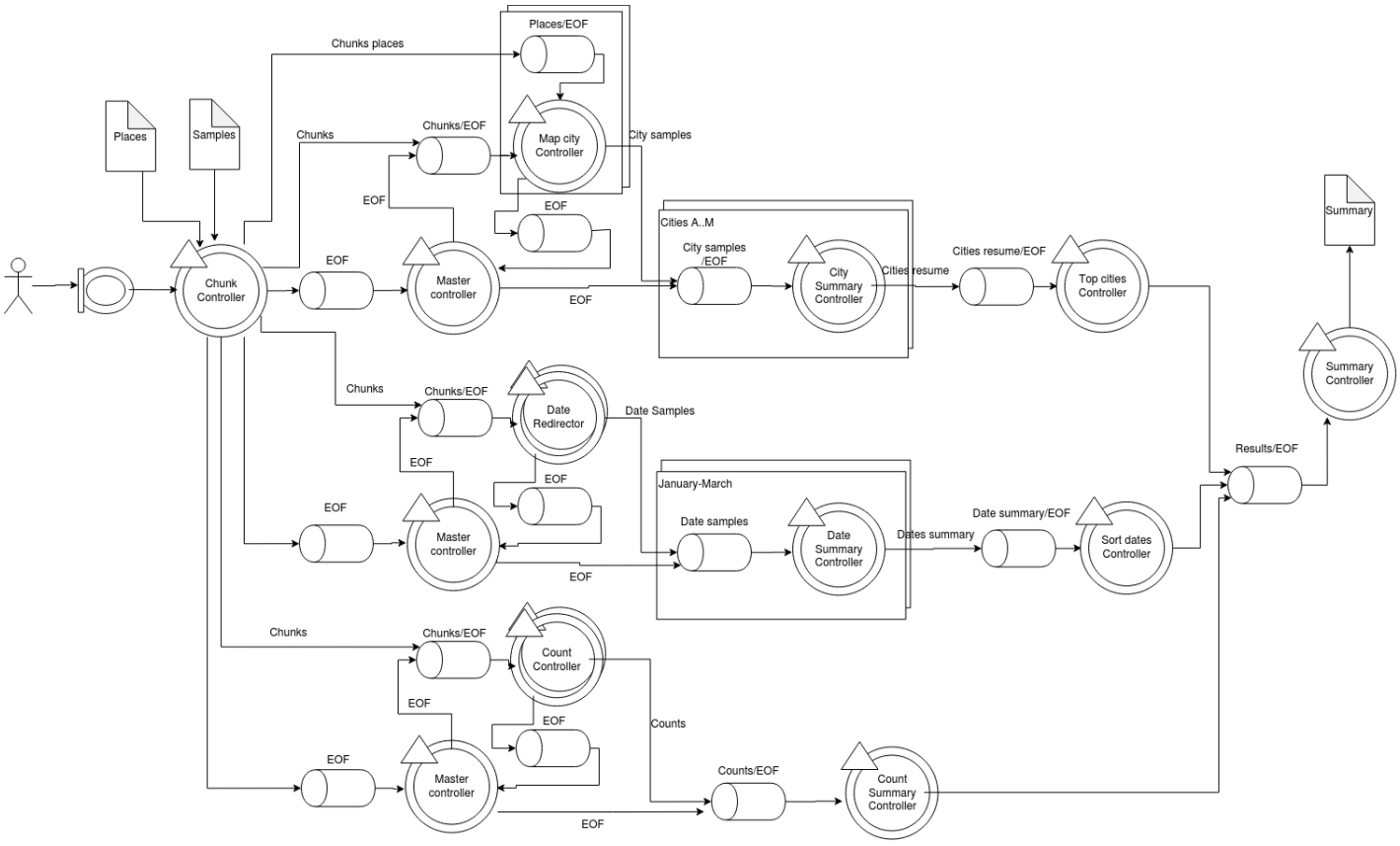

Figura 3: Diagrama de robustez completo

En la cual se encuentran distribuidas las responsabilidades de la siguiente forma:

- Chunk manager: Se encarga de leer del archivo de datos (Samples) y el archivo de lugares (Places) y envíar información de a partes a cada cola, cada una correspondiendo a una operación diferente (Top 3 ciudades con más positivos, agrupamiento de casos por fecha y porcentage de decesos con respecto a positivos).

- Master Controller: Se encarga de transmitir correctamente el fin de archivo (EOF) recibido del Chunk Manager para finalizar el trabajo de los Workers y propagar el fin de archivo (EOF) a los siguientes procesos.
- Map City Controller: Se encarga de calcular, filtrar y establecer la ciudad más cercana a la latitud y longitud de los datos recibidos para luego propagarlo a la cola correspondiente según ciudad. Tiene una cola para recibir los lugares sobre los que debe procesar.
- Date Redirector: Se encarga de transformar y redirigir los datos a la cola correspondiente según la fecha.
- Count Controller: Se encarga de contar ocurrencias de resultados (positivos o negativos) de los datos recibidos y propagar los resultados una vez recibido el fin de archivo.
- City Summary Controller: Se encarga de agrupar positivos por ciudad y propagarlos a la siguiente cola. A su vez, propaga el fin de archivo (EOF) a la siguiente etapa.
- Date Summary Controller: Se encarga de agrupar positivos y negativos por fecha y propagarlos a la siguiente cola. También, propaga el fin de archivo a la siguiente etapa.
- Count Summary Controller: Recibe resultados parciales de los trabajadores (Count Controllers) y calcula el resultado final agrupando la información recibida, finalmente la transmite a la cola de salida.
- Top Cities Controller: Recibe los resultados de agrupamiento por ciudad de cada controlador y calcula el top 3 con los resultados obtenidos, procede a enviarlos a la cola de salida.

- Sort Dates Controller: Recibe los resultados de agrupamiento por fecha, los ordena y los reenvía a la cola de salida.
- Summary Controller: Recibe toda la información y se encarga de escribir en un archivo los resultados obtenidos de cada operación.

6.1.1. Ventajas y desventajas del diseño

Poseemos las siguientes ventajas:

- Agregar nuevas operaciones implica agregar un camino diferente que no requiere modificación a grandes rasgos de la arquitectura definida.
- Podemos aprovechar las generalizaciones de las operaciones mencionadas para realizar otros tipos de análisis con la información dada.
- Permite una buena distribución de trabajo (Múltiples trabajadores) para realizar operaciones costosas de transformación de datos.
- Permite una buena distribución del volumen de datos, a fin de cuentas los nodos que poseen operaciones de agrupamiento (Resume/Summary) terminan dividiendo los datos de entrada según la cantidad de nodos que se quieran utilizar.

Sin embargo, tenemos también desventajas, como las siguientes:

- Tenemos varios puntos importantes de falla, algunos provocados por la propia naturaleza del problema (Ciertas operaciones requieren el agrupamiento de todos los datos) y otros generados por el uso de nodos maestro que controlen a los diferentes trabajadores.
- Hay más costo de transmisión de datos, el hecho de que el Chunk Manager transmita información replicada a traves de 3 colas implica más costo de transporte de datos.

6.2. Diagrama de despliegue

Hay múltiples propuestas para esto, dado el diagrama de robustez podemos aplicar distintos cortes, entre ellos uno podría ser ubicar todos los controladores maestros en un único nodo o también ubicar todos los controladores de trabajo (Workers) en un único nodo, sin embargo, se propone la siguiente forma de desplegar:

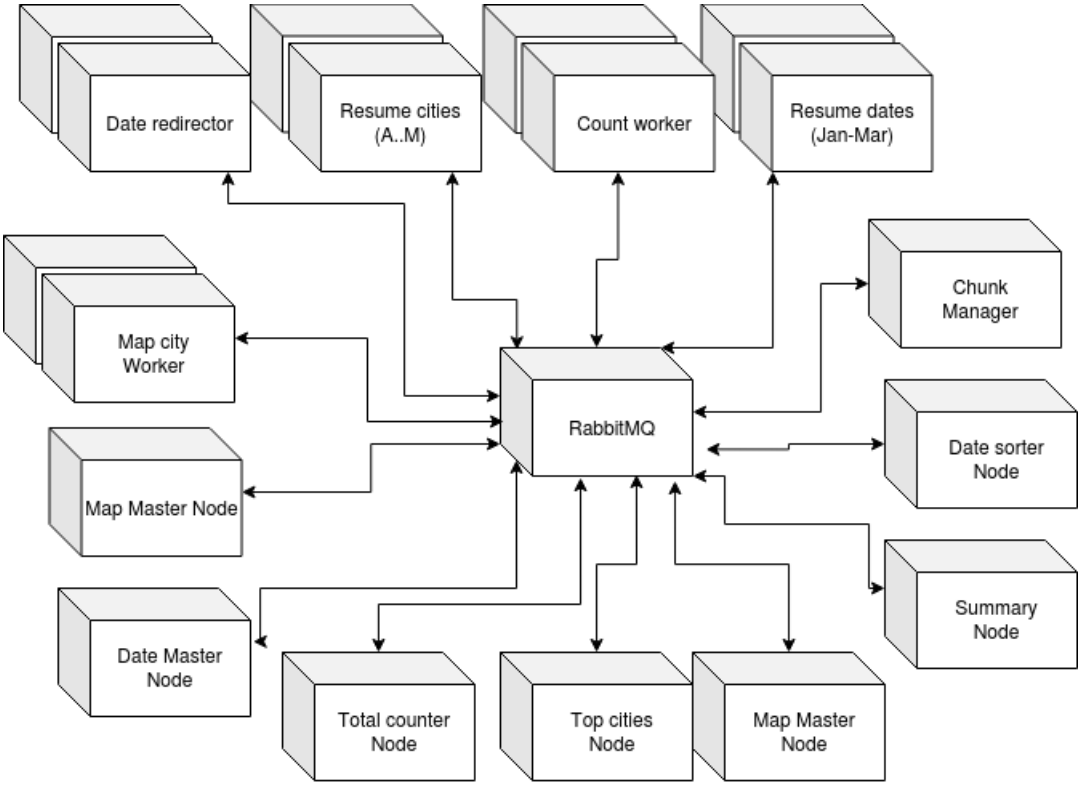

Figura 4: Diagrama de despliegue

Esta forma permite que cada nodo efectúe una única o muy pocas operaciones. Pero igual que antes, vemos nodos que son críticos y ante una caída provocarían la caída total del sistema. Observamos una fuerte dependencia al nodo de Rabbitmq.

7. Vista de desarrollo

A continuación, se van a mostrar los artefactos que componen al sistema, en este caso se optó por utilizar un diagrama de paquetes:

7.1. Diagrama de paquetes

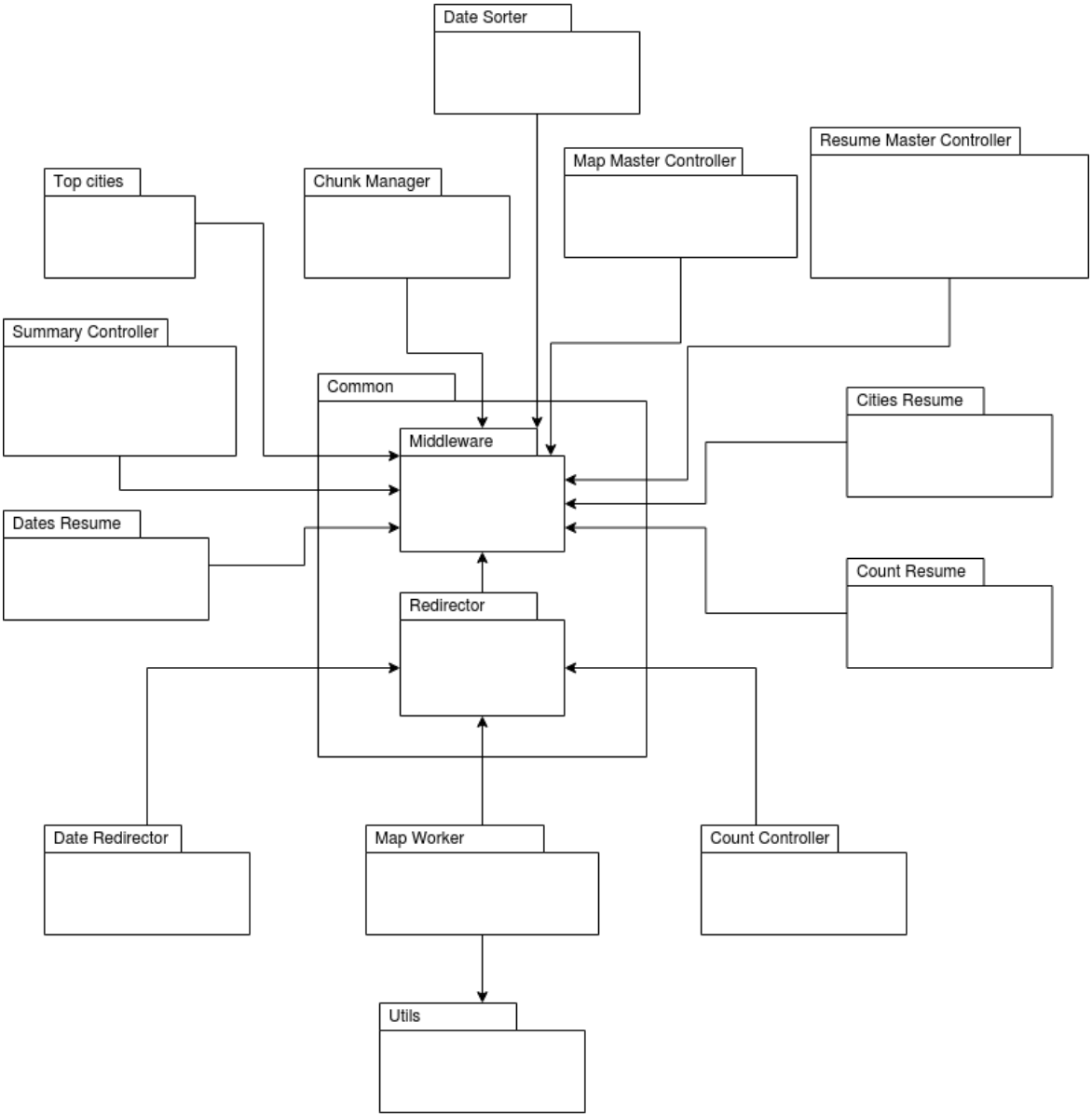

Figura 5: Diagrama de paquetes

En este caso, el paquete de Middleware es muy utilizado porque es el que permite la comunicación entre los nodos, así como también el paquete Redirector que se encarga de redirigir información a varios nodos.

8. Vista de procesos

A continuación, se van a mostrar diagramas de actividades y secuencia para el flujo de las operaciones del calculo de Top 3 de ciudades con más positivos.

8.1. Diagrama de actividades

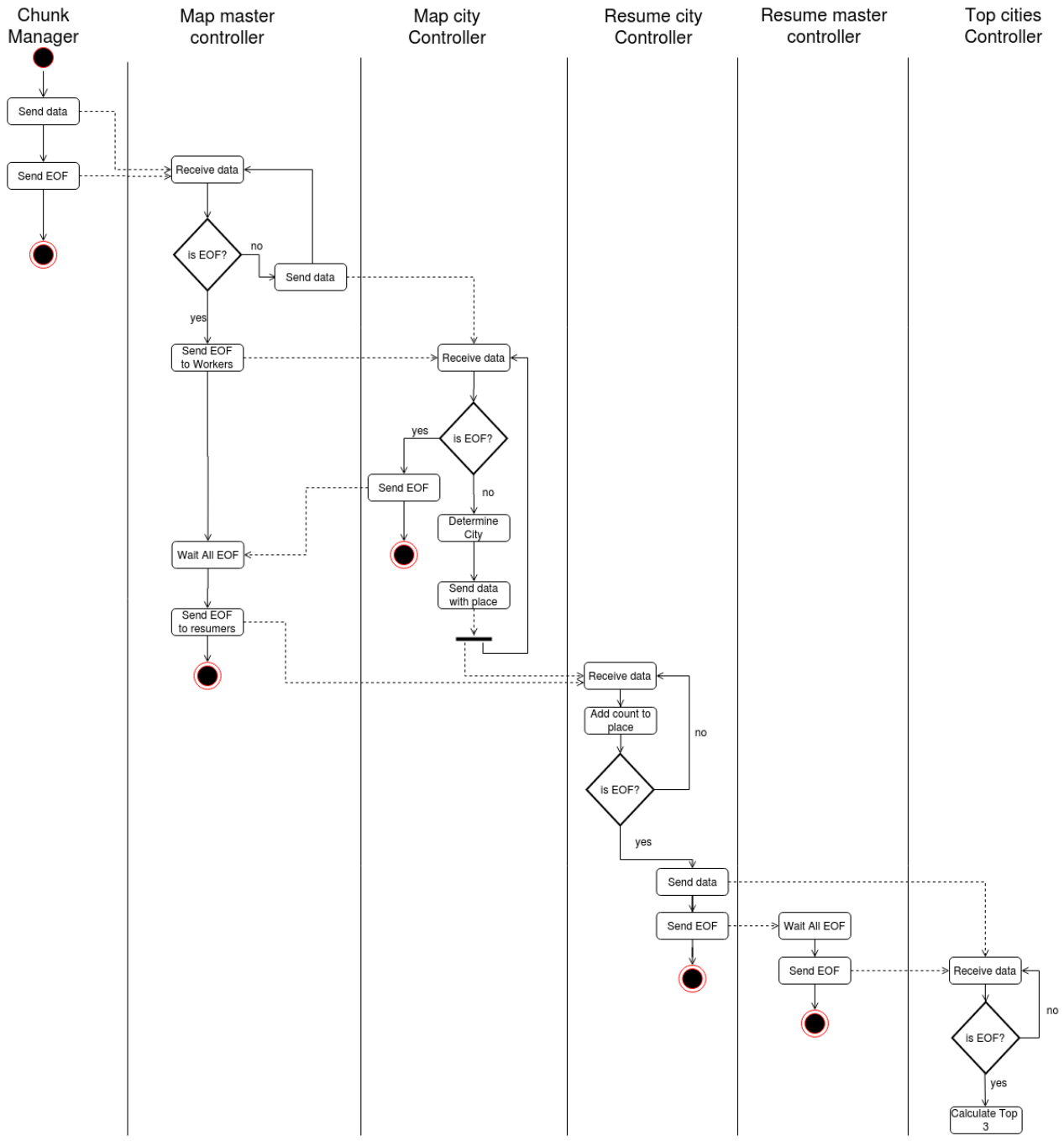

Figura 6: Diagrama de actividades del calculo de Top 3 ciudades. Parte 1/2

En esta primera parte se ve el flujo de la información hasta el calculo del

Top 3 de ciudades con más casos positivos. Algunos comentarios al respecto:

- Respecto a "Send Data"del Chunk Manager: Esta operación implica un bucle infinito de enviar información hasta llegar al fin de archivo, no se impuso en el diagrama para que no sea tan pesado.
- Vease que para este diagrama solo se incluye un úninco trabajador (Map City Controller) y un único nodo de resumen de positivos por ciudad (Resume City Controller).

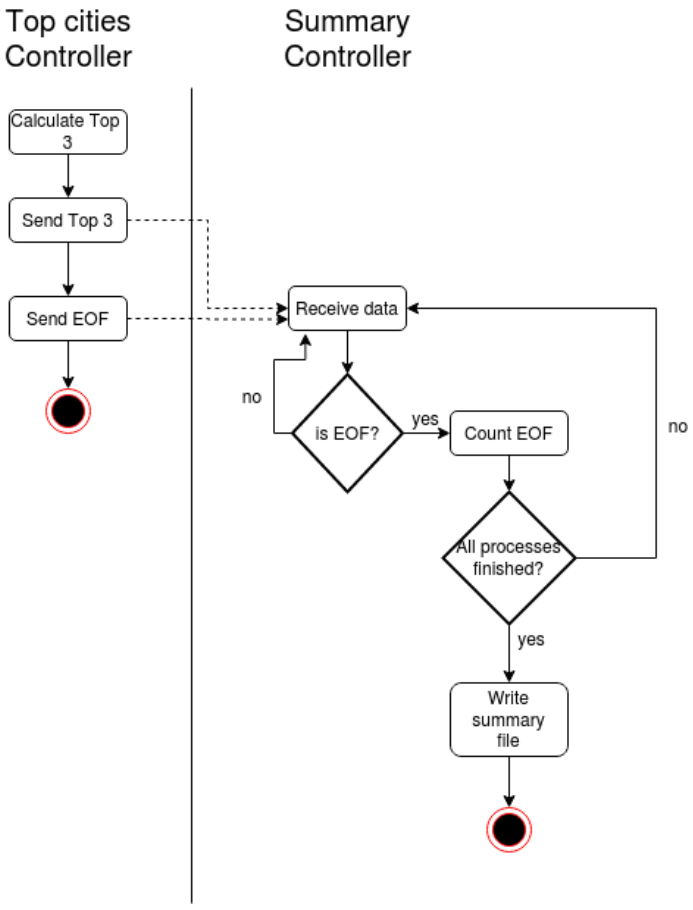

Figura 7: Diagrama de actividades del calculo de Top 3 ciudades. Parte 2/2

En esta segunda parte se ve el flujo de datos desde el calculo del top 3 de ciudades con más casos positivos hasta la escritura en el archivo. Se hace

una excepción asumiendo que es la única operación a realizar y por eso se realiza la escritura en el archivo.

8.2. Diagrama de secuencia

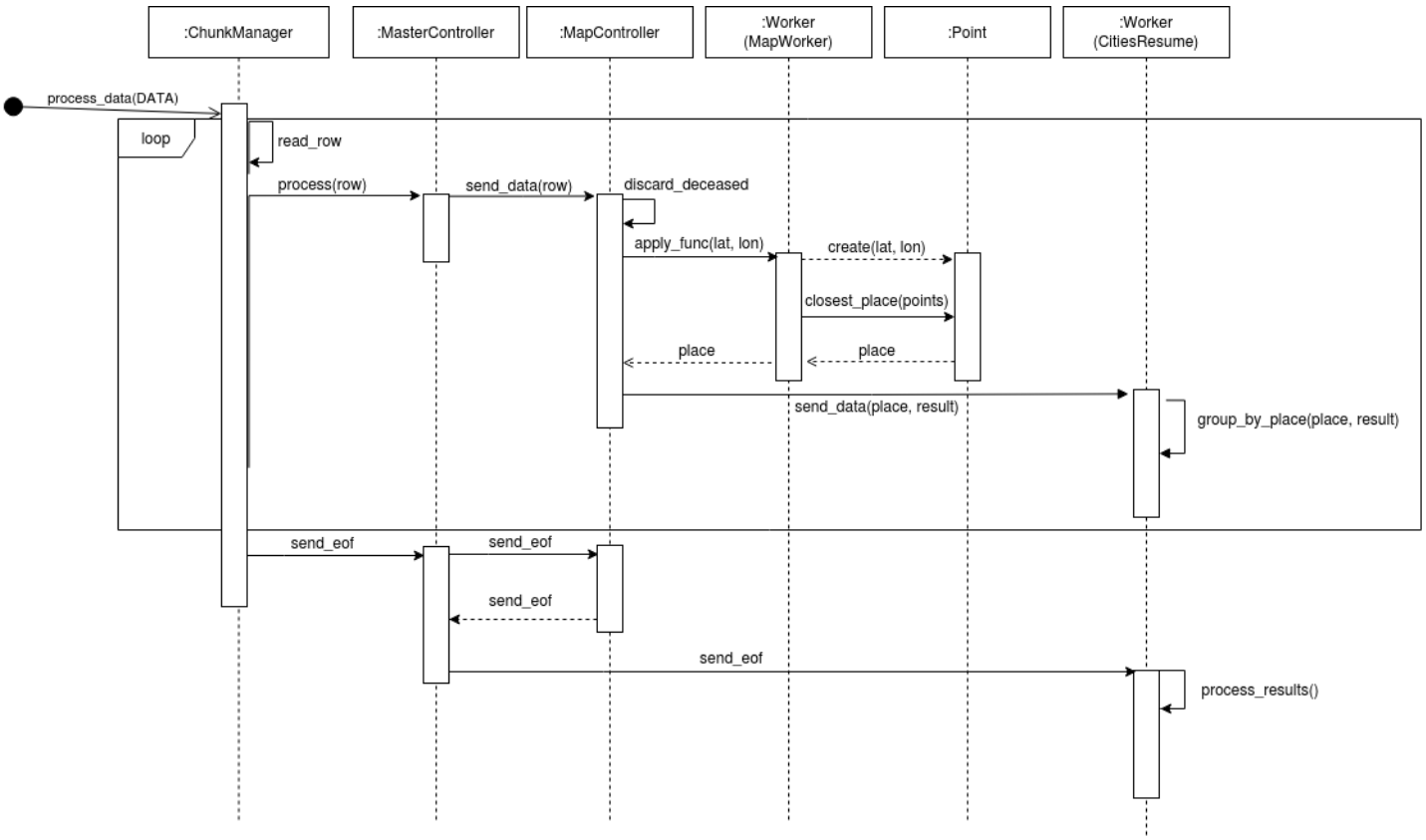

Figura 8: Diagrama de secuencia del calculo de Top 3 ciudades. Parte 1/2

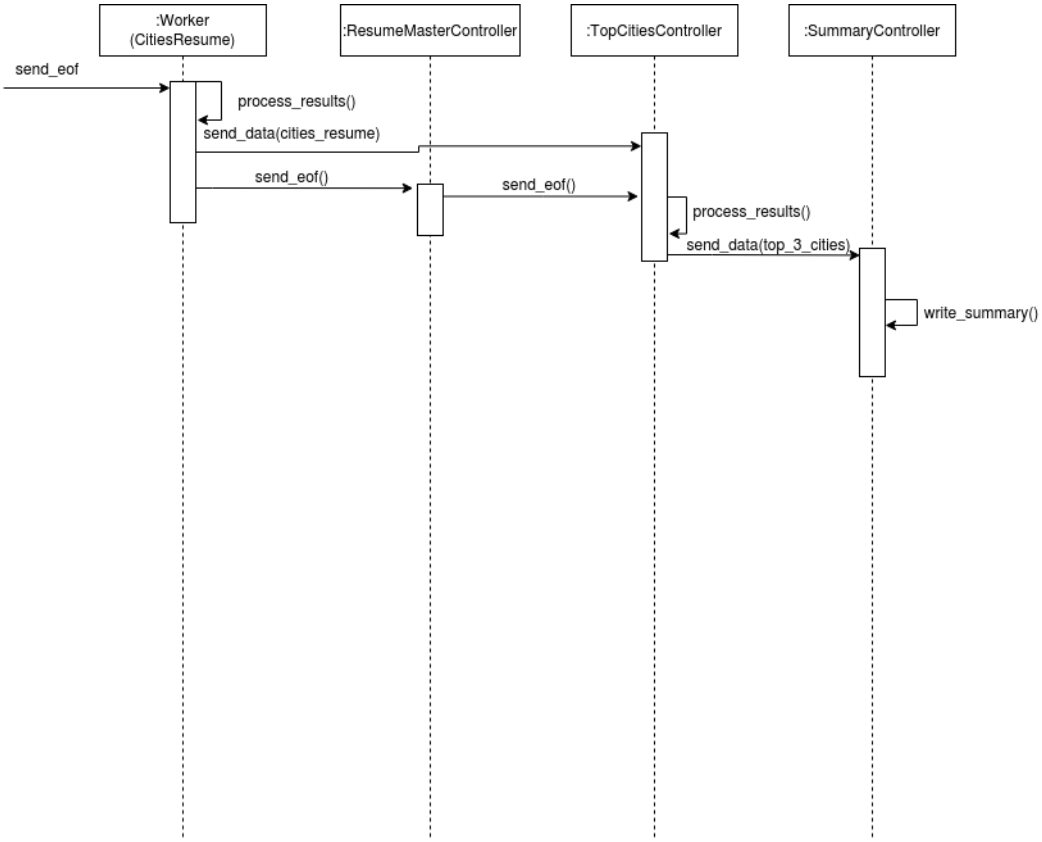

Figura 9: Diagrama de secuencia del calculo de Top 3 ciudades. Parte 2/2

En este diagrama se hace la misma suposición que en diagrama anterior, un único trabajador (Worker) y un único nodo de agrupación de datos (Resumer). Para que no sea un diagrama tan denso no se incluye el flujo de datos hacia el protocolo de comunicación, ya que este solo se encarga de recibir mensajes y transmitírselo a los próximos objetos.

9. Vista lógica

Se propone a mostrar los diferentes diagramas de clases que agrupan las clases de los paquetes mencionados en la sección de Vista de desarrollo. Respecto al diagrama de estados, no se realizará uno ya que no hay muchos

estados que compongan al sistema, en general puede verse el sistema como procesando información o leyendo un fin de archivo.

9.1. Diagrama de clases

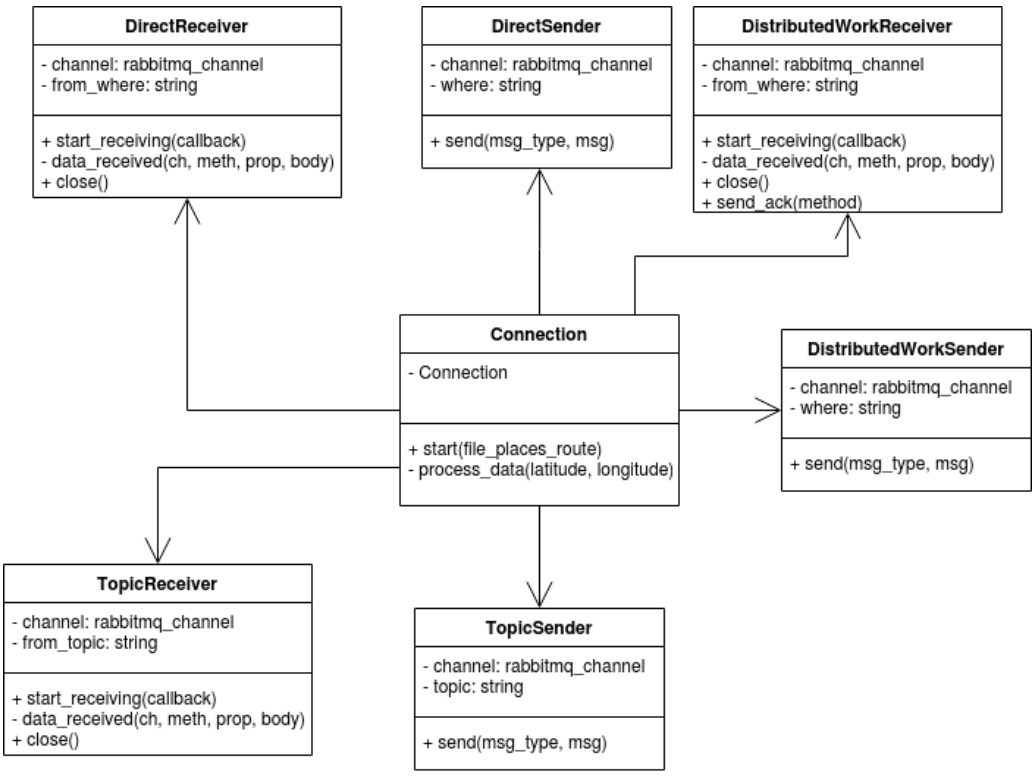

Figura 10: Diagrama de clases del Middleware

Descripción de clases:

- Connection: Maneja la conexión al middleware usando RabbitMQ. Se encarga de crear diferentes formas de comunicación entre procesos distribuidos.
- DirectReceiver: Crea un canal de comunicación para recibir información de procesos (Comunicación punto a punto).
- DirectSender: Crea un canal para transmitir comunicación a un canal

del tipo DirectReceiver.

- DistributedWorkReceiver: Crea un canal de comunicación para recibir información de varios que pueda ser leída por alguno de muchos procesos que utilicen este canal (Comunicación uno a uno con múltiples trabajadores).

- DistributedWorkSender: Crea un canal de comunicación para enviar información a un canal del tipo DistributedWorkReceiver para distribuir la carga de trabajo.

- TopicReceiver: Crea un canal de comunicación para recibir información bajo cierto tópico.

- TopicSender: Crea un canal de comunicación para envíar información bajo un cierto tópico.

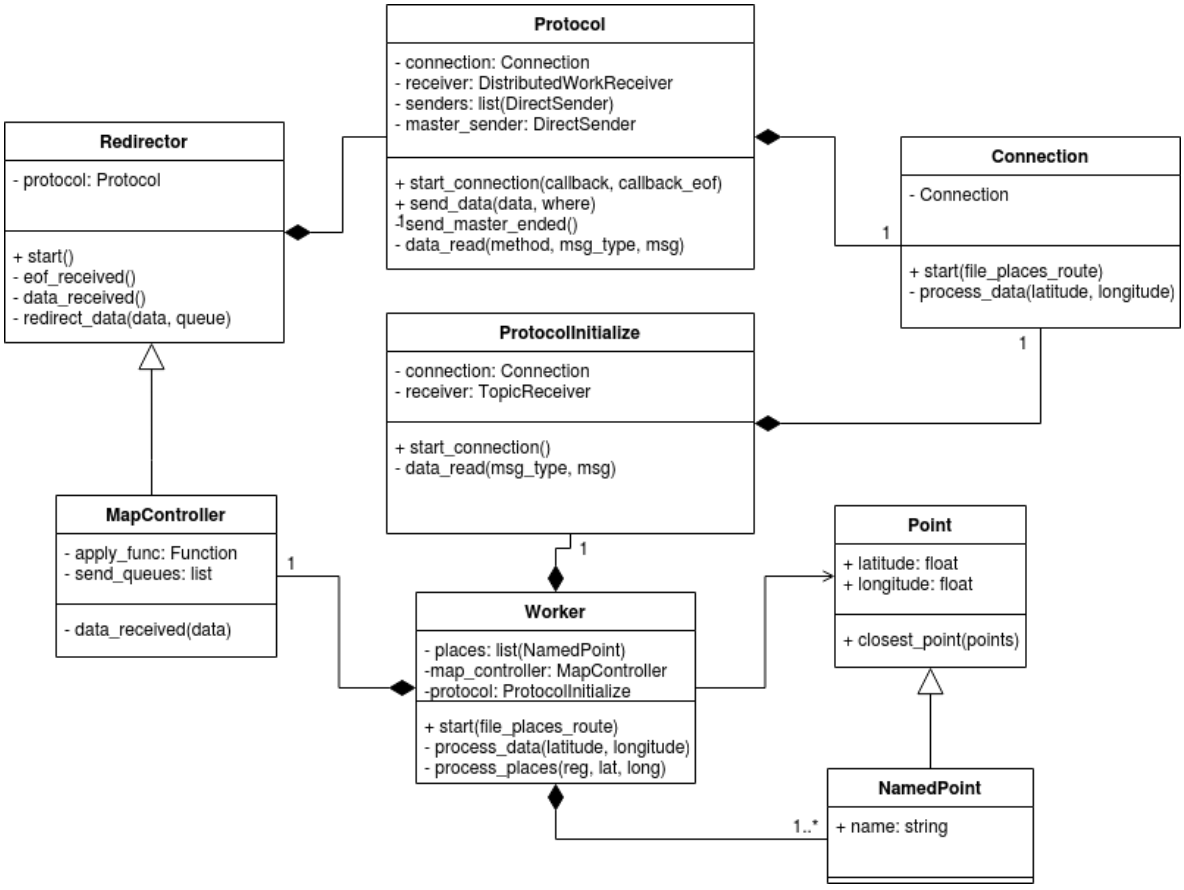

Figura 11: Diagrama de clases del Map Worker

Descripción de clases:

- Worker: Calcula la ciudad más cercana a la latitud y longitud recibida de un caso y procede a enviar los resultados a traves del MapController.
- Point: Clase que representa un punto en el espacio, incluye un método para calcular el punto más cercano (Usando distancia de haversine) de una lista de puntos con respecto a si mismo.
- NamedPoint: Clase que representa un punto con nombre.
- MapController: Se encarga de recibir información a través del Redirector, aplicarle la función apply_func y transmitir los resultados al

próximo nodo que corresponda.

- Redirector: Clase abstracta que se encarga de recibir información a traves de un canal de comunicación, envíar los resultados a alguno de múltiples canales y de envíar información de fin de archivo a un nodo maestro a través de un canal de comunicación especial.

- Protocol: Protocolo para manejar la creación y destrucción de canales de comunicación así como también manejar el envío de información, de fin de archivo y finalizar la comunicación.

- ProtocolInitialize: Protocolo para manejar la inicialización de los datos de los lugares para procesar los cálculos.

Las clases del paquete del Middleware se sobreentienden que conectan con la clase Connection aquí mostrada, no se incluye para no hacer denso el gráfico. Las clases del paquete Redirector están incluidas en este diagrama (Redirector, Protocol y Connection), por lo que no se incluirá diagrama de clases para este paquete.

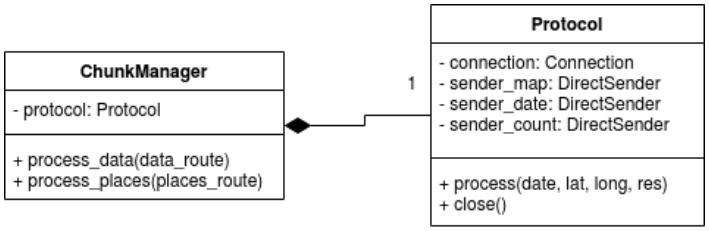

Figura 12: Diagrama de clases del Chunk Manager

Descripción de clases:

- ChunkManager: Se encarga de leer los datos de ambos archivos y transmitir cada fila a través del protocolo.

- Protocol: Maneja la creación de los canales de comunicación para la transmisión de datos a diversas colas para efectuar las operaciones.

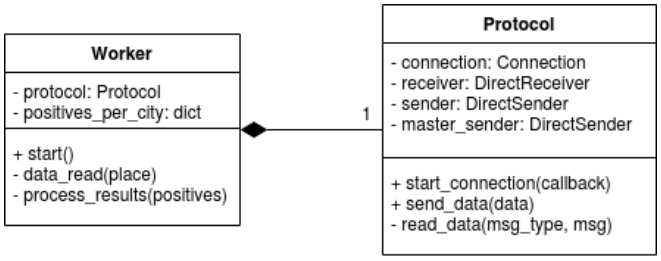

Figura 13: Diagrama de clases del Cities Resume

Descripción de clases:

- Worker: Se encarga de recibir la información de casos por ciudad y agruparlas por ciudad de manera de contar casos por ciudad, luego, al recibir toda la información procede a enviarla a la siguiente etapa.
- Protocol: Maneja la creación de los canales de comunicación para la transmisión de datos a diversas colas para efectuar las operaciones.

El resto de paquetes tienen todos un diagrama de clases similar, un worker que procesa la información, realiza alguna acción y envía la información a través de un Redirector (Que ya incluye un protocolo de comunicación) o a través de un propio Protocolo de comunicación.

10. Manual de uso

Para utilizar este sistema, basta con ingresar el archivo de datos nombrado **data.csv** y el archivo de países **places.csv** en la carpeta data dentro de **chunk_manager**, los resultados serán escritos en el archivo: **summary_controller/summary/summary.txt**

Para correr el programa, ejecutar:

make docker-compose-up map_workers=<TotalMapWorkers>
date_workers=<TotalDateWorkers>count_workers=<TotalCountWorkers>
Indicando la cantidad de trabajadores que se utilizarán. Con respecto a la cantidad de nodos de agrupamiento (Date Summary Controllers y Cities Summary Controller) estos deben ser modificados manualmente, actualmente no está automatizada la forma de creación de los mismos y debe de insertarse en el código y en los archivos de docker-compose.

Luego, para ejecutar el cliente, basta ejecutar:
make client-run

11. Problemas encontrados y mejoras disponibles

Este sistema presenta inconvenientes y mejoras disponibles a realizarse pero por falta de tiempo, no pudieron realizarse, entre ellas destacamos:

- Automatizar la creación de nodos de agrupación.
- Escribir clases más generales para no utilizar tantas clases de Protocolo (Refactor de código).
- Renombrar paquetes y clases a nombres más simples de entender (Refactor de código).
- No utilizar varios de canales de comunicación de salida del ChunkManager (Que funcionan de manera secuencial por el momento) y utilizar comunicación vía tópicos (Respecto a esto, hubo problemas por inicialización de colas para no perder información). De esta forma, se podrían agregar nuevas operaciones sin incluir un nuevo canal de comunicación a mano.

- Separar el archivo de docker-compose en múltiples archivos para cada operación, de manera que sea más legible y cómodo de usar.
- Sería útil incluir un nodo de monitoreo para monitorear el estado del sistema.
- Se incluye código para manejar configuraciones a través de archivo pero actualmente se están utilizando variables de entorno, sería mejor migrar a los archivos de configuración.
- Se podría paralelizar tranquilamente la ejecución del envío de datos de ambos archivos en el Chunk Manager.