Sistemas distribuidos (75.74)

TP-4 COVID19 Statistics

Apellido y Nombre	Padrón	Correo electrónico
Funes Federico	98372	fede.funes96@gmail.com
Damián Cassinotti	96618	dcassinotti@fi.uba.ar

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.
- El sistema debe ser tolerante a fallas y estar altamente disponible.

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:

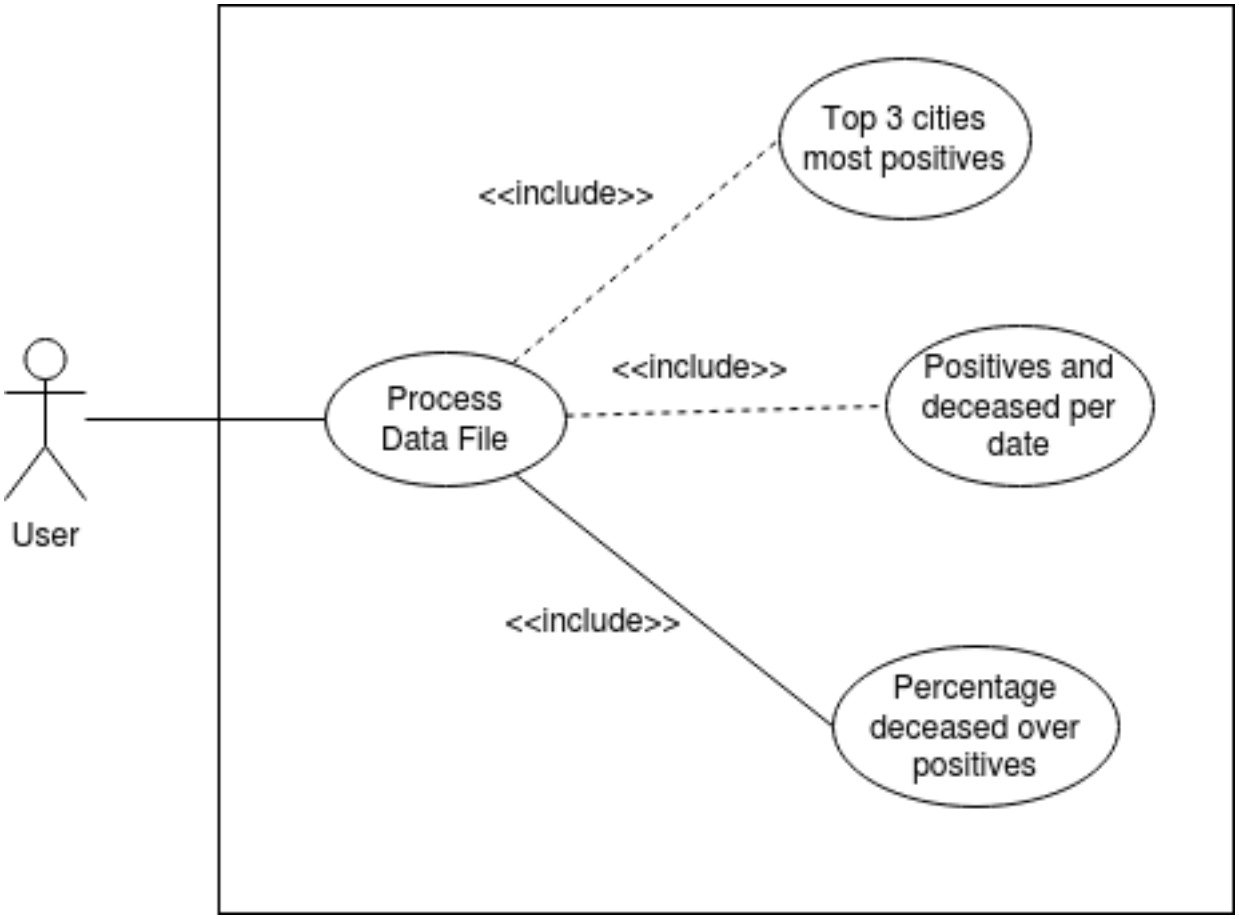

Figura 1: Casos de uso

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:

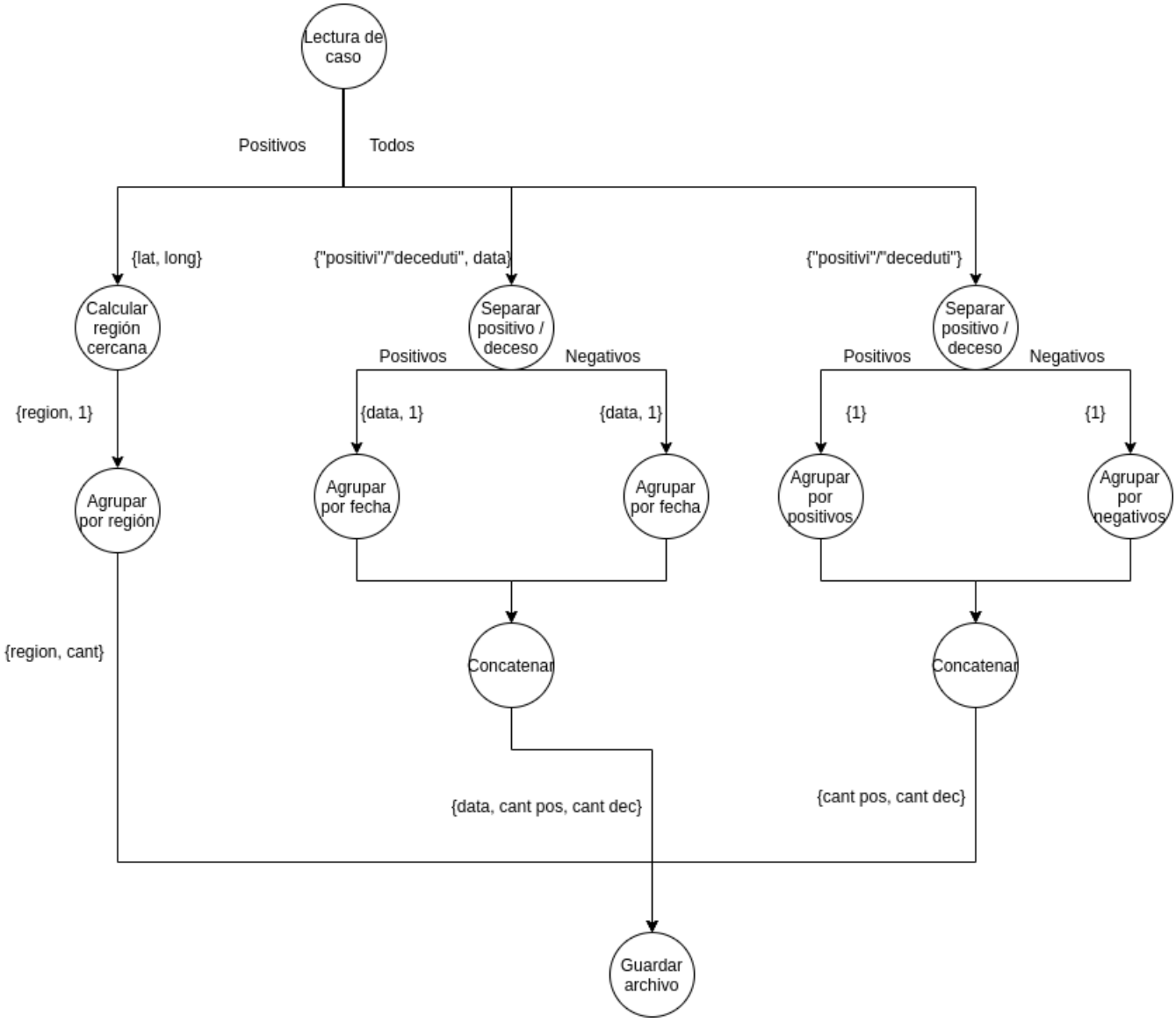

Figura 2: Directed Acyclic Graph (DAG) de operaciones

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:

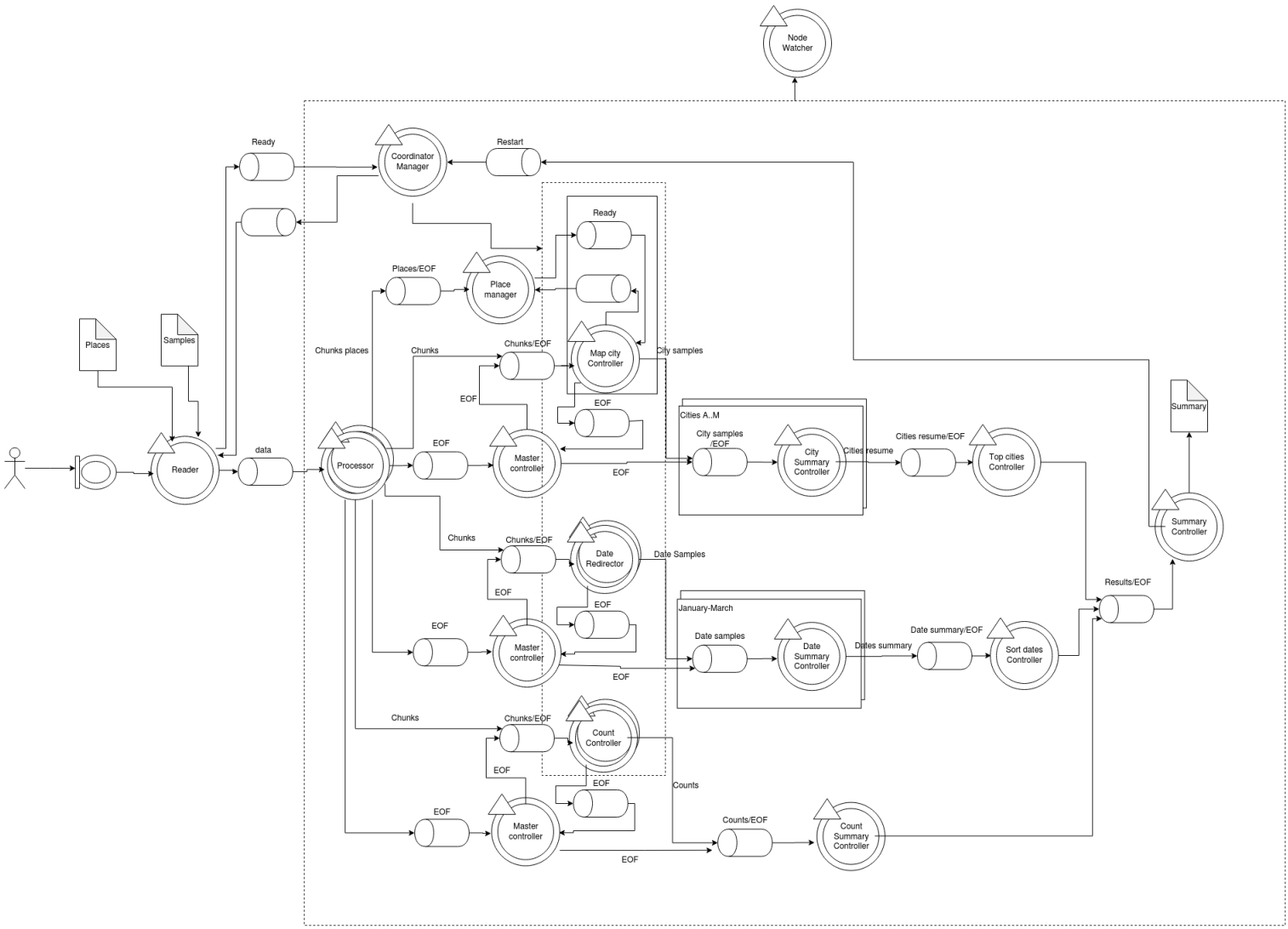

Figura 3: Diagrama de robustez completo

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

- Reader: Se encarga de procesar los archivos de entrada del lado del cliente y enviarlos al servidor.
- Processor: 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.
- 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.
- Place Manager: Recibe la información de los lugares y los guarda en un cluster de almacenamiento estable.
- Node Watcher: Vigilan el sistema y se encargan de levantar nodos que se encuentren caidos.
- Coordinator Manager: Se encarga de coordinar el inicio y fin de una conexión de manera que solamente un cliente se encuentre ejecutando por vez. También coordina a los redirectores para determinar cuando deben de estar ejecutandose y cuando esperando por una nueva conexión.

También incluimos un sistema de almacenamiento seguro basado en lideres de la siguiente forma:

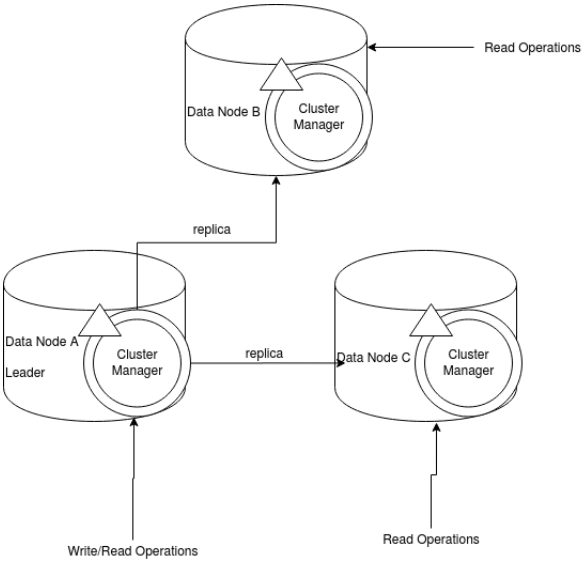

Figura 4: Sistema de almacenamiento seguro Leader-based

Este sistema es utilizado por los nodos para almacenar datos (guardar estados) y poder filtrar mensajes duplicados. Funciona utilizando el algoritmo de Bully para la elección de líder, de forma tal que solo el líder es el único nodo habilitado para realizar escrituras mientras que el resto de los nodos (Y él mismo) son capaces de realizar operaciones de lectura. El sistema opera de la siguiente manera:

- Escritura: Ante una operación de escritura, el nodo líder se encarga de replicar la operación al resto de los nodos. Una vez que dichos nodos finalizaron, recién ahí se notifica al cliente que la escritura fue realizada.

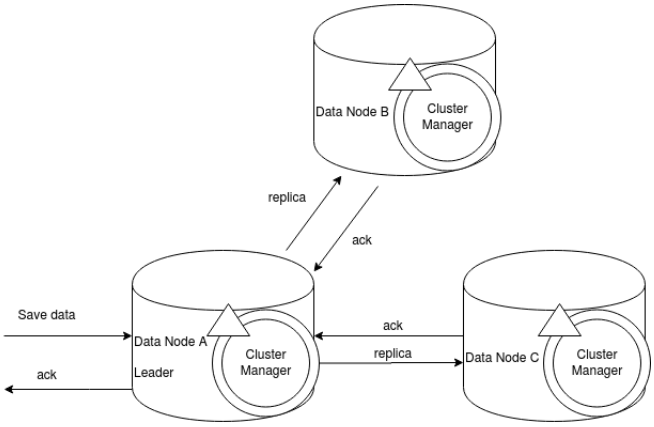

Figura 5: Sistema de almacenamiento seguro realizando una escritura

Por como se puede ver en la imagen, el nodo leader encargado de las escrituras debe esperar por la confirmación de las replicas antes de confirmarle al cliente que la operación fue exitosa.

- Lectura: Simplemente se lee de cualquiera de los nodos, se garantiza la consistencia causal de las operaciones en secuencia de un cliente, por ejemplo, si un cliente escribe un archivo y luego intenta leerlo, siempre leerá lo que escribió. En cambio, si otro cliente intenta leer en durante la escritura, se encontrará con información vieja o, si tiene suerte, podrá leer la nueva información si lo hace justo luego de la escritura.

Desde el punto de vista de un diagrama de robustez que incluya colas, podemos ver que el diseño del cluster de datos es de la siguiente forma:

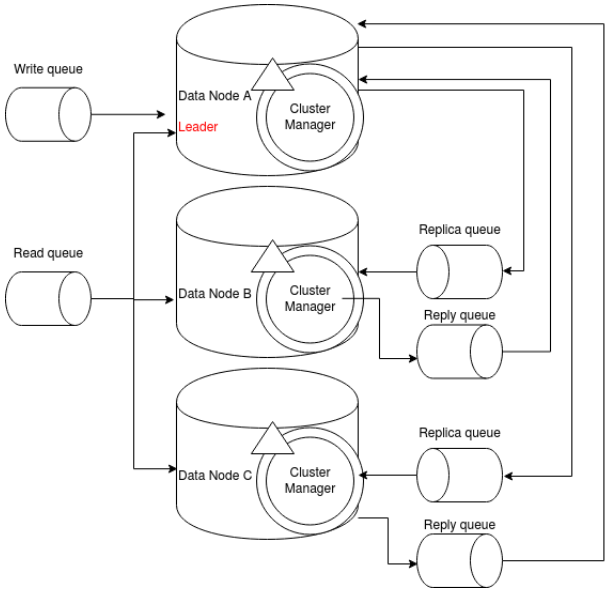

Figura 6: Diagrama de robustez del sistema de almacenamiento seguro

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:

- Hay más costo de transmisión de datos, el hecho de que el Processor transmita información replicada a traves de 3 colas implica más costo de transporte de datos.
- Actualmente el sistema opera con todos los nodos de almacenamiento activos, en caso de un nodo caído, el sistema debe esperar a que se levante para proseguir, esto se debe a que, en caso de que un nodo de almacenamiento se levante, el hecho de garantizar la consistencia entre todos los nodos es bastante complejo.
- Solo se realizan operaciones de sobreescritura para garantizar la consistencia de datos, es decir, no se puede concatenar información ya existente a un archivo garantizando la consistencia de los datos. Esto se debe a que no hay un mecanismo para invalidar una transacción cuando la replica para algún nodo de almacenamiento falla.

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:

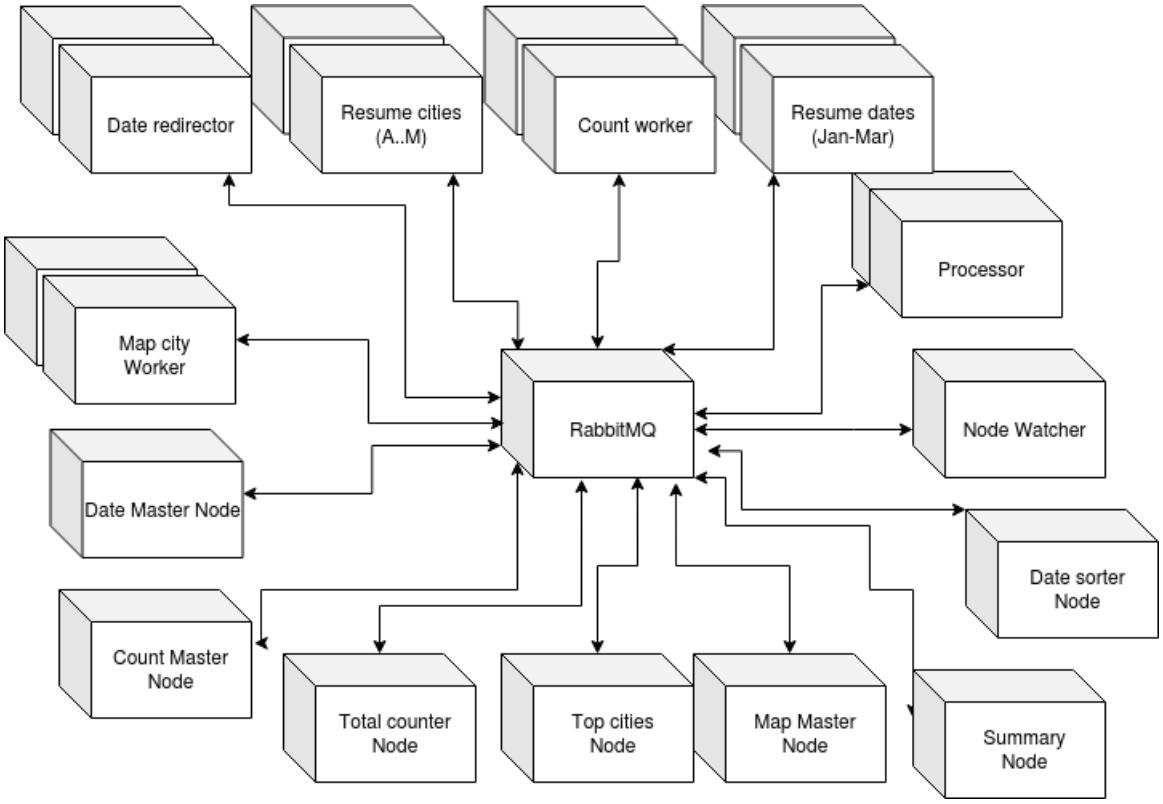

Figura 7: Diagrama de despliegue

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.

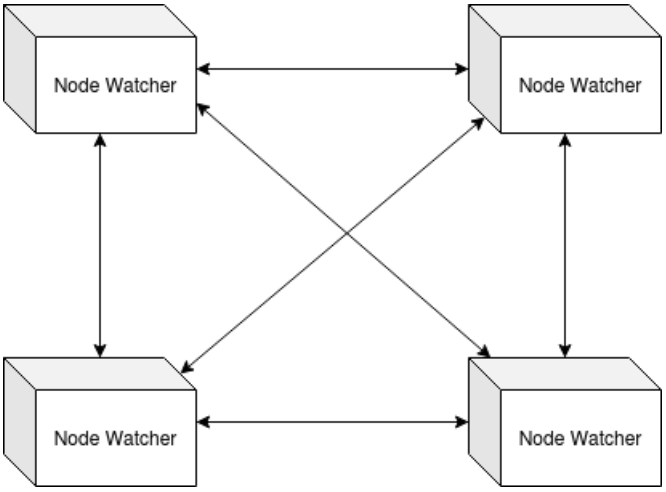

Figura 8: Diagrama de despliegue de los node watcher

Por otro lado, tenemos los node watcher encargados de levantar nodos caídos que están comunicados entre ellos sin usar RabbitMQ y que utilizan un sistema basado en un líder utilizando el algoritmo de Bully para consenso de líder. El líder de estos nodos es el encargado de recibir estados de salud por parte del resto de los nodos, y, ante la falta de un estado de salud por parte de algún nodo durante un tiempo mayor al preestablecido, se toma el trabajo de levantar al nodo asumiendo que se encuentra caído.

7. Algoritmo de elección de líder

Como ya se mencionó anteriormente, se optó por utilizar el algoritmo de elección de líder Bully tanto para los nodos de almacenamiento como para los nodos watcher. A continuación, mostramos el funcionamiento del algoritmo para los nodos watcher, ya que el algoritmo funciona de igual forma para ambos tipos de nodos:

Inicialmente tenemos todos los nodos enviándose estados de salud entre ellos, cuando un nodo detecte que otro nodo está caído, si es el lider inicia una

elección, caso contrario simplemente lo remueve de su lista de conexiones. Al levantarse todos los nodos al comienzo, automáticamente el nodo de mayor valor (Nodo D) se convierte en el lider.

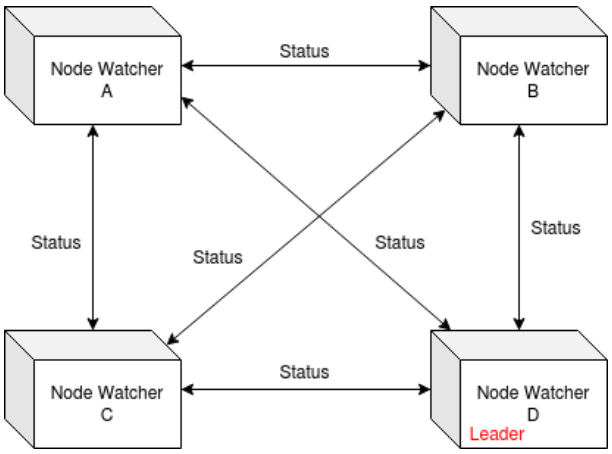

Figura 9: Algoritmo de Bully: Conexión inicial

Cuando algún nodo (En este caso Nodo A y Nodo B) detecta que el nodo lider está caido, comienza una nueva elección

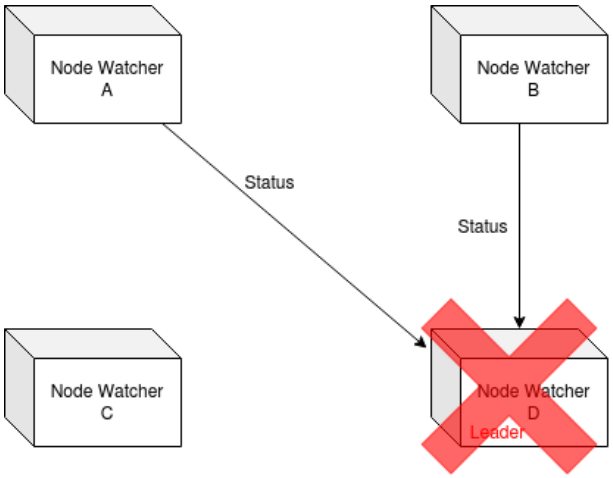

Figura 10: Algoritmo de Bully: Lider caído

El nodo que inicia la elección envía un mensaje de tipo elección a aquellos nodos con un identificador mayor el suyo (En este caso, A le envía a B y C, y B le envía a C)

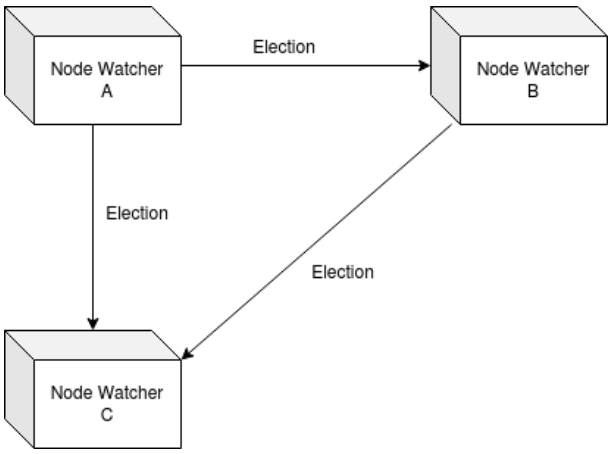

Figura 11: Algoritmo de Bully: Inicio de elecciones

Aquellos nodos que reciban un mensaje de elección deben de responder al nodo que les envío la elección para ahora ellos hacerse cargo de la elección (En este caso, el único nodo capaz de proseguir con la elección es el nodo C)

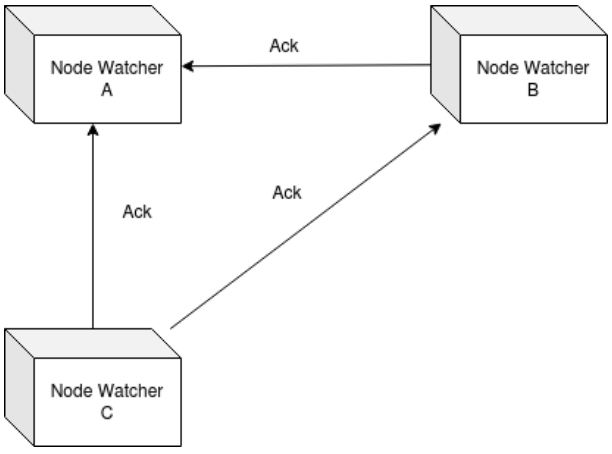

Figura 12: Algoritmo de Bully: ACK elecciones

Como el nodo C inició una elección y no hay ningún nodo con un identi-

ficador mayor al suyo, se autoproclama líder y envía un mensaje de tipo líder al resto de los nodos (Siempre el nodo con mayor identificador será el nuevo lider).

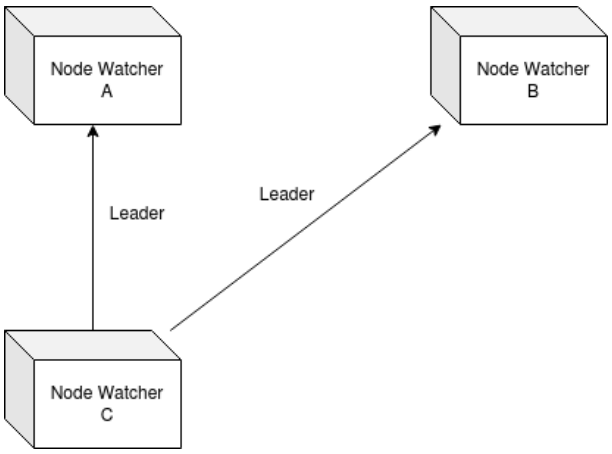

Figura 13: Algoritmo de Bully: Nodo C nuevo lider

Nuevamente el sistema vuelve al caso inicial, todos los procesos enviándose mensajes de estado de salud con aquellos nodos que se encuentren activos.

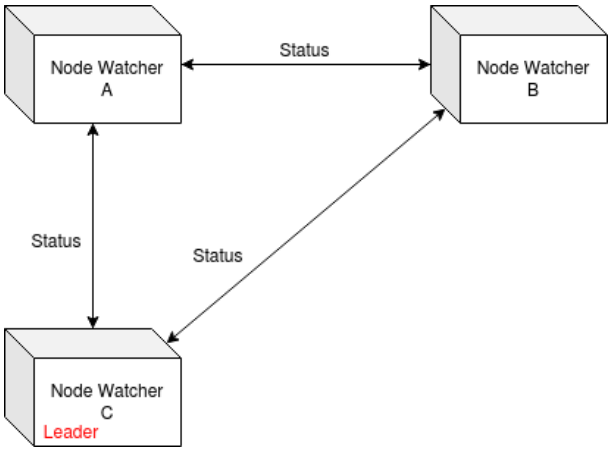

Figura 14: Algoritmo de Bully: Sistema con nuevo lider

Si eventualmente un nodo se levanta, este se comunica con el resto de los nodos para restablecer las conexiones pérdidas e inicia una nueva elección.

En caso de ser el nodo con mayor identificador de los que se encuentran vivos, automaticamente se proclamará como lider (En este caso, el nodo D se autoproclama lider)

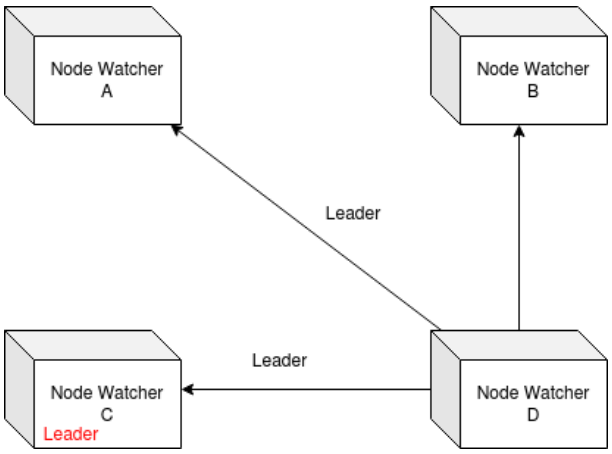

Figura 15: Algoritmo de Bully: Nodo levantado

Nuevamente el sistema se mantiene enviando estados de salud entre todos los nodos activos

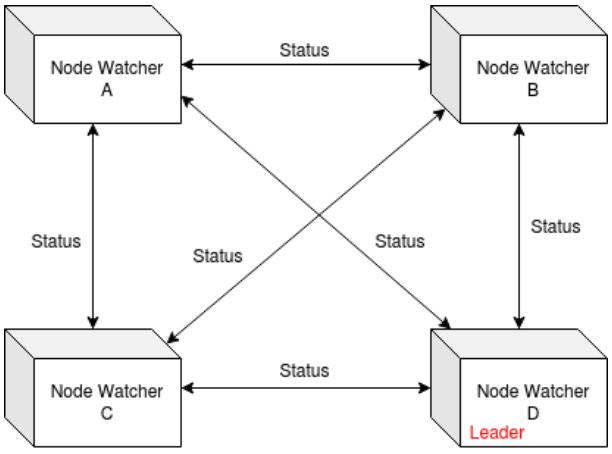

Figura 16: Algoritmo de Bully: Conexión final

8. 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:

8.1. Diagrama de paquetes

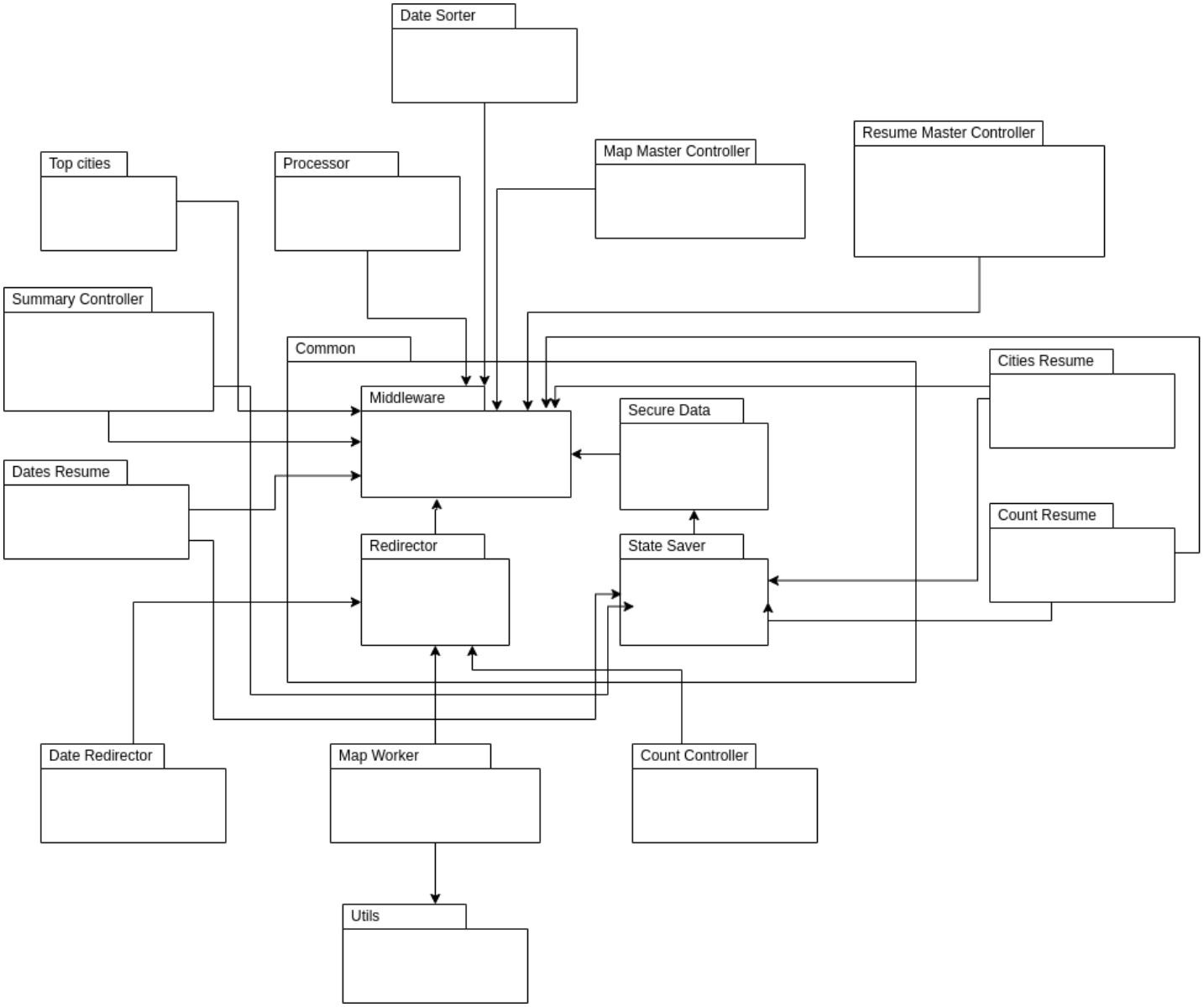

Figura 17: Diagrama de paquetes

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.

9. 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.

9.1. Diagrama de actividades

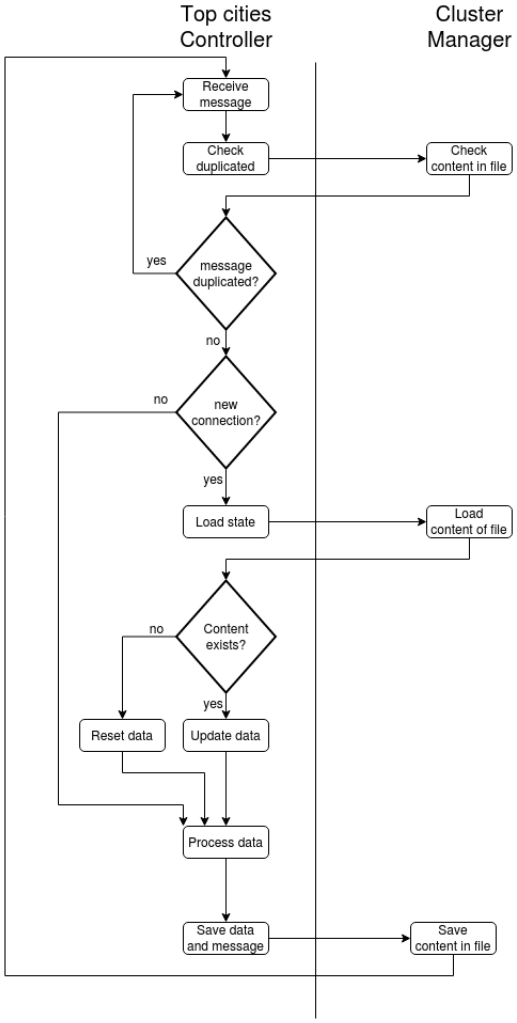

Figura 18: Diagrama de actividades del filtro de duplicados y guardado de estados del Top cities controller

Al igual que para el Top cities controller, para el resto de los reducers se sigue el mismo algoritmo, primero se filtran duplicados, luego se chequea si el mensaje recibido pertenece a la misma conexión para cargar datos y finalmente guardarlos antes de devolver un ACK a la cola correspondiente. Esto trae como consecuencia que, en el peor de los casos, se envíe más de 1 vez el mismo mensaje a la próxima etapa, pero la misma contendrá un filtro de duplicados con lo que no hay problemas.

10. 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.

10.1. Diagrama de clases

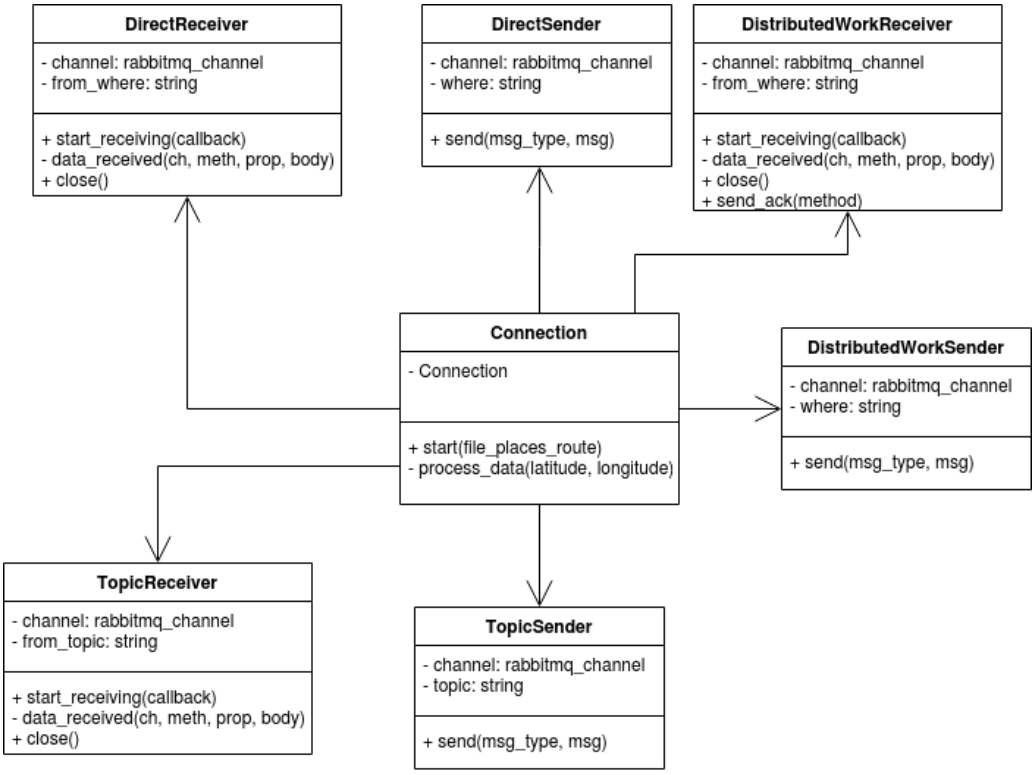

Figura 19: Diagrama de clases del Middleware

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.

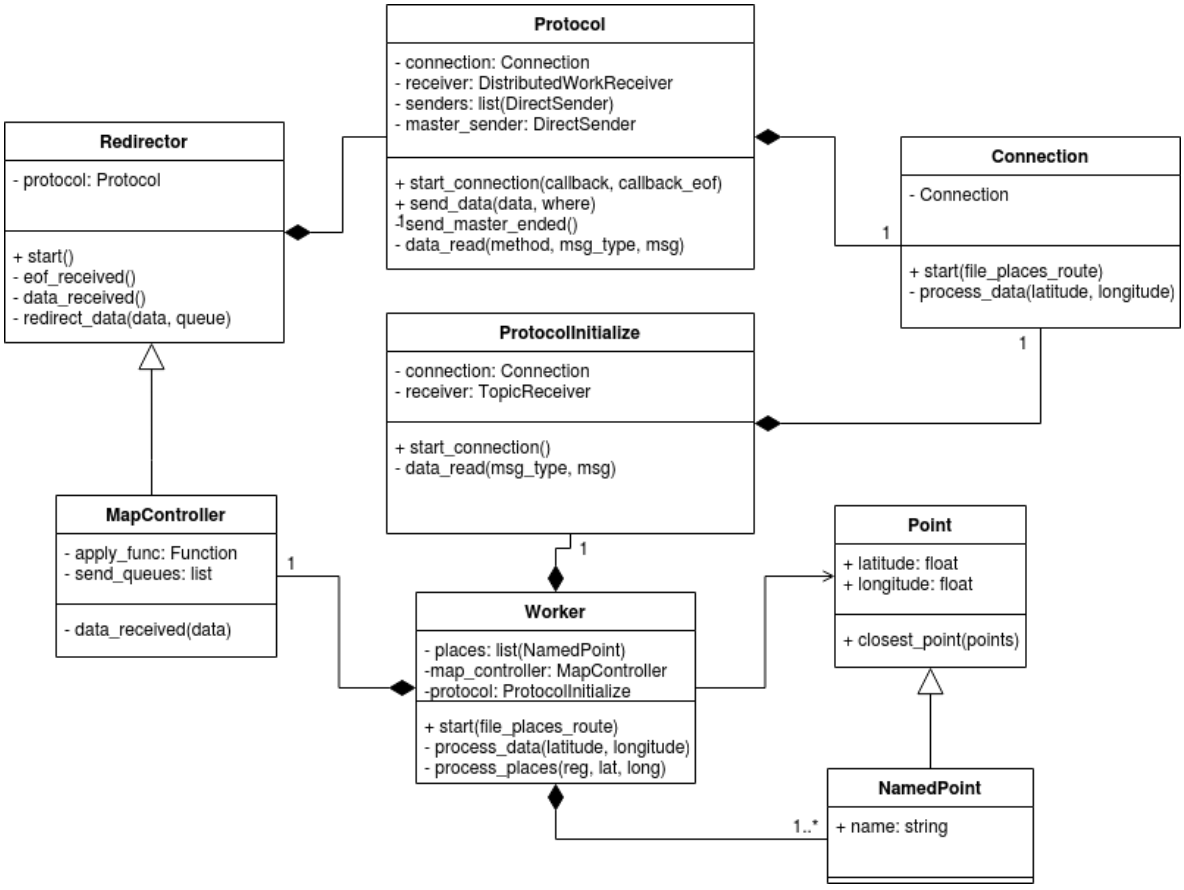

Figura 20: Diagrama de clases del Map Worker

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.

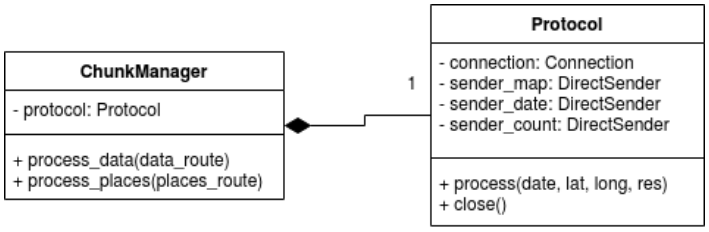

Figura 21: Diagrama de clases del Chunk Manager

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.

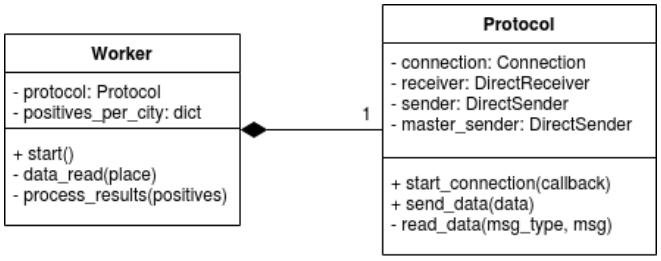

Figura 22: Diagrama de clases del Cities Resume

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.

11. 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 **reader**, los resultados serán escritos en el archivo:
summary_controller/summary/summary_<connection_id>.txt
Para correr el programa, ejecutar:

make system-run

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 processors=<processors>

12. Limitaciones del sistema

Veamos ahora las limitaciones que posee el sistema. Por un lado, para que el sistema realice su flujo normal sin trabas, debemos contar con al menos un nodo de cada tipo. En caso de que algún nodo de procesamiento (processors, workers, reducers, etc.) se caiga, automáticamente será levantado por un nodo Watcher.

Por otro lado, podemos no contar con los nodos Watcher (es decir, pueden caerse todos), pero tenemos la desventaja de que no se levantan entre ellos. Es decir, si perdemos alguno de nuestros Watchers no sería inconveniente, ya que se seleccionará un nuevo líder y el funcionamiento será el mismo, es decir el flujo normal continúa. Pero, en caso de caerse todos, nuestro sistema queda vulnerable a caídas de otros nodos, que no serán notificadas ni se levantados.

Por último, es importante notar la diferencia con los Data Nodes. Debido a que manejan archivos, se decidió que no se levanten automáticamente, sino de forma manual. Además, esto genera que, en caso de faltar nodos, no se cumpla con la replicación (recordemos que para que la escritura sea efectiva se debe escribir en todos los nodos) y por lo tanto el sistema deje de funcionar correctamente.

13. 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).
- Separar el archivo de docker-compose en múltiples archivos para cada operación, de manera que sea más legible y cómodo de usar.
- 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.