75.74 - Sistemas Distribuidos I

TP4 - Alta Disponibilidad y Tolerancia a Fallos

Reddit Memes Analyzer

1° Cuatrimestre de 2022

Eliana Gamarra - 100016

Gianfranco Conti - 102740

Alejandro Pernin - 92216

Índice

Introducción

En el siguiente informe se presenta una solución a un sistema distribuido para el análisis de datos extraídos de Reddit.

Dicho sistema posee un middleware que permite la utilización de queues de RabbitMQ. Se lanzarán varios servicios en simultáneo que se coordinarán para escribir y leer los datos de las queues.

El cliente enviará en una conexión los datos de los posts y comments en diferentes queue y estos datos completaran el recorrido para obtener tres salidas de los datos:

- Promedio de score de todos los posts
- URLs de memes que gustan a estudiantes (con comments sobre university, college, student, teacher, professor y con score mayor al promedio)
- Descarga del meme con mejor sentiment promedio

A continuación se presentan una serie de diagramas para facilitar el entendimiento del sistema.

Arquitectura

La arquitectura de este sistema se basa en la realizada para el trabajo práctico anterior pero se le agregó lo necesario para que el sistema sea tolerante a fallas y los nodos vuelvan a levantarse en caso de una falla. Para esto se hizo que aquellos nodos que almacenan algún tipo de estado interno lo persistan para poder restaurarlo luego de una falla, y además se agregó un **Health Check** para monitorear a los nodos y reiniciarlos en caso de fallas.

Vista lógica

En el siguiente dag se muestra como se irán procesando los datos que vayan ingresando al sistema, estos no cambia respecto de la anterior iteración

DAG

Vista Física

Diagrama de Robustez

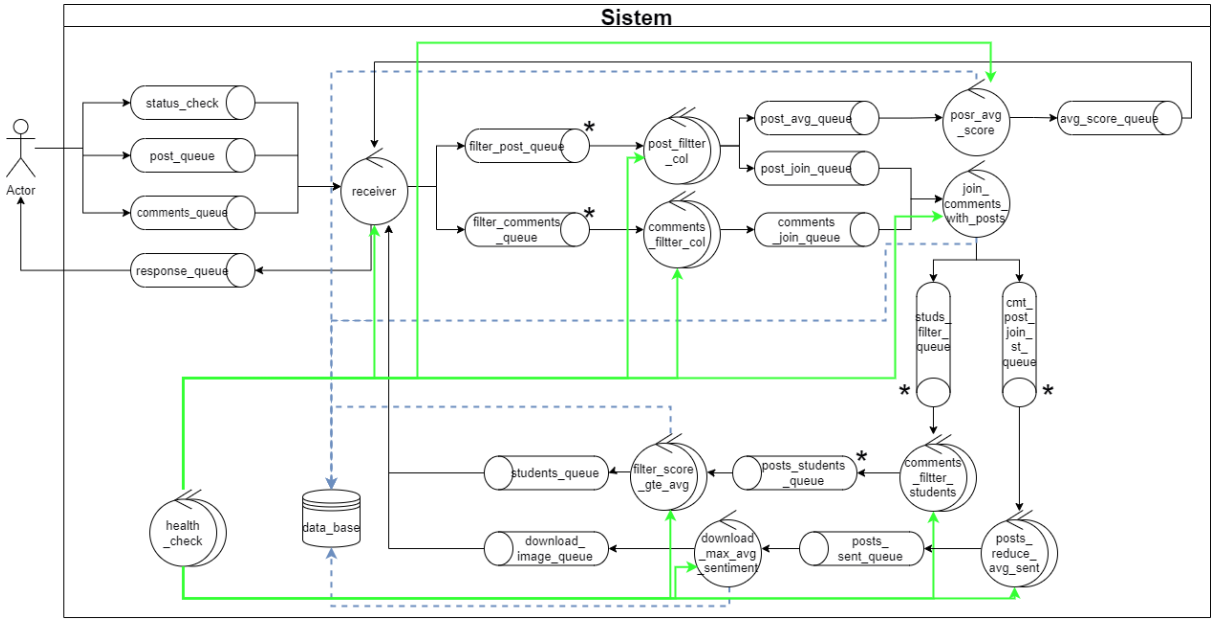


Diagrama de los distintos procesos involucrados y de qué manera se comunican por medio de queues de RabbitMQ.

Para establecer las diferentes conexiones con Rabbit se usó la librería Pika. Que crea una conexión y canal de comunicación, este será el Middleware. En el diagrama también se puede observar la base de datos indicada como un cilindro y los procesos de healt check (lo cuales también se comunican mediante colas de rabbit pero para mantener la legibilidad del diagrama fueron obviadas).

También se tuvo en cuenta que las conexiones de rabbit pueden fallar y reiniciarse por eso es que las queues fueron señaladas como durables, y por la misma razón se marcan los mensajes en modo persistente al comenzar a enviar mensajes.

Cliente

A la izquierda se tiene el punto de entrada del usuario a la app, donde el cliente, que está conectado a la red del sistema, pregunta el estado actual del sistema. Si el sistema le responde que esta disponible, es decir que no hay ningún nodo conectado actualmente, el cliente comienza a enviar mediante chunks las lineas leidas del CSV de post y comments a sus respectivas queues.

Estos son enviados al receptor que automáticamente los envía al resto del flujo para comenzar a procesarse.

El cliente va a estar consultando su estado cada X tiempo para saber si el sistema ya terminó de procesar el set de datos enviado.

Si otro cliente quiere conectarse cuando el sistema está ocupado, este último le responde que está ocupado.

Sistema

Una vez que el receiver lee las queue de comments y posts que envio el cliente entran al sistema y empiezan a procesarse

Ya que el archivo tiene muchas cantidad de filas a procesar se eligió tener varios workers (la cantidad es configurable) para filtrar los datos innecesarios.

He de aclarar que los servicios que debían esperar la totalidad de los datos para poder enviar el resultado obtenido (como lo son: Join Comments With Post, Post Avg Score y Download Max Avg Sentiment) deben mantener persistido su estado.

La salida de cada procesamiento se vuelven a comunicar por medio de queues que recibirán los distintos procesos hasta que el receptor obtiene los resultados obtenidos y estos son enviados nuevamente al cliente junto con un mensaje de finalización para que el cliente se cierre.

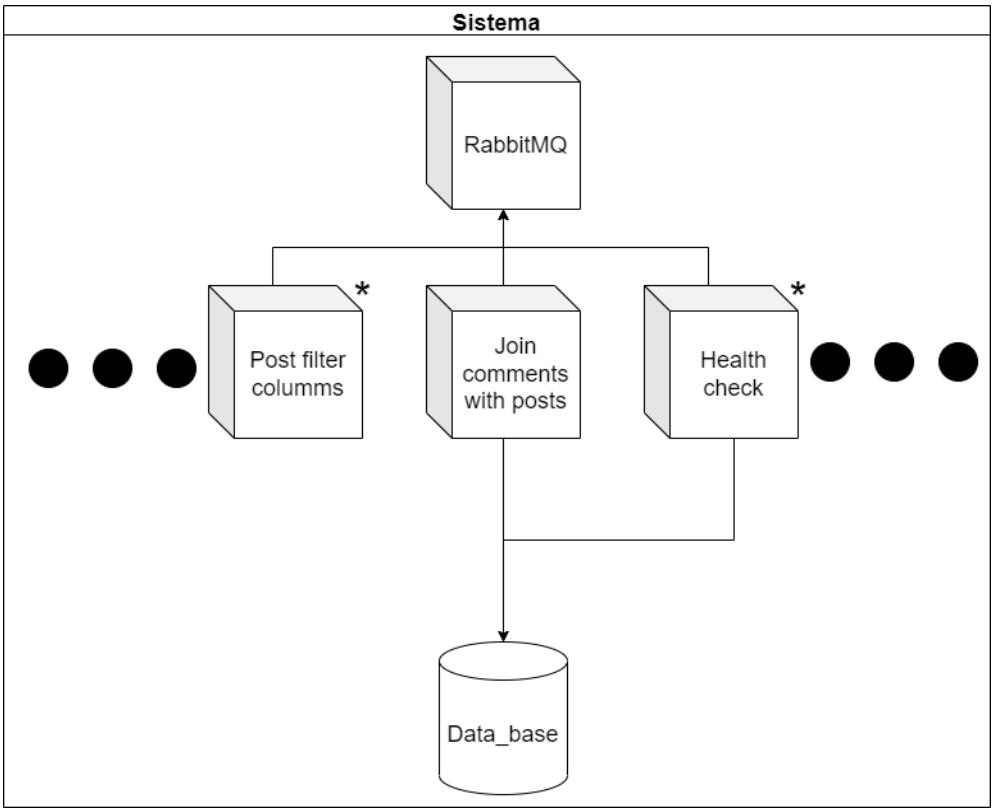
Para la persistencia de los datos en los nodos que tienen que esperar a la totalidad de los datos para poder responder o que tienen un estado se hace un guardado en un archivo de su estado actual por cada dato recibido, así si ocurre un fallo y se cae el sistema esté al levantarse se reanuda en el último estado disponible

Health Check:

Se tiene un algoritmo de elección de leader del tipo bully que envía mensajes para comprobar el estado actual de cada uno de los nodos hasta el momento, si el nodo líder no está disponible se debe iniciar el algoritmo de bully para elegir un nuevo líder

Diagrama de despliegue

El despliegue del sistema es bastante sencillo ya que todos los nodos se comunican a través de rabbitMQ por lo que se puede desplegar cada nodo por separado siempre en cuando se puedan conectar con rabbit y los nodos que requieren persistencia de datos puedan acceder a la base de datos. En el siguiente diagrama de despliegues se muestra justamente algunos de estos nodos y sus conexiones.



Vista de Procesos

Diagrama de Actividad

A continuación se muestra el diagrama de actividad para obtener los posts que cumplen gustarle a los estudiantes (con comments sobre university, college, student, teacher, professor) y con score mayor al promedio

No se tiene en cuenta en el diagrama el paso inicial de filtrar columnas, ya que no tiene significancia.

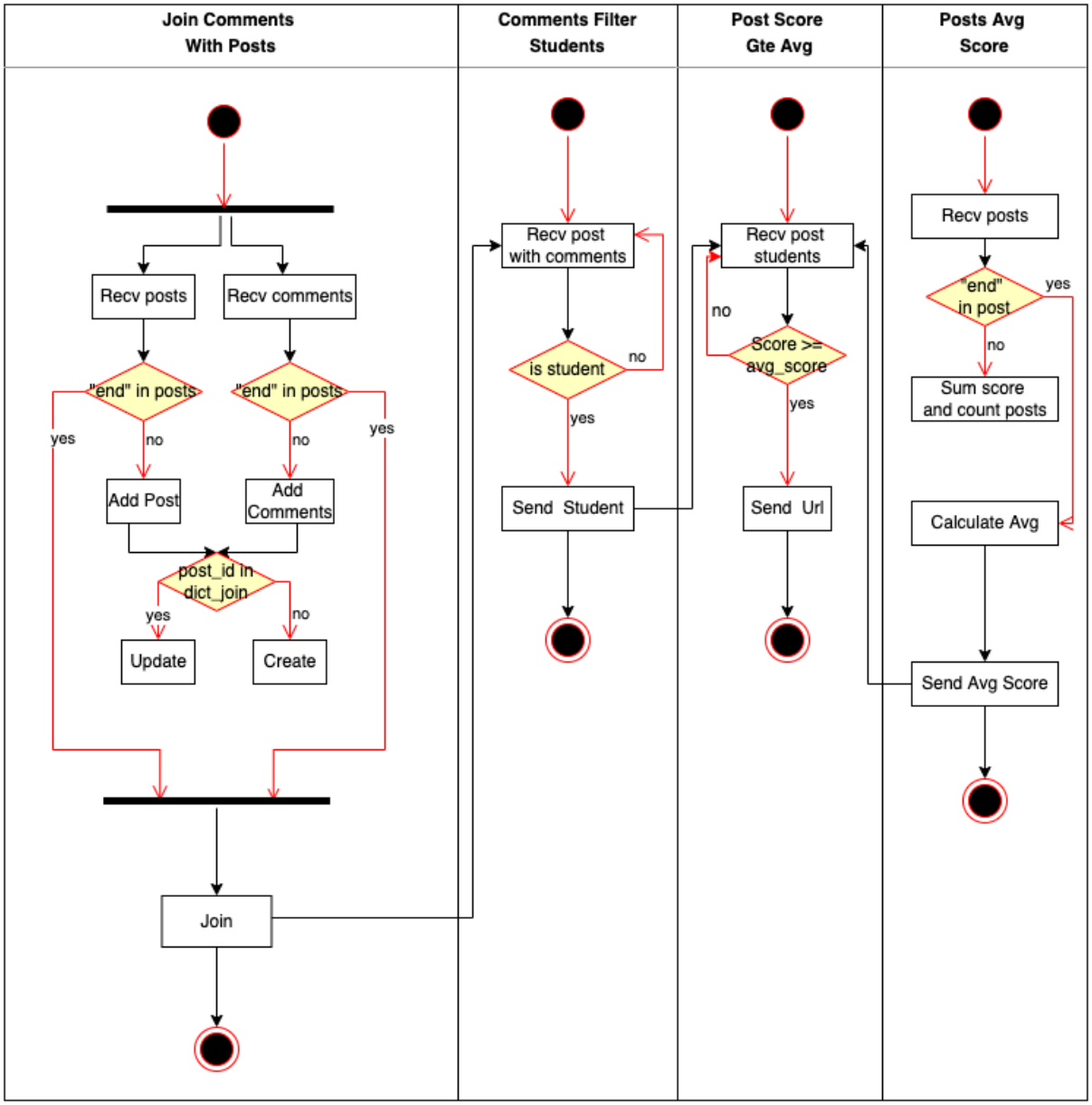
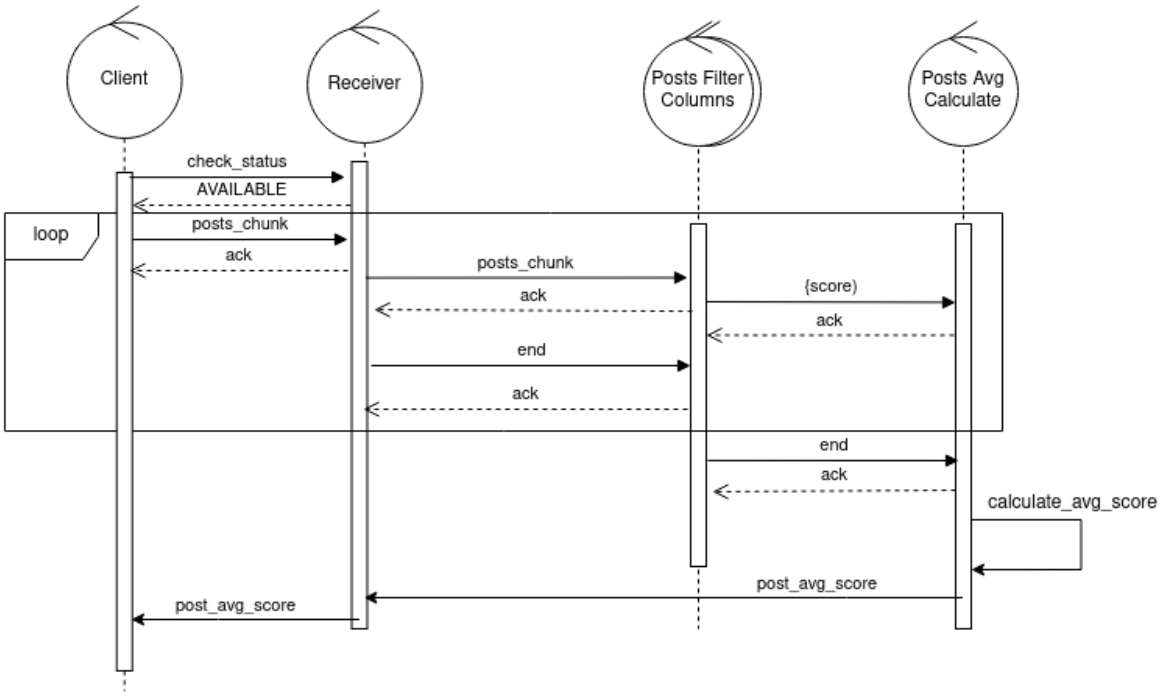


Diagrama de secuencia

Se muestra a continuación el diagrama de secuencia del recorrido hasta obtener el avg score de todos los posts.

El primer mensaje que el cliente envía al receiver es para checkear el status del sistema, si el receiver indica que el sistema está disponible el cliente puede comenzar a enviar datos. Luego de ser aceptado el cliente comienza enviando todos los post del csv, que luego Post Filter Columns elimina los que fueron eliminados o no tienen url y se los que restan se lo envía al Post Avg Source. Este va recibiendo y sumando el score y contando la cantidad de post recibidos en variables internas. Cuando finalmente recibe el mensaje de finalización, calcula el avg_score y por último envía al receiver el resultado obtenido y este último lo reenvía al cliente.

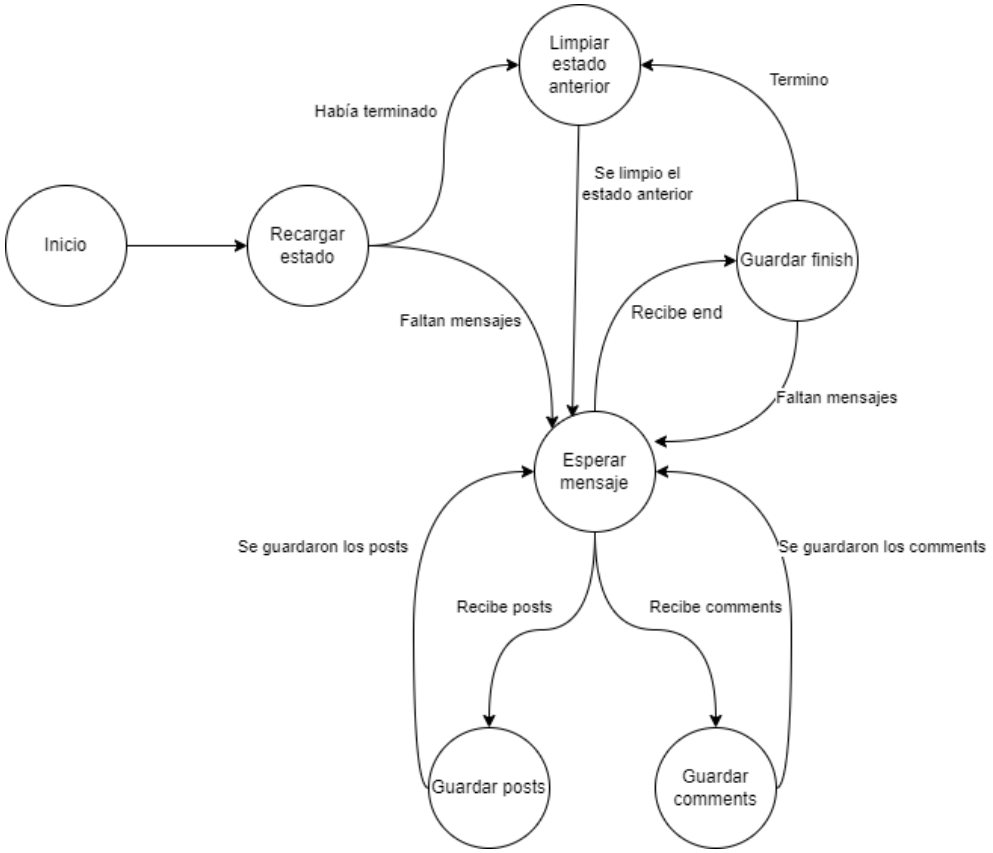


Persistencia

Para aquellos nodos que guardan estado es importante que en caso de morir este estado se conserve para ser recuperado. Para esto cada nodo es levemente diferente ya que se guardan distintos valores y la lógica de cada uno de estos nodos es diferente, pero a modo de ejemplo se describira como funciona la persistencia del nodo de join.

En el nodo join se deben persistir, los posts recibidos, los comentarios correspondientes a este post, y los ends. Además el join cuenta con un vector de hashes de mensaje para saber si un mensaje es repetido y en ese caso descartar. La lógica del join consta de guardar el mensaje recibido para poder ser recuperado en caso de morir(se guarda cada mensaje para poder utilizar writes atómicos, por lo que para recrear el estado simplemente se lo agrega como si fuese un mensaje nuevo), y además se guarda un vector para saber que filtros ya han terminado. El ack del mensaje se realiza en caso de recibir información nueva luego de guardarlo en la base de datos y en caso de recibir un end se hace o inmediatamente luego de guardarlo en memoria en caso de no ser el último, o caso contrario luego de enviar los resultados del join y limpiar el estado. El problema que tiene este modelo es que en caso de morir el nodo durante el envío de los resultados este enviará duplicados de los mensajes ya enviados lo cual es muy poco performante aunque no genera conflicto ya que los nodos posteriores filtraran los duplicados.

Para el manejo de duplicados del join se hashea el mensaje recibido para que en caso de recibir un mensaje con el mismo hash identificar el duplicado y filtrar.



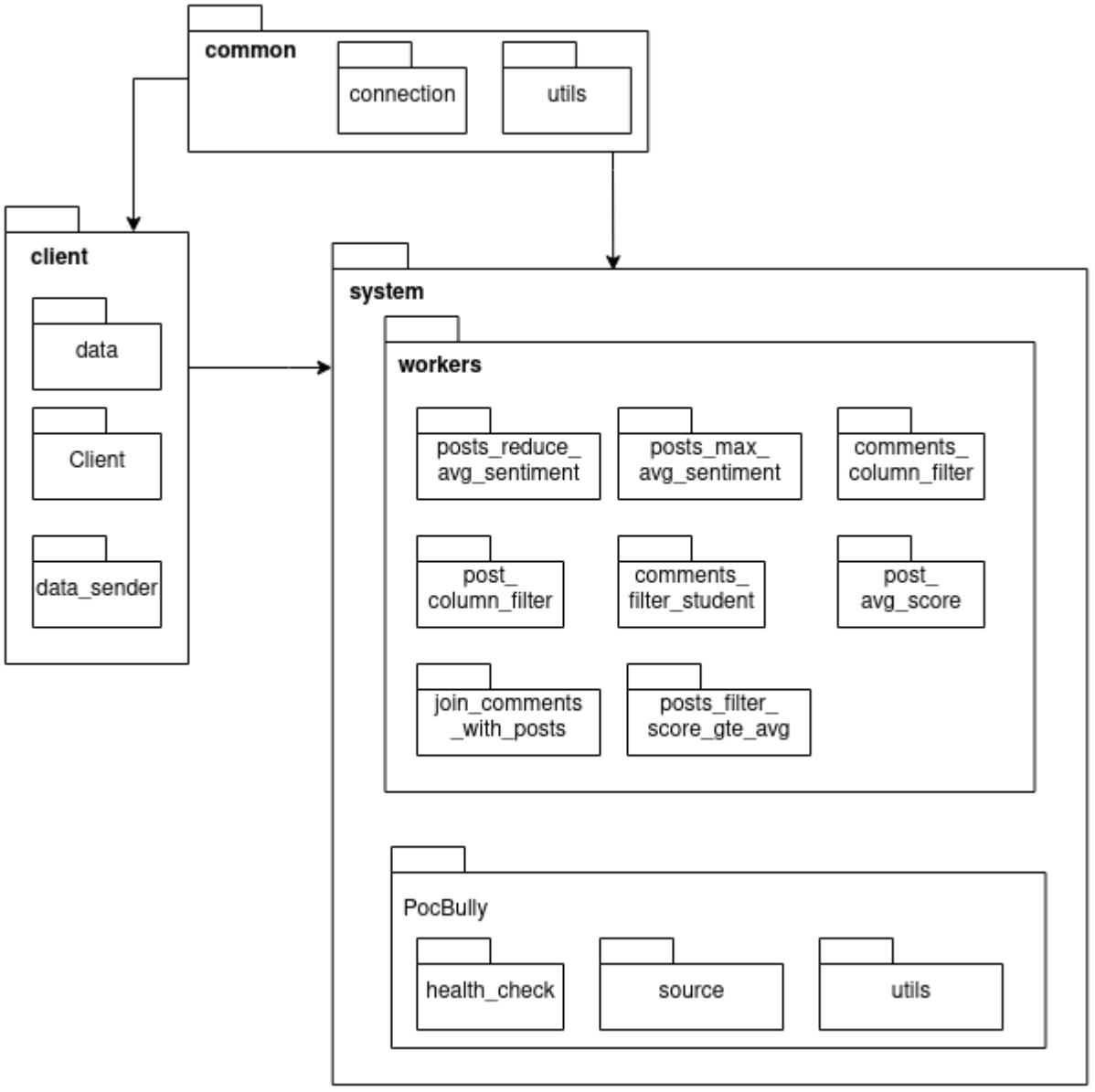
Vista de desarrollo

Diagrama de Paquetes

En el siguiente diagrama se muestra el diagrama de paquetes y cómo se relacionan unos con otros.

Cada servicio es un paquete diferente, que se despliegan en diferentes nodos, pero a su vez todos dependen de un paquete common que contiene el middleware y un paquete de útiles para la configuración de logging

Fuera del sistema se tiene el cliente que se conecta con el sistema y el sistema tiene los workers y el paquete del algoritmo de bully



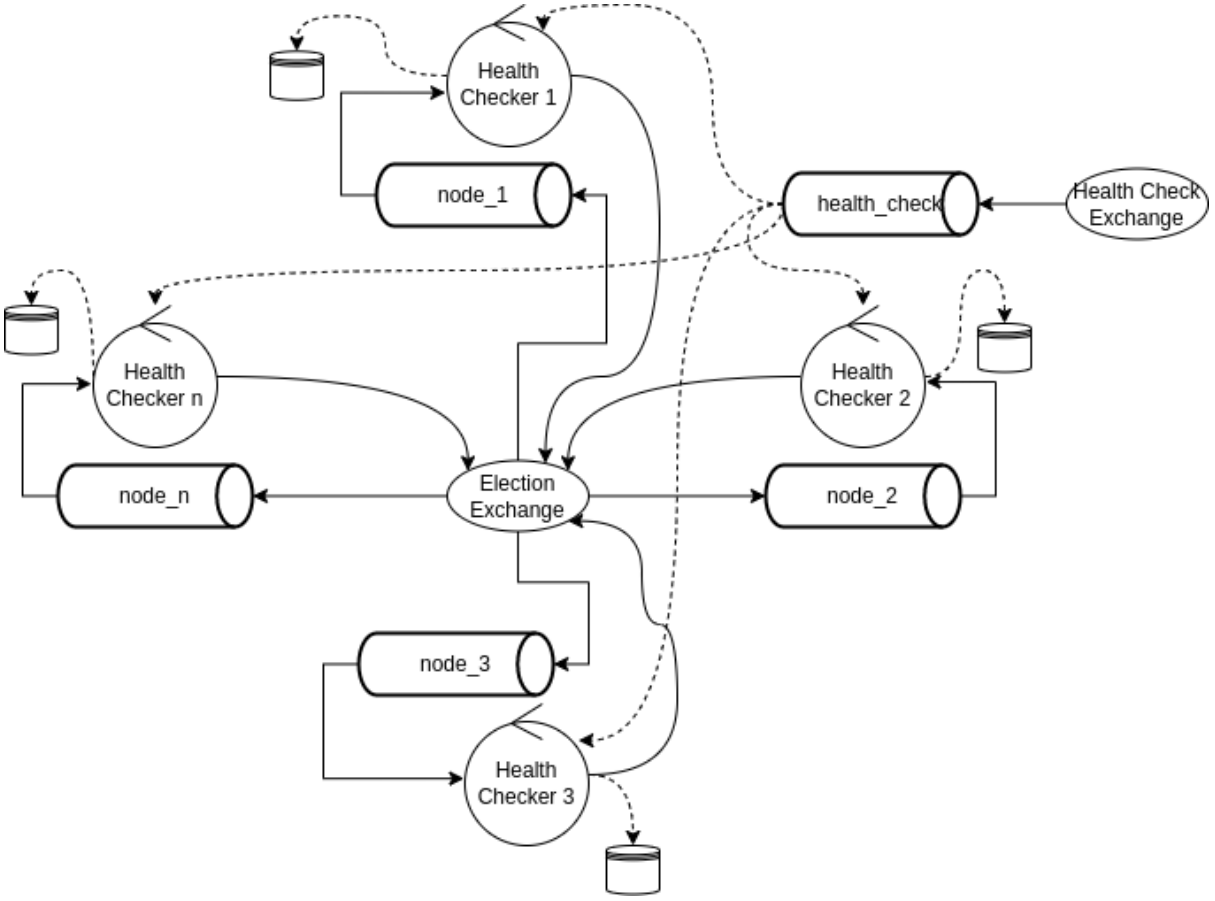
Health Check

El sistema posee un cluster de nodos llamados ‘health-checkers’ que se encargan de monitorear el estado de los nodos del sistema y levantarlos en caso de encontrar uno caído.

El cluster es tolerante a fallos, empleando un nodo líder que es el que efectivamente realiza el monitoreo del sistema y el resto de los nodos estarán a disposición por si el líder deja de operar.

El cluster responde al patrón de Bully y posee la siguiente arquitectura

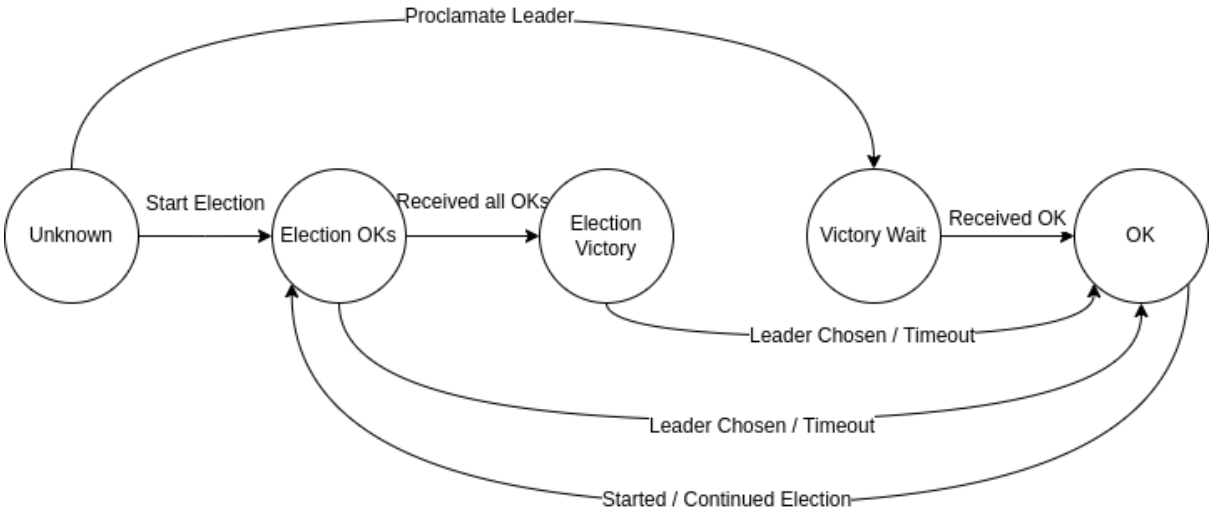
- Exchange y queue de ‘Health Check’, éstos recibirán los mensajes de los nodos del sistema (‘health-beats’) los cuales serán consumidos por el nodo health-checker líder.
- Election Exchange: Dicho exchange distribuirá todos los mensajes dentro del cluster, tanto los destinados a la organización de una elección como el ‘hearbeat’ del líder.
- Queues: Cada nodo del cluster posee su propia queue
- Persistencia: Existe una persistencia que será empleada por el nodo líder para resguardar los datos de los nodos monitoreados.



A su vez cada nodo implementa una máquina de estados para representar sus estados y transiciones posibles

- Unknown: El nodo recién se inicia y desconoce el estado del cluster. Puede iniciar una elección o autoproclamarse líder (si posee el id más alto)
- Election OKs: El nodo espera el OK de los nodos con id mayores que continuarán la elección. Al recibir OK de todos los nodos superiores pasa a esperar el resultado de la elección. Si sucede un timeout se proclama líder.

- Election Victory: El nodo espera el resultado de la elección y pasar a OK, de haber timeout se proclama lider.
- Victory Wait: El nodo se autoproclamó lider (no mediante elección) y espera a recibir un OK de un potencial lider actual. Una vez que lo obtiene u ocurre un timeout, pasa al estado OK.
- OK: Es el estado estable de un nodo. Procesa los mensajes recibidos por los otros nodos del cluster.



Implementación de Monitoreo

Para facilitar la adaptación del sistema a ser monitoreado se implementó una clase *mixin*
la cual implementa dos métodos

- Start: Inicia un subproceso el cuál consiste en un loop que manda un ‘healthbeat’ cada N segundos (configurable)
- Exit: Termina el subproceso y espera su terminación

Por lo cuál la clase que implementa un nodo actual sólo debe heredar de la misma y emplear dichos *start* y *end* en su lógica. Como único requisito externo se tiene que el nombre del container debe ser accesible mediante la variable de entorno ‘CONTAINER_NAME’.

Implementación Health Check

El nodo líder del health check posee dos subprocesos

- Heart Beat: Envía un mensaje al cluster cada N segundos para informarles de que continúa operativo
- Health Check: Consume de la queue de health check los mensajes (health beats) enviados por los nodos del sistema. Cada vez que recibe un mensaje persiste en un archivo todos los nodos que están siendo monitoreados con el timestamps de la última actualización. En caso de que el líder se caiga, esta información va a estar disponible para el nodo que tome su lugar.

Escenario

El cliente consulta el estado del sistema, si está disponible especifica los CSVs de comentarios y posts, luego el cliente espera los resultados del sistema uno por uno hasta que recibe la totalidad y recibe un mensaje de finalización.

Diagrama Casos de Uso

En este pequeño diagrama se muestran los causo del sistema