Trabajo Práctico 2

Ventura Julián
Padrón: 102391

Sistemas Distribuidos I
1ºC 2022
Fecha de Entrega:
11/06/2022

Contenido

Decisiones Tomadas

A continuación, se listan una serie de decisiones tomadas a la hora de diseñar e implementar la arquitectura.

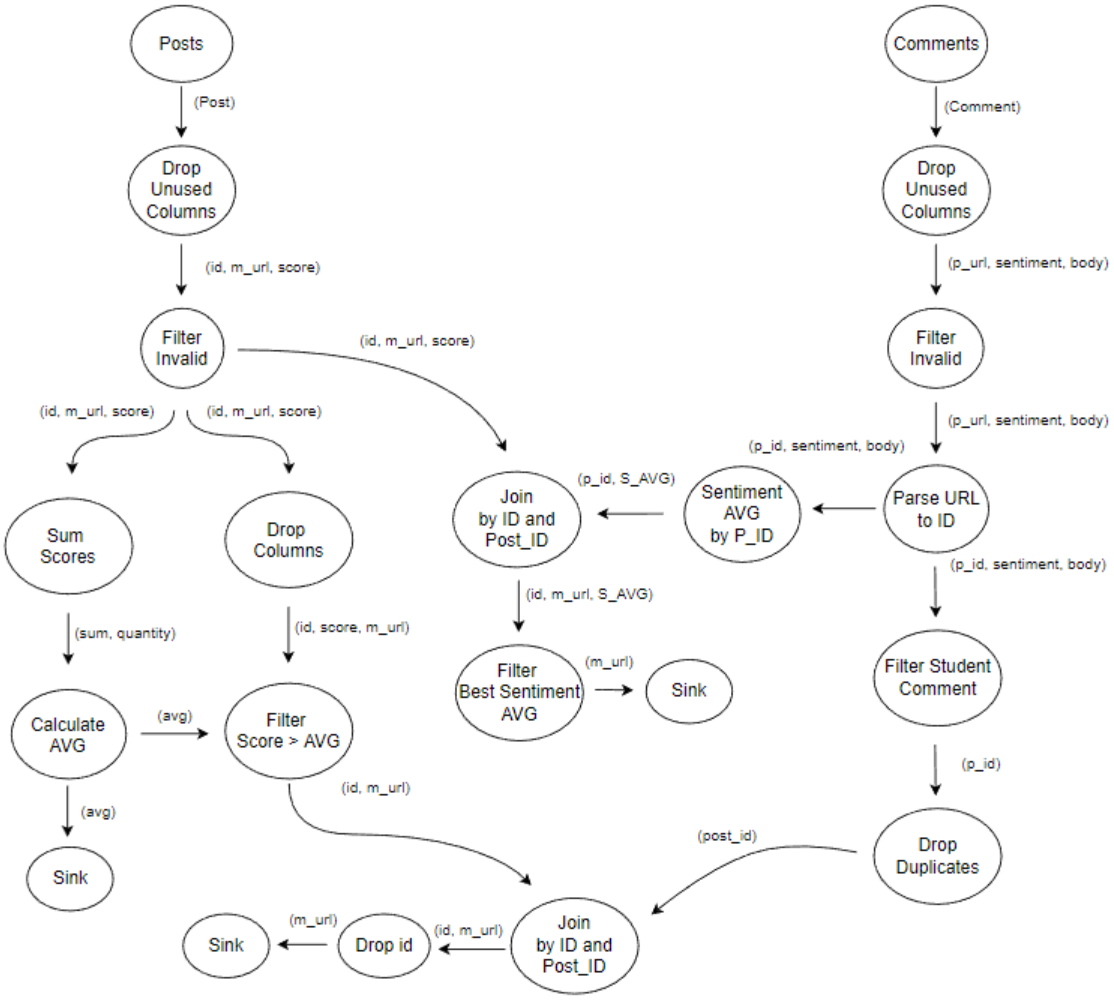
- El protocolo se implementará en binario, con el objetivo de minimizar el transporte de información entre cliente y servidor.
- El servidor deberá soportar una conexión con un único cliente, la cual disparará la ejecución de tres operaciones sobre el stream de datos recibido. Tras la culminación de estas, el servidor finalizará.
- El empleo de la señal SIGTERM para la finalización de los procesos será necesaria en toda ejecución del sistema.
- Al recibir una señal SIGTERM, todo proceso del sistema buscará realizar una salida graceful, interrumpiendo las operaciones y abriendo la posibilidad a perdidas de datos.
- El stream de datos, tanto de posts como de comentarios se recibe de forma paulatina e intercalada por parte del cliente.
- El servidor conoce el formato de los datos a recibir, pero no posee datos precargados.
- La conexión con el cliente se realiza por TCP, a partir de un protocolo binario hecho a medida para esta problemática.
- La ejecución de las operaciones y almacenamiento temporal de los datos se realizará en memoria.
- Los datos de entrada, tanto de posts como de comentarios, pueden tener datos incompatibles que deberán ser filtrados.
- La obtención meme de sentimiento promedio máximo será realizada luego de ejecutar una operación de join entre comentarios y posts, para garantizar la obtención de un resultado, siempre y cuando el join no sea vacío. Esto debido a que puede no tenerse información del post cuyo sentiment promedio resultó ser el mayor, sea porque fue filtrado o porque no existía en el set de datos.

Arquitectura Propuesta

A continuación, se explicará la arquitectura utilizada en el sistema, con la ayuda de diagramas siguiendo un modelo de vistas 4+1.

Vista Lógica

En el siguiente diagrama DAG se puede observar cómo es el flujo de datos desde la recepción de los posts y comentarios, hasta la obtención de los resultados finales del cómputo en los nodos marcados como sink.



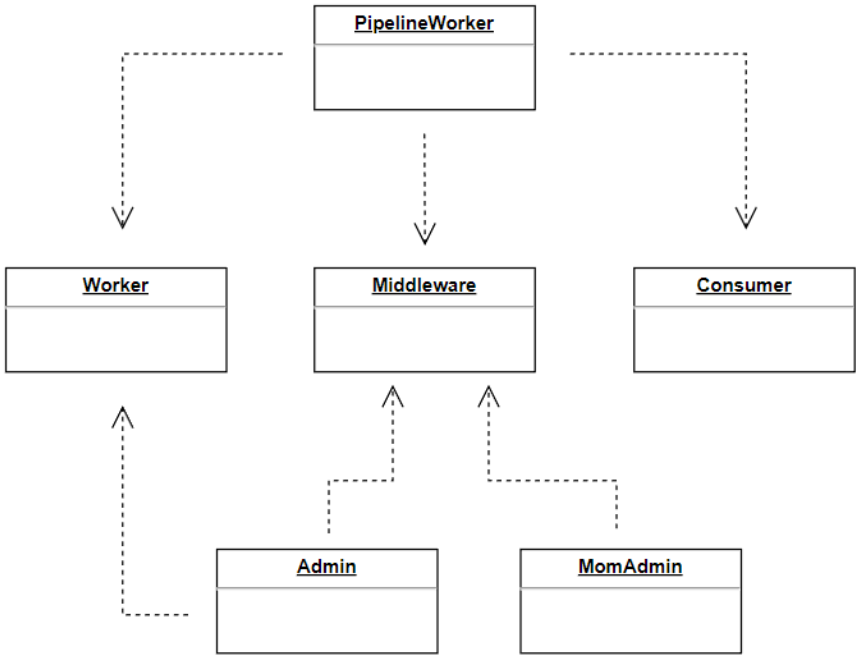
Las tuplas que acompañan a las aristas representan las tuplas de datos que viajan por el pipeline de una etapa a la otra.

Como se ha comentado en las decisiones tomadas, se incluyen etapas de filtrado de datos para ambos streams. Adicionalmente, la etapa de join de sentimiento promedio antecede al cálculo del mejor sentimiento promedio, para garantizar un resultado ante la posible falta de posts.

En los siguientes diagramas de clases se puede apreciar como el DAG anterior fue traducido a entidades del sistema.

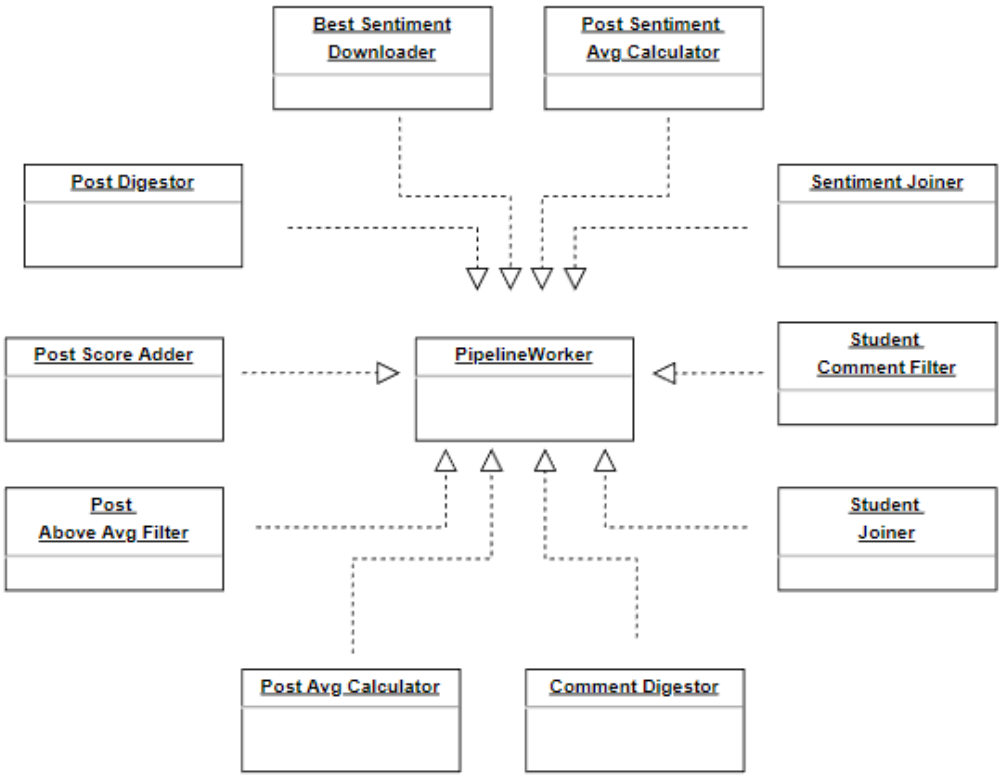
Como comentario, cada uno de estos diagramas se centran en las relaciones entre las entidades y no en los métodos y variables que estos posean, debido a eso se omiten los mismos.

Adicionalmente, se ha introducido una abstracción no existente en la arquitectura "PipelineWorker" para facilitar el entendimiento y fraccionamiento de los diagramas



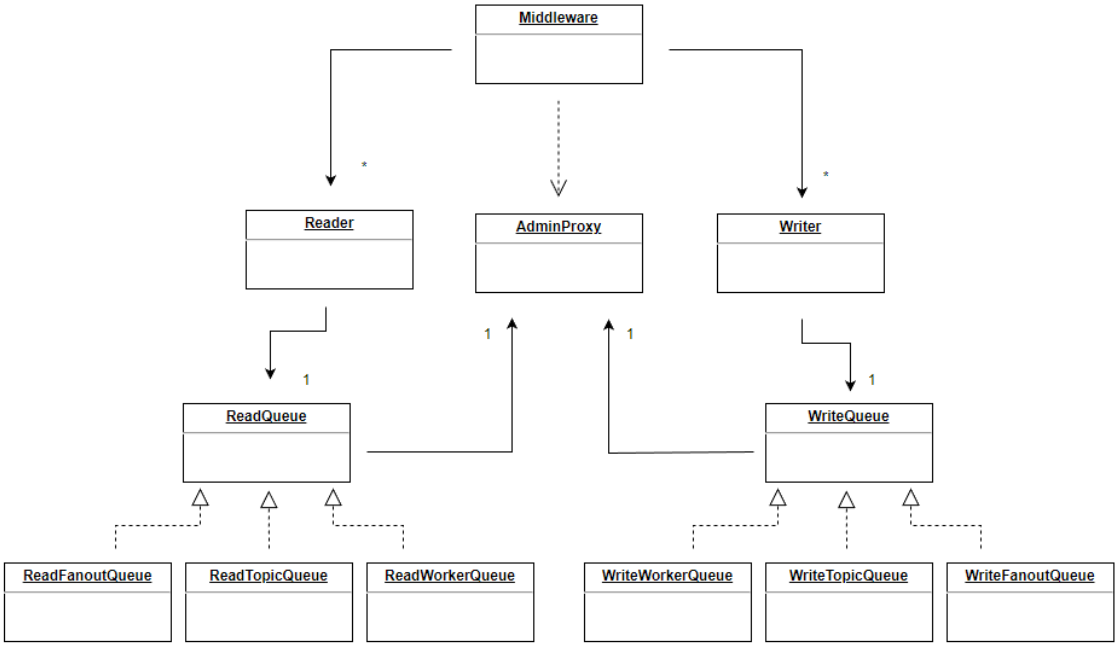
- Middleware: Contiene la lógica de cliente del middleware implementado. A su vez depende de otras estructuras no incluídas.
- MomAdmin: Contiene lógica de coordinación de creación y cierre de colas del middleware.
- Worker: Contiene toda la lógica necesaria para iniciar un nuevo proceso en el sistema, incluyendo el parseo de archivos de configuraciones y la conexión con el middleware.
- Consumer: Utilizado para consumir una cola de mensajes hasta recibir una señal de finalización o cierre de cola.
- Admin: Encargado de proveer la interfaz del servidor al cliente, recibiendo el stream de datos y aportando los resultados obtenidos.
- PipelineWorker: Cada proceso involucrado en el cómputo del sistema.

En el siguiente diagrama es posible observar gran parte de los vértices del DAG, transformados en entidades del sistema.



- Post Digestor: Realiza el filtrado de los posts que arriban al sistema, eliminando aquellas filas inválidas y columnas innecesarias.
- Post Score Adder: Realiza la sumatoria de los scores de los posts recibidos de la etapa anterior.
- Post Avg Calculator: Calcula el promedio de los scores de los posts, a partir de las sumatorias recibidas de la etapa anterior.
- Post Above Avg Filter: Elimina aqueyos posts cuyo score no supere el del promedio de todos los posts.
- Comment Digestor: Trabajo análogo al del Post Digestor, pero sobre los comentarios.
- Student Comment Filter: Elimina todos los comentarios cuyo cuerpo no posea alguna palabra relacionada al ámbito escolar.
- Post Sentiment Avg Calculator: Calcula el sentimiento promedio de cada post a partir del sentimiento de cada comentario asociado a él, con su id.
- Sentiment Joiner: Permite unir la información de los posts y el cálculo de sentimiento de cada uno.
- Student Joiner: Permite unir la información de los comentarios del ámbito escolar con los posts que superen el promedio de score.
- Best Sentiment Downloader: Permite descargar el meme correspondiente al post con mejor sentimiento promedio.

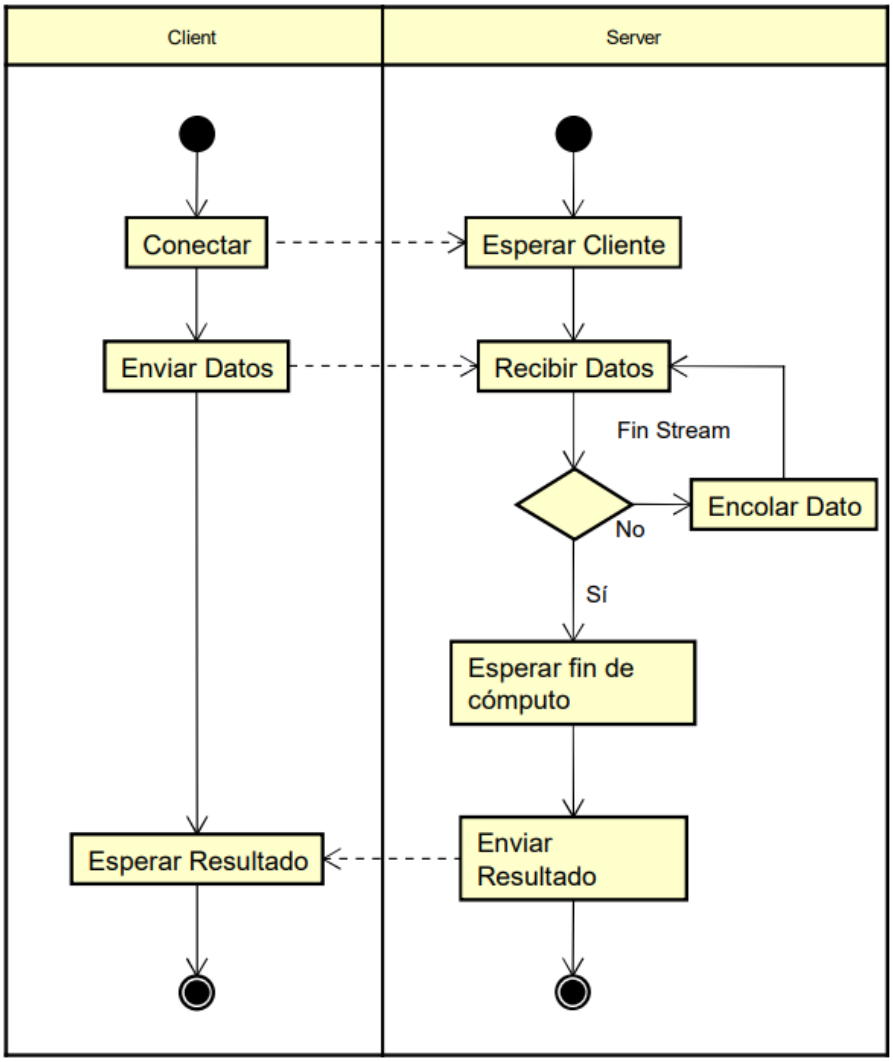
En el siguiente diagrama se muestra el modelado realizado en el middleware de mensajes implementado.



- Reader: Encargado de realizar la traducción de los mensajes recibidos del servidor de RabbitMQ a mensajes que cumplan la interfaz provista por el middleware implementado.
- Writer: Encargado de realizar la traducción de mensajes provenientes de procesos escritores al servidor de RabbitMQ.
- ReadQueue: Abstracción sobre una cola que permite realizar lecturas.
- ReadFanoutQueue: Permite realizar lecturas sobre una cola de tipo fanout.
- ReadTopicQueue: Permite realizar lecturas sobre una cola de tipo topic.
- ReadWorkerQueue: Permite realizar lecturas sobre una cola de tipo worker.
- WriteQueue: Abstracción sobre una cola que permite realizar escrituras.
- WriteFanoutQueue: Crea y permite realizar escrituras sobre una cola de tipo fanout, la cual replicará los mensajes a cualquier proceso que escuche en ella.
- WriteTopicQueue: Crea y permite realizar escrituras sobre una cola de tipo topic, sobre la cuál se escribirán mensajes indicando un tópico particular. Este podrá ser utilizado por cualquier proceso lector que escuche en ella para discriminar los mensajes que desee obtener.
- AdminProxy: Establece una interfaz de comunicación al administrador del middleware, permitiendo el reporte de nuevas colas y el cierre de estas.

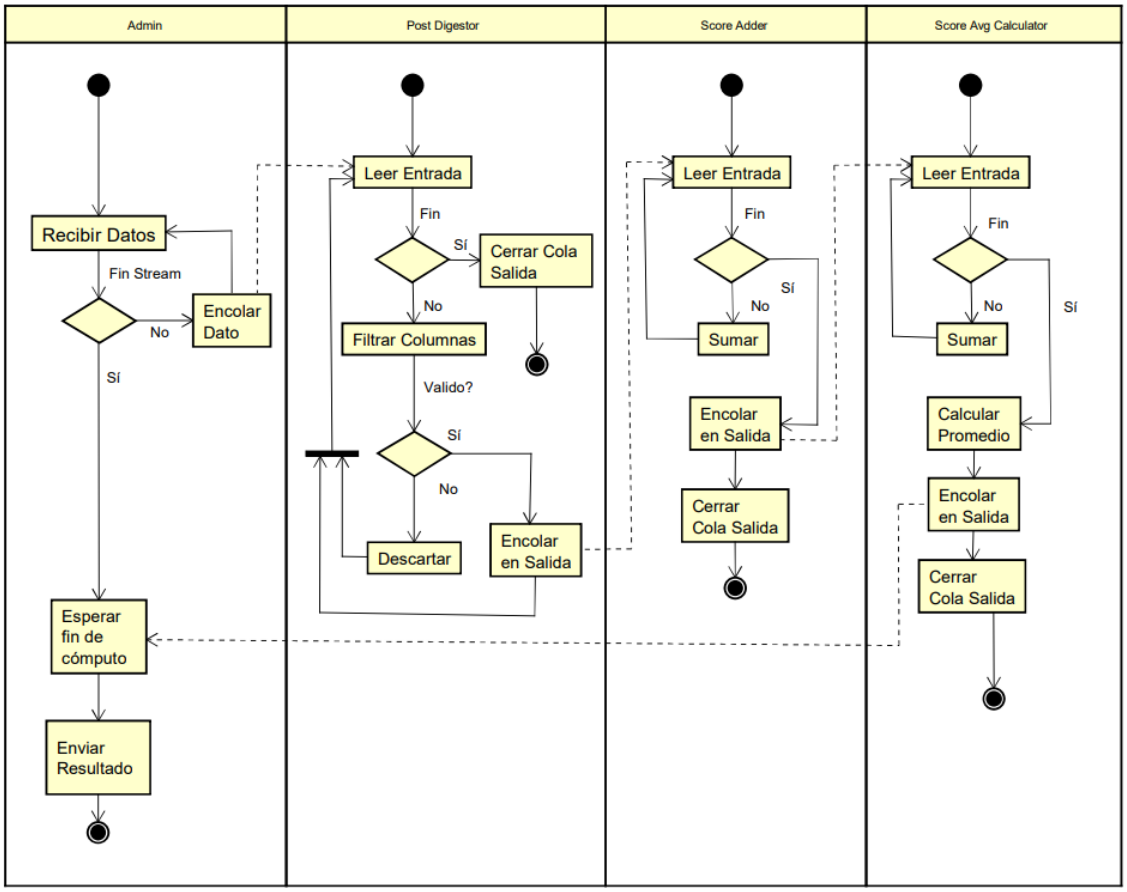
Vista de Procesos

En el siguiente diagrama de actividad se muestra cómo es el flujo del programa al realizar la conexión entre la aplicación cliente y el servidor.



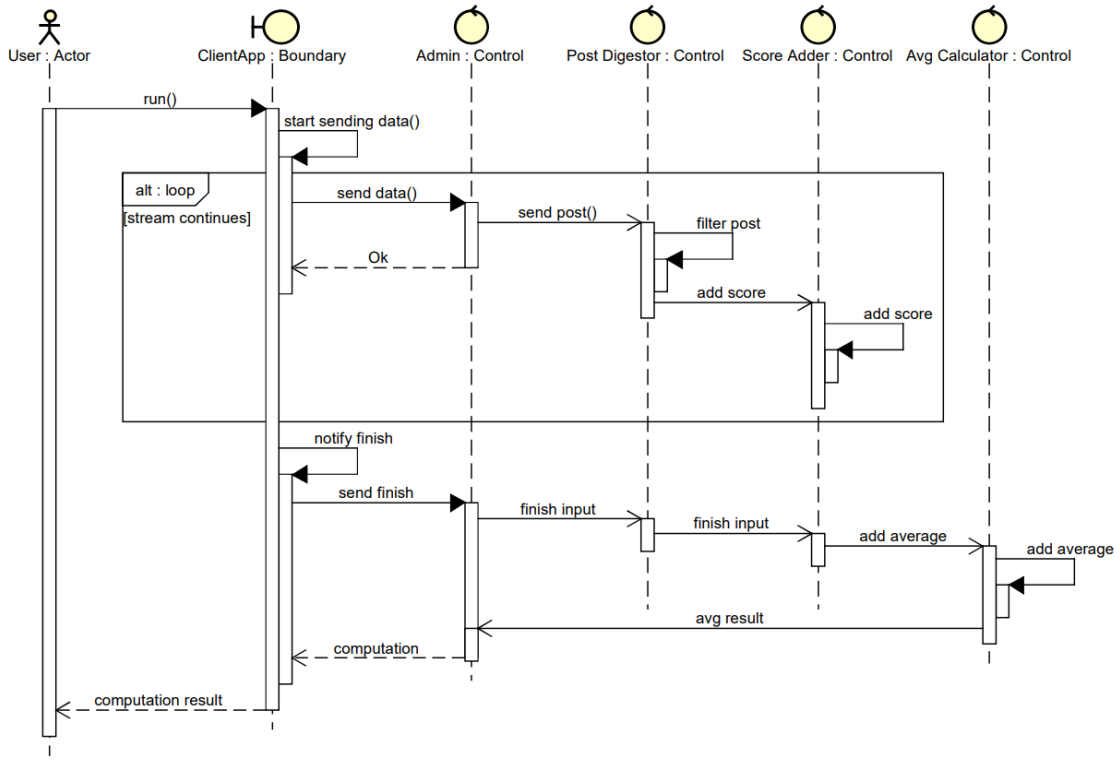
Los datos que se reciben son encolados en una cola de posts o en una de comentarios para ser procesados por etapas posteriores, según corresponda.

En el siguiente diagrama se observa un ejemplo de procesamiento del score promedio de posts, en donde se ven involucrados distintos procesos de forma asincrónica.



De este diagrama es importante destacar que la sincronización entre procesos se logra de manera sencilla cuando los procesos escritores cierran la cola en la cuál se encuentran escribiendo, esto gracias a la abstracción provista por el middleware.

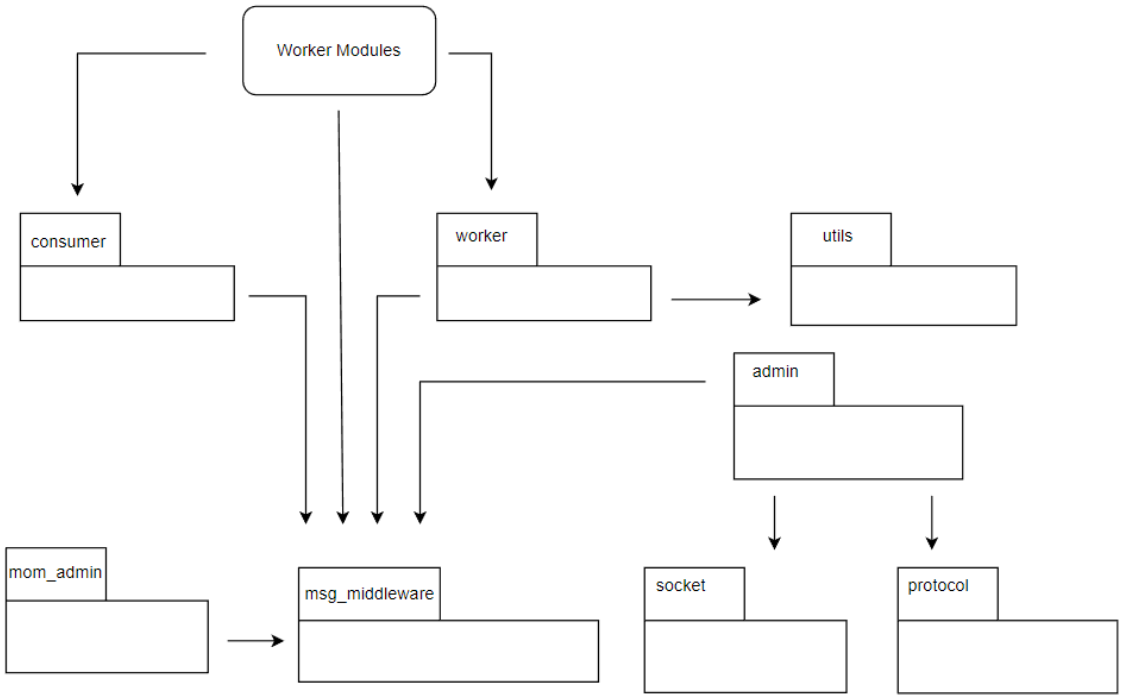
En el siguiente diagrama se observa la misma tarea, pero desde un punto de vista secuencial de los datos.



Vista de Desarrollo

El siguiente diagrama de alto nivel muestra como se compone la arquitectura de la aplicación desde la vista de la interacción entre los paquetes creados.

Debido a que la lógica de negocio es muy sencilla, se observará mucha similitud con los diagramas de clase antes introducidos.

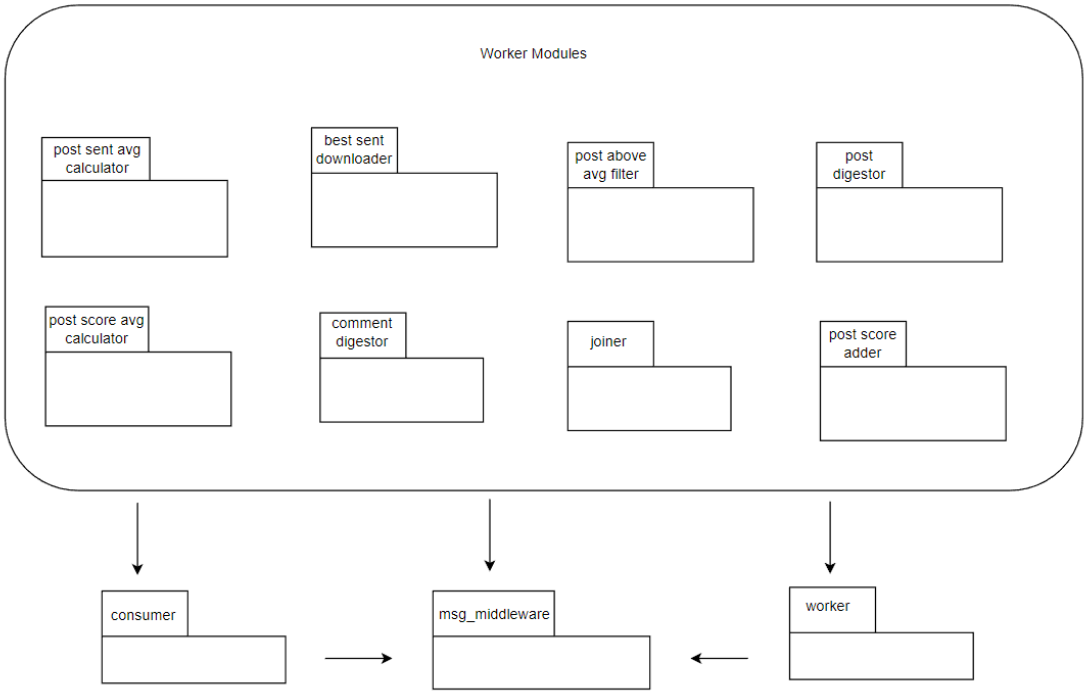


Como puede apreciarse, el paquete “msg_middleware” que contiene al middleware de mensajes implementado cumple un rol central ya que de el depende la mayor parte del código del servidor. En el se encuentra todo el código asociado a la lógica de cliente del middleware, el cual se comunicará con el servidor de RabbitMQ para propagar los mensajes necesarios.

Con respecto al mom_admin, este será un paquete que corresponde a un proceso que brindará coordinación al middleware implementado.

El rectángulo “worker modules” no es un paquete como tal del sistema, sino que engloba a cierto número de paquetes que corresponden a cada uno de los procesos que forman parte del pipeline de ejecución.

En el siguiente diagrama, se ve en más detalle cómo es la composición de dicha abstracción



Como ya se ha mencionado, existe una relación bastante directa entre el diagrama DAG, los diagramas de clases y este diagrama aquí mostrado.

Vista Física

En los siguientes diagramas de robustez observaremos una vista de la interacción entre los controladores, actores y entidades del sistema. En este caso, cada controlador fue implementado como un proceso corriendo en una máquina aislada, con conexión a una red común con los demás procesos del sistema.

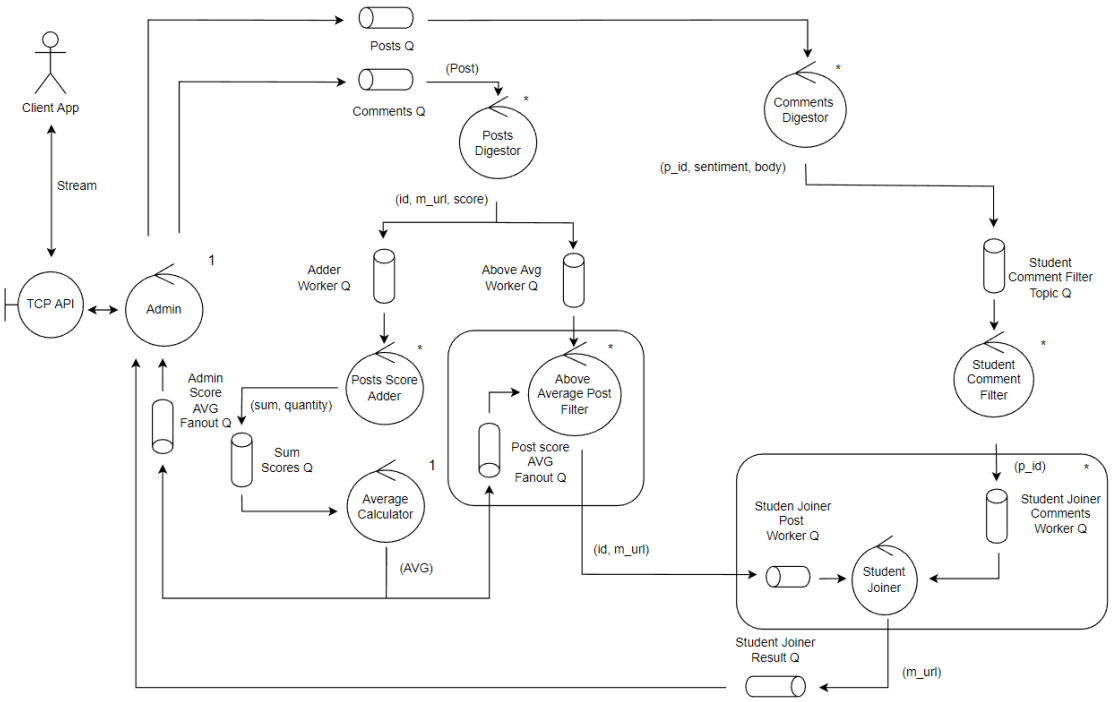
Las aristas representan con su sentido, cómo fluyen los datos entre los distintos controladores. Los datos en cuestión son representados, al igual que en el DAG, como tuplas.

Se añade una vista lógica de la conexión entre los distintos controladores, a partir de las colas de mensajes ya introducidas:

Debido a la complejidad del diagrama, se decidió fraccionarlo en dos partes, cada una de las cuales hace foco en los procesos involucrados en las operaciones que se pretenden detallar.

Adicionalmente, se omitió al proceso administrador de mensajes “middleware_admin”, como así también a los procesos que forman parte del servidor de RabbitMQ debido a que no aportaban a la comprensión de los diagramas.

El siguiente diagrama muestra a los procesos involucrados en los cálculos de puntaje promedio de posts y posts con algún comentario relacionado al ámbito escolar, que superen al promedio en puntaje.



Es importante destacar que al haber omitido el trabajo que realizan los exchanges de RabbitMQ, puede dar la impresión que los procesos que escriben en más de una cola de salida lo hacen de forma autónoma (conocen a todas las colas de salida), sin embargo este no es el caso. Todo proceso escribe su resultado en una cola única, sobre la cuál luego otros procesos podrán unirse. Una forma más explicita de representarlo sería uniendo la salida de una cola con la entrada de otra, sin embargo esto se omitió ya que terminaba no solo ensuciando el diagrama sino que además se consideró que podría ser inadecuado conectar dos colas sin un controlador intermedio.

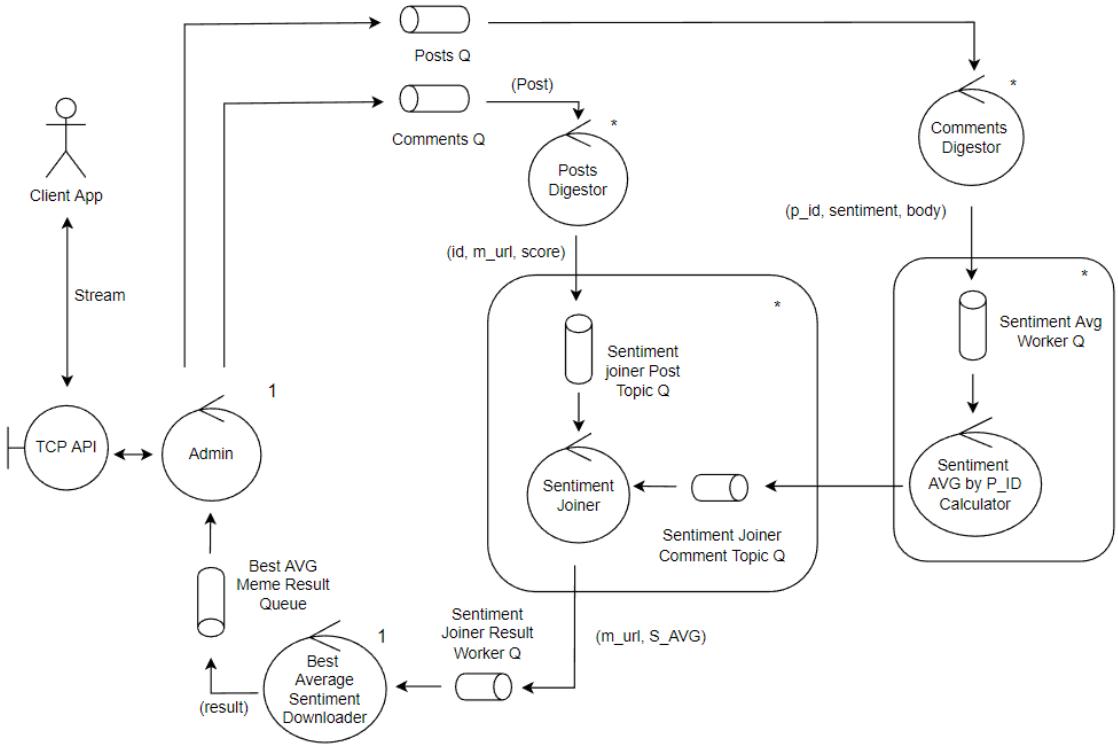
A continuación se brindará una explicación de los procesos que forman parte del diagrama.

Cuando se haga mención a “load balance” este hará referencia a la técnica de encolar en una cola de salida por tópico utilizando como tópico el resultado de aplicar una función de hash sobre el post id de una tupla, teniendo en cuenta el número de procesos del espacio de salida.

- Post Digestor: Recibe los posts a través de una cola de entrada de tipo worker, sobre la cuál se repartirán los datos entre los procesos integrantes de este grupo. Luego escribirá el resultado en una cola por tópico haciendo load balance teniendo en cuenta el número de procesos “Sentiment Joiner”.
- Post Score Adder: Recibe la entrada de una cola tipo worker sobre la cuál se repartirán los datos entre los distintos procesos del grupo. Dicha cola será vinculada a la salida del post digestor con un tópico que matchea con cualquier salida. Finalmente escribirá sus resultados en una cola tipo worker que comparten todos los procesos del grupo.

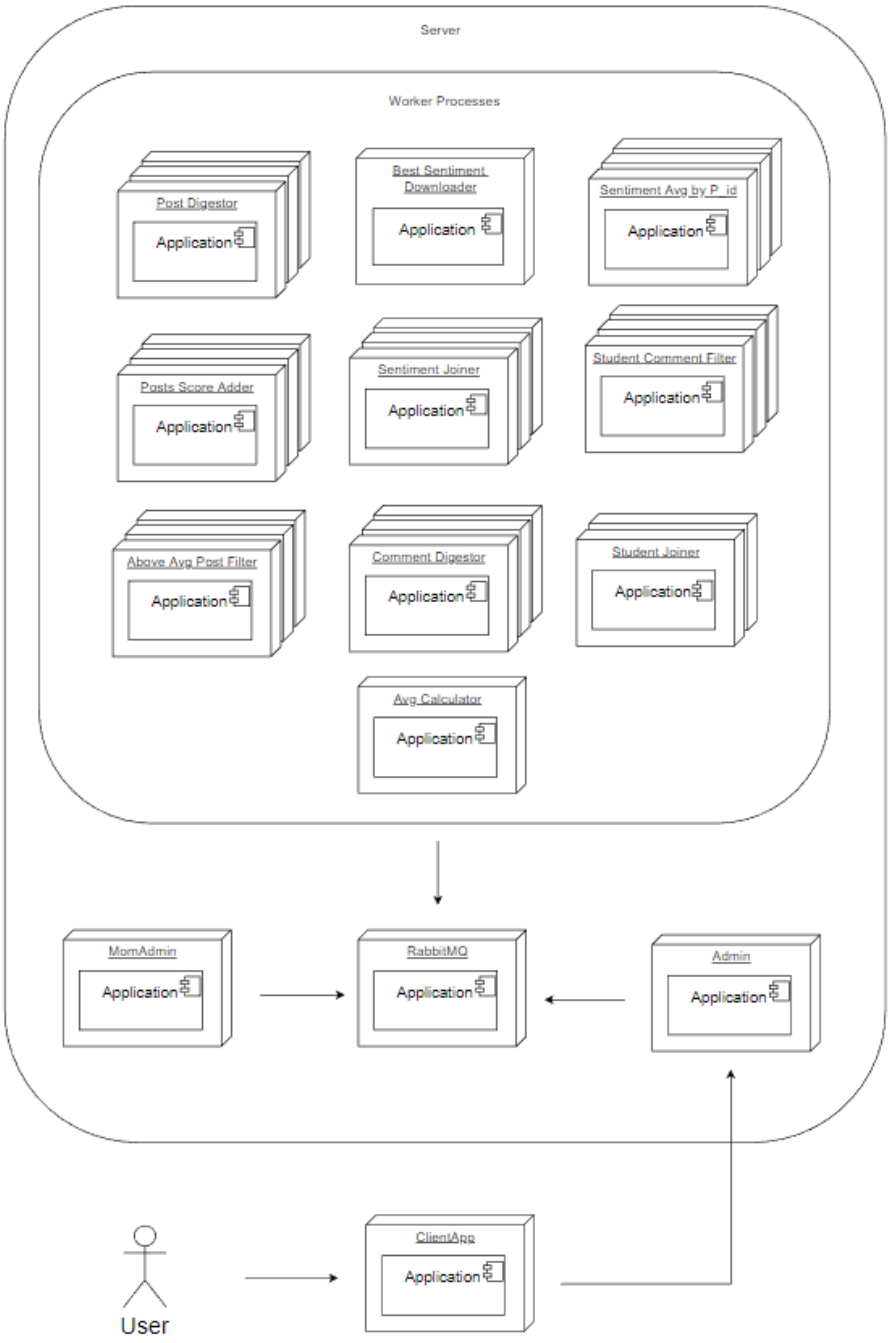
- Average Calculator: Sólo existirá un único proceso de este tipo ejecutándose. El mismo lee de la cola tipo worker resultado de la etapa anterior. Luego de realizar el cálculo pertinente escribirá su resultado en una cola tipo fanout.
- Above Avg Post Filter: Leerá por una cola tipo worker compartida entre todos los procesos del grupo, la cuál se vinculará a todos los topicos de la cola de salida del Post Digestor. Además recibirá el valor del promedio de score de posts por una cola tipo fanout exclusiva de cada proceso del grupo.
- CommentsDigestor: Recibe los comments a través de una cola de entrada de tipo worker, sobre la cuál se repartirán los datos entre los procesos integrantes de este grupo. Luego escribirá el resultado en una cola por tópico haciendo load balance teniendo en cuenta el número de procesos “Sentiment Avg Calculator”.
- StudentCommentFilter: Recibe los comentarios de una cola tipo worker compartida entre todos los procesos que integran el grupo. Escribirá el resultado en una cola por tópico haciendo load balance teniendo en cuenta el número de procesos “Student Joiner”.
- Student Joiner: Cada proceso del grupo tendrá dos colas exclusivas a él unidas por tópico a la salida de “Above Avg Post Filter” y “Student Comment Filter”. El tópico utilizado estará compuesto por el id del proceso en el grupo al que pertenece. Finalmente escribirá el resultado en una cola tipo worker compartida por todos los procesos del grupo.
- Admin: Escribirá en dos colas tipo worker los comentarios y posts que reciba del cliente. Leerá el resultado del cómputo por tres colas, obteniendo promedio por una cola tipo fanout exclusiva a él, el mejor meme por sentiment promedio por una cola tipo worker, y los mejores memes escolares por una cola tipo worker.

El siguiente diagrama muestra a los procesos involucrados en la obtención del meme cuyo post tuvo el mejor sentimiento promedio entre sus comentarios.



- Sentiment Avg Calculator: Cada proceso del grupo tendrá una cola exclusiva a él unida por tópico a la salida de “Comment Digestor”. El tópico utilizado estará compuesto por el id del proceso dentro del grupo. Escribirá la salida en una cola por tópico realizando load balance teniendo en cuenta el número de procesos “sentiment joiner”
- Sentiment Joiner: Cada proceso del grupo tendrá dos colas exclusivas a él unidas por tópico a la salida de “Post Digestor” y “Sentimen Avg Calculator”. El tópico utilizado estará compuesto por el id del proceso en el grupo al que pertenece. Finalmente escribirá el resultado en una cola tipo worker compartida por todos los procesos del grupo.
- Best Avg Sentiment Downloader: Existirá un único proceso de este tipo. Leerá su entrada de una cola tipo worker sobre la cúal escribirá la etapa anterior. Escribirá su salida en una cola tipo worker.

El siguiente diagrama muestra cómo sería el despliegue del sistema en un medio físico y su interacción con el cliente



Como puede observarse, el mismo fue diseñado para poder ser desplegado en un entorno multicomputing, en el cuál las únicas conexiones conocidas entre los procesos son por parte del cliente al admin (a través de su dirección ip y puerto) y de cada proceso del servidor al servidor de RabbitMQ.

Se decidió utilizar un doble cuadro que envuelve a varios procesos por dos razones. El primer cuadro permite delimitar al servidor del cliente, mientras que el segundo cuadro permite

englobar a los procesos denominados “Worker Process” en una categoría en la cuál todos tienen una conexión únicamente con el servidor de RabbitMQ.

Coordinación de Servicios y Middleware

La coordinación de servicios fue realizada a partir de un middleware de mensajería centralizado. Este middleware se encarga abstraer la utilización de las colas de mensaje, para que las mismas sean análogas a los canales nativos del lenguaje utilizado, Golang.

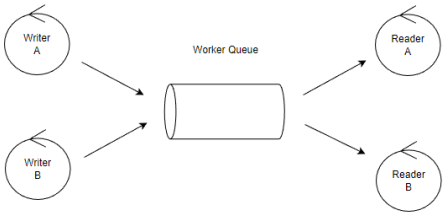
Para esto, el middleware basa su funcionamiento en un Message Oriented Middleware como es RabbitMQ.

Abstracción

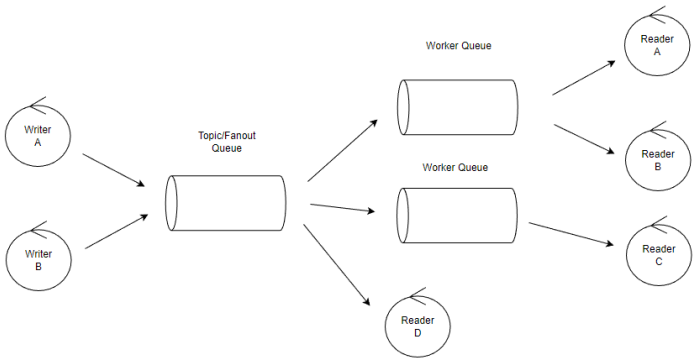
Previamente se introdujo el modelado de colas utilizado por los procesos del sistema, el cual fue basado en la siguiente abstracción:

Worker Queue

Se trata de una cola de mensajes productor consumidor habitual, la cual permite múltiples consumidores y productores, como puede observarse en el siguiente diagrama.



Sin embargo, este tipo de cola se extendió para permitir la posibilidad de defiir un “source” de los datos a leer, de forma tal de poder crear composiciones como la siguiente:



Como se puede observar, un grupo de procesos lectores podrán realizar trabajo repartiendo la carga creando una cola productor consumidor, que tenga como productor a otra cola de tipo “topic” o “fanout”, teniendo la posibilidad de indicar un tópico en el primer caso. En el diagrama, los lectores A y B crearon este tipo de patrón.

Por otro lado, como muestra el diagrama, esto no impide que otros procesos se suscriban a la cola source de la forma que deseen, ya sea a través de una cola worker intermedia o directamente. Es importante destacar que en la abstracción realizada, si se desea que más de un proceso comparta una misma cola (multiple consumer), dicha cola deberá tener un nombre asignado y deberá ser de tipo worker.

Topic Queue

Las colas de tipo tópico de escritura permiten escribir mensajes con un tópico particular, los cuales puedan luego ser filtrados por procesos lectores que se unan a la misma.

Cuando un proceso crea una cola de lectura tipo topic, deberá proporcionar el tópico al cuál desea suscribirse. La cola resultante será creada de forma exclusiva a él, por lo que cualquier nombre provisto será ignorado y, como consecuencia, no podrá ser compartida a otros procesos.

Fanout Queue

Las colas de tipo fanout tienen una funcionalidad similar a las de tipo topic, con la salvedad de que todos los mensajes serán siempre emitidos a todos los procesos lectores, sin discriminar por tópico.

Composición

La composición del middleware puede dividirse en dos partes, la biblioteca que provee una iterfaz al usuario y el administrador que se ejecuta en un proceso distinto.

Interfaz de Cliente del Middleware

La interfaz del cliente, correspondiente a la primera parte de la división, es la que tiene la mayor lógica. La interfaz que provee al usuario es la siguiente:

- Crear nueva cola de tipo lectura, en base a configuración.
- Crear nueva cola de tipo escritura, en base a configuración.
- Crear nueva cola, tipo a determinar en base a configuración.

La configuración mencionada para cada cola es la siguiente:

- Name: Indica el nombre de la cola que será creada, tomando vacío como una cola anónima.
- Class: Indica el tipo de cola a crear, entre Worker, Fanout y Topic
- Topic: Indica el topico al cuál la cola deberá suscribirse.
- Source: Indica a qué cola deberá suscribirse la cola actual a crear, para tomar información.
- Direction: Indica la dirección de la cola a crear, entre Read y Write.

Con esta interfaz es muy sencillo crear colas a partir de un archivo de configuraciones. De esta forma, el proceso que las utilice puede quedar no solo abstraído de su funcionamiento, sino que también de su creación.

Internamente, la interfaz de cliente utiliza la interfaz de RabbitMQ para crear las abstracciones antes mencionadas, declarando colas y exchanges y bindeandolos, valiéndose de la propiedad de idempotencia de declaración que provee el MOM de Rabbit.

Por cada cola creada, el middleware creará un worker en una gorutina encargado de hacer la traducción entre la información provista por rabbit al usuario y viceversa. Cuando la cola es de lectura, el worker traducirá los mensajes provenientes de rabbit en mensajes que respeten la interfaz provista por el mom implementado. Cuando la cola sea de escritura, hará el camino contrario, tomará los mensajes que respetan la interfaz y los enviará por rabbit en formato string.

Para mejorar la performance del sistema, se implementó un envío en batch dentro del propio mom. Este consiste en la acumulación de mensajes recibidos por parte de los procesos escritores y su correspondiente envío a la red cuando alguna condición se cumpla:

- Timeout de envío: Tiempo máximo transcurrido entre el envío de dos batches.
- Tamaño mínimo del mensaje: Se ha superado el tamaño mínimo para el envío de un mensaje

Cuando esto suceda, se realizará el envío de un único mensaje que contendrá un vector de todos los mensajes emitidos hasta el momento (mensaje batch).

Dado que RabbitMQ permite el envío de mensajes en formato binario, se implementó un protocolo que es comprendido por los procesos lectores y escritores, que realiza la codificación de estos mensajes en batch a una tira de bytes. En la siguiente sección se profundizará sobre este protocolo.

Problemas en la implementación

El problema de la implementación surge cuando se desea notificar el cierre de un canal de comunicación, por ejemplo, la finalización del stream de posts.

Sería ideal que el proceso escritor pudiese cerrar el canal de golang y que esa acción fuese transmitida de alguna forma al proceso lector, que verá su canal (también de golang) cerrado. Esta idea, aunque muy sencilla, llevó varios dolores de cabeza que se detallarán en una sección siguiente.

Para implementar este comportamiento, se realizó lo siguiente:

- Un proceso escritor cierra el canal de golang
- Eventualmente, el worker traductor del mom recibirá esta señal, por lo que enviará un mensaje “finish” por la cola de rabbit.
- El worker lector del otro extremo, al recibir el paquete “finish” cerrará el canal de golang
- Eventualmente, el proceso recibirá la señal de cierre del canal.

A simple vista esto resulta muy sencillo, sin embargo, tiene dos grandes problemáticas. La primera es que no se puede simplemente cerrar un canal cuando hay múltiples escritores, porque, aunque uno haya finalizado, los demás pueden seguir escribiendo. La segunda es que si se tiene lectores compitiendo por los mensajes en una cola (una cola worker), solo uno de ellos recibirá el mensaje de finalización, por lo que el otro se quedará en deadlock.

Para la solución de ambos problemas se decidió crear un administrador.

Administrador del Middleware

El administrador del middleware es un proceso que corre de forma independiente y cumple una simple pero crucial tarea. Deberá controlar el envío de la señal finalización de cada cola. El mismo tiene comunicación con los procesos cliente de la mom a través de una cola de control tipo worker.

Como se detalló anteriormente, cuando un usuario crea una cola de escritura, se creará un worker traductor a nivel del mom. Al mismo tiempo que este worker es creado, el cliente del mom emitirá un mensaje “new,<queue_config>” al administrador.

Cuando otro proceso crea una cola de tipo worker y la misma es unida a través del campo “source” a una cola de escritura previamente creada, entonces el cliente del mom emitirá un mensaje “new_reader, <queue_name>” al administrador.

El administrador mantendrá una estructura interna con todas las colas de escritura creadas, el número de escritores que cada una de ellas tiene y el número de lectores (en cola de tipo worker) que posee.

Cuando el administrador recibe el mensaje que le informa la aparicion de una nueva cola, este creará una nueva entrada en dicha estructura o aumentará en uno el contador de escritores.

Cuando el administrador recibe el mensaje que le informa la aparición de un nuevo lector, aumentará en uno el contador de lectores.

Cuando un writer del cliente recibe la señal de finalización, enviará un mensaje “finish,<queue_name>” al administrador.

El administrador, al recibir este mensaje reducirá el contador de escritores de dicha entrada. Si el contador llega a cero, emitirá tantos mensajes como número de lectores haya registrados, por dicha cola (que recordemos tiene acceso porque se le ha enviado la configuración de inicio), con el mensaje “finish”.

Protocolo Implementado

Para la comunicación entre el cliente y el servidor se utilizó un protocolo binario similar al utilizado en el trabajo práctico 1.

Se detallarán brevemente las características:

- Cada mensaje enviado tiene un header de 4 bytes que indican el largo, en bytes, del cuerpo del mensaje.
- El cuerpo del mensaje tiene como primer byte un opcode que permite determinar el tipo de mensaje, de entre los cuales se encuentran:
    - Post
    - Comment
    - PostFinished
    - CommentFinished
    - Response
    - Error
- Cada mensaje tiene su codificacion particular, dependiendo de su composición. En forma general, cada mensaje se codifica como la composición de la codificación de sus tipos, los cuales son:
    - El OPCode antes mencionador, que se codifica como un byte sin necesidad de conversiones.
    - Un número se codifica simplemente como su representación bianria en BigEndian
    - Una string se codifica como su representación binaria en UTF8, anteponiendo un entero de 4 bytes que indica el largo del string.

Para la comunicación entre los procesos del sistema a través del middleware se utilizaron mensajes en formato texto, los cuales son codificados en UTF8 a binario antes de ser enviados al cliente del mom, para su correspondiente envío por la red.

El cliente del mom provee una interfaz de mensaje a los procesos que tiene los siguientes campos:

- Topico del mensaje, en formato texto.
- Cuerpo del mensaje, en formato binario.

Como se comentó anteriormente, el mom implementa envío de mensajes en batch para mejorar la performance del servidor.

Para realizar este envío se utilizó un protocolo binario de codificación que permite traducir un vector de mensajes binarios a una tira de bytes. Entonces, cada mensaje enviado al servidor de Rabbit tendrá la siguiente codificación:

- Un header de 4 bytes en BigEndian que indica el tamaño del vector de mensajes
- Por cada mensaje a enviar:
    - Se indica el tamaño del mensaje en un entero de 4 bytes BigEndian
    - Se escribe el mensaje tal como fue recibido.

Correcciones

- Se reescribió el informe teniendo en cuenta el modelo 4+1 vistas.
- Se incluyeron más diagramas, entre ellos el diagrama de secuencia faltante en la primera entrega.
- Se corrigieron los diagramas de actividad, robustez y despliegue en base a errores cometidos.
- Se refactorizó el diagrama de robustez, para ser más fiel a la arquitectura.
- Se reformularon las explicaciones del middleware y protocolo en base a los cambios realizados.