Trabajo práctico N°2: Gestión de reputación en Twitter

Sistemas Distribuidos I (75.74)

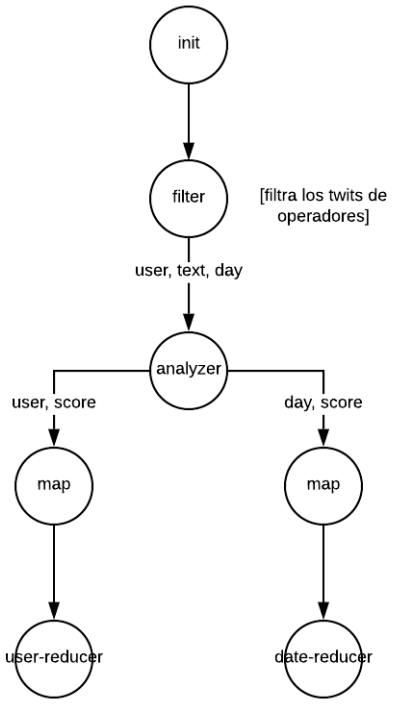
Ana Czarnitzki

7 de junio de 2019

Para este trabajo se pide la realización de un sistema que haga análisis de sentimientos sobre comentarios en *twitter* y genere reportes en base a los mismos, enumerando aquellos usuarios con más de tres comentarios negativos y el total de comentarios negativos y positivos por día.
Se pide además optimizar la eficiencia de todo este análisis, creando un *pipeline* para realizar todo el procesamiento necesario, dividiendo el trabajo en etapas lo más pequeñas posibles.

1. Vista lógica

Como ya se ha mencionado en la introducción, se modela al sistema como un *pipeline* donde se modifica la información obtenida en cada nodo hasta llegar a un resultado final en el último nodo. En particular, se piensa según el modelo *mapreduce* y el sistema tiene el siguiente *DAG* (*Direct Acyclic Graph*).

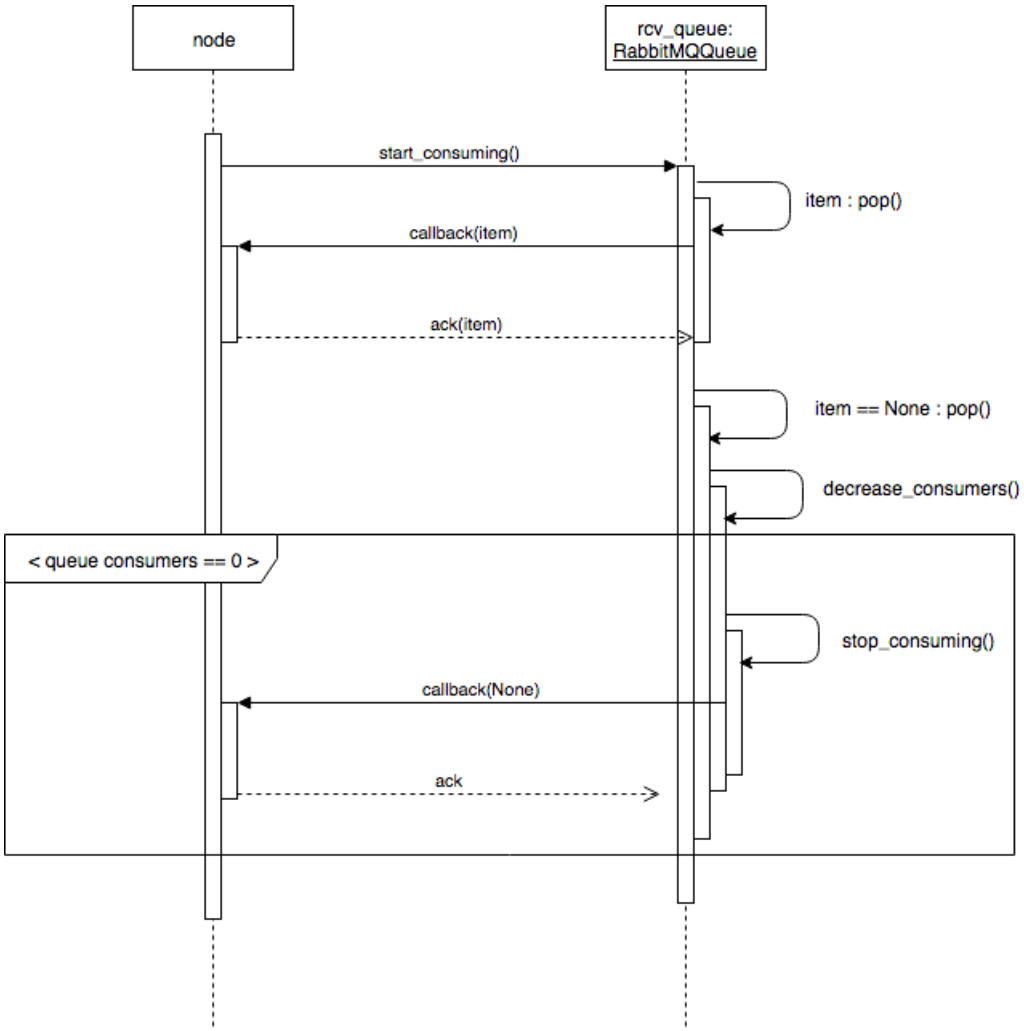


Los nodos de este *DAG* se detallan a continuación:

- *init*: es el encargado de iniciar todo el procesamiento y de *inyectar* al sistema de los *twits*.
- *filter*: filtra los *twits* de operadores, dejando pasar sólo los que sean de clientes, también filtra aquellos *twits* que no tengan el formato correcto (que les falte un campo, por ejemplo).
- *parser*: realiza un parseo inicial de los *twits*, eliminando los campos que no se utilizan y transformando los campos de tiempo en días.
- *analyzer*: realiza el análisis de sentimiento del texto de los *twits* y le otorga un puntaje a cada uno.
- *mappers*: realiza un mapeo de cada *twit* a su correspondiente *reducer*.

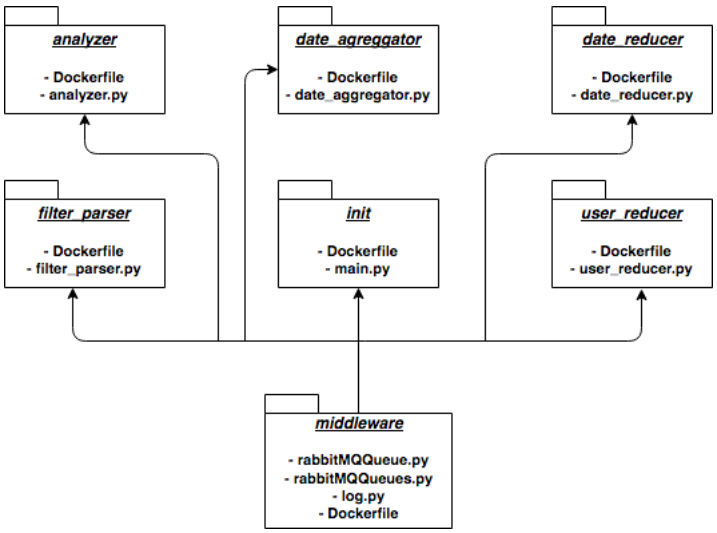
- *user-reducer*: realiza la *reducción* para los usuarios (es decir, cuenta la cantidad de comentarios negativos por usuario).
- *date-reducer*: realiza la *reducción* para los días (es decir, cuenta la cantidad de mensajes negativos y positivos para cada día).
- *agregator*: junta la información que generan los *date-reducers*.

El comportamiento de cada nodo puede verse en el siguiente diagrama de secuencia, dónde cada función de *callback* realiza la transformación de los datos recibidos.



2. Vista de desarrollo

El código de este trabajo tiene siete módulos, uno para cada uno de los nodos del *DAG* (ver agrupación en procesos en la siguiente sección) y uno de *middleware* (en este paquete se encuentran las abstracciones de comunicación que se utilizan, en particular la *Queue* de *RabbitMQ*).
Estos módulos se pueden ver a continuación:



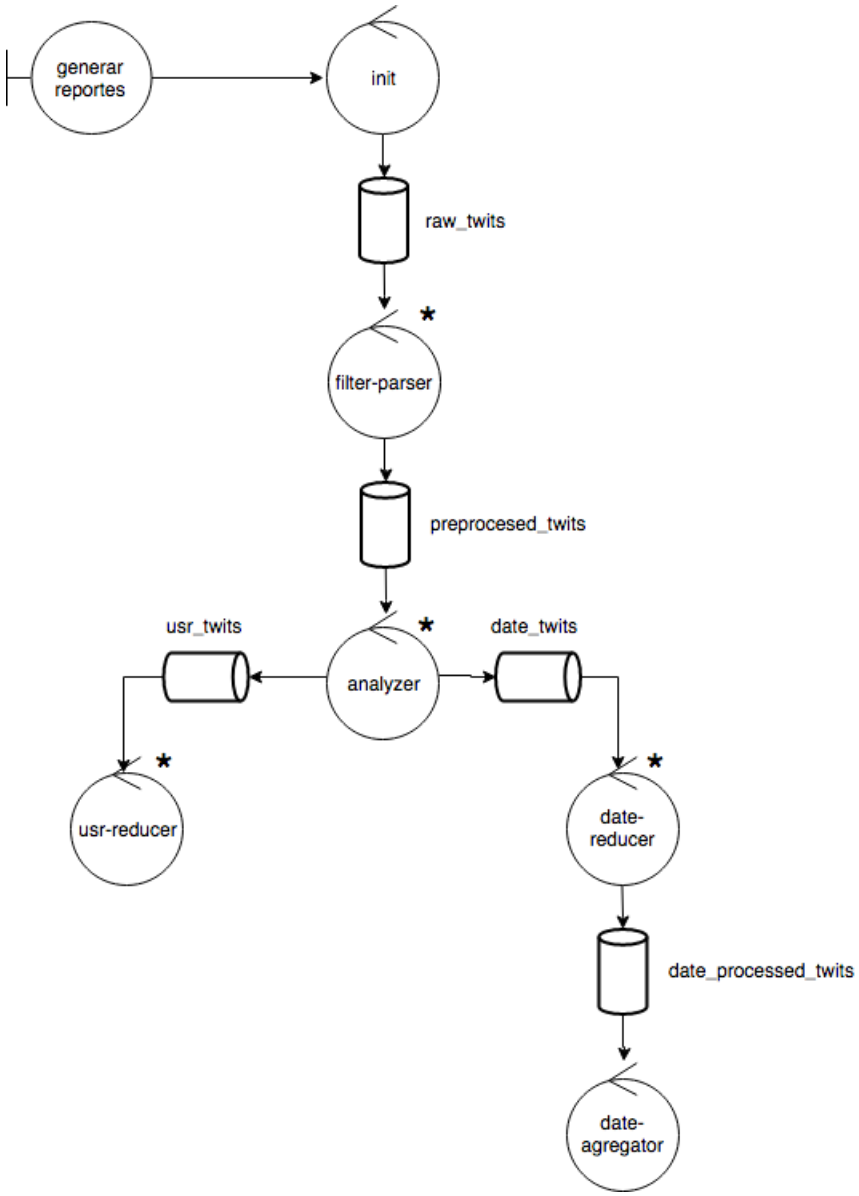
3. Vista de proceso

Para la implementación del gestor, se agrupan los distintos pasos del *DAG* en un menor número de procesos, en particular:

- *init*: Es el proceso que abre el archivo de *twits* y envia sus líneas al *filter-parser*.
- *filter-parser*: se encarga de filtrar los *twits* que son de operadores y parsear los de clientes, quedándose sólo con los campos de usuario, fecha (transformada a día) y texto.
- *analyzer*: analiza los textos de los *twits* y les asigna un puntaje, luego envia los mismos a los *reducers* tanto de usuarios como de fechas con los campos adecuados para cada uno.
- *user-reducer*: se encargan de contar para cada usuario cuantos comentarios negativos dejó y, en caso de llegar a la cantidad de alerta, escribir el nombre del mismo en un archivo (es decir, reportarlo).
- *date-reducer*: se encargan de contar para cada día cuantos comentarios positivos y negativos hubo en el mismo.

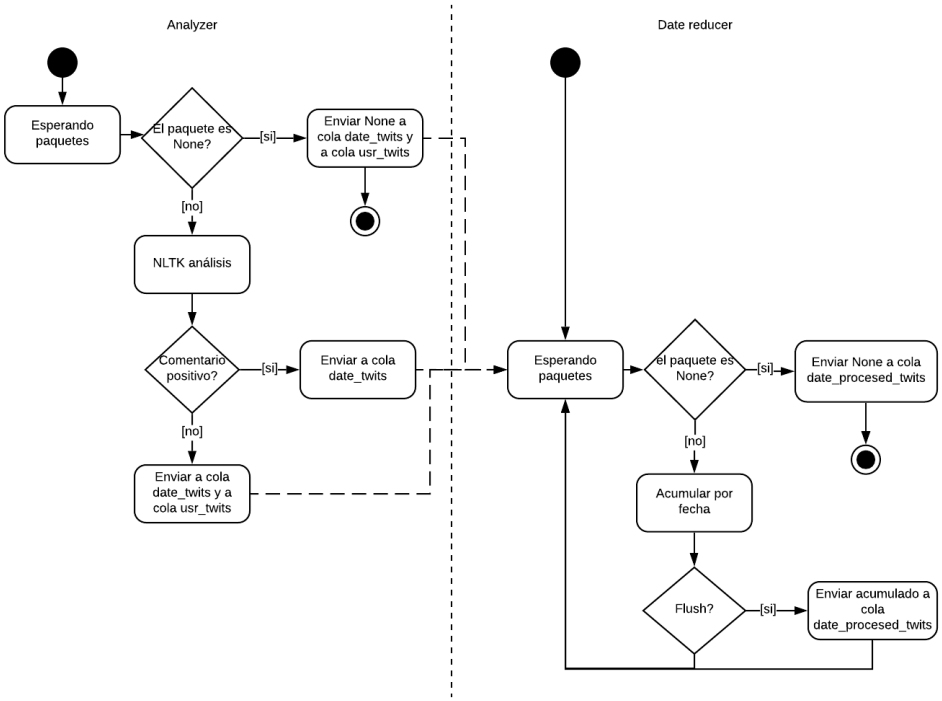
- *date-aggregator*: junta la información de todos los *date-reducers* y escribe la misma en un archivo (es decir, lo reporta).

Tanto el *init* como el *date-aggregator* son procesos de multiplicidad única, en el siguiente gráfico se puede ver esto junto con la comunicación entre todos los procesos:



(*) refiere a procesos escalables

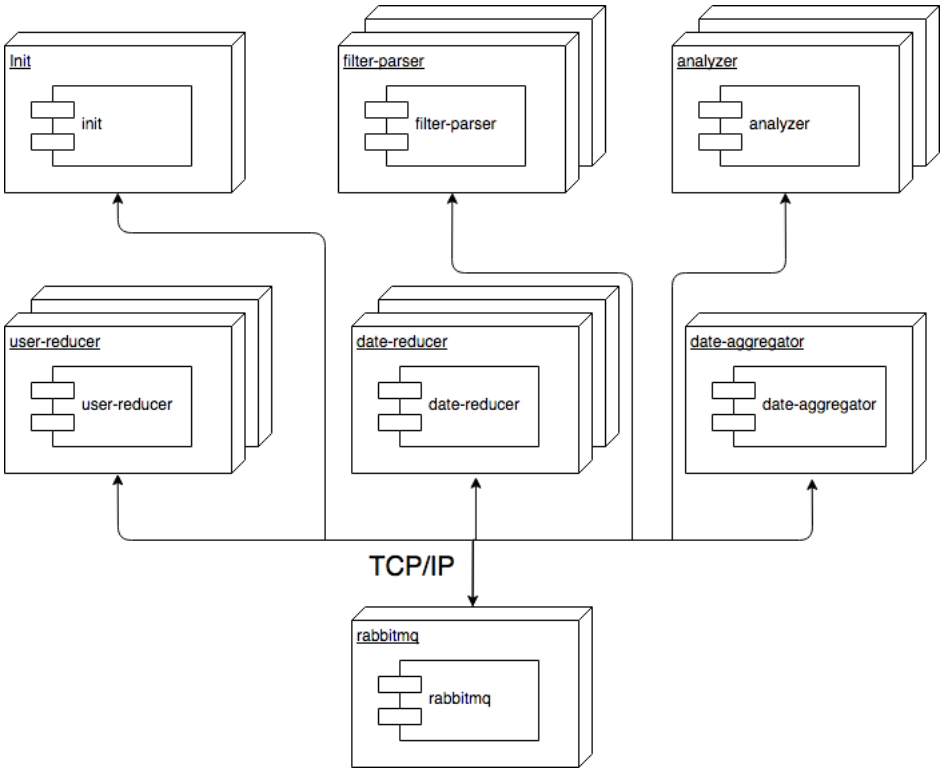
En el siguiente diagrama de actividades se puede ver la coordinación en más detalle entre el analyzer y el date reducer.



La cantidad de cada conjunto de procesos es configurable según un script que genera archivos de tipo *docker-compose* que se encuentra en el repositorio.

4. Vista física

Para la implementación de este trabajo, todos los procesos del pipeline se encuentran en su propio *container* mientras que el servidor de *RabbitMQ* se lanza en otro *container* separado.
Todos los componentes del primer container se comunican por colas de *RabbitMQ* como ya se marcó en los anteriores diagramas.



5. Escenarios

Este sistema modela dos escenarios separados, el primero es el de reportar todos los usuarios con más de tres comentarios negativos, el segundo el de reportar la cantidad total de comentarios negativos y positivos por día.
En ambos casos la forma de interacción con el usuario es la misma: se debe guardar el archivo que contiene los *twits* en la carpeta de reportes y luego levantar el sistema, al finalizar los reportes se encontrarán en esa misma carpeta.

6. Mejoras y trabajo futuro

A continuación se listan las posibles mejoras para este trabajo.

- Generar más abstracciones para los nodos del pipeline. En particular, se podrían crear clases que reciban información de una cola y escriban a otra (o a un conjunto de), esto serviría para modelar tanto el filter-parser como el date-reducer. Otra opción posible sería generar también clases de tipo *fuente* y de tipo *sumidero* para el date-agregator y el init, respectívamente.

- Mejorar la abstracción de *Queue* para *RabbitMQ* del *middleware* para que resulte más sencilla de usar. En particular, hoy la misma necesita saber cuántos consumidores y cuántos productores tiene para manejar los envios de finalización. Una opción posible sería que la misma se controle en memoria compartida y guarde aquellos procesos que se subscriben a ella. Se decidió mover esta lógica a los inicializadores de todos los procesos, de forma que las clases de los nodos no deban conocer esta información.