75.74 - Distribuidos I - TP2

Facultad de Ingeniería
Universidad de Buenos Aires

Mermet, Ignacio Javier
98153

Mayo 2022

Índice

1 Sobre la entrega

El código de la entrega se puede encontrar en GitHub.

2 Estructura del proyecto

El proyecto fue desarrollado en `python`[2] y empaquetado con `poetry`[1]. Tiene dos CLIs asociadas:

- `rma`:
- `rma_client`:

En la carpeta `docker` se encuentra disponibles el `Dockerfile` asociado al paquete.

3 Instalación y ejecución

Referirse al archivo `README.md` provisto en el repositorio para ver las instrucciones de instalación.

4 DAG

El diseño del DAG se hizo en base a la premisa de filtrar la mayor cantidad de datos lo antes posible. Del mismo, modo, evitar reprocesamiento que tenga la misma salida.

Los nodos se plantearon no en base a unidades de ejecución que reciben el mismo input, sino en unidades que conceptualmente hacen el mismo tipo de tareas. Por ejemplo, filtros o transformaciones.

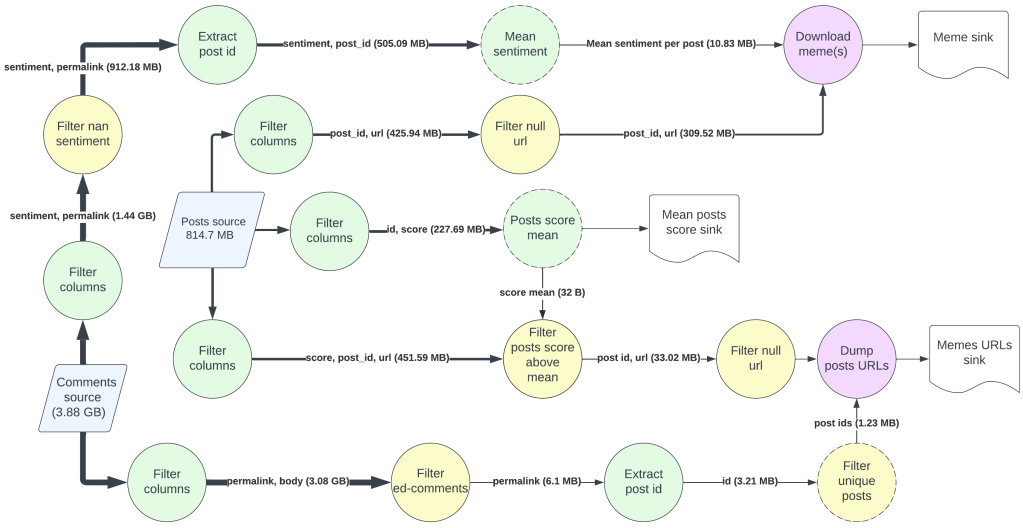


Figura 1: DAG diseñado

Nota sobre los colores.

- Rosa: `Join`
- Verde: `Transformation`
- Azul: `Source`
- Amarillo: `Filter`

Las líneas punteadas significan que los resultados de esos nodos no pueden ser obtenidos hasta haber procesado la totalidad de los datos entrantes. Los números entre paréntesis son valores aproximados del tamaño de la salida de cada nodo. No sería adecuado pensarlos en valores absolutos, sino más bien, son ilustrativos en valor relativo. Estos valores se obtuvieron con `pandas`, viendo el tamaño en memoria del dataframe resultante de la operación (con los parámetros `memory_usage="deep"` y `index=False`).

4.1 DAG Generado desde RMA

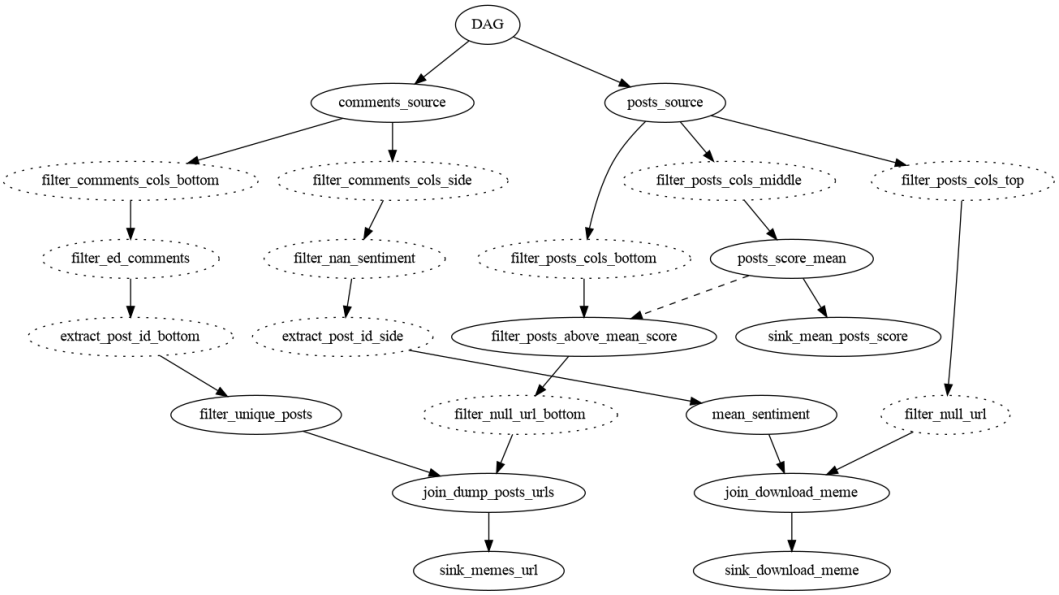


Figura 2: Render automático del DAG implementado.

Este grafo es generado con `graphviz` automáticamente al renderizar el DAG mediante `poetry run rma render-dag`. Los nodos con lineas punteadas indican que esa tarea está siendo paralelizada. Las aristas punteadas indican dependencias entre tareas.

Cada componente corre en su propio container de Docker. Necesita conocer las direcciones de las cuales sacar sus datos de entrada. Es por la complejidad asociada a generar este deploy que el presente TP incluye un módulo para definir DAGs y renderizar su `docker-compose.yaml` asociado.

Notar no todos los nodos que pueden ser paralelizados se implementaron de modo de hacerlo. Algunas simplificaciones con respecto al diseño se hicieron en base al esfuerzo requerido. Por ejemplo, los joins son paralelizables, pero recordemos que el DAG se diseño de modo tal que la cantidad de datos que les llega es la menor posible. O en el caso de `filter_posts_above_mean_score`, siendo el único nodo que recibe la salida de otro como dependencia, se evitó complicar la implementación paralelizándolo.

5 Arquitectura

5.1 Diagrama de Secuencia

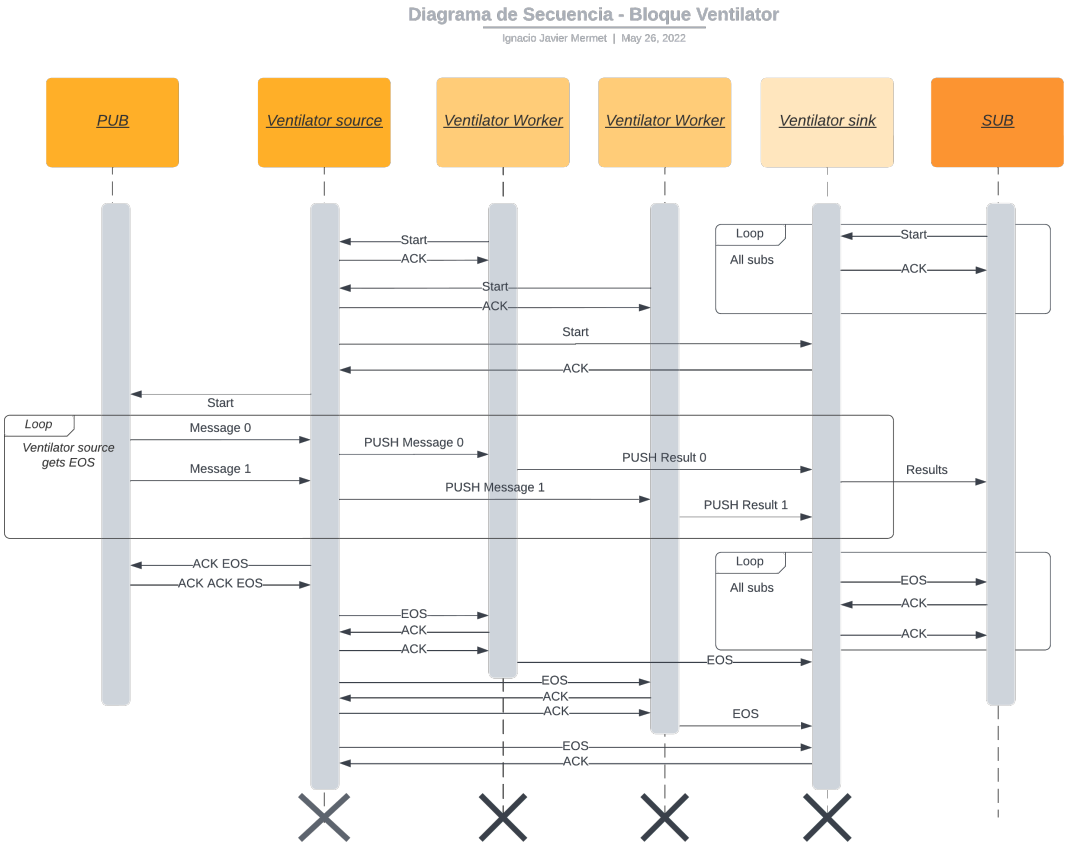


Figura 3: Diagrama de secuencia para bloques Ventilator.

5.2 Diagrama de Robustez

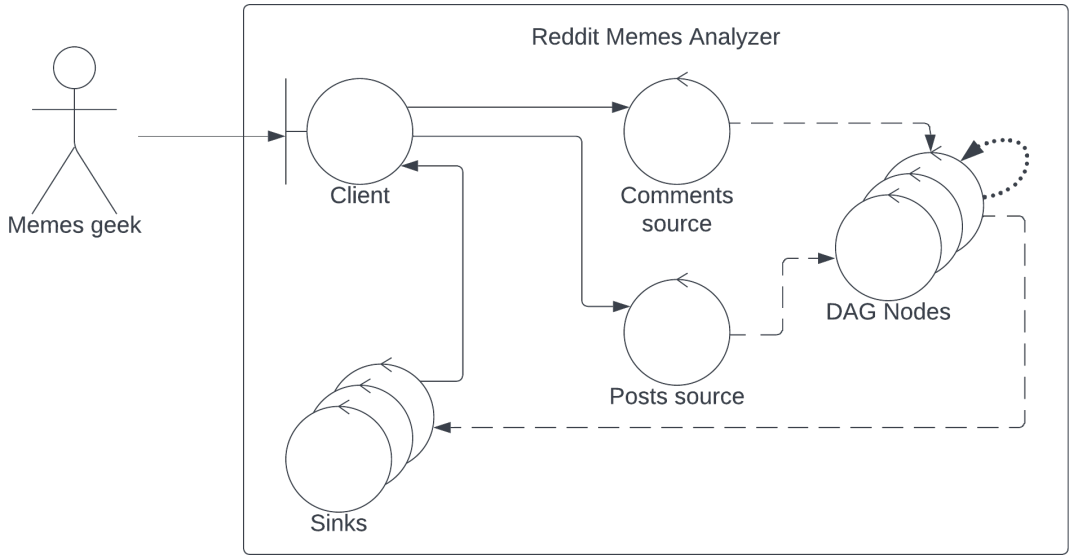


Figura 4: Diagrama de robustez.

5.3 Diagrama de Despliegue

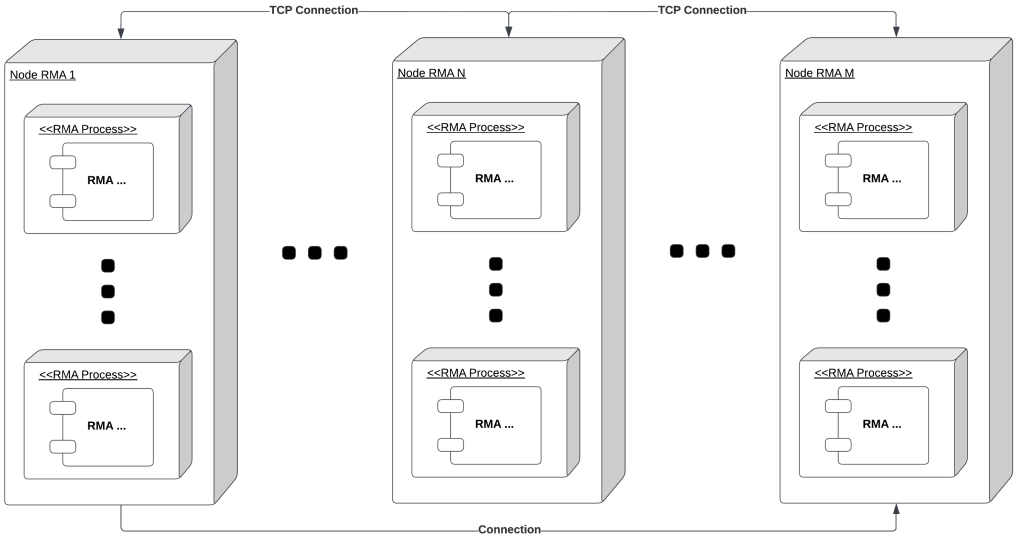


Figura 5: Diagrama de despliegue del sistema.

Si bien sencillo, en este diagrama vemos que podemos tener n nodos, cada nodo con C_i containers ejecutando un entrypoint de `rma`. Mientras sean accesibles por red entre sí, cualquier configuración de nodos y containers por nodo será válida.

5.4 Diagrama de Paquetes

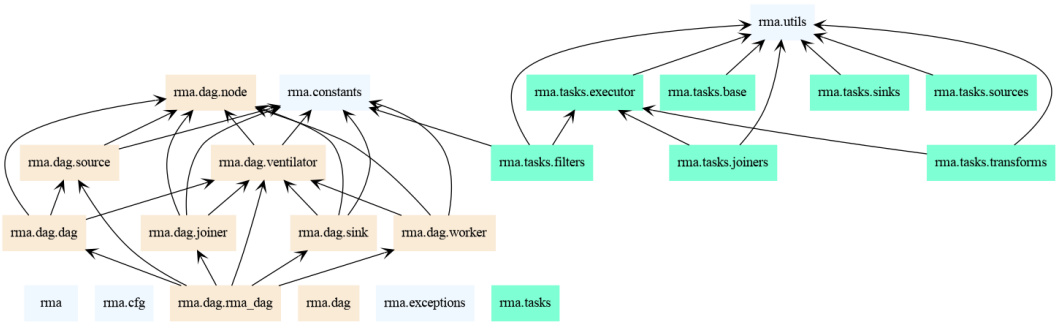


Figura 6: Diagrama de paquetes generado con `pyreverse`.

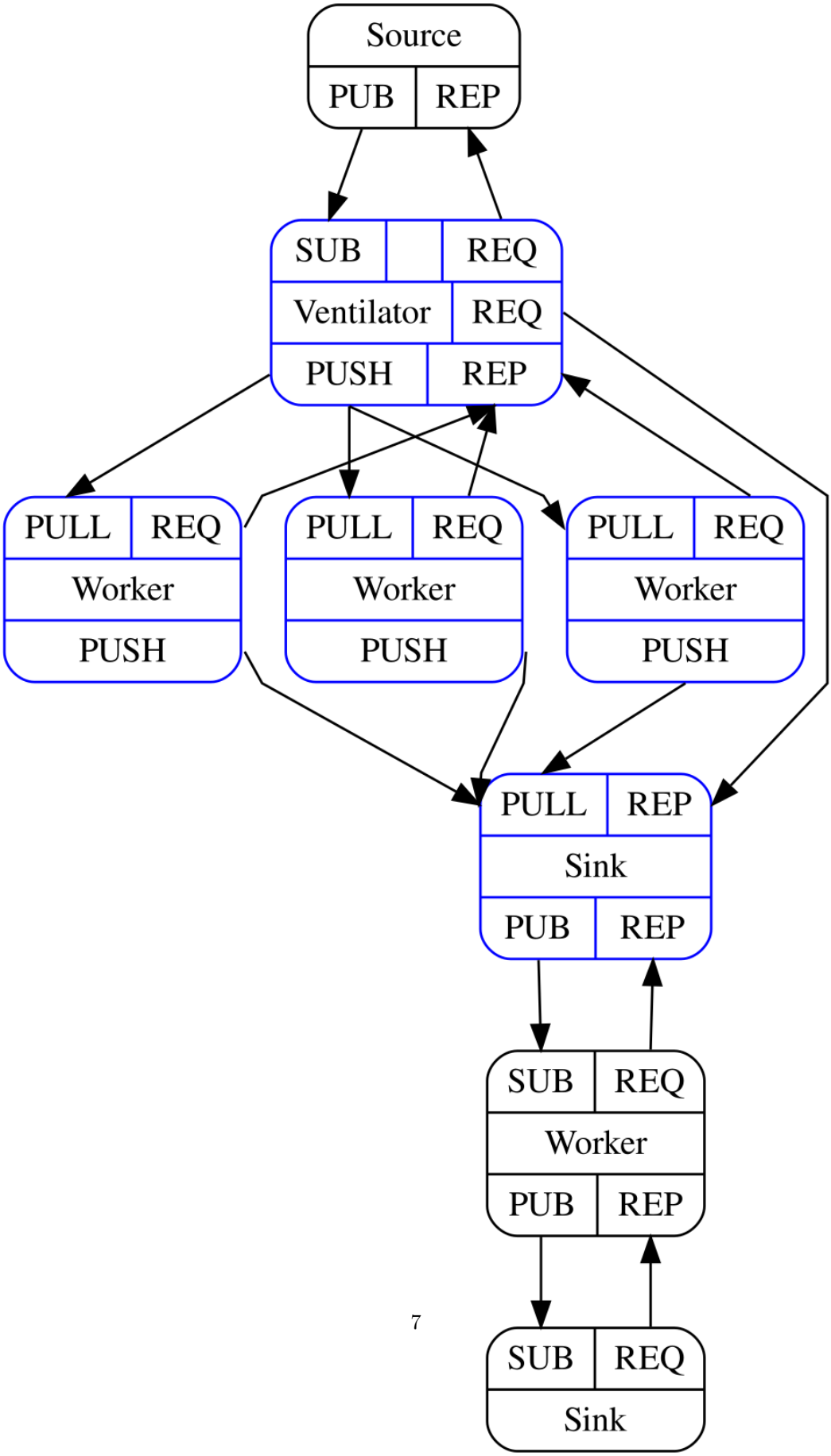
6 Interfaz de los bloques del DAG

De modo de poder interfeacear cómodamente los distintos nodos del DAG, se define una interfaz en común. Se usa `PUB/SUB` para entrada/salida de datos y `REQ/REP` para coordinar actividades.

Vemos en el siguiente diagrama los principales bloques del sistema:

- Sources
- Sinks
- Ventilator Blocks: Bloques que paralelizan transformaciones o filtros
- Workers: Ejecutan transformaciones o filtros sin paralelismo

Los `Joins` cumplen con la misma interfaz y son muy cercanos a los `Workers` en comportamiento, con la salvedad de que reciben dos inputs.


Source
PUB
REP
SUB
REQ
Ventilator
REQ
PUSH
REP
PULL
REQ
Worker
PUSH
PULL
REQ
Worker
PUSH
PULL
REQ
Worker
PUSH
PULL
REP
Sink
PUB
REP
SUB
REQ
Worker
PUB
REP
SUB
REQ
Sink

Referencias

[1] *poetry*. URL: https://python-poetry.org/.

[2] *python*. URL: https://www.python.org.