Programación de Objetos Distribuidos
Trabajo Práctico 2
Aeropuertos y Movimientos

Segundo Cuatrimestre 2019

Grupo 2

Integrantes

Della Sala, Rocío - 56507

Giorgi, María Florencia - 56239

Rodríguez, Ariel Andrés - 56030

Santoflaminio, Alejandro - 57042

Introducción

El objetivo del trabajo fue implementar una aplicación de consola que utilice el modelo de programación MapReduce junto con el framework HazelCast para el procesamiento de datos de vuelos. La idea principal fue la utilización implementaciones de colecciones distribuidas para cargar la data a clusters y hacer procesamiento distribuido.

Decisiones de diseño e implementación

Lo primero que se realizó fue la separación en 3 módulos; `client`, `api` y `server`.
En el primer módulo se encuentran las querys y parsers implementados, además del propio cliente. El cliente hace uso de dos parsers, `AirportParser` y `MovementParser`, para obtener los datos y validar parámetros. Los parsers solo guardan los campos que serán utilizados en las querys o aquellos mencionados en la consigna como campos relevantes. Por ejemplo, en el caso de los aeropuertos solo se guarda el código oaci, nombre y provincia.
El cliente hace llamados a las queries, dependiendo de cual fuera requerida, las cuales llaman a métodos del módulo `api`. En este módulo además de las queries, encontramos clases que permiten modelar las filas que serán volcadas a archivos. El vuelco a archivos es logrado mediante el *FileManager* que hace uso de un *BufferedWriter*.

En el segundo módulo se encuentra la implementación de las querys a través de MapReduce.
A continuación se explicará el funcionamiento de cada query, pero hay que destacar que todas tienen un comportamiento general: Para cada query se creó una clase específica para su mapper, que implementa la interfaz Mapper, lo mismo sucede para los reducers, que implementan la interfaz *ReducerFactory*. Se implementó un combiner para las querys como etapa intermedia post-mapper para hacer más eficiente el envío de datos por la red, aunque en algunas otras no fue necesario utilizarlo. El combiner trabaja como el reducer, salvo que acumula hasta que se acaban y se emite el resultado parcial. Por último en todas las querys se implementó un collator para post-procesamiento.

Query 1

El mapper emite el código *oaci* del aeropuerto origen de los que estén despegando o aeropuerto destinos de los vuelos que estén aterrizando junto con un 1, representando la cantidad. Utilizamos combiner debido a que por cada movimiento se emite un par clave-valor. El reducer genera el par final clave-valor donde la clave será el código del aeropuerto y el valor la cantidad de movimientos del aeropuerto. El collator ordena descendente por movimiento y luego alfabéticamente por código *oaci.*

Query 2

El mapper emite el nombre de la aerolínea junto con un 1, representando la cantidad, solo si el vuelo es de tipo cabotaje. Si la aerolínea es N/A se emite en vez del nombre de la aerolínea, el string “Otros”. Utilizamos combiner debido a que hay 3 tipos de vuelos y en su mayoría son de cabotaje o internacionales por lo cual tendremos muchas emisiones de pares clave-valor. El reducer genera el par final clave-valor donde la clave es el nombre de la aerolínea y el valor será el número de vuelos de cabotaje de dicha aerolínea. El collator calcula los porcentajes respecto al total de movimientos de cabotaje y se queda con los n pares de mayor porcentaje ordenándolos descendentemente y luego alfabéticamente por nombre de aerolínea. Al final se agrega el par cuya clave es “Otros”.

Query 3

En esta query se hace uso del mapper de la query 1, y luego se implementa otro mapper. En este segundo mapper, se emite la cantidad de movimiento en miles junto con el código *oaci*. En este caso no se hace uso de un combiner, ya que al haber obtenido los datos procesados por el mapReduce de

la Query 1, la cantidad de emisiones va a ser baja. El reducer genera finalmente un par clave-valor, donde la clave es una cantidad de movimientos en miles y el valor el par de aeropuertos (en forma de lista de strings). Finalmente el collator devuelve los datos ordenados de forma descendente por grupo (de miles de movimientos) y los pares de forma alfabética.

Query 4

El mapper recibe como parámetro el string del aeropuerto origen sobre el cual se quiere realizar la query. Emite código *oaci* junto con un 1, si el movimiento evaluado tiene como origen al aeropuerto pasado por parámetro. Se hace uso de un combiner porque se consideró que se trataba una query con gran cantidad de emisiones. El reducer genera los pares clave-valor, siendo la clave los códigos de los aeropuertos destino y el valor, la cantidad de movimientos. Finalmente el collator, devuelve únicamente los N aeropuertos con mayor cantidad de movimientos, en forma ordenada (descendente).

Query 5

El mapper emite el código *oaci* del aeropuerto origen o destino junto con un 1, representando la cantidad, solo si el aeropuerto pertenece al dataset de aeropuertos y el vuelo es privado tanto con matrícula nacional como extranjera. Si el vuelo no es privado, entonces se emite el código *oaci* del aeropuerto junto con un 0. Utilizamos combiner debido a que se realizan dos emisiones para todos los movimientos, salvo aquellos que contengan un aeropuerto de origen como de llegada que no pertenezca al dataset. Emite pares clave-valor donde la clave será el código oaci y el valor será un par de dos longs, el primero representando los valores acumulados, es decir solo los movimientos privados, y el segundo representando el total de movimientos, sean privados o no. El reducer genera el par final clave-valor donde el valor pasará a ser el porcentaje. El collator se queda con los n pares de mayor porcentaje ordenándolos ascendentemente y luego alfabéticamente por código *oaci*

Query 6

El mapper recibe por parámetro un mapa con códigos *oaci* como clave y provincias como valor. Por cada movimiento se emite un par que contiene origen y destino junto con un 1. Nuevamente combiner debido a que se realizan dos emisiones para todos los movimientos, si es que ese vuelo está registrado en el dataset por su llegada y por su despegue. El reducer generará un par junto con su cantidad de movimientos en común. Finalmente el collator, devuelve los datos en orden descendente y con los pares ordenados de forma alfabética.

Resultados - Análisis de Tiempos

A continuación se presentan los tiempos totales de resolución para las diferentes consultas. Las consultas fueron ejecutadas, de la forma que se indicó en campus en el archivo “Comandos Salida”. Se tratan de ejecuciones en forma local en una sola computadora.

Consulta	Tiempo con 1 nodo	Tiempo con 2 nodos	Tiempo con 3 nodos	Tiempo con 4 nodos
Query 1	2.281s	2.276s	4.244s	3.966s
Query 2	2.148s	3.378s	3.391s	2.381s
Query 3	2.582s	2.611s	2.801s	4.013s
Query 4	2.356s	2.118s	3.265s	3.261s

Query 5	3.066s	3.04s	4.039s	3.522s
Query 6	3.412s	2.974s	4.506s	5.171s
Promedio	2.641s	2.733s	3.708s	3.719s

Conclusiones

Se observa en la tabla, que a mayor cantidad de nodos, mayor tiempo toma la resolución de las queries. Consideramos que teóricamente, los resultados deberían dar de forma inversa. Sin embargo atribuimos estos números, a que el set de datos de los archivos CSV utilizados es relativamente chico por lo tanto no reflejan tanto el comportamiento teórico. Es posible que con grandes archivos, a mayor cantidad de nodos, obtengamos menor tiempo de resolución.