Trabajo Práctico Especial 2

Programación de Objetos Distribuidos

Consulta de datos de vuelos

Grupo 3:
Esteban Kramer - 55200
Oliver Balfour - 55177
Martina Scomazzon - 55410
Bianca Ritorto - 57082

Objetivo

Diseñar e implementar una aplicación de consola que utilice el modelo de programación MapReduce junto con el framework HazelCast para el procesamiento de datos de vuelos desde y/o hacia aeropuertos nacionales, basado en datos reales elaborados por EANA (Empresa Argentina de Navegación Aérea) y la ANAC (Administración Nacional de Aviación Civil).

Diseño de la solución

Estructura

Decidimos dividir el sistema en tres módulos:

1. API: contiene las clases que son compartidas por el cliente y el servidor.
2. Client: obtiene los parámetros ingresados por el usuario para ejecutar las queries.
3. Server: crea el nodo que va a pertenecer al cluster.

Carga de datos

Los datos de los archivos movimientos.csv y aeropuertos.csv son cargados por MovementCsvParser y AirportCsvParser respectivamente. Ambas clases implementan la interfaz CsvParser y leen los archivos línea por línea para crear los modelos de Movement y Airport con la información extraída.

Queries

Todas las queries implementan la interfaz Query cuyo unico metodo es *run*. Cada query tiene su propia implementación de de un Mapper, Reducer y Combiner.

Query 1

En esta query se calculan la cantidad de movimientos por aeropuerto. Para lograr esto, mapeamos los movimientos por aeropuertos usando la implementación de Query1Mapper. En donde emitimos un uno con el origen o destino OACI dependiendo si el movimiento fue un despegue o aterrizaje.

Luego, en el Query1ReducerFactor, se retorna la suma de los valores acumulados, es decir, los movimientos por código OACI.

Para lograr mayor eficiencia y evitar enviar información extra por la red, implementamos el Query1Combiner.

Query 2

Esta query pide el top N aerolíneas según el porcentaje de movimientos de cabotaje, ordenado descendentemente por dicho porcentaje.

El Query2Mapper emite como salida uno si el tipo de vuelo fue local y cero en caso contrario. Luego, el Query2Reducer se encarga de contar solo los vuelos de cabotaje. Por último, el Query2Collator se encarga de calcular los porcentajes y ordenarlos descendentemente. En un primer instante se pensó calcular los porcentajes teniendo una variable global que se incrementa cuando se emiten los valores en el mapper pero se decidió abandonar está solución ya que podría tener errores cuando hay múltiples nodos. El Query2Combiner funciona de manera similar al query anterior.

Query 3

La query calcula los pares de aeropuertos que registran la misma cantidad de miles de movimientos.

Para obtener los movimientos por código OACI utilizamos el Query1Mapper. Luego, realizamos un segundo map reduce, con el Query3Mapper se mapea cada combinación [OACI y cantidad de movimientos] a [miles de movimientos y OACI], solo tomando en cuenta los valores mayores o iguales a 1000 movimientos. Luego, en el Query3ReducerFactory, se genera una lista en la que se agregan los códigos del mismo grupo.

Query 4

Esta query pide los n aeropuertos destino con mayor cantidad de movimientos de despegue que tienen como origen a un aeropuerto oaci.

En primer lugar, con el Query4Mapper por cada movimiento, emitimos un uno con el destino OACI si el OACI de despegue del movimiento coincide con el OACI pasado por parámetro. Luego, en el Query4CombinerFactory, sumamos los valores para un aeropuerto dado y en el Query4ReducerFactory sumamos todos los valores. Por ultimo, con el Query4Collator, ordenamos descendente por movimientos y luego alfabéticamente y limitamos la respuesta a n valores.

Query 5

La query imprime los n aeropuertos con menor porcentaje de vuelos privados. A diferencia de los casos anteriores necesitábamos obtener sólo aquellos aeropuertos que figuren en ambos archivos, aeropuertos.csv y movimientos.csv, ¿Cómo logramos esto? Le pasamos al Query5Mapper un argumento con todos aquellos aeropuertos que no poseen sus OACIS vacíos ya que esos, indudablemente no nos van a servir.

Nos trasladamos a la clase Query5Mapper por unos instantes, allí podemos ver como en el map buscamos un "join" entre el conjunto de aeropuertos y movimientos respectivamente. Si logramos encontrar un movimiento cuyo OACI ya sea de salida o destino coincida con el OACI de algún aeropuerto habremos conseguido el OACI deseado.

Dado que luego debíamos obtener un porcentaje y no tan solo una suma en todo momento procuramos no solo sumarle uno a cada coincidencia, sino que también a un total que se utiliza luego para dividir cada elemento por él y obtener así, un porcentaje.

Una forma menos eficiente, y a menos nos referimos con prácticamente inaplicable. Sin utilizar los beneficios de Hazelcast se podría haber intentado de realizar un join entre los dos files chequeando que el OACI del aeropuerto no esté vacío y coincida ya sea con él OACI destino o el OACI de salida. Y así obtener un conjunto de movimientos a quienes aplicarles el mapper, combiner, reducer y submit. De hecho, lo intentamos, como podrán suponer, tardaba más de 10 minutos en poder resolver la query. Automáticamente, descartamos la propuesta.

Query 6

La query calcula los pares de provincias que comparten al menos min movimientos. En este caso se evaluó la posibilidad de utilizar múltiples map reduces, pero luego de debatirlo, optamos por encontrar la forma de solucionarlo con un único map reduce, a continuación profundizaremos en lo implementado.

Al igual que en la Query 5, le pasaremos al, en este caso Query6Mapper, un argumento, un mapa previamente diseñado que tenga como Key: OACI del aeropuerto, y Value: la provincia a la que corresponde. En el Query6Mapper, obtenemos las dos provincias y emitiremos un uno, solo en aquellos casos en donde las dos provincias sea distintas. El combiner, junta los valores emitidos y luego; el reducer suma los valores. Para finalizar, el Query6Collator ordena el resultado obtenido.

Resultados de las ejecuciones

	Query 1 [sec]	Query 2 [sec]	Query 3 [sec]	Query 4 [sec]	Query 5 [sec]	Query 6 [sec]
1 nodo	0.944	0.734	0.751	0.636	0.697	0.677
2 nodos	2.101	1.842	2.268	1.712	2.742	2.329
3 nodos	2.162	1.876	3.188	2.693	2.761	2.998
4 nodos	2.927	3.002	4.186	2.724	3.660	3.018

Conclusiones

Se puede observar que a mayor cantidad de nodos, se aumenta el tiempo de ejecución de las queries. Aunque este observación va en contra de la lógica de que si se cuenta con más nodos, el procesamiento es más rápido, se puede explicar por qué esto sucede. Al ser los archivos .csv analizados relativamente pequeños, más nodos en el cluster no ayudan al procesamiento, sino que lo entorpecen. Si fueran considerablemente más grandes, pasado un límite de tamaño, los tiempos comenzarían a reducirse cuantos más nodos hubiera.