Informe de Trabajo Práctico

72.42 - Programación de Objetos Distribuidos

2° Cuatrimestre 2019

Lund, Marcos (57159)

Ferrer, Tomás (57207)

Katan, Jonathan (56653)

Atar, Marcos Ariel (57352)

Introducción

A modo de introducción se describe la estructura general del proyecto. El proyecto consiste en tres carpetas principales: ***api***, ***client*** y ***server***.

- La carpeta ***api*** contiene las clases e interfaces generales que se usan tanto en el cliente como en el servidor. Contiene los modelos Airport y Movement (que contienen los campos relevantes para este trabajo e implementan las interfaces DataSerializable y Comparable), algunas clases auxiliares y un paquete por cada consulta, conteniendo cada uno de éstos los componentes necesarios para la operación MapReduce a realizar.
- La carpeta ***client*** contiene la clase Client (donde se encuentra el main del cliente), dos parsers de archivos CSV, y un paquete conteniendo una clase por tipo de consulta (desde aquí se envían los datos requeridos para el MapReduce y se espera a que la operación termine, para luego escribir los resultados en un archivo CSV.
- La carpeta ***server*** contiene la clase Server, que levanta una instancia de HazelCast según la configuración especificada en hazelcast.xml, incluído como recurso del servidor.

Diseño e implementación

Respecto del cliente se planteó inicialmente tener una clase con el método main por cada consulta a realizar. Sin embargo, la preparación previa a la consulta y las acciones posteriores a la consulta resultan muy similares por lo que no se tomó este camino. Para correr las consultas se decidió entonces tener un script en bash por cada consulta (no tienen la extensión .sh para respetar el formato de ejecución expresado en la consigna).

Inicialmente se consideró que el cliente guardara en una lista los movimientos a medida que iban parseándose del CSV de movimientos. Luego, los movimientos serían enviados desde esta lista a Hazelcast. No obstante, se modificó este comportamiento para que los movimientos fueran enviados a medida que iban parseándose, sin estructura intermedia.

Cada consulta implementa un mapper distinto según las necesidades. Luego, el combiner toma los datos emitidos por el mapper y los procesa de tal modo que no se transmita por la red cada valor emitido por el mapper. Nótese que, a excepción de la Query3, todas las queries utilizan un combiner. En la Query3 se emiten designadores OACI (identificadores únicos), por lo que no hay forma de combinar los mismos ahorrando transmisiones de bytes por la red. Luego, el combiner transporta por la red los datos combinados y el reducer los recibe, operando según necesario para cada query. Finalmente, cada reducer pasa los resultados al collator, donde se filtra y ordena según los requisitos de cada consulta. De esta forma se logra que todos los pasos necesarios se realicen dentro de una misma operación MapReduce, sin necesidad de que el cliente deba preocuparse por reordenar los resultados fuera de la operación MapReduce, bastándole con únicamente iterar por los mismos. La única excepción es el redondeo de números Double

en la Query2 (el cliente realiza esto al momento de escribir los resultados en el archivo de salida).

En muchos casos, los combiners y reducers son reutilizados ya que tienen el mismo comportamiento. Para la generación de logs se utilizó slf4j y desde los scripts se especifica a qué archivo escribir los logs (además, siempre se loggea por STDOUT).

Todas las queries utilizaron un solo trabajo MapReduce, a excepción de la Query3 donde se contabilizó en primer lugar la cantidad de movimientos de cada aeropuerto, para luego realizar otro trabajo MapReduce que agrupe aeropuertos con la misma cantidad de miles de movimientos. Sin embargo, dicha tarea podría haberse implementado en el Collator del primer trabajo pero resultó más apropiado dividir tareas distintas.

Al momento de crear un mapa de movimientos de tipo IMap para la utilización por parte del cluster, se concatenó el instante actual al nombre del mismo para evitar conflictos que involucraran distintas corridas en simultáneo de una misma query. De no hacerlo, esto hubiera significado que los mapas tuvieran el mismo nombre, provocando que los clientes accedieran a un mismo mapa distribuido en vez de a cada uno su propio (lo cual sería deseado, más aún en caso de que los archivos de input fueran distintos).

Utilizamos un archivo XML para configurar cada nodo del cluster, en vez de configurarlo programáticamente desde Java. El discovery se realiza con el mecanismo de auto-descubrimiento multicast. Se consideró utilizar el modo TCP/IP pero multicast resultó más simple.

Detalle de cada trabajo MapReduce:

- Query1: para cada movimiento, se emite el designador OACI apropiado y un ‘1’, de manera que el reducer sume cada entrada que recibe, obteniendo finalmente la cantidad de movimientos totales por aeropuerto.
- Query2: para cada movimiento de cabotaje, se emite el nombre de aerolínea (u “Otros”) y un ‘1’, de manera que el reducer sume cada entrada que recibe, obteniendo la cantidad de movimientos totales por aerolínea. Luego es posible calcular un porcentaje en base al total de vuelos de cabotaje.
- Query3: se obtiene mediante la Query1 la cantidad de movimientos por aeropuerto; luego para cada aeropuerto se emite como llave la cantidad de miles de movimientos y como valor el designador OACI, para luego desde el reducer guardar en listas los aeropuertos con misma cantidad de miles de movimientos.
- Query4: para cada movimiento, se emiten los designadores OACI destino (si corresponde) y un ‘1’, de manera que el reducer contabilice, para cada aeropuerto, la cantidad de movimientos que tienen como origen el aeropuerto deseado.
- Query5: para cada movimiento, se emiten designadores OACI origen y destino junto con un boolean describiendo si se trata de un vuelo privado o no, de manera que el reducer contabilice, para cada aeropuerto, la cantidad total de movimientos y la cantidad de vuelos privados del mismo (aquellos valores que hayan llegado como ‘true’). De esta manera se calcula luego un porcentaje.
- Query6: para cada movimiento, se emite una estructura de tipo ProvinceContainer con provincias origen y destino y un ‘1’, de manera que el reducer contabilice la cantidad de movimientos totales entre cierto par de provincias.

Análisis de rendimiento

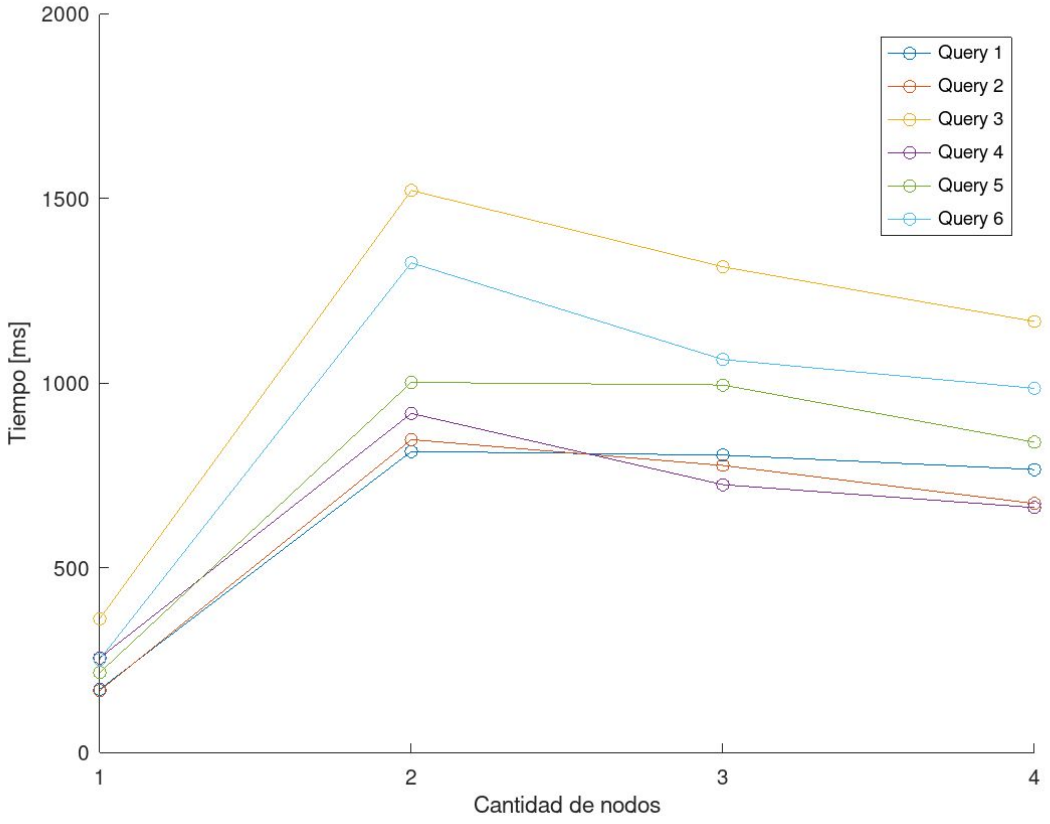

Figura 1: Análisis de tiempos de ejecución de MapReduce variando cantidad de nodos

Se corrieron pruebas para cada consulta realizada variando la cantidad de nodos en el cluster y se registró el tiempo de ejecución del trabajo MapReduce. Todos los nodos utilizados fueron computadoras del Laboratorio C de la sede Madero de la universidad, cada uno con una IP única. El cliente que realizó las consultas también utilizó una IP distinta.
Debido a que el conjunto de datos no tiene un volumen de datos significativo, se observó que con un único nodo las pruebas obtuvieron los menores tiempos de ejecución porque se evitó el overhead que involucra el envío de datos entre nodos por la red. Sin embargo, utilizando más de un nodo, se observó que a mayor cantidad de nodos se registraron menores tiempos en todos los casos, debido a la distribución de la carga entre mayor cantidad de nodos.
La consulta 3 obtuvo los mayores tiempos, seguramente debido al hecho de que ejecutó dos trabajos MapReduce, a diferencia del resto.