Trabajo Práctico Especial

Programación de Objetos Distribuidos
72.42
2017Q2

GRUPO 7

Benítez, Julián
Marcantonio, Nicolás
Raies, Tomás Agustín
Saqués, M. Alejo

Observaciones generales

Objetos involucrados

En el modelo de objetos, se posee una clase, `InhabitantRecord`, que sirve de contenedor para los datos de cada entrada del censo. Se ha buscado que todos aquellos campos cuyo dominio se conoce de antemano sean `enum`s: la condición de empleo, la provincia (y su correspondiente región).

Deserialización de los datos en CSV e inserción

Para la deserialización del contenido de los censos, se ha utilizado la librería de **Apache Commons**, en particular las clases `CSVFormat` y `CSVRecord`.

Los `InhabitantRecord`s generados son luego insertados en un `IMap` de Hazelcast. La inserción en dicho mapa distribuído se realiza de a *chunks* de un máximo de 5000 entradas, siendo dicha inserción realizada por un *pool* de threads. Todo el procesamiento de los `InhabitantRecord`s corre a cuentas del *thread* principal de la aplicación cliente.

Manejo de los datos en la red

Se ha procurado evitar enviar información innecesaria a través de la red, haciendo que `InhabitantRecord` implemente sus propios métodos de serialización y deserialización del objeto.

Por un lado, se ha procurado que los `enum`s no envíen su nombre completo a través de la red.

Por otro lado, dado que ninguna *query* precisa la totalidad de los datos de las entradas del censo, se explicitan cuáles son los campos que deben ser enviados a través de la red. De esta forma, se elimina el tráfico innecesario, como por ejemplo los nombres de los departamentos (cuyo tamaño es muy variable).

Descripción de las implementaciones

Nota: Para cada *query*, se exhibe un conjunto de resultados temporales para tiempo de carga de la información en el sistema distribuído, como así también del procesamiento de la *query* sí.

Las pruebas se han realizado en el *hardware* del laboratorio 3 de SDT, utilizando un **n=10** y **provincia=Santa Fe** (para donde corresponda).

Query 1

Descripción

Total de habitantes por región del país, ordenado descendentemente por el total de habitantes.

Implementación

Para contar la población por regiones, el *mapper* mapea cada `InhabitantRecord` a la región de su provincia, y luego el *reducer* cuenta la cantidad de `InhabitantRecord`s por región.

Tiempos

- **Carga:** 3.329 s
- **Trabajo *map/reduce*:** 2.607 s

Query 2

Descripción

Los "n" departamentos más habitados de la provincia "prov".

Implementación

Tomando la Provincia y la cantidad de departamentos por parámetro, con `ProvinceFilterMapper` se emite únicamente los `InhabitantRecord`s que corresponden a la provincia pasada como parámetro, y se emiten con su departamento como clave. Luego, `InhabitantsPerDepartmentReducer` cuenta la cantidad de habitantes para cada departamento.

Después de esto, se aplica un `Collator` que utiliza una `PriorityQueue` donde se almacenan todos los pares departamento-población, usando como comparador la población en orden descendente. Luego, se toman los primeros n valores que devuelve la `PriorityQueue`, donde n es el parámetro de la *query*.

Como alternativa al `Collator` implementado, una opcion podria haber sido ordenar todos los valores obtenidos según su población, almacenandose primero en una lista. Otra, podría haber sido recorrer la lista una sola vez, llevando siempre los n departamentos con mayor población hasta el momento, y comparando cada valor de la lista contra el enésimo departamento con mayor población, para decidir si se reemplaza o no a alguno de los departamentos de la lista de mayores.

Consideramos la solución implementada es un buen compromiso al ser computacionalmente menos compleja que la primera opción descrita como alternativa, y menos compleja de implementar que la segunda.

Tiempos

- **Carga:** 3.17 s
- **Trabajo *map/reduce*:** 2.337 s

Query 3

Descripción

Índice de desempleo por cada región del país, ordenado descendentemente por el índice de desempleo.

Implementación

Primero, el *mapper* mapea cada `InhabitantRecord` a la región de su provincia, y luego el reducer cuenta la cantidad de habitantes empleados y desempleados. Con esos datos, se calcula el *ratio* y luego se ordena los resultados por el mismo.

Tiempos

- **Carga:** 3.059 s

- **Trabajo *map/reduce*:** 2.29 s

Query 4

Descripción

Total de hogares por cada región del país ordenado descendentemente por el total de hogares.

Implementación

Primero, el *mapper* mapea cada `InhabitantRecord` a la región de su provincia, y luego el reducer *hashea* los IDs de hogar para no obtener repetidos. Luego, retorna el tamaño del *set* y el `Collator` ordena los resultados obtenidos de mayor a menor según la cantidad de hogares.

Tiempos

- **Carga:** 2.967 s
- **Trabajo *map/reduce*:** 1.162 s

Query 5

Descripción

Por cada región del país el promedio de habitantes por hogar, ordenado descendentemente por el promedio de habitantes, mostrándose siempre dos decimales.

Implementación

Similar al anterior, pero ahora se cuenta también la población total de cada región, y luego se divide la población de cada región sobre la cantidad de hogares, para obtener el promedio de personas por hogar. También se ordena de mayor a menor por el promedio de personas por hogar.

Tiempos

- **Carga:** 3.061 s
- **Trabajo *map/reduce*:** 2.741 s

Query 6

Descripción

Los nombres de departamentos que aparecen en al menos n provincias, ordenado descendentemente por el número de apariciones.

Implementación

La idea es que el *mapper* emita pares <"Nombre de Departamento", "Provincia"> por cada entrada del censo. De tal forma, cada uno de los reducers de los departamentos encontrados mantiene un contador y un *set* de las provincias ya contabilizadas (para no contar más de una vez la aparición de un departamento en una provincia dada). Luego, un `Collator` se ocupa de filtrar los nombres de departamento cuyo número de coincidencias sea menor al n requerido, como así también se encarga de ordenar los resultados por dicho número de coincidencias.

Tiempos

- **Carga:** 4.829 s
- **Trabajo *map/reduce*:** 2.914 s

Query 7

Descripción

Los pares de provincias que comparten al menos n nombres de departamentos, ordenado descendentemente por la cantidad de coincidencias.

Implementación

Similarmente al caso anterior, el *mapper* se encarga de emitir pares <"Nombre de Departamento", "Provincia">. El reducer, en cambio, irá añadiendo a un *set* las provincias en las que aparece un determinado departamento, para luego emitir un *set* de `ProvincePair` (un par de provincias ordenado según el orden alfabético de las mismas).

El `Collator` se ocupará de contabilizar las múltiples apariciones de un ProvincePair entre los diferentes departamentos, eliminará los resultados con menor n que el requerido y los devolverá ordenados.

Tiempos

- **Carga:** 3.275 s
- **Trabajo *map/reduce*:** 4.886 s