Programación de Objetos Distribuidos

Trabajo Práctico Especial

Grupo

Hufflepuff

Integrantes

Leonel Badi

Lucas Farré

1. Ejecución

Para simplificar la puesta en marcha del trabajo práctico se decidió hacer que el mismo sea un proyecto *Maven*. De esta manera la generación del *jar* se logra simplemente corriendo el siguiente comando desde la raíz del proyecto:

```
mvn package
```

Este *jar* contiene todas las dependencias incluyendo *hazelcast*, así, el seteo de variables de entorno se vuelve mucho más simple, como veremos a continuación.

1.1. Levantar un nodo

Para levantar un nodo debe procederse primero a setear la variable entorno *CLASSPATH* con la ruta del *jar* generado anteriormente. Lo hacemos de la siguiente manera:

```
CLASSPATH=/path/absoluto/al/proyecto/target/concurrentJson-1.0-SNAPSHOT.jar
```

Luego, debe especificarse la interfaz de red en *hazelcast.xml* que se encuentra en la raíz del proyecto. En el caso de la red del ITBA lo hacemos así:

```
<interfaces enabled="true">
      <interface>10.6.0.*</interface>
</interfaces>
```

Ahora sí, basta con ejecutar el siguiente comando para levantar el nodo:

```
java com.hazelcast.console.ConsoleApp
```

1.2. Correr el cliente

Ponemos en marcha el cliente de la siguiente forma:

```
java -jar /path/absoluto/al/proyecto/target/concurrentJson-1.0-SNAPSHOT.jar
query=1 n=10 path=/path/absoluto/al/proyecto/resources/imdb-200.json
```

2. Diseño

Para realizar la lectura del archivo y el parseo de datos se utilizó la librería *Gson*. Con esta se logró simplificar el parseo al solo tener que definir la clase *Movie* para que corresponda con la estructura de *JSON* provista.

En una primera instancia se completó el *IMap*(estructura distribuida entre los nodos) de manera sincrónica. Este proceso era eficiente al correrlo en un solo nodo, pero cuando se realizaron pruebas en varios nodos simultáneamente agregaba un overhead considerable al tener que estar sincronizando los nodos para cada dato ingresado. Por esta razón se decidió agregar los valores asincrónicamente con las funciones que provee *hazelcast*.

Para la primera *query* se realizó un mapeo, utilizando como key el nombre de cada actor y emitiendo en el value el score de la película en la que actuó. De esta manera en el reducer sumarizamos todos los score para cada actor contando sobre una variable. El reducir emite la suma de todos los *score* del actor que está reduciendo. Por último tenemos un *Collator* para solamente mostrar los *N* actores más populares.

Para la segunda *query* se realizó el mapeo utilizando como *key* el año de una película (siempre y cuando este año sea mayor que el año tope). De esta manera el reducer recibo para un año (*key*) , la lista de todas las películas. Luego el reducer solo se queda con las que tienen mayor metascore y para cada año devuelve la lista de las mejores.

Para llevar a cabo la tercera *query* se realizó un mapeo por actor y como *value* la lista de todos los compañeros de esa película, excluyéndose él mismo. De esta manera el *reducer* hace el registro de todas la parejas para un actor, contando la cantidad de veces que actuaron junto. Para realizar esto utiliza un mapa que tiene como *key* el nombre del compañero y como *value* la entidad *Partner*. Luego al finalizar el *reducer* sólo devuelve las parejas que más apariciones tuvieron con ese actor. En una última instancia hay un *Collator* que filtra y sólo devuelve las que más apariciones tuvieron entre todo el conjunto de actores.

Para la cuarta query se hizo un mapeo que tiene como *key* el nombre del director y como *value* la lista de actores que dirigió en una película específica. El *reducer* al recibir el director con las listas para las películas que dirigió cuenta cuáles fueron los actores que tienen más apariciones. Para realizar esto utiliza un mapa que tiene como *key* el nombre del actor y como *value* la cantidad de apariciones.

3. Rendimiento

Las siguientes pruebas se realizaron utilizando una *MacBook Air* con un procesador Intel Core i5 de 1.3GHz con 4GB de RAM. Los tiempos siguientes son medidos en segundos y tienen la suma de todos los tiempos(lectura del archivo, parseo y ejecución).
Parámetros de Q1: N = 10.
Parámetros de Q2: TOPE= 2005.

	Q1	Q2	Q3	Q4
1 nodo	8	4	12	5
2 nodos	9	5	17	8
3 nodos	9	4	21	9
4 nodos	10	6	25	5

Se puede observar que aunque se aumente la cantidad de nodos, en la mayoría de los casos el rendimiento empeora. Esto puede deberse al trabajo extra que cuesta sincronizar el IMap al principio y la comunicación entre los nodos.