Sesión 4
Java API y serialización

Experto Big Data

Profesor: Iván de Prado
Alumno: David Suárez Caro

Ejercicio 1. Histograma con Writables

Para este ejercicio hemos utilizado dos jobs MapReduce. El primero llamado CalculateMaxMin lo utilizamos para calcular como su propio nombre indica el máximo y el mínimo valor de entre todos los valores de entrada.

CalculateMaxMinJob

NumbersTuple:
Writable propio que posteriormente utilizaremos. Se compone de dos DoubleWritable max y min respectivamente.

CalculateMaxMinDriver:
Aquí configuramos el archivo de entrada y las clases Mapper y Reducer que vamos a utilizar (CalculateMaxMinMapper, CalculateMaxMinReducer).

También configuramos las salidas del Mapper (IntWritable, NumbersTuple) y del job (IntWritable, NumbersTuple).

CalculateMaxMinMapper:
En este Mapper simplemente recibimos un (Text, NullWritable) y enviamos una nueva tupla formada por un IntWritable(1) y un Writable creado por nosotros llamado NumbersTuple el cual hemos explicado anteriormente.

CalculateMaxMinReducer:
Se recibe un IntWritable, cuyo valor en este caso siempre es 1, para unificar todas las tuplas, y un iterable de NumbersTuple que contiene los máximos y los mínimos respectivamente. Todos estos se recorren para obtener el máximo y mínimo total de toda la secuencia de números procesada.

HistogramJob

HistogramDriver:
Aquí configuramos las tuplas de entrada y las clases Mapper y Reducer que vamos a utilizar (HistogramMapper, HistogramReducer).

También configuramos las salidas del Mapper (IntWritable, IntWritable) y del job (IntWritable, IntWritable).

HistogramMapper:
Recibimos un DoubleWritable y procedemos al enrutamiento de números teniendo ya del job anterior el máximo y el mínimo y el numero de barras pasadas por parámetro.

HistogramReducer:
Muy sencillo en este caso calcula la suma del total de números que van a esa barra.

Ejercicio 2. Los amigos de mis amigos con Writables

Para este ejercicio hemos utilizado un solo job MapReduce el cual hemos llamado Friends.

Friends

FriendsTuple:
Writable propio que posteriormente utilizaremos. Se compone de dos Text bool y friend respectivamente.

FriendsDriver:
Aquí configuramos el archivo de entrada y las clases Mapper y Reducer que vamos a utilizar (FriendsMapper, FriendsReducer).

También configuramos las salidas del Mapper (Text, FriendsTuple) y del job (Text, Text).

FriendsMapper:
En este Mapper simplemente recibimos un (Text, Text) con la implicación A es amigo de B y enviamos dos nuevas tuplas formadas por (A, (true, B)) y (B, (false, A)) cuyos tipo son (Text, FriendsTuple).

FriendsReducer:
En el reducer vamos añadiendo a dos listas (directRel y reverseRel) respectivamente que luego vamos recorriendo de manera conjunta para emitir los amigos de nuestros amigos.

Ejercicio 3. Writables con comparadores eficientes

Para este ejercicio hemos heredado de la clase WritableComparator y hemos implementado el método compare.

NumbersTupleComparator

```
public class NumbersTupleComparator extends WritableComparator{

      public NumbersTupleComparator(){
        super(NumbersTuple.class);
    }
    @Override
    public int compare(WritableComparable o,WritableComparable o2){
        NumbersTuple m = (NumbersTuple)o;
        NumbersTuple m2 = (NumbersTuple)o2;
        int cmp = m.getmax().compareTo(m2.getmax());
        if (cmp != 0) {
            return cmp;
            } else
            return m.getmin().compareTo(m2.getmin());
    }
}
```

FriendsTupleComparator

```
public class FriendsTupleComparator extends WritableComparator{
      public FriendsTupleComparator(){
        super(FriendsTuple.class);
    }
    @Override
    public int compare(WritableComparable o,WritableComparable o2){
      FriendsTuple m = (FriendsTuple)o;
      FriendsTuple m2 = (FriendsTuple)o2;
        int cmp = m.getBool().compareTo(m2.getBool());
        if (cmp != 0) {
            return cmp;
            } else
            return m.getFriend().compareTo(m2.getFriend());
    }
}
```

Github Repository:
https://github.com/davsuacar/Session4_HadoopJavaAPI