Lab Lecture #3.2

Word count in document

The goal of this phase is to count the total number of words for each document, in a way to compare each word with the total number of words. The mapper will receive as input a key that is, by default, the byte offset of the current line in the current file processed (`LongWritable` object) and a value that is the line read from the file (`Text` object). It must output another (key, value) pair. The problem is again coding the (word, *n*) pair in a single object. We will use the "string trick", emitting a simple string composed by the word, the special character "`/`" and a string representing the number of occurrences of the word in the document. In order to split a string `line` in an array of strings with a custom char you can use the following statement:

```
String[] tokens = line.split("@");
```

The reducer will receive two `Text` objects as keys and values representing the (doc name, (word, *n*) couple. It just needs to sum the total number of values in a document and pass this value over to the next step, along with the previous number of keys and values, as necessary data for the next step, writing in the files a line for each ((word, doc name), (*n*, *N*)) couple's couple ☺.

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCountInDocument
{
   public static class NewMapper extends Mapper<LongWritable, Text, Text, Text>
   {
      public void map(LongWritable key, Text value, Context context)
         throws IOException, InterruptedException
      {
         String[] wordAndDocCounter = value.toString().split("\t");
         String[] wordAndDoc = wordAndDocCounter[0].split("@");
         context.write(new Text(wordAndDoc[1]),
                       new Text(wordAndDoc[0] + "=" + wordAndDocCounter[1]));
       }
   }

   public static class NewReducer extends Reducer<Text, Text, Text, Text>
   {
      protected void reduce(Text key, Iterable<Text> values, Context context)
         throws IOException, InterruptedException
      {
         int sumOfWordsInDocument = 0;
         Map<String, Integer> tempCounter = new HashMap<String, Integer>();
         for (Text val : values) {
            String[] wordCounter = val.toString().split("=");
            tempCounter.put(wordCounter[0], Integer.valueOf(wordCounter[1]));
            sumOfWordsInDocument += Integer.parseInt(val.toString().split("=")[1]);
         }
         for (String wordKey : tempCounter.keySet())
            context.write(new Text(wordKey + "@" + key.toString()),
                          new Text(tempCounter.get(wordKey) + "/" + sumOfWordsInDocument));
       }
   }

   public static void main(String[] args) throws Exception
   {
      Configuration conf = new Configuration();
      Job job = new Job(conf, "word count in document");
      job.setJarByClass(WordCountInDocument.class);

      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);

      job.setMapperClass(NewMapper.class);
      job.setReducerClass(NewReducer.class);

      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));

      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```