Lab 5: Simple Hadoop Programs

Due date: February 18, 11:59pm.

Lab Assignment

Assignment Preparation

This is an individual lab. I expect every person to complete it without consulting others.

Overview

In this lab you will create a number of Hadoop MapReduce programs that perform a variety of simple tasks on simple data inputs. All programs you will write share the same features:

- The inputs are files with simple, easy-to-understand structure, with each record stored in a single line.
- The `map()` and `reduce()` methods will not need to use any functionality beyond core Java libraries.
- The MapReduce job you are asked to implement is an example of a specific type of a data-processing operation discussed in class.

Program 1: Filtering

Create a MapReduce job named `repeatLetters.java` that works as follows.

Input. The input to the MapReduce process is a file containing a list of words, one word per line.

Processing. Your MapReduce job shall output a list of words that have double letters in them. For example, words like "hello", "book", "barrow" shall be emitted, while words like "table", "break", "analysis" shall be filtered out. The MapReduce job shall be case-insensitive, so, "Llan", for example, shall be printed out.

Output. The `reduce()` method shall emit the word as the key, and the letter that is doubled as the value. For example, for "Hello", the output shall be

```
Hello    l
```

For words with multiple double letters, e.g., "toothiness", report one of the letters (either "o" or "s").

Program 2: Transformation

Create a MapReduce job named `invert.java` that works as follows.

Input. The input to the MapReduce process is a file containing a list of words, one word per line.

Processing. Your MapReduce job shall output the list of word pairs: the first word of the pair (the key) is the original word, and the second word - the original word with characters in each word has been reversed. For example if the input word is "stream", the program shall produce as output the following pair:

```
stream     maerts
```

Output. The `reduce()` method shall output the input word as the key, and the transformed word as the value.

Program 3: Grouping and Aggregation

Create a MapReduce job named `scores.java` that works as follows.

Input. The input to the MapReduce process is a file that contains records of the following format:

```
<itemId>, <numPurchased>, <pricePerUnit>, <shippingCost>
```

Each record is designed to mimic information taken from a collection of sales invoices. Each record specifies details of a single purchase of one specific product identified by `<itemId>`. The number of units purchased is in

<numPurchased> field, the remaining two fields are the price per unit and the shipping cost added to the order to ship the items. A sample set of records may look as follows:

```
1, 3, 45.5, 2.33
2, 1, 100.00, 0.00
1, 2, 46, 1.50
```

Processing. The MapReduce job shall compute for each item, how many units were sold, and what was the total profit from selling the units. The profit is computed as follows. If fewer than 100 units of an item were sold (overall), profit is 2.25% of the total revenue (cost of the items plust cost of the shipping). If more than 100 units of an item were sold, profit is 2.25% of the total revenue for the first 100 items (as they are found in the input file), and 3% of the total revenue for the remaining items (split the shipping costs proportionally, where necessary).

Output. The `reduce()` method shall output the item Id as the key, and the pair <numberOfUnitsSold>, <Profit> as the value. The output shall look as follows (minor syntactic deviations are ok):

```
1 126, 347.63
```

Program 4: Inverting

Construct a MapReduce job `invertedIndex.java` that works as follows.

Input. The input to the process is a text file of the form

```
<messageID>, <message text>
```

Here, <messageID> is a number, and <message text> is a string in double quotes. A sample line of input can look as follows:

```
4052, "I think. MongoDB is awesome!"
```

Processing. Your MapReduce job shall output for each word found in the entire input the following information:

- Number of unique occurrences
- Number of documents in which it occurs
- List of documents in which it occurs

Output. In the output, use the word as the key, and output a printable object that consists of the two values and the list specified above. For example, the output may look as follows:

```
MongoDB     8, 5, [1972, 4052, 7402, 9734, 10934]
```

Program 5: Putting it all together

Create a MapReduce program `mixture.java` that behaves as follows.

Input. The input to the prgram is a file in which each line has the following format:

```
<word>, <word>, <word>
```

Sample input lines can be:

```
dense, giraffe, staple
always, decided, fun
```

Processing. Your MapReduce job shall output pairs of words that satisfy the following conditions:

- the two words appeared on the same line
- the first word is shorter than the second
- the first word appeared in multiple lines
- the second word is the longest word that co-occurred on the same line with the first word

If there is more than one word that can fit the last rule, your program must output each pair separately.

For example, if the input lines are

```
band, brand, stubble
brand, rattles, door
band, door, structure
```

the output shall be (in no particular order):

```
band   structure
brand  stubble
brand  rattles
door   structure
```

You can use the first word as the key and the second word as the value for your `reduce()` output.

Submission

Submit your Java programs.

All submitted files must contain your name on them.

Submit all your code in a single archive (zip or tar.gz).

Use `handin` to submit as follows:

Section 01:

```
$ handin dekhtyar lab06-01 <FILES>
```

Section 03:

```
$ handin dekhtyar lab06-03 <FILES>
```

Good Luck!