Project: Part 1
Hadoop MapReduce

Assignment

Starting from the code of Exercise 1, you should provide the following MapReduce programs:

Exercize A: MaReduce job concatenation A MapReduce program for transposing the output of the word count program. More precisely, the program should associate each frequency (i.e., the number of times a word appears in the text) with the words with that frequency.

For instance, given the output

```
car 3
the 6
house 3
phone 5
pen 3
glass 3
battery 5
```

the following output has to be generated:

```
3 car pen house glass
5 battery phone
6 the
```

Hints. To solve this exercize, you should concatenate two MapReduce jobs inside the same program, using the output of the first as the input of the second. To this aim, you can rely on the usage of `SequenceFileOutputFormat` and `SequenceFileInputFormat` classes, in order to store the result of the first

job in a (temporary) file, maintaining the (key, value) structure, and to read such file in the second job. The class `Driver`, made available for this exercize, will help you to achieve this behavior.

Exercize B: MaReduce job concatenation A MapReduce program, obtained by modifying the one developed for Exercize A, that counts the number of words with the same frequency.

For instance, given the input

```
car 3
the 6
house 3
phone 5
pen 3
glass 3
battery 5
```

the following output has to be generated:

```
3 4
5 2
6 1
```

Exercize C: Combiner A MapReduce program for reducing the amount of data that are sent over the network from the mappers to the reducers, through the usage of a Combiner.

Hints. The combiner for a job has to be set with:

```
job.setCombinerClass(Combiner.class);
```

where `Combiner` is the class you want to use as the combiner.

Define, *if required*, a Combiner class and configure the job to use the Combiner in one of the transpose programs, designed as solutions for Exercizes A and B.

Exercize D: Sequence-based output storage Modify the programs developed for Exercize A and Exercize B in order to use `SequenceFileOutputFormat` also as output format for the final data.

Then, run the modified transpose program (Exercize A) passing as output directory `wordcount/wordtrans` and look at the result. Run the modified frequency program (Exercize B) passing as output directory `wordcount/wordfreq`

and look at the result. The stored files will be used as input for the next exercizes.

Hints. If you print the result in `wordcount/wordtrans` or `wordcount/wordfreq`, since the files are no more stored in text format, you will see a different result with respect to those obtained for Exercizes A and B.

Exercize E A MapReduce program for computing maximum frequency and its associated words. Use `wordcount/wordtrans` as input directory.

For instance, with input:

```
3 car pen house glass
5 battery phone
6 the
```

the output will be:

```
6 the
```

Hints. In this exercize, you should set the number of reducers to 1.

You should then start by computing the 'local' maximum for each Mapper tasktracker, that is the maximum among the values generated by all map function calls managed by that Mapper, and send it to the reducer. To this purpose, you need to define some variables local to the Mapper.

Hadoop uses reference reusing, thus, in order to assign keys and values to an instance variable, copy by value should be used by calling the `set` method or using a costructor.

For instance, assume `words` is an instance variable of type `Text` and assume you want to assign `value` to `words`. In order to understand what is the problem related to reference reusing, look at the following code:

```
protected void map(IntWritable key, Text value, Context ctx) {
  words = value; // wrong
  words.set(value) // ok
  words = new Text(value); // ok
}
```

You should also take into account that in the `reduce` function, once you consume a value from the `Iterable` representing the input value, the consumed value is no more accessible.

For instance, assume `value` is a list containing only one element. In order to undestand the problem related to `Iterable` management, look at the following code:

```
protected void reduce(IntWritable key, Iterable<Text> value, Context ctx) {
  Text v = value.iterator().next();
  value.iterator().hasNext() // false
}
```

Exercize F * A MapReduce program for computing both the maximum and the minimum frequency and their corresponding associated words.

Exercize G* A MapReduce program for computing the average frequency of words in a text. The input for such MapReduce program is `wordcount/wordfreq`.

For instance, with input:

```
3 4
5 2
6 1
```

the output will be:

```
4.0
```

Hints. The solution to this problem is quite similar to that proposed for Exercize E and F; however, since average is not associative, you should send to reducers all information required to correctly compute the global average. The final output should be written relying on the `TextOutputFormat` class. In case you do not want to associate any value in output pairs, you should use type `NullWritable` as output value type.

Exercize H A MapReduce program for performing a simple join-like operation between the output data generated in Exercize D.

More precisely, in Exercise D, you generated transposed data (stored in `wordcount/wordtrans`) like:

```
3 car pen house glass
5 battery phone
6 the
7 a
```

You also generated count frequency data (stored in `wordcount/wordfreq`) like:

```
3 4
5 2
6 1
```

Now, you should join the rows of the two files sharing the same key, obtaining:

```
4 car pen house glass
2 battery phone
1 the
```

Hints. Use as input the directories `wordcount/wordfreq` and `wordcount/wordtrans` made in Exercise D.

The `Driver` class, made available for this exercize, relies on `MultipleInputs` that allows you to deal with multiple mappers, each working on a different file.

You may also need the class `ValuesWritable`, made available for this exercize. This class implements `Writable` and thus can be used as value type in map/reduce functions. It is not mandatory to use class `ValuesWritable`, if you find other ways to solve the exercise.

Rules for project development and delivery

- The project, Part 1, can be developed by groups of up to two persons.
- Each student should upload on AulaWeb, a single zip file of name CognomeN (where N is the initial name letter). The file should contain: (i) one folder of name ExercizeX containing the source files (driver, map, reduce, and possibly combiner classes) developed for Exercize X; (ii) a short document specifying the names of the group members and describing the proposed solution for each exercize.
- The zip file should be uploaded by November 23.
- To Project Part 1 will be assigned a rating among {A+, A, B, C, D}, accordin to the following rules:
  - A+: all exercizes, from A to H, have been correctly solved
  - A: all exercizes except F and G have been correctly solved
  - B: exercizes A, B, C, D, E have been correctly solved
  - C: exercizes A, B, C, D have been correctly solved

– D: exercizes A and B have been correctly solved
– In all other cases, no rating will be provided.