CSCI4180 (Fall 2015)

Assignment 1: Counting by MapReduce

Due on Oct 22, 2015, 23:59:59

Introduction

The goals of this assignment are to let students get familiar with Hadoop MapReduce and simple MapReduce development.

Part 1: Getting Started on Hadoop (20%)

In this part, you need to demonstrate the following:

1. Configure Hadoop in *fully distributed mode* on all your assigned virtual machines.
2. Run the provided WordCount program on your Hadoop platform.

Please show to the TAs that you achieve the above requirements.

We provide two sample datasets for you to test your program. One is the KJV Bible, and another one is the complete works of Shakesphere. Note that the sample datasets are not of large scale (each of them contains no more than 6MB), so you may not see the performance benefits when multiple mappers are used.

More instructions will be provided in the tutorials. Make sure that you strictly follow the instructions (e.g., the versions of JDK and Hadoop, the configuration of the software); otherwise the TAs cannot troubleshoot the problems and you will risk losing marks!

Part 2: Word Length Count (20%)

Please extend the WordCount program to count the length of each word with the optimization technique *in-mapper combining*. Call the program *WordLengthCount.java*.

Output Format:
Each line contains a tuple of (length, count), separated by a space character. For example, "`3 5`" means that there are a total of five words of length three. The output is not necessarily printed in any increasing/decreasing order of length or count.

Sample Input:

```
The quick brown fox jumps
over the lazy dog.
```

Sample Output:

```
3 3
```

```
5 3
4 3
```

Time Limit: Note that the program should be completed within a reasonable timeframe for a given dataset. The time limit is set by the TAs. Marks will be deducted if the program runs too long.

Note: The punctuations should also be counted toward the length of a word for your convenience. For example, in the sample input, the word "dog." should be counted as a word with length 4 instead of 3.

Part 3: Counting Initial Sequences of N-grams (25%)

Using *WordLengthCount.java* as a starting point, extend it to count the initial sequences of N-grams (call the program *NgramInitialCount.java*). An *N-gram* is a sequence of N consecutive words, where N is the input in the command-line argument. For a given N, we define an *initial sequence* as the first letters of the N words in an N-gram.

Output Format: Each line contains a tuple of (1st word's initial, 2nd word's initial, ..., N-th word's initial, count), where the adjacent items are separated by a space character. In this problem, please only output the count of the initial sequences that are *all in alphabets*. You can output the count of each initial sequence in any order, but the initials should be case-sensitive. That is, "`A a`" and "`a a`" should be counted and displayed separately. See the example below, where $N = 2$.

Command Format:

```
$ hadoop jar [.jar file] [class name] [input dir] [output dir] [N]
```

Sample Input:

```
‘The quick brown fox jumps
over the lazy dogs.
```

Sample Output:

```
q b 1
b f 1
f j 1
j o 1
o t 1
t l 1
l d 1
```

Notes:

- We skip "`‘The`" because it starts with the single quote "`‘`".
- The word in the end of a line and the word in the beginning of the next line can also form an N-gram. For example, "`jumps over`" is an N-gram for $N = 2$. Thus, we need to include "`j o`" as well.
- Two N-grams may have the same initial sequence, such as "`The dog`" and "`The duck`", and we count the initial sequence twice.
- We may miss some N-grams that appear in two different HDFS blocks, but it's okay.
- You don't need to implement in-mapper combining here, but you are strongly encouraged to think of any possible way to speed up your program.

Time Limit: Note that the program should be completed within a reasonable timeframe for a given dataset. The time limit is set by the TAs. Marks will be deducted if the program runs too long.

Part 4: Computing Relative Frequencies of Initial Sequences of N-grams (25%)

Extend Part 3's program to compute the relative frequencies of the initial sequences of N-grams (call the program *NgramInitialRF.java*). Again, in the problem, you only need to consider the initial sequences whose initials are in all alphabets. For a given N, the relative frequency of an initial sequence is defined as count("`X Y Z`")/count("`X *`"), where count() is a function that counts the occurrence of an input, `X Y Z` is an initial sequence (for $N = 3$ here), and `*` stands for any alphabet initials.

Output Format: Each line contains a tuple of (1st word's initial, 2nd word's initial, $\cdots$, N-th word's initial, frequency), where the adjacent items are separated by a space character. You only need to output the initial sequences whose relative frequencies are at least θ, where θ is a configurable parameter specified in the command-line argument and $0 \leq \theta \leq 1$. See the following example for $N = 2$ and $\theta = 0.6$.

Command Format:
`$ hadoop jar [.jar file] [class name] [input dir] [output dir] [N] [`θ`]`

Sample Input:
```
the quick brown fox jumps over the lazy dogs.
the Quick Brown fox jumps Over the Lazy dogs.
the Quick Brown fox Jumps Over the Lazy ‘‘dogs’’.
```

Sample Output:
```
f j 0.66666667
q b 1
b f 1
o t 1
l d 1
d t 1
Q B 1
B f 1
O t 1
L d 1
J O 1
```

Time Limit: Note that the program should be completed within a reasonable timeframe for a given dataset. The time limit is set by the TAs. Marks will be deducted if the program runs too long.

Notes:

- For the initial sequence "`f j`", since count("`f j`") = 2 and count("f *") = 3, the relative frequency is $2/3 > \theta$.
- For the initial sequence "`L d`", we can see that count("`L d`") = 1 and count("L *") = 1. Since we only consider alphabet initials, "`L ‘‘`" should not be counted into count("`L *`").

Part 5: Configure Hadoop on Azure (10%)

In this part, you need to demonstrate the following:

1. Configure Hadoop in fully distributed mode on Windows Azure, using 2 compute instances that we assign to each group. In particular, you should configure one instance to run as the JobTracker, and another instance to run as the TaskTracker.

2. Run the provided WordCount program on Azure.

Please show to the TAs that you achieve the above requirements.

Bonus (5%)

The top 3 groups whose Part 4's programs have the smallest running time will receive the bonus marks. You may consider to optimize your programs or configure some parameters in Hadoop to make the programs perform better. If more than 3 groups have the best performance, we will still give out the 5% bonus to each group. Note that the program must return the correct answer in order to be considered for the bonus mark.

Submission Guidelines

You *must* at least submit the following files, though you may submit additional files that are needed:

- *WordLengthCount.java*
- *NgramInitialCount.java*
- *NgramInitialRF.java*

Your programs must be implemented in Java. Any program that is not compilable will receive zero marks!!

The testing platform during demo is provided by the TAs. If you need to change the configuration of Hadoop, make sure that you do it dynamically inside the code. You are not allowed to modify any configuration files during demo.

Please refer to the course website for the submission instructions.

The assignment is due on Oct 22, 2015, 23:59:59. Demo will be arranged on the next day.

Have fun! :)