CSCI4180 (Fall 2024)

Assignment 1: Counting by MapReduce

Due on Oct 17, 2024, 23:59:59

Introduction

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

In Parts 1-4, you only need to configure Hadoop in *pseudo-distributed mode*. For the bonus part, you will need to configure Hadoop in *fully distributed mode*.

Part 1: Getting Started on Hadoop (25%)

In this part, you need to demonstrate the following:

1. Configure Hadoop in *pseudo-distributed mode* on one of 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 Java 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 (25%)

Please extend the WordCount program to count the length of each word with the optimization technique *default 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.

In-mapping combining is *not* allowed for this part.

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

Using *WordLengthCount.java* as a starting point, extend it to count the sequences of N-grams (call the program *NgramCount.java*). An N-*gram* is a sequence of N consecutive words, where N is the input in the command-line argument.

Output Format: In this problem, we define a word as a sequence of English *alphanumeric* characters (e.g., dogs, cat31, etc.). Each line contains a tuple of (1st word, 2nd word, ..., N-th word, count), where the adjacent items are separated by non-alphanumeric characters (e.g., space, tab, newline, or punctuations). You may assume that the datasets contain only printable ASCII characters, all in English. You can output the count of each sequence in any order. Words are case-sensitive. That is, "`Apple apple`" and "`apple apple`" 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:

```
The quick 1
quick brown 1
brown fox 1
fox jumps 1
jumps over 1
over the 1
the lazy 1
lazy dogs 1
```

Notes:

- We treat the punctuations "`'`" and "`.`" as delimiters. We treat only "`The`" and "`dogs`" as words.
- For a word like "`ca't2e`", we treat it as two words "`ca`" and "`t2e`".
- 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 $N = 2$.
- Multiple non-alphanumeric characters are treated as a single delimiter. For example, "`quick ,,,,, brown`" will form a 2-gram "`quick brown`".
- We may miss some N-grams that appear in two different HDFS blocks, but it's okay.
- You don't need to implement any combining technique 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 Sequences of N-grams (25%)

Extend Part 3's program to compute the relative frequencies of the sequences of N-grams (call the program *NgramRF.java*). For a given N, the relative frequency of a 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 a sequence (for $N = 3$ here), and $*$ stands for any word sub-sequences.

Output Format: Each line contains a tuple of (1st word, 2nd word, $\cdots$, N-th word, frequency). You only need to output the 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:

```
fox jumps 0.66666667
quick brown 1
brown fox 1
over the 1
lazy dogs 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 sequence "`fox jumps`", since count("`fox jumps`") = 2 and count("fox *") = 3, the relative frequency is $2/3 > \theta$.
- For the sequence "`lazy dogs`", we can see that count("`lazy dogs`") = 1 and count("lazy *") = 1, so the relative frequency is 1.

Bonus (5%)

(a) (2%) Configure Part 4's programs to run on Hadoop in fully distributed mode.

(b) (3%) The top 3 groups whose Part 4's programs have the smallest running time in fully distributed mode 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 bonus 3% 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*
- *NgramCount.java*
- *NgramRF.java*
- Declaration form for group projects: (`http://www.cuhk.edu.hk/policy/academichonesty/Eng_htm_files_(2013-14)/p10.htm`)

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.

Demo will be arranged on the next day.

Have fun! :)