CPSC 313: Distributed and Cloud Computing
Spring 2022

Lab 2: Count words in files: Map Reduce version

Summary

You will create a single python code file named "mr_count_words.py" that takes a list of files as input, and counts all the words in the list of files, printing out the list of words with their counts. This time, however, you will use a basic map reduce class to distribute the work. You are provided with the map reduce class in a file named “basicMR.py” attached to this assignment.

This will use similar code to your lab1 implementation, though your implementation for counting words will be much simpler as it’s applied through map reduce.
You should have the following constants & functions:

```
# stop words are words that don’t count – conjunctions, extensions, prepositions, etc. Use this list to check each word you find, and if it’s in this list you ignore it.
STOP_WORDS =
['a', 'an', 'and', 'are', 'as', 'be', 'by', 'for', 'if', 'in', 'is', 'it', 'of', 'or', 'py', 'rst', 'that', 'the', 'to', 'wit
h',]

def find_words(file_name):
“”” take a file and return a list of lists where each sublist is a list with the word and the number 1 for that file. Please note that you can use dictionaries instead of lists
  This may look like: [['foo', 1], ['bar', 1], [‘foo’, 1], ...]
“””


def count_words(item):
“””
  Take an item that is a list or tuple (or dictionary), get its word and count and return a tuple (or list or dictionary) that is the word with it’s summed count.

This is a two line function, but it’s a bit counter-intuitive because you should use the sum() function that takes an iterable. In the map reduce file you’ll notice that we can take a list of iterators over tuples of the form [<’word1’, 1>, <’word2’, 1>, <’word1’, 1>, …] and create a single iterable that gives you the entire chain of tuples. The sum function takes this iterable and applies sum to the numbers (all or 1 for this exercise) to get the count.

“””
def main():
“””
 main function: get the files from input (or *args) or just assume they or in an appropriate directory and use all .txt files there. Iterate through the files counting words in each file, then combine results from each
```

inside this main function you'll start a timer using time.perf_counter (you'll import the time library), create the SimpleMapReduct class with the find_words and count_words functions, call the objects __call__ method with the input files and a chunksize, sort the list by count in reverse order (highest count first), end the timer with another call to time.perf_counter(), calculate the time spent and you have the map_reduce

You should also call your previous implementation with the timer calls and then you can compare the time taken to do a set of rather large files (> 1GB) using both approaches.

"""

```
if __name__ == '__main__':
    main()
```

Requirements

- You should have a test file that tests both your original count_words functions and the new map_reduce output to see that they match. This is in addition to your individual unit tests for each implementation.
- I suggest you use pytest as your test framework, though any python test framework is fine. Look up pytest to see how to use it, though it's simple.
    - Install pytest – in your terminal within vscode or in a powershell window:
        - >> pip install pytest
    - Make sure in your test file each function (or class) you use has "test" in the name. As a convention, I tend to start each function with "test". So if I'm testing my word_count function, I'd have a "test_word_count" function in my test file.
    - To run the tests, just use the command "pytest" in your powershell or terminal
- You must log relevant information from your working code into a log file: "count_words.log"
    - Use the Python "logging" library. To see how to use that library, search for instructions.
- You must compute the total time it takes to count all the words in all the files, and output that clearly to the log file.
- You must follow the style guide (document is in Teams) with the following exceptions:
    - naming: Single letter variable or function names are NOT allowed.
        - All names must be meaningful and descriptive. Instead of "for x in list:" you may use "for loop_control in list" or something more meaningful.
    - You may use tabs instead of spaces, and your indentation is encouraged to be tab or 4 spaces instead of 2 spaces.
- You can find test files by searching for "random word test file"

- For this assignment you ARE required to test large files. We will go over an easy way to do this in class. Some hints:
    - You can call a REST service (https://random-word-api.herokuapp.com/word?number=1000) where you can use whatever number of words you want that will return a list of random words from a dictionary
    - That gives you a corpus of words, now you can randomly take words from this list and add them to a file with a space as a separater until you reach the file size you want

Submission

- Your python file (mr_count_words.py)
- Your test file (test_CW.py)
- A log file that shows the output of your test run and any log statements from the code