Central Washington University

Advance Algorithm

Winter 2020

Title-cased Word Count with MapR

Student: Tin Tan Nguyen
Email: ntin@cwu.edu

February 27, 2020

- **Project description**

  This project will use the concept of MapReduced to find the frequency of title-cased words in many large text files. For this projects, multiple of text books will be used to test the run time analysis of the algorithm.

- **MapR description:**

  Spawn multiple processes to process divided workloads in parallel in order to optimize run-time execution.

- **Project planning :**

  First we need to down load a large text file, preferably books from the internet.

  Python multiprocessing Pool package will be used to create parallel processes to read and count separate section of the text file simultaneously.

  We will start with 2 threads and go to to 8 threads to see if the run-time get improved as threads are increasing.

  We then can compare the run-time of MapR method vs sequential method and observe if there is improvement.

- **Run-time analysis:**

  First we need to process the text files to remove all spaces, lines and special characters in the texts. Then we need to put all the words from the text in a list and then we apply both sequential and MapReduce method to count how many title-cased words.

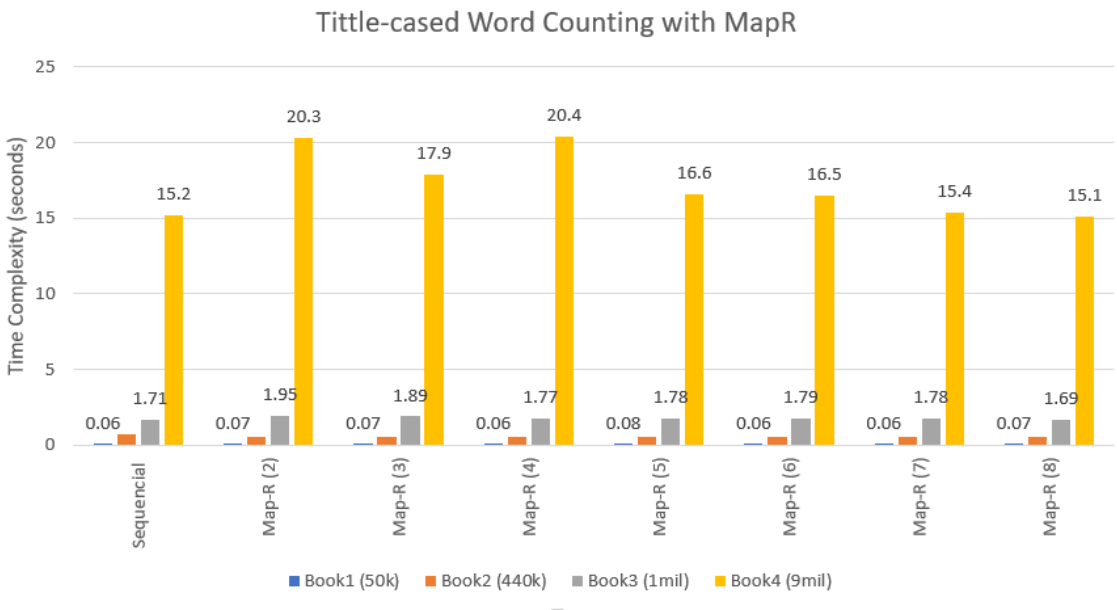

Figure 1: Time Complexity Analysis

From looking at the chart above, run-time of sequential method is a lot better than MapReduce method for smaller and larger samples (¡9mil words) because it costs time to create threads and do scheduling. If the sample size is small enough and the task is lightweight, sequential always beat MapReduce method.

We can then increase the workload of the task by adding the file processing along with word counting in the main task. This will add more workload for the CPU and this time we should expect to get better run-time of the MapReduce method compare to the sequential method.

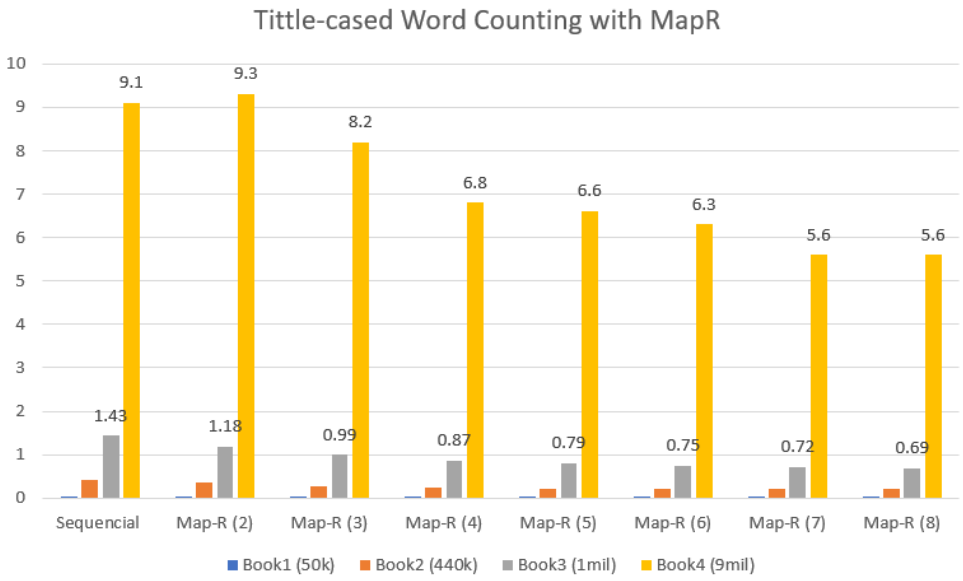

Figure 2: Time Complexity Analysis

As expected, run-time of MapReduce method is much better compare to sequential method in this second experiment just by increasing the workload. For this experiment, MapReduce run-time get saturated when number of processes reach to 7 and the run-time is about 40% faster compare to sequential method.

- **Python MapR Code:**

```
def read_count_MapR(number_of_process, filename):
    file_detail = split_file(filename)
    p = Pool(number_of_process)
    subSize = int(len(file_detail) / number_of_process)
    results = []
    for i in range(number_of_process):
        initial = int(i * subSize)
        if i == number_of_process - 1:
            end = len(file_detail)
        else:
            end = int((i + 1) * subSize)
        handle = p.apply_async(string_count,([file_detail[initial:end]]))
        results.append(handle)
    word_tracking_mapR = {}
    for r in results:
        d = r.get()
        for key in d:
            if key in word_tracking_mapR:
                word_tracking_mapR[key] = word_tracking_mapR[key] + d[key]
            else:
                word_tracking_mapR[key] = d[key]
    p.close()
    p.join()
    return word_tracking_mapR
```

Figure 3: Python MapR Code