Central Washington University

Advanced Algorithms

Winter 2019

Project 3 Report

Author:
Hermann Yepdjio

Professor:
Dr. Razvan Andonie

February 20, 2019

Contents

1 Introduction

The purpose of this project was to implement a multi-processing Map Reduce program to find frequencies of title-cased words in a large text-file. We then were supposed to run the program and record its efficiency for different file sizes and number of cores used. Those results have been compared to those of a similar but sequential program and the results will be discussed below on this report.

2 Experimentation Process

We wrote the program with the assumption that every word in the text file that starts with a capital letter is a title-cased word. So basically the program looks for words that start with an uppercase letter, then for each of them, counts and records how many times it appears in the file.

- "find_frequencies(args)" is the function that does this part,
- "args" is a list of size 2, containing the name of the file at index 0 and an uppercase letter to search for at index 1.

During the experimentation, we first tried to compare the results of the Map Reduce and the sequential search using different files of different sizes, and then we tried to compare only the results of the Map Reduce when using different number of cores.

2.1 Sequential Search Vs Map Reduce

For this part, we ran the experimentation 10 times using 10 files of different sizes and the results were recorded.

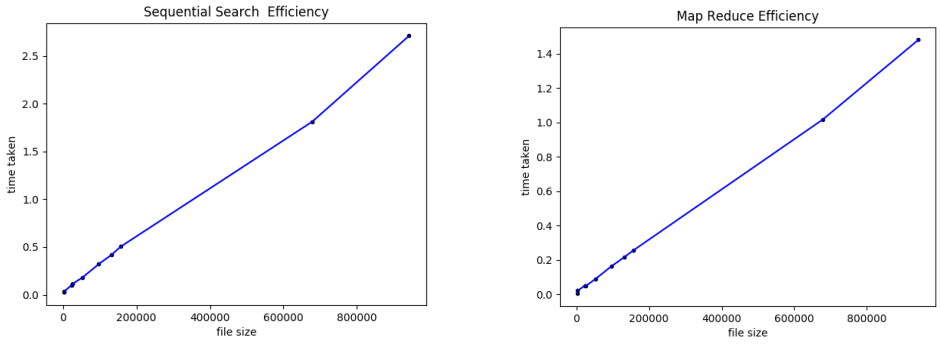
2.2 Varying the Number of Cores

Here we were only experimenting with the Map Reduce. we varied the number of cores to be used between 2 and 8 to observe if the performance would change and we recorded the results.

3 Results

3.1 Sequential Search VS Map Reduce

The first part of the experimentation was to compare the efficiencies of the sequential search and Map reduce. Figure 1 shows the results that we obtained.



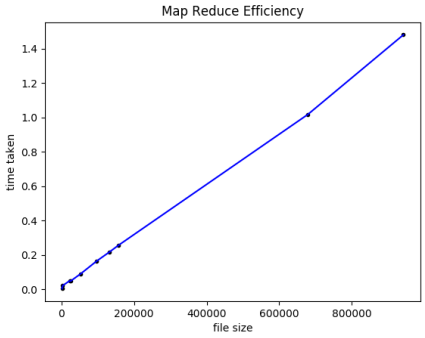
(a) Sequential Search (b) Map Reduce

Figure 1: Experiment_1 using 2 cores for Map Reduce

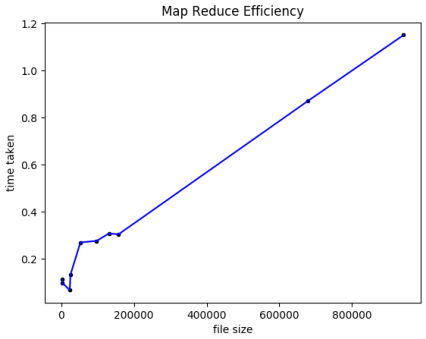
As we can see from Figure 1 above, the Map Reduce performs twice better than the sequential search.

3.2 Varying the number of cores

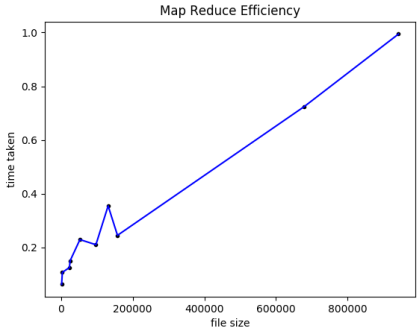
In the second part of the experimentation we varied the number of cores between 2 and 8 for the Map Reduce, and the results are depicted in Figure 2 below.



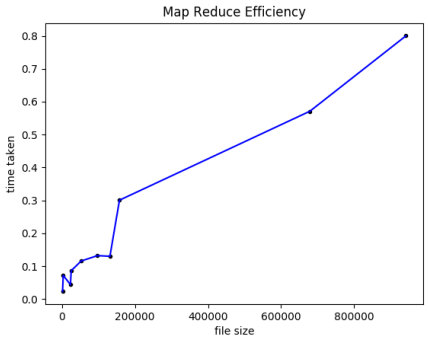
(a) Map Reduce with 2 Cores



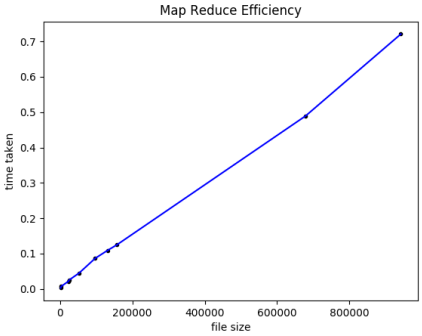
(b) Map Reduce with 3 Cores



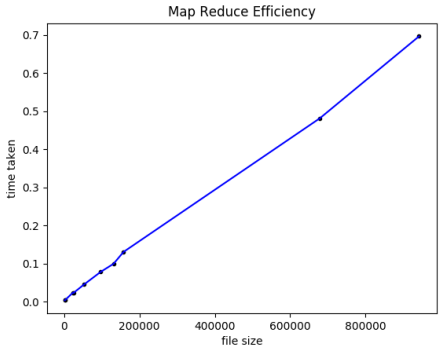
(c) Map Reduce with 4 Cores



(d) Map Reduce with 8 Cores



(e) Map Reduce with 16 Cores



(f) Map Reduce with 32 Cores

Figure 2: Experiment_2 using more than 2 cores for Map Reduce

As we can see from Figure 2 above, the efficiency of the Map Reduce increases by about 50% when we switch from 2 cores to 4 cores and about 20% when switching from 4 to 8 cores. Figures 2-e and 2-f also show that this

improvement reaches a limit which is probably determined by the capability or the architecture of the computer which the program is run on.

4 conclusion

From the experimentation above, we can conclude that Map Reduce is a really efficient technique for solving problems that can be parallelized. It significantly improve the processing time and allows to fully use the capacity of the computer. However, the improvement percentage is not proportional to the number of cores used as it decreases as the number of cores used increases. In other words, doubling the number of cores does not necessarily mean that the program will run twice as fast as it ran before.