ENSC 351

Lab 3

Map Reduce

Steven Luu (301150253)

Younghoon Jee (301177482)

Introduction

Map Reduce is a programming implementation that processes and generates data sets running in parallel. The Map Reduce is primarily consisted of two procedures: map procedure and reduce procedure. Map procedure is responsible for filtering and sorting data while reduce procedure is performing summary operation. [1] For instance, for word count part of lab 3, we created map function that read input data and generated a key-value pair. Once each map pairs with certain key and value got created, we sorted them in same pair with same key. After that, we used Reduce function to reduce pairs with same key. This will eliminate redundancy of pairs with same key.

Explain your workload, how did you come up with it and why is it faster than single-threaded implementation.

For our workload, we chose to implement a census that divides a group of people by a specified age group and by gender. After doing some research on MapReduce, we came upon a webpage that listed some simple examples of MapReduce applications [2]. From here, we decided to have our workload determine the number of males and females under the age of 50 and over the age of 50, given a list of gender and age.

Our implementation has two threads running to map the key-value pairs and two threads to reduce the key-value pairs. We chose to only run two threads as our program was created on a Windows machine with four cores and did not want to make our program too complex. First the file is read at the input stage where the gender and age are read as a pair of strings and saved in a list. Next, one thread will map every even pair and another thread will map every odd pair to key-value pairs; the key-value pairs from both threads are then combined into another list. The key of each pair is the gender and the value of the pair is the age, and our workload now starts sorting the pairs by gender and put them into two gender groups. Here we have one thread that checks a gender group for ages under 50, reduce the key-value pairs, and then repeated for the next gender group. The second thread at this step will check the gender group for ages above 50, reduce the pairs and check the next gender group again. The results of the first thread are saved in one list and the results of the second thread are saved in a second list. The two results list are now displayed by the result stage.

The multi-threaded implementation of this census program is faster than the single-threaded implementation. The map step runs fasters because we have two threads mapping the input values instead of mapping each input incrementally, and at the reduce step we have each thread only looking at their specified age range rather than checking the groups twice, once for each age group. A comparison of runtimes for multi-threaded and single-threaded version of our census program are shown in Table 1.

Comment on efficiency of MapReduce of word counts as compared to the efficiency of single-threaded implementation.

Single-threaded word count implementation is more efficient than multi-threaded word count implementation. From Table 1, we can conclude that multi-threaded word count took more than twice as much run time. In MapReduce of word count, we initially read input from text file. When program read each word, we created map pairs by putting each word into key and set the value ‘1’ indicating each word count. In the end of file, we had a list of key-value pairs with different keys with same value of ‘1’. As the map consisted of the same values from each pair, sorting needs to compare only keys to group pairs of same keys. Consequently, having multi-threaded word count makes sorting over-complicated as values of each pair are insignificant. So, multi-threaded word count does not benefit from additional complexity of the program.

Comment on which kind of workload is best suited to MapReduce

Word loads that works with a lot of data which have significant values for each are best suited for MapReduce. In our Word Count programs, we saw that the key-value pair of each word had the same value, so only important data is the number of pairs with a specific key. In this case using MapReduce makes the program more complex and increases runtimes. In our census programs we must check both the key and value of each pair because we want to group the key-value pairs by their key and by the values. Both the key and value are significant and checking them leads to prolonged run times. Applying MapReduce to our census program improved the runtime as it had more threads working to check key and values of the pairs. Additionally, MapReduce does not benefit from working with a small amount of data because there are more efficient ways of working with small amount of data than using MapReduce.

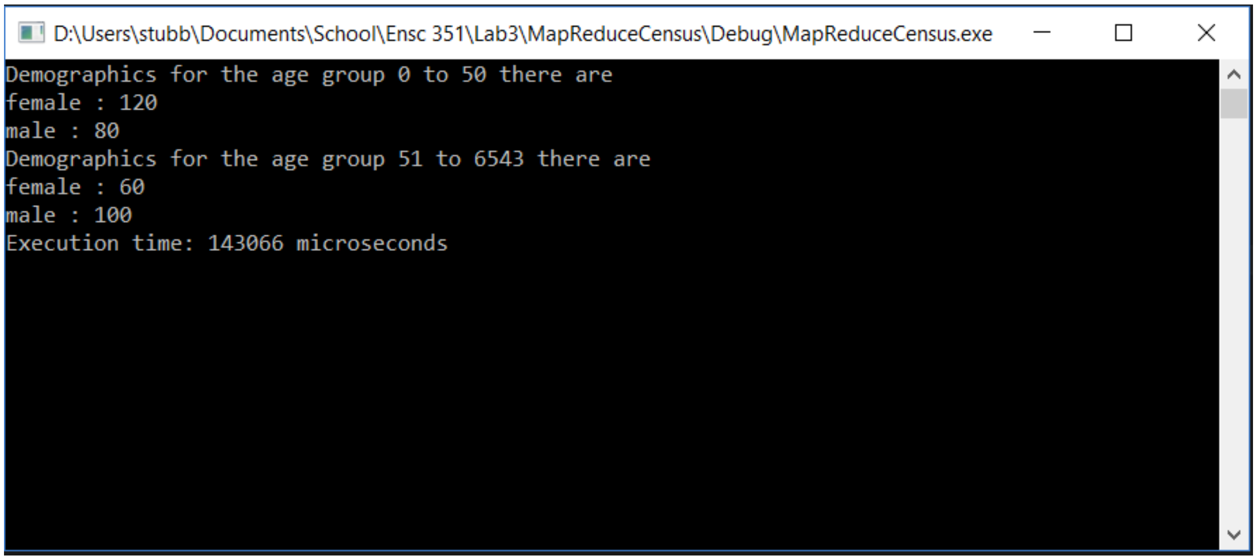
MapReduce is often run with the map phase spread across multiple computers. How would this impact the execution speed?

Running the map phase across multiple computers will improve execution speed as there are more cores available which will allow more threads to map the key-value pairs. Having more threads allows the task of mapping the key-value pairs to be further divided up between the available threads which reduced the program’s runtime. There is a limit adding threads because at a certain point the time it takes for the data to be sent between computers will exceed the improved execution time, negating the benefits.

Appendix 1

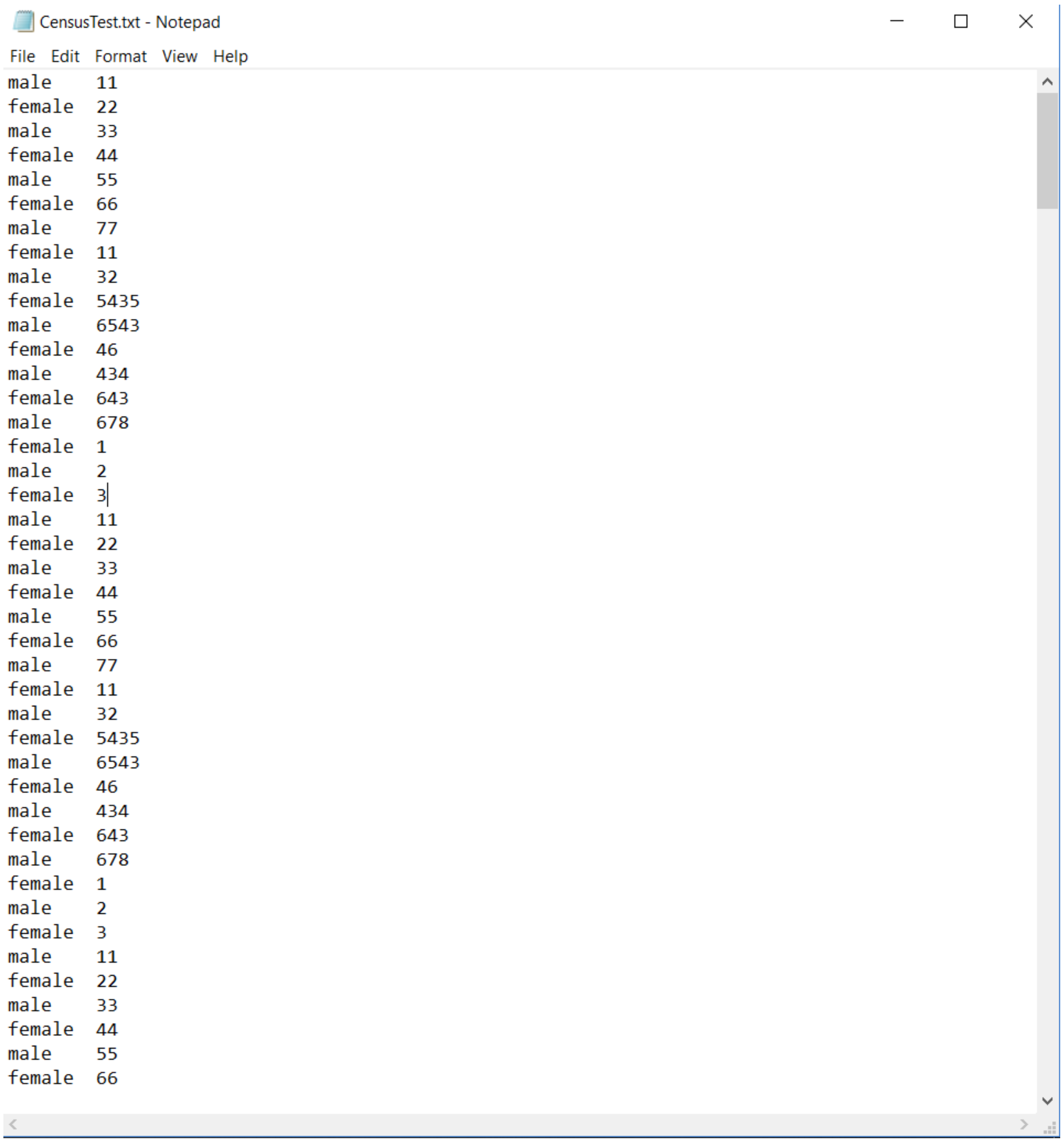
Run	Word Count Single-Threaded (404 inputs)	Word Count Multi-Threaded (404 input)	Census Single-Threaded (360 inputs)	Census Multi-Threaded (360 inputs)
1	508.038	914.414	384.317	291.99
2	413.83	1097.034	403.848	219.866
3	496.8	1120.307	388.635	258.403
4	432.796	1219.391	463.576	202.134
5	447.051	1027.205	488.569	322.319
6	451.304	975.839	331.827	293.029
7	522.481	1067.686	537.765	214.005
8	482.024	1196.179	363.302	332.065
9	441.725	981.903	372.682	237.464
10	434.514	1078.7	469.888	200.031
Average time (ms)	463.0563	1067.8658	420.4409	257.1306

Table 1. Run time of different Map Reduce implementations in milliseconds


D:\Users\stubb\Documents\School\Ensc 351\Lab3\MapReduceCensus\Debug\MapReduceCensus.exe

```
Demographics for the age group 0 to 50 there are
female : 120
male : 80
Demographics for the age group 51 to 6543 there are
female : 60
male : 100
Execution time: 143066 microseconds
```

Figure 1. Results of Census Work load



```
male    11
female  22
male    33
female  44
male    55
female  66
male    77
female  11
male    32
female  5435
male    6543
female  46
male    434
female  643
male    678
female  1
male    2
female  3
male    11
female  22
male    33
female  44
male    55
female  66
male    77
female  11
male    32
female  5435
male    6543
female  46
male    434
female  643
male    678
female  1
male    2
female  3
male    11
female  22
male    33
female  44
male    55
female  66
```

Figure 2. Input File of Census Work load

References

[1] Wikipedia, "MapReduce," [Online]. Available: https://en.wikipedia.org/wiki/MapReduce. [Accessed 12 10 2018].

[2] StackOverFlow, "Good Map Reduce Example," 4 9 2012. [Online]. Available: https://stackoverflow.com/questions/12375761/good-mapreduce-examples . [Accessed 14 10 2018].