CS434: Introduction to Parallel and Distributed Computing

Ndze'dzenyuy, Lemfon K.

April 2021

Lab 4 Final Project Report

1 Part I

1.1 MapReduce Implementation Used

For this submission, I used the Mrs MapReduce implementation of map reduce. My choice to of using MapReduce was driven by two major considerations; level of abstraction and flexibility. After exploring other implementations like Pheonix++, MapReduce-MPI, MapReduceForC and Mr Job, I realized that Mrs MapReduce had a level of abstraction that was adequate for the amount of time I had to work on the project. I initially started out working with Pheonix, but simple things like manipulating strings and other data formats were quite burdensome. Mrs MapReduce, owing on the one hand to the fact that Mrs MapReduce is implemented in Python, but more so to its user-friendly design, turned out to be very easy to use as it let me focus on the very important things while doing the other things trivially. Another dimension in which this abstraction was evident was the fact that with Mrs MapReduce had elegant documentation. As one who started out by exploring Pheonix and MapReduceForC, I came to heavily appreciate the good documentation of the MapReduce framework. In Pheonix I often had to step through the code to determine on my own what was going on, and did a lot of trial and error. On the contrary, Mrs Map Reduce let me understand at a go from the documentation what was going on, what I could do or could not do and why certain things were done in a certain way. Secondly, the fact that Mrs Map Reduce had several examples (especially the pi-estimation program) turned out to be very important, as it strenghtened my understanding of the MapReduce paradigm, while giving me a solid grasp of the Mrs MapReduce ecosystem. These, I believe, were due to the fact that Mrs MapReduce stayed true to its promised design principles; keeping it simple, reducing the burden on programmers, and making sure that developers do not repeat themselves.

I did not face any challenges with installing Mrs MapReduce. I dowloaded the tar file, simply ran the setup.py script as was specified in the documentation and I was good to go.

The code for the Wordcount program in Mrs MapReduce further demonstrated the concept of abstraction that I touched on earlier. The code was succint, barely 37 lines of code. This number of lines of code look rather small when compared to the almost 200 lines of code for the same problem when implemented in Pheonix++. As is common with other Mrs MapReduce programs, the Countword program was simply a class that extended the default Mrs MapReduce class and overwrote the run and reduce functions.

2 Part II

To solve the matrix multiplication problem in MapReduce, a similar approach as was used in the last implemented MPI program for matrix multiplication was used. The incoming matrices were divided into rows and columns, and the rows and columns were organized into key value pairs, and then the key value pairs were used to calculate the final result. The organization was done in such a way that if the desired value of P to be used is 16, 16 key value pairs will be created and then reduced to obtain the result. Each key value pair will have a number that identified it, and helps us in determining which part of the final matrix it was calculating. When the algorithm starts, we call a function that divides the first matrix into rows based on the value of P and N, divides the second matrix into columns and then pairs both into a key value pair with an id as previously discussed.

The reduction is done simply by extracting the portions of the key value pair that represent the rows from the first matrix, the part of the value that represent the columns from the second matrix, and then calculating the partial matrix that will result from this calculation. Using the key, we then compute the region of the result matrix that has been calculated and fill in into the matrix appropriately.

2.1 Discussing Results

In what follows, I present a comparison of the results obtained with some common possible combinations of N and P, and compare them to results obtained when we attempt the matrix multiplication in a serial manner.

N	P	Runtime in seconds	
		Serial Runtime	MapReduce Runtime
4	4	0.000191211	0.170
6	9	0.0002167225	0.163
6	36	0.0002167225	0.171
16	16	0.00398421	0.170
16	64	0.00398421	0.173
20	100	0.00641	0.180
28	196	0.018023	0.188
80	256	0.1373887	0.336
100	400	0.23156118	0.496
144	576	0.5862841	0.981
196	784	1.48336338	2.027

To comment on the results obtained, we must begin by realising that such an approach is not the best for comparing the performance of algorithms, as such things as the number of background applications and the power saving mechanisms of the computer can influence the amount of time that is taken to compute the result.

Howver, the results presented above give us a sense that as the size of the matrix increases, the run time of the sequential algorithm grows much faster than that of the MapReduce algorithm. This is consistent with expected results, as the advantage of using an approach like MapReduce can only be felt when the size of the data on which the operation is being caried out is considerably large.