15-440 Distributed Systems
Project 4
Hira Yasin Dhamyal

Experimentation and Analysis

Performance:

Fixed Data Set size = 40 million

Time for MPI = 93.700049 sec
Time for sequential = 600 sec
Time for Map Reduce = 3701 sec

MPI runs faster than both the MapReduce and the sequential. But what surprises me is the time for MapReduce. It is very slow even with respect to the sequential implementation. The time recorded for the Map Reduce was taken with undefined number of map tasks and 3 reduce tasks. The website showed that the number of map tasks run were 18, each taking around 3 mins, 30 secs. The number of reduce tasks that were run were 3. And each reduce task was taking time around 7 mins, 30 secs while reduce task went to 13 mins. The reduce tasks were taking a lot of time, so I thought lets increase the number of reduce tasks to 10 and then see the time for this. But to my surprise again, the time remained approximately the same which is very sad. Either it is because of something highly time consuming in my code, which I doubt, or it is because of some other reason, which we will examine more in the Analysis section. But this section concludes that MPI is faster of all and MapReduce did not give us many benefits at least with my implementation with the default settings.

Development Effort:

Hadoop MapReduce was the easiest to implement among the 3. It might be because the algorithms were the same as I had previously done and so I did not have to spend time thinking about them. But the idea was straight forward, easy to understand and quick to implement. MPI took me a lot of time to understand and error handling was really bad in it. It took me hours to find the error and fix it. This problem didn't occur in sequential or in the Hadoop implementation.

One problem that is in Hadoop is that it was difficult to debug the program if the problem arose. It is a long process to debug the code because I had to go check the output files and see the results, so it took me time to debug the code. But the errors were less for Hadoop and so it was quick to fix them as compare to the other ones.

Experience in applying MapReduce to the K-means clustering algorithm:

One of tricky part in MapReduce was to analyze what the keyIN, valueIN, keyOUT, valueOUT were supposed to be for the map and reduce tasks. The rest were the same algorithms as implemented previously. So once that was figured, it did not take me time to write it. Another important task was to figure out what files to write output to and how to do that. It took me

time to figure this out and sorting out the different functions that could be used to complete this task. Also it took me time to figure out what to change in the configuration file in order to change the map tasks and block size.

Insights concerning the performance trade-offs of MPI and MapReduce with K-Means:

MapReduce was easier and quicker to implement than MPI. But MPI produces results quicker than MapReduce in file sizes that are less for example for 1 million data points. MapReduce took a lot of time, 34 mins approx., on the default settings of the number of map, reduce tasks and the block sizes. In comparison, MPI 's time was closer to 80 secs, which is a lot better than MapReduce time.

Thoughts on the applicability of K-Means to MapReduce:

The applications of K-Means algorithm by itself are a lot and we just did 2 of them. A little search shows that it is very important for Search engines and in wireless sensor networks. So it seems very important in practical applications. When wanting to use K-Means on very large data set sizes such as when Google has to do, using MapReduce makes the K-Means algorithm fast on large data sets which means that in practical applications, the algorithm can provide the results in a short time.

Recommendations regarding the usage of MapReduce for algorithms similar to k-Means

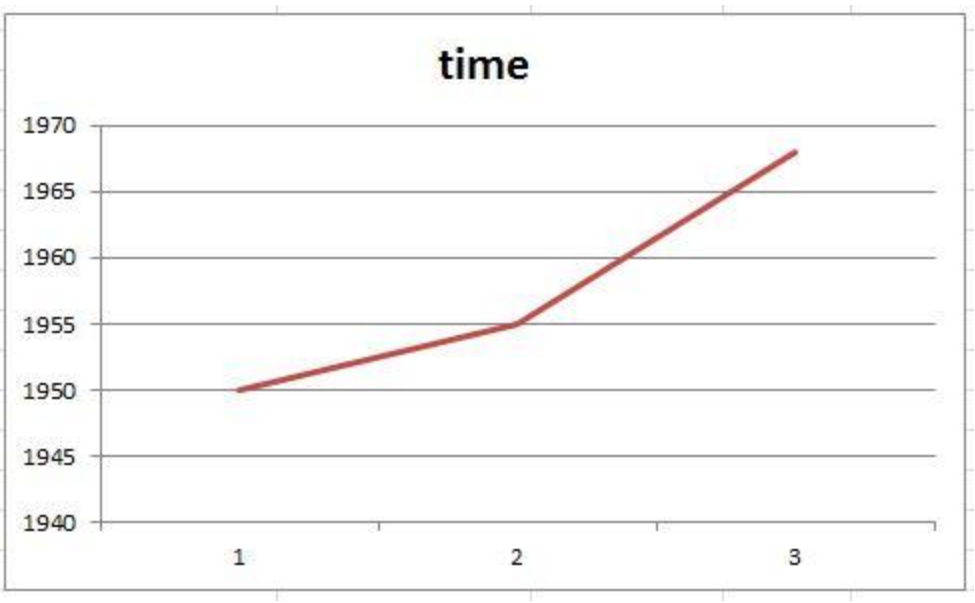
MapReduce could be used for any algorithm which involves large datasets and where work can be divided by the simultaneous work of the map tasks and the reduce tasks start before the map even finish. Any such problem would require the usage of MapReduce and can be done in less amount of time.

Analysis:

Scalability Study on Number of Map Slots per each Virtual Machine

Data Set points = 10,000,000
Data set size = 276.48 MB
HDFS Block Size = 64 MB
Virtual Machines = 4

no. of Map Slots / virtual machine	**Time (Secs)**
1	1950
2	1955
4	1968



Time is on the y-axis and number of map slots on the x axis.

1 Map Slot per VM
Per Wave = number of VMs * slots per VM = 4 * 1 = 4 Map Slots
waves = (Data Set size / HDFS block size) / per wave = 4.32 / 4 = 1 waves required.

2 Map Slot per VM
Per Wave = number of VMs * slots per VM = 4 *2 = 8Map Slots
waves = (Data Set size / HDFS block size) / per wave = 4.32 / 8 = 1 wave required.

4 Map Slot per VM
Per Wave = number of VMs * slots per VM = 4 * 4 = 16 Map Slots
waves = (Data Set size / HDFS block size) / per wave = 4.32 / 16 = 1 wave required.

With the increase in the number of Map Slots per VM, the time increased because the number of waves decreased. And as the data set size that I was working on was not that big, the increase in the number of map slots did not cause a lot of difference in the time required to finish the program. But I can say that the time increase because there was more interleaving in the processing at each virtual machine. Hence the time increased.

Scalability Study on HDFS Block Size

HDFS BLOCK SIZES	Time (Secs)
32MB	1927.5
64MB	1955
128MB	1925

Data Set Points = 10,000,000
Data set size = 276.48 MB
Virtual Machines = 4
Number of map slots = 2

32 MB HDFS block size
Number of map tasks = (Data Set size / HDFS block size) = 276.48 / 32 = 8.64
Per wave = number of VMS * number of slots = 4 * 2 = 8
Number of waves = (Number of map tasks / per wave) = 8.64 / 8 = 2

64 MB HDFS block size
Number of map tasks = (Data Set size / HDFS block size) = 276.48 / 64 = 4.32
Per wave = number of VMS * number of slots = 4 * 2 = 8
Number of waves = (Number of map tasks / per wave) = 4.32 / 8 = 1

128 MB HDFS block size
Number of map tasks = (Data Set size / HDFS block size) = 276.48 / 128 = 2.16
Per wave = number of VMS * number of slots = 4 * 2 = 8
Number of waves =(Number of map tasks / per wave) = 2.16 / 8 = 1

It is the same scenario here. The fluctuation in time is very less and that is because my data set size is too low. The number of waves do not change and so the time difference does not differ much. If the data set size was bigger like 40 million, the time would decrease. As the number of waves would decrease and this would cause less overlap between map and shuffle.

Scalability Study on the number of VMs.

Data Set Points = 10,000,000
Data set size = 276.48 MB
Number of map slots = 2
HDFS Block Size = 64 MB

number of VM	Time (Secs)
1	2431
2	2290
3	2045
4	1955

1 Virtual Machines
Number of Map tasks = (Data Set size / HDFS block size) = 276.48 / 64 = 4.32
Per wave = number of VMS * number of slots =1 * 2 =2
Number of waves =(Number of map tasks / per wave) = 4.32 / 2 = 3

2 Virtual Machines
Number of Map tasks = (Data Set size / HDFS block size) = 276.48 / 64 = 4.32
Per wave = number of VMS * number of slots = 2 * 2 =4
Number of waves =(Number of map tasks / per wave) = 4.32 / 4 = 2

3 Virtual Machines
Number of Map tasks = (Data Set size / HDFS block size) = 276.48 / 64 = 4.32
Per wave = number of VMS * number of slots = 3 * 2 =6
Number of waves = (Number of map tasks / per wave) =4.32 / 6 = 1

4 Virtual Machines
Number of Map tasks = 276.48 / 64 = 4.32
Per wave = number of VMS * number of slots = 4 * 2 =8
Number of waves =(Number of map tasks / per wave) = 4.32 / 8 = 1

The time decreases as the number of virtual machine increases because the number of waves decreases. The calculation shows why do they decrease. Also with the decrease in the number of VMs, there is less parallelism in the code and more work to be done and less number of map slots per VM as well. This causes the time to decrease

Improvement in the analysis:

The analysis can be improved if the data set size was large like 80 million points. This would show the difference significantly. Unfortunately, because of the lack of time, this was not possible. But even with less significant changes, the changes are still visible and the analysis shows why the time increases or decreases.