MATRIX OPERATION

REPORT: Performance Analysis

Prateek Kumar Singh

Introduction:

MapReduce

MapReduce is a processing method and a model for a distributed computing. Map and Reduce are two crucial jobs that make up a MapReduce algorithm. A set of data is transformed into another set through a map, where each element is separated into tuples (key/value pairs). The second work is a reduce task, which takes a map's output as input and concatenates the data tuples into a smaller collection of tuples. The reduce job is always carried out after the map job, and hence the name MapReduce.

In this Assignment we are going to perform operations on matrices using the MapReduce Technique. We are given 3 matrices namely M, N and X, these matrices are supplied to the MapReduce algorithm as text files (.txt file format). The goal is to obtain the result of X-MN.

A matrix is a rectangular grid of numbers organised into rows and columns. The term "matrix element" or "entry" refers to each individual number in a matrix. In this assessment the product of matrices M and N is a dot product, which is denoted as MN. To carry out the dot product of two matrices, the number of columns in the first matrix should be the same and equal to the number of rows of the second matrix. The first matrix's number of rows and the second matrix's number of columns are combined to form the final matrix, or the matrix product.

Algorithm:

The MapReduce programm runs in two phases: the map phase and the reduce phase.

The Map Phase:
The processing of the input data is the mapper's responsibility. The input data is stored in the Hadoop file system as a file or directory (HDFS). The mapper function receives the input file line by line. The mapper transforms the data and produces numerous tiny data bits.

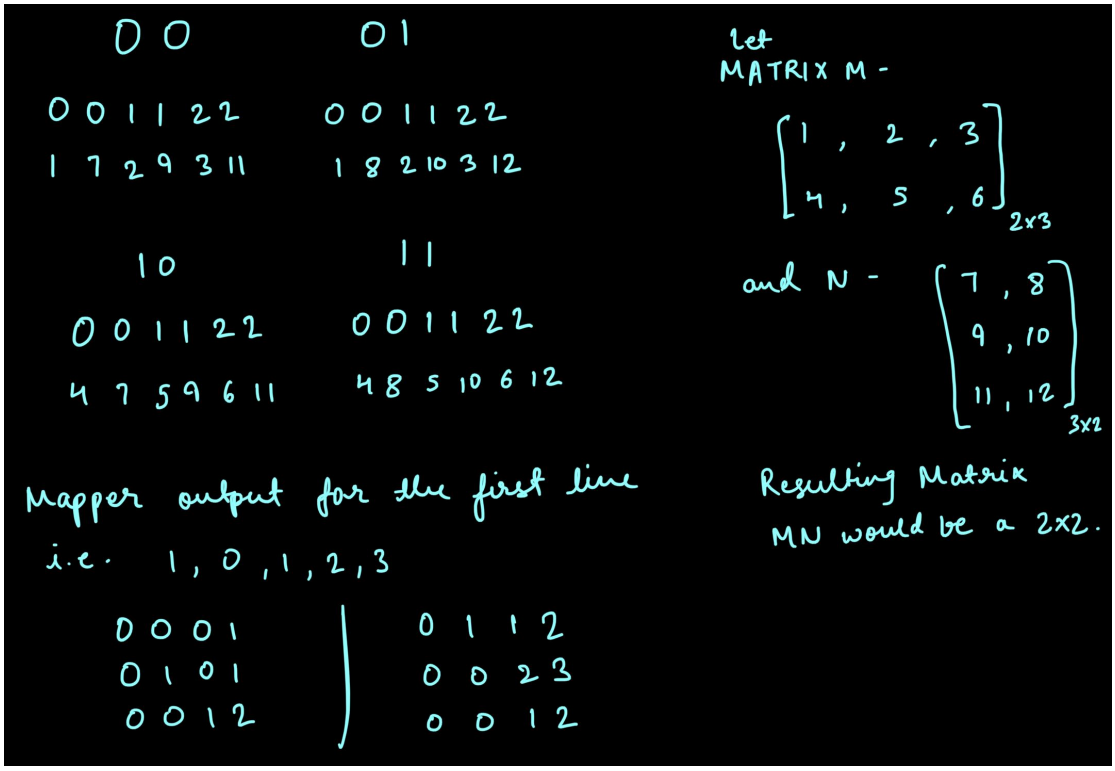

Figure 1: Intuition of Map Algorithm

The output of the mapper is a key value pair. Figure 1 give the intuition of the mapper algorithm. The mapper emits the key pairs for the input matrices M, N, and X. The key value pairs are of the format k1, k2, k3, V.

The Reduce Phase:
Reducer processes the data that arrives from the output of the mapper. After processing, it generates a fresh set of output which is kept in the HDFS. Here, the reducer's inputs are the key value pairs k1, k2, k3, V generated by the mapper. Now, to perform the expected task i.e. X-MN, the key value pairs are stored in empty dictionaries and are checked in every iteration to perform the multiply and add operations. The operations are explained by the following example.

For instance if the values V1 and V2 are consistent with all three keys i.e. k1, k2, and k3, then V1 and V2 are multiplied. k1, k2 corresponds to the indices of the resulting output matrix and therefore values with same k1, k2 are summed up to obtain the final result.
Here the values V for the X matix are multiplied with a negative one, so that it subtracts automatically when we are adding the the inputs with same k1, k2. Doing so will give us MN-X. In order to obtain the expected operation, the final result is also multiplied with a negative one and thus X-MN is obtained.

Performance Analysis:

A.

Matrix, Reducers	1	3	6	9
6x6	11510	14250	15530	18550
20x20	12650	14350	17130	20530
50x50	16280	18860	23020	26770
100x100	21550	26980	33270	37030
200x200	195280	231020	233370	194400

Figure 2: CPU Time Spent in ms.

MATRIX SIZE	Map Input Records	Map Output Records
6x6	18	468
20x20	60	16400
50x50	150	252500
100x100	300	2010000
200x200	600	16040000

Figure 3: Map I/P & O/P records for different matrix sizes.

Test results for the execution of Task 1 on M, N, and X matrices for 5 different test sizes i.e. 6x6, 20×20, 50×50, 100×100, and 200×200 with varying number of reducers i.e. 1, 3, 6, and 9.

B. Performance Based on Size of the Matrix

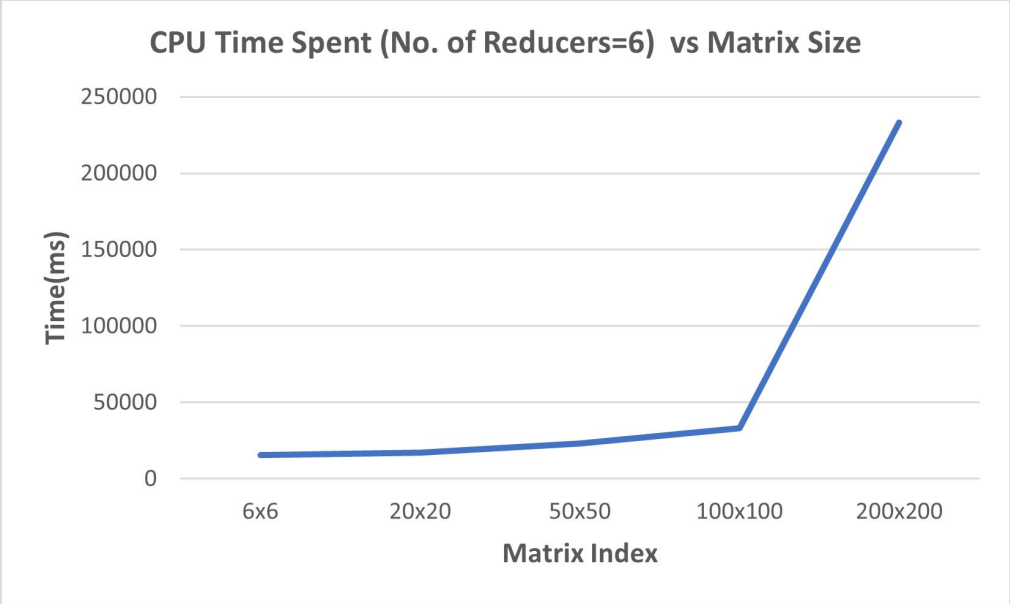

Figure 4: CPU Time(ms) for 6 reducers with different matrix sizes.

It can be observed from Figure 4, that the CPU time spent increases with the increase in the size of the matrix. This is an expected result, since the CPU has to deal with the increased number of tasks including the generation of increased number of key value pairs by the mapper and processing the increased number of inputs for the reducer.

C. Performance Based on Number of Reducers

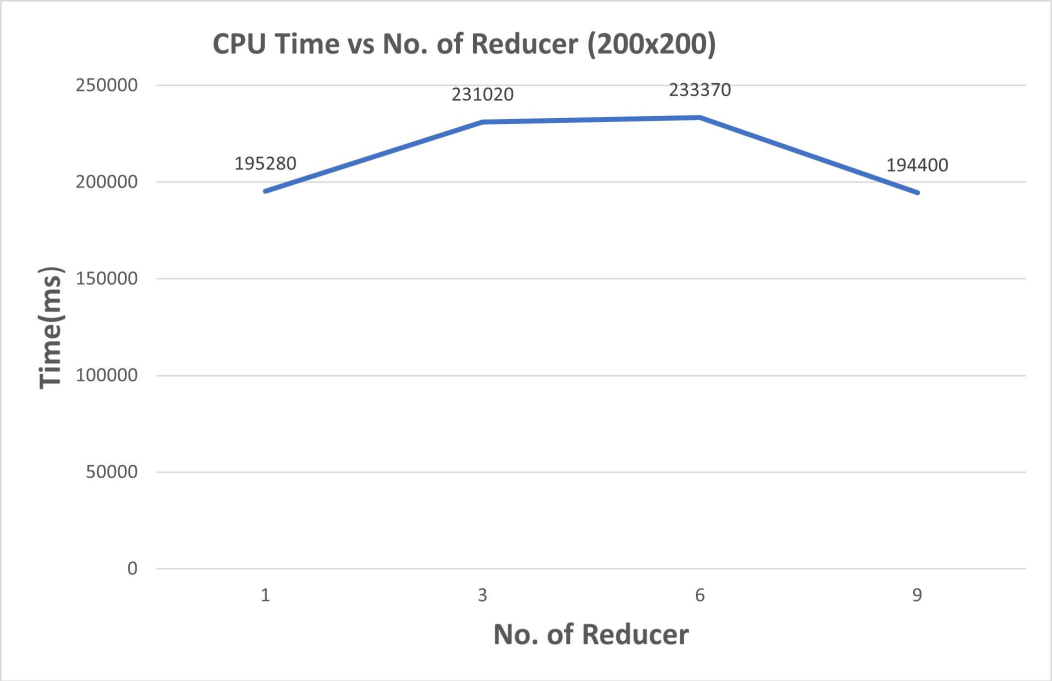

Figure 5: CPU Time(ms) for increasing reducers in 200x200 matrix.

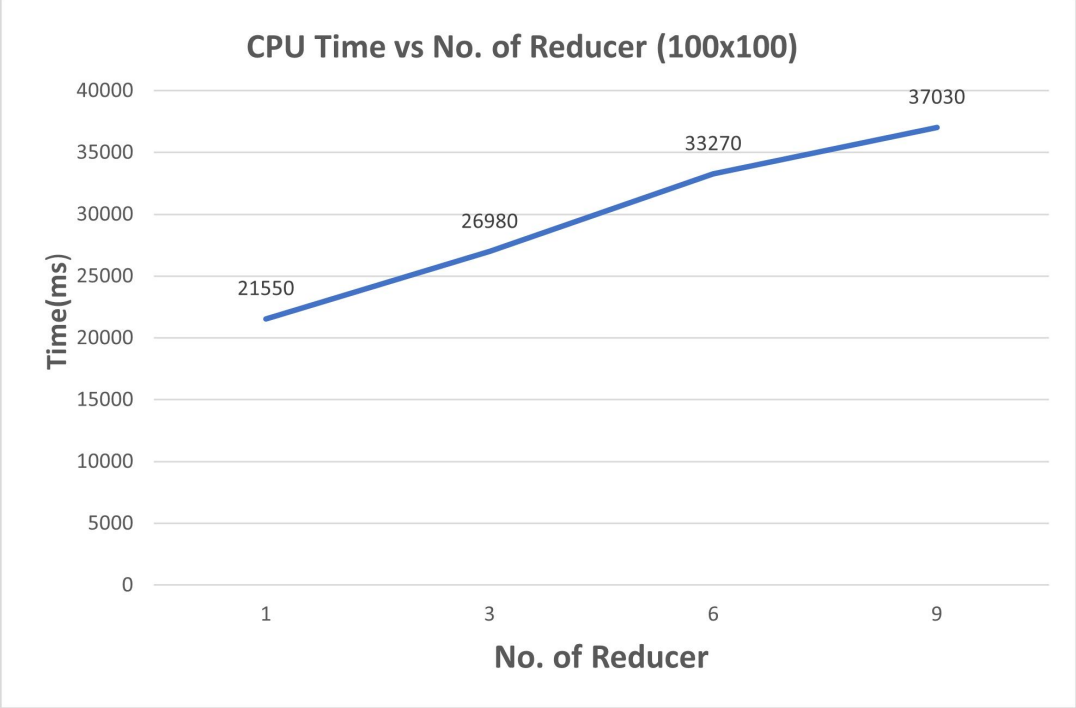

Figure 6: CPU Time(ms) for increasing reducers in 100x100 matrix.

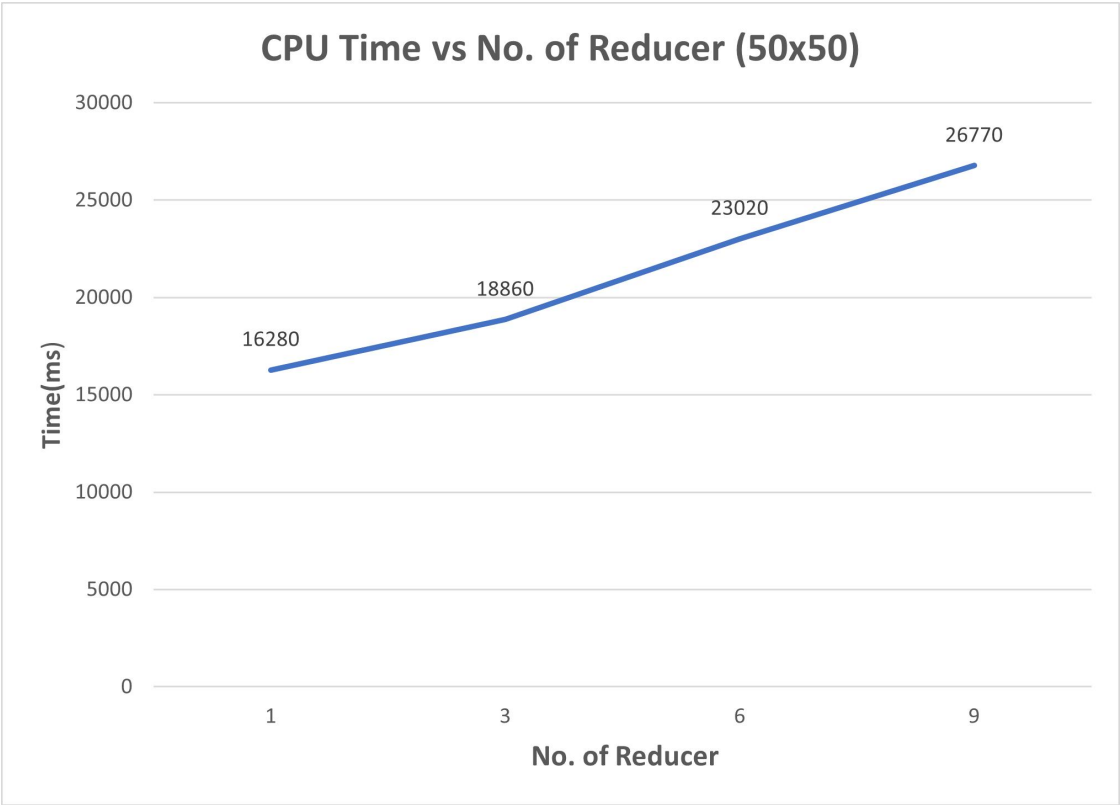

Figure 7: CPU Time(ms) for increasing reducers in 50x50 matrix.

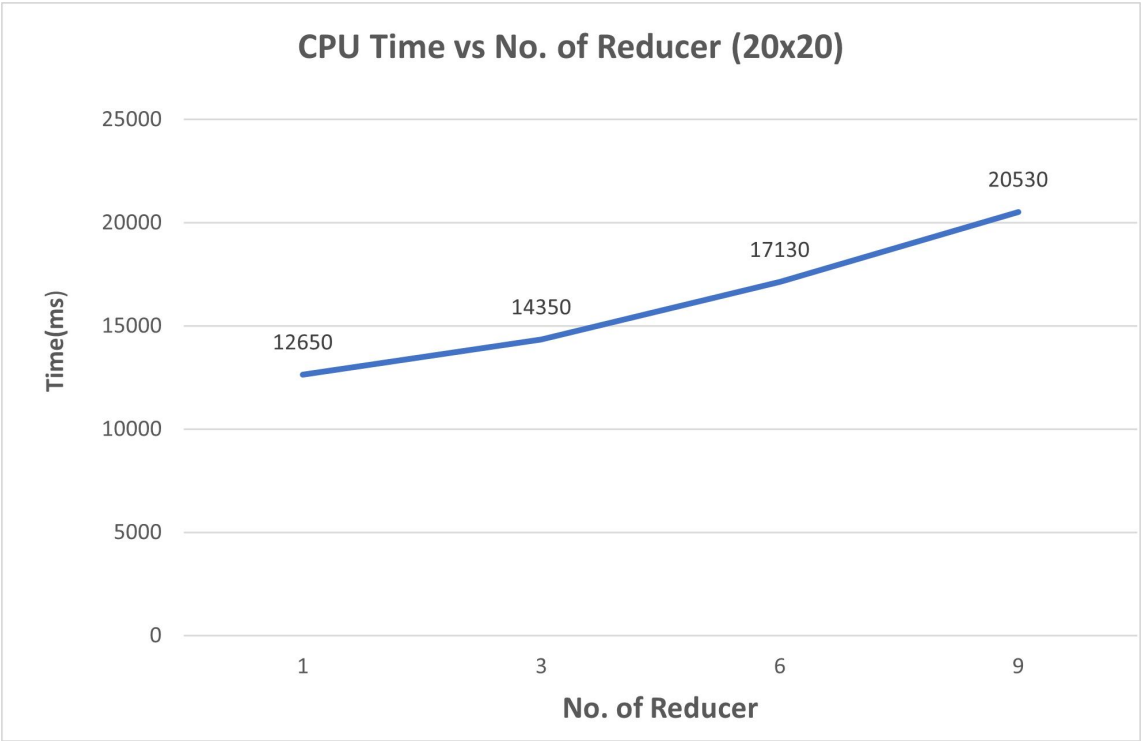

Figure 8: CPU Time(ms) for increasing reducers in 20x20 matrix.

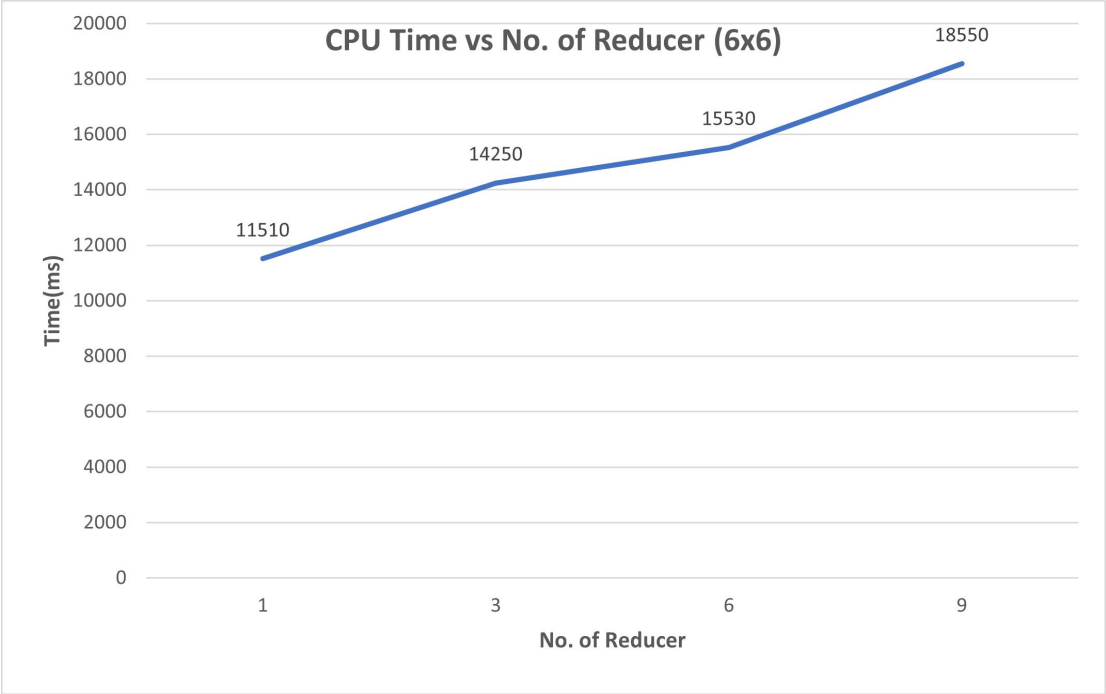

Figure 9: CPU Time(ms) for increasing reducers in 6x6 matrix.

As observed from the figures above, the trend is same with all the matrices i.e. the CPU time increases or performance decreases with the increase in the number of reducers. The reason for the trend is explicable, since each reduce task need to start up and be created/instantiated in the nodes, which result in an increase of startup time. Additionally, data must be divided among all reducers, which increases the amount of time needed for network transfers and processing.

So, to answer the question that adding more reducers always improve performance. The answer is no, as evident from the tests above. Since too many reducers will produce too many little HDFS (output) files, which is undesirable and will strain HDFS due to the necessary housekeeping and thus affecting the performance negatively.