CSP -554 Project Report
Ramesh Chandra Reddy Musilanna – A20521309
Department of Computer Science
Illinois Institute of Technology, Chicago.

Project Title:
Dot Product of Vectors, Matrixes using Big Data Technologies

Project Statement:

Matrix/Vector multiplication is the one of the most fundamental operations that most of the machine learning algorithms rely on. Knowing the working of matrix multiplication in a distributed system provides important insights on understanding the cost of our algorithms.

Proposed Solution:

Naïve Matrix Multiplication:

Suppose *A* and *B* are two matrices with dimensions respectively *(am,an)* and *(bm, bn)* and that you computing A * B.

1. You can perform the matrix multiplication if and only if a*n = bm*. In other words, you can only perform if the number of columns of the left matrix is equal to the number of rows of the right matrix.
2. The result of the multiplication is a matrix with dimensions *(am, bn)*.
3. Each entry of the new matrix will be the sum of the product of the corresponding row in A and column in B.

we repeat the same process for every row of the left matrix and every column of the right one.

Algorithm Pseudocode:

```
Data: A[am][an], B[bn][bm]
Result: R[][]={0}
```

```
If an == bn then
    For m=0; m<am,m++ do
        For r=0; r<bm;r++ do
            For k=0;k<bn;k++ do
                Q[m][r] += A[m][k] * B[k][r];
            End
        End
    End
End
```

Time Complexity Analysis:

The naive matrix multiplication contains three nested loops. For each iteration of the outer loop, the total number of the runs in the inner loops would be equivalent to the length of the matrix. Here, integer operations take O(1) time. In general, if the length of the matrix is N, the total time complexity would be $O(N^3)$

Matrix Multiplication using Map Reduce:

Before working on the code to compute multiplication using map reduce, we need to understand the data representation.
First column represents the matrix name of which the value belongs.
Second column represents the row of matrix in which the value belongs.
Third column represents the column of matrix in which the value belongs
Fourth column represents the value.

Mapper Function:
The mapper will read the matrixes and to differentiate them we keep a count of line number we are reading. We keep a key value, which represents the elements that need to be multiplied and the elements that need to summed.

{0} {1} {2} are the part of key and {3} is the value.

$$\begin{array}{ccc} A & B & A * B \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} & \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} & = \begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix} \end{array}$$

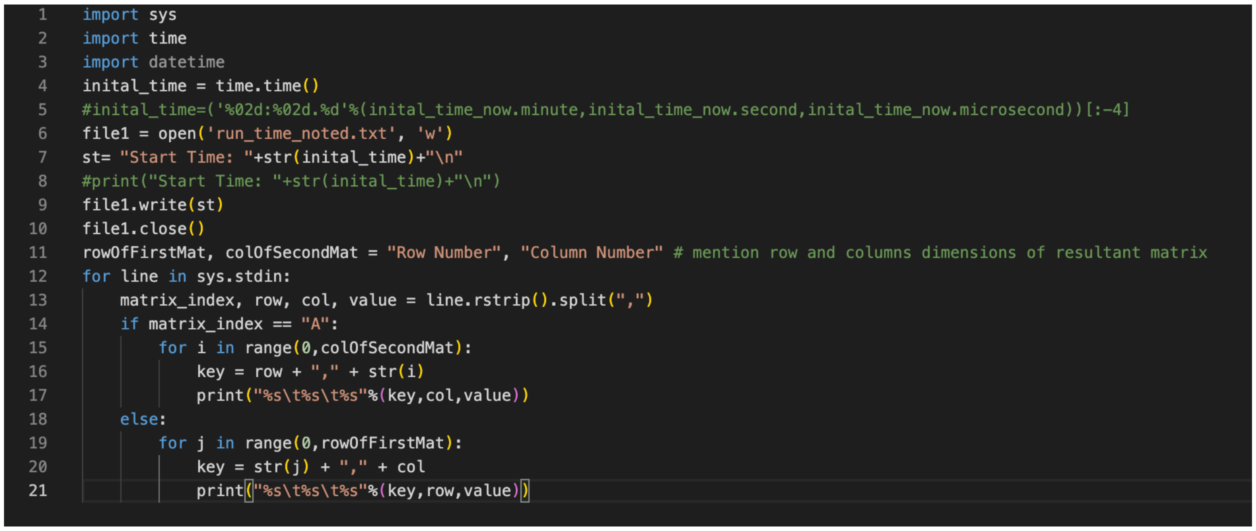
{0} {1} {2} actually represents the position of element from A or B to A*B

- {0} is the row position of the element
- {1} is the column position of the element
- {2} is the position of the element in addition. (like 1, 6 are at position 0 in addition and 2,5 are at position 1)

Elements with same key will get multiplied and the elements with same first two numbers of the key are part of sum in same row.

After mapper produces its result, Hadoop will sort the mapper results by key and provide it as an input to Reducer Function.

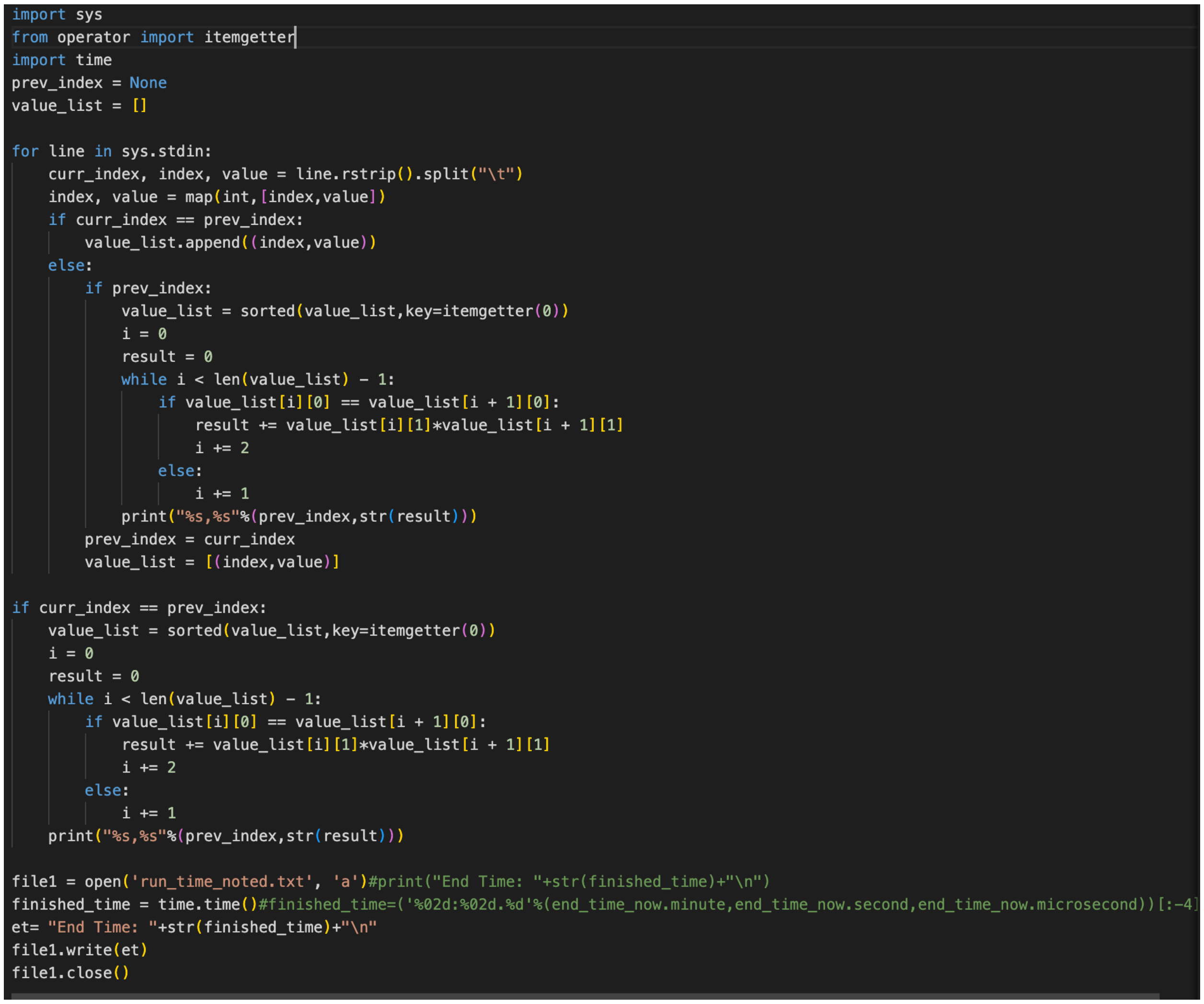
Mapper Code:



```python
import sys
import time
import datetime
inital_time = time.time()
#inital_time=('%02d:%02d.%d'%(inital_time_now.minute,inital_time_now.second,inital_time_now.microsecond))[:-4]
file1 = open('run_time_noted.txt', 'w')
st= "Start Time: "+str(inital_time)+"\n"
#print("Start Time: "+str(inital_time)+"\n")
file1.write(st)
file1.close()
rowOfFirstMat, colOfSecondMat = "Row Number", "Column Number" # mention row and columns dimensions of resultant matrix
for line in sys.stdin:
    matrix_index, row, col, value = line.rstrip().split(",")
    if matrix_index == "A":
        for i in range(0,colOfSecondMat):
            key = row + "," + str(i)
            print("%s\t%s\t%s"%(key,col,value))
    else:
        for j in range(0,rowOfFirstMat):
            key = str(j) + "," + col
            print("%s\t%s\t%s"%(key,row,value))
```

Reducer Function:

The Reducer function does the multiplication of the values with same key and sums up the values.



```python
import sys
from operator import itemgetter
import time
prev_index = None
value_list = []

for line in sys.stdin:
    curr_index, index, value = line.rstrip().split("\t")
    index, value = map(int,[index,value])
    if curr_index == prev_index:
        value_list.append((index,value))
    else:
        if prev_index:
            value_list = sorted(value_list,key=itemgetter(0))
            i = 0
            result = 0
            while i < len(value_list) - 1:
                if value_list[i][0] == value_list[i + 1][0]:
                    result += value_list[i][1]*value_list[i + 1][1]
                    i += 2
                else:
                    i += 1
            print("%s,%s"%(prev_index,str(result)))
        prev_index = curr_index
        value_list = [(index,value)]

if curr_index == prev_index:
    value_list = sorted(value_list,key=itemgetter(0))
    i = 0
    result = 0
    while i < len(value_list) - 1:
        if value_list[i][0] == value_list[i + 1][0]:
            result += value_list[i][1]*value_list[i + 1][1]
            i += 2
        else:
            i += 1
    print("%s,%s"%(prev_index,str(result)))

file1 = open('run_time_noted.txt', 'a')#print("End Time: "+str(finished_time)+"\n")
finished_time = time.time()#finished_time=('%02d:%02d.%d'%(end_time_now.minute,end_time_now.second,end_time_now.microsecond))[:-4]
et= "End Time: "+str(finished_time)+"\n"
file1.write(et)
file1.close()
```

Time Complexity Analysis:

Time complexity of Map function: O(1).
Time complexity of Reduce function: $O(N^2)$
Assuming the number of clusters is K. then overall Time complexity is $O((N^2)/k)$.

Results and Analysis:

I have implemented few mapper files with respect to sample size (matrix size).
And to note the time differences I have created a text file to note the start and end time every time we run the program.

Reference:

https://en.wikipedia.org/wiki/Matrix_multiplication#Outer_product
https://www.youtube.com/watch?v=c3loR2znLDI
https://www.geeksforgeeks.org/matrix-multiplication-with-1-mapreduce-step/
https://datascienceguide.github.io/map-reduce
http://infolab.stanford.edu/~ullman/mmds/ch2n.pdf