Name of the student:	Tanmay Prashant Rane	**Roll No.**	8031
Practical Number:	4	**Date of Practical:**	
Relevant CO's	**At the end of the course students will be able to use tools like hadoop and NoSQL to solve big data related problems.**		
Sign here to indicate that you have read all the relevant material provided before attempting this practical		**Sign:**	

Practical grading using Rubrics

Indicator	**Very Poor**	**Poor**	**Average**	**Good**	**Excellent**
Timeline (2)	More than a session late (0)	NA	NA	NA	Early or on time (2)
Code design (2)	N/A	Very poor code design with no comments and indentation(0.5)	Poor code design with very comments and indentation (1)	Design with good coding standards (1.5)	Accurate design with better coding satndards (2)
Performance (4)	Unable to perform the experiment (0)	Able to partially perform the experiment (1)	Able to perform the experiment for certain use cases (2)	Able to perform the experiment considering most of the use cases (3)	Able to perform the experiment considering all use cases (4)
Postlab (2)	No Execution(0)	N/A	Partially Executed (1)	N/A	Fully Executed (2)

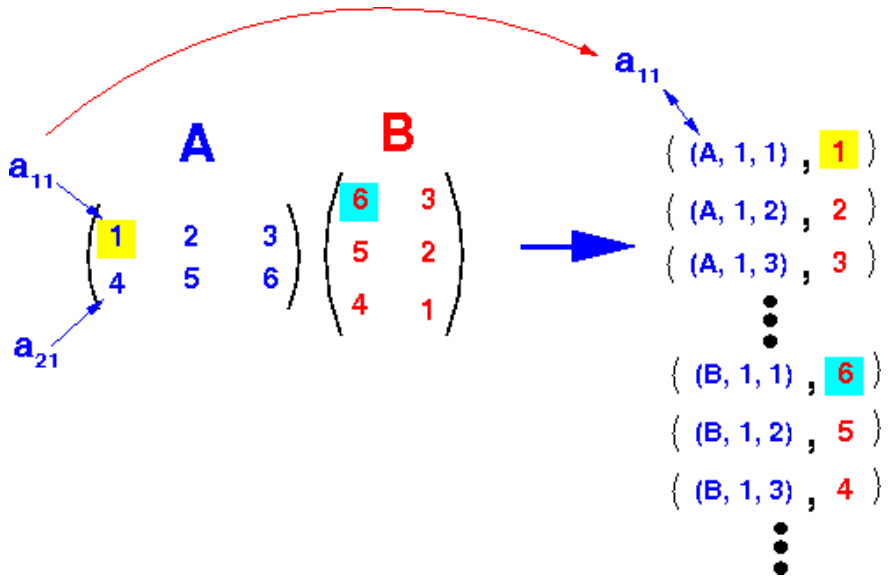
Total Marks (10)	**Sign of instructor with date**

Practical

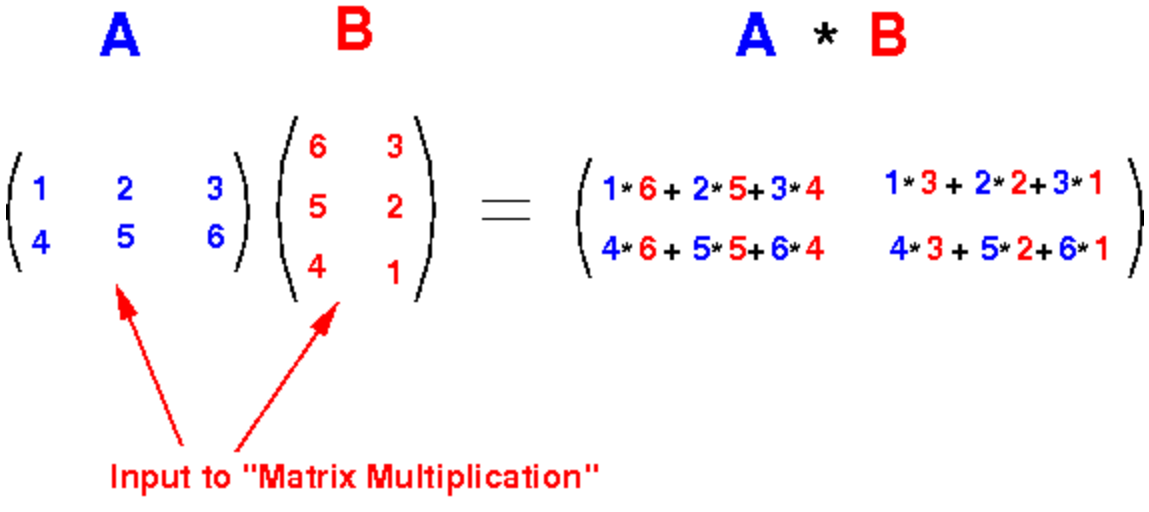
Course title: Big Data Analytics
Course term: 2019-2020
Instructor name: Saurabh Kulkarni

Problem Statement: Perform matrix multiplication using one step map-reduce

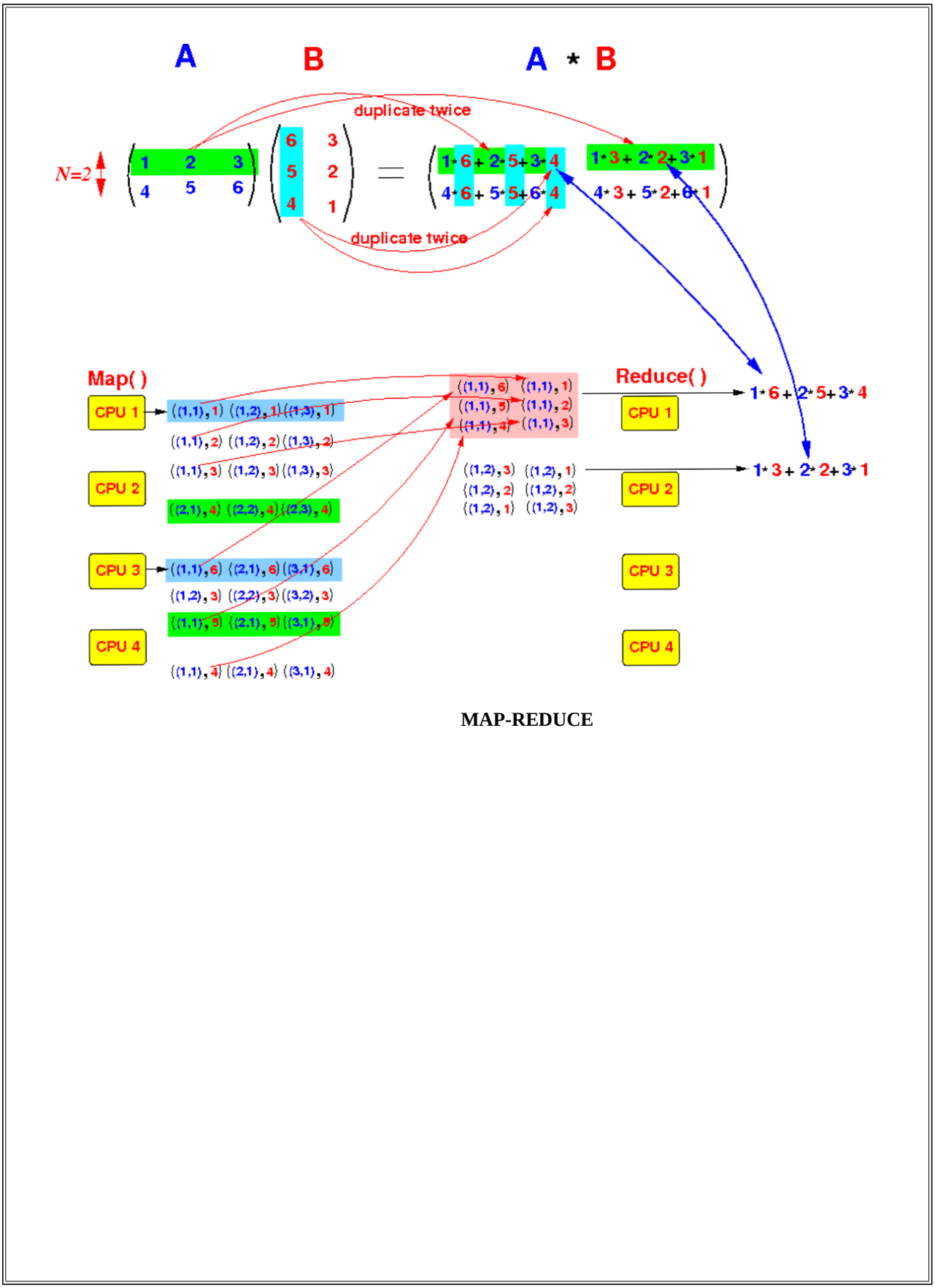
Theory:Explain the concept of matrix multiplication using one step map-reduce with the help of an example



MAP



INPUT


A
B
A * B
duplicate twice
N=2
duplicate twice
Map( )
Reduce( )
CPU 1
CPU 2
CPU 3
CPU 4

MAP-REDUCE

Code:

code for mapper:

Code for Reducer:

Code for Driver Class:

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


public class Map extends Mapper<LongWritable, Text, Text, Text>
{

        public void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException
        {
                //Create a configuration object using context
                Configuration conf = context.getConfiguration();

                // Consider a matrix of size m*n and other matrix is of size n*p. Get the values of m and p using
get() method of configuration object
                int m = Integer.parseInt(conf.get("m"));
    int p = Integer.parseInt(conf.get("p"));

    //convert Text value to string
    String line = value.toString();

    // Split each line into tokens separated by comma.
    String[] indicesAndValue = line.split(",");

    // Define outputkey of type Text()
    Text outputKey = new Text();

    //Define outputvalue of type Text()
    Text outputValue = new Text();

    //If first token is A
    if (indicesAndValue[0].equals("A"))
    {

        // Vary k from 0 to no. of columns-1 of 2nd matrix
        for (int k=0; k < p; k++) {
                //set outputkey as tokens[1],k i.e.i,k
                outputKey.set(indicesAndValue[1] + "," + k);
                // set outputkeyvalue as A,tokens[2],tokens[3] i.e. A,j,mij
```

```
outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2] + "," + indicesAndValue[3]);
              //outputValue
              context.write(outputKey, outputValue);

       }
            }

            // Vary i from 0 to no.of rows-1 of 1st matrix
           for (int i=0; i < m; i++)
           {
             //set outputkey as i,tokens[2] i.e.i,k
              outputKey.set(i + "," + indicesAndValue[2]);

              //set outputkey value B,toknes[1],tokens[3] i.e. B,j,nkj
              outputValue.set("N," + indicesAndValue[1] + ","
                      + indicesAndValue[3]);
              // write key and value to context
              context.write(outputKey, outputValue);
           }
}
}
```

//REDUCER

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Reduce extends Reducer<Text, Text, Text, Text> {

            public void reduce(Text key, Iterable<Text> values, Context context)
                                    throws IOException, InterruptedException {
                        // process values

                        // define value as String type array
                        String[] value;
                        //create 2 hashmaps (hashA,hasB) of Integer,Float to store values from A and B matrix
                        HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
       HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
       //for each value in values
       for (Text val : values) {
                // convert each value to string as it is Text and split it by comma. Store this in value i.e. string type array
defined above.
            value = val.toString().split(",");
               //if value[0] is A then
            if(value[0].equals("A")) {
                        hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            }
                        //put value[1] i.e.j and value[2] i.e. mij in hashA
```

```
else
            {
                        hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            }
      }
                        //else put value[1] i.e.j and value[2] i.e. njk in hashB

      // take value of n which is common in both the matrices here from context object
      int n = Integer.parseInt(context.getConfiguration().get("n"));
      // define result of type float and assign value 0 to it.
      float result = 0.0f;
      float m_ij;// define a_ij of type float
      float n_jk;// define b_jk of type float

      //define a loop variable j and iterate till it is less than n
      for (int j = 0; j < n; j++)
      {
          // check if value exists for a key j in hashA. if yes assign it to a_ij else assign 0 to a_ij.
          m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
          // check if value exists for a key j in hashB. if yes assign it to b_jk else assign 0 to b_jk.
         n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
       // multiply a_ij with b_jk and add this product to result which is of type float and declared above.
         result += m_ij * n_jk;

      }
      // if value of result is not 0 then
      if (result != 0.0f)
      {
          // write key,value to context. key is null and value is (i,k,result)
          context.write(null, new Text(key.toString() + "," + Float.toString(result)));
      }
      }
            }
//DRIVER

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MatDriver {

            public static void main(String[] args) throws Exception {
                        Configuration conf = new Configuration();
      // A is an m-by-n matrix; B is an n-by-p matrix.
      conf.set("m", "3");
      conf.set("n", "3");
      conf.set("p", "3");
      @SuppressWarnings("deprecation")
          Job job = new Job(conf, "MatrixMatrixMultiplicationOneStep");
      job.setJarByClass(MatDriver.class);
      job.setOutputKeyClass(Text.class);
```

```
job.setOutputValueClass(Text.class);

        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path("/media/tanmay/Data/SEM-8/BDA/EXP4/2by2data"));
        FileOutputFormat.setOutputPath(job, new Path("/media/tanmay/Data/SEM-8/BDA/EXP4/MatOut3"));
        job.waitForCompletion(true);

}

}
```

PostLab:

1. Generate a 100x100 matrix in the format that above map-reduce code understands using su programming language
2. Compute execution time for a 100x100 matrix using suitable APIs

Code for postlab question

```
//DRIVER FOR 100 X 100 MATRIX MULTIPLICATION

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MatDriver {

        public static void main(String[] args) throws Exception {
                Configuration conf = new Configuration();
    // A is an m-by-n matrix; B is an n-by-p matrix.
                long startTime = System.nanoTime();
    conf.set("m", "100");
    conf.set("n", "100");
    conf.set("p", "100");
    @SuppressWarnings("deprecation")
        Job job = new Job(conf, "MatrixMatrixMultiplicationOneStep");
    job.setJarByClass(MatDriver.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path("/media/tanmay/Data/SEM-8/BDA/EXP4/post_input_1"));
    FileOutputFormat.setOutputPath(job, new Path("/media/tanmay/Data/SEM-8/BDA/EXP4/postout"));
    if (job.waitForCompletion(true)) {
        long endTime = System.nanoTime();
        long timeElapsed = endTime - startTime;
                System.out.println("Execution time in milliseconds  : " + timeElapsed/1000000);
                        System.out.println("Job Completed");
                        return;
                }


}

}

//EXECUTION TIME : 15136 ms ~ 15.136 s
```