Name of the student:	*Yash Sankpal*	**Roll No.**	*8695*
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: 2021-2022
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

*For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. Thus, the matrix dimensions can be given as A: i * j and B: j * k.*
For applying map-reduce paradigm to matrix multiplication, the rows from matrix A need to be grouped with columns of matrix B.
Thus, in mapper, we form key-value pairs as (i, k) -> (A, j, val) where for matrix A, k loops in range of number of columns in B and for matrix B, k loops in range of number of rows in A.
Thus, for matrix A, we get pair i, k where for each row i, the reducer can work with k columns of B and for matrix B, we get pair i, k to correspond with row-wise pairs for A.
For eg:
A: 1 3 B: 4 5
4 6 7 8
After mapper, we get:
A: (1,1) -> (A, 1, 1)
(1,1) -> (A, 2, 3)
(2, 1) -> (A, 1, 4)
(2, 1) -> (A, 2, 6)
(1, 2) -> (A, 1, 1)
(1, 2) -> (A, 2, 3)
(2, 2) -> (A, 1, 4)
(2, 2) -> (A, 2, 6)
B: (1,1) -> (B, 1, 4)
(1, 1) -> (B, 2, 7)
(1,2) -> (B, 1, 5)
(1, 2) -> (B, 2, 8)
(2, 1) -> (B, 1, 4)
(2, 1) -> (B, 2, 7)
(2, 2) -> (B, 1, 5)
(2, 2) -> (B, 2, 8)
In reducer, we make:
(1, 1) -> Alist: [[A,1,1], [A,2,3]]
Blist: [[B,1,4], [B,2,7]]
…(for all keys)
Here, we multiply values with same j value and sum all products to get result matrix value for location (i, k)

Code:

code for mapper:

```
package exp4;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Mapper4 extends Mapper<LongWritable, Text, Text, Text> {

  public void map(LongWritable ikey, Text ivalue, Context context) throws IOException, InterruptedException {
    Configuration conf = context.getConfiguration();

    int m = Integer.parseInt(conf.get("m"));
    int p = Integer.parseInt(conf.get("p"));

    String str = ivalue.toString();
    String st[] = str.split(",");

    Text outputKey = new Text();
    Text outputValue = new Text();

    if(st[0] == "A") {
      for(int i=0;i<p;i++) {
        //set outputkey as tokens[1],k i.e.i,k
        outputKey.set(st[1]+","+i);
        // set outputkeyvalue as A,tokens[2],tokens[3] i.e. A,j,mij
        outputValue.set(st[0]+","+st[2]+","+st[3]);
        context.write(outputKey,outputValue);
      }
    }
    else {
      for(int i=0;i<m;i++) {
        //set outputkey as i,tokens[2] i.e.i,k
        outputKey.set(i+","+st[2]);
        //set outputkey value B,toknes[1],tokens[3] i.e. B,j,nkj
        outputValue.set(st[0]+","+st[1]+","+st[3]);
        context.write(outputKey,outputValue);
      }
    }

  }

}
```

Code for Reducer:

```
package exp4;

import java.io.IOException;
import java.util.HashMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Reducer4 extends Reducer<Text, Text, Text, Text> {

  public void reduce(Text _key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
    // process values
    // process values
    // define value as String type array
    String[] value;

    //create 2 hasmaps (hashA,hashB) of Integer,Float to store values from A and B matrix
    HashMap<Integer,Float> aStore = new HashMap<Integer,Float>();
    HashMap<Integer,Float> bStore = new HashMap<Integer,Float>();


    //for each value in values
    for(Text i: 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 = i.toString().split(",");

      //if value[0] is A then
      if(value[0].equals("A")) {
        //put value[1] i.e.j and value[2] i.e. mij in hashA
        aStore.put(Integer.parseInt(value[1]),Float.parseFloat(value[2]));
      }
      else {
        //else put value[1] i.e.j and value[2] i.e. njk in hashB
        bStore.put(Integer.parseInt(value[1]),Float.parseFloat(value[2]));
      }
    }

    Configuration conf = context.getConfiguration();

    // take value of n which is common in both the matrices here from context object
    int n = Integer.parseInt(conf.get("n"));


    // define result of type float and assign value 0 to it.
    float result = 0.0f;

    // define a_ij of type float
    float a_ij;

    // define b_jk of type float
    float b_jk;

    //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.
      a_ij = aStore.containsKey(j) ? aStore.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.
      b_jk = bStore.containsKey(j) ? bStore.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 += a_ij * b_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)));
    }
  }
}
```

```
package exp4;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver4 {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        conf.set("m", "3");
        conf.set("n", "3");
        conf.set("p", "3");
        Job job = Job.getInstance(conf, "MatrixMultiplication");
        job.setJarByClass(Driver4.class);
        // TODO: specify a mapper
        job.setMapperClass(Mapper4.class);
        // TODO: specify a reducer
        job.setReducerClass(Reducer4.class);

        // TODO: specify output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // TODO: specify input and output DIRECTORIES (not files)
        FileInputFormat.setInputPaths(job, new Path("hdfs://localhost:9000/input/exp4_postlab"));
        FileOutputFormat.setOutputPath(job, new Path("hdfs://localhost:9000/output/exp4__"));

        System.out.println(job.waitForCompletion(true));
    }

}
```

PostLab:

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

Code for postlab question

```java
package exp4_postlab;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver4 {

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("m", "100");
    conf.set("n", "100");
    conf.set("p", "100");
    Job job = Job.getInstance(conf, "JobName");
    job.setJarByClass(Driver4.class);
    // TODO: specify a mapper
    job.setMapperClass(Mapper4.class);
    // TODO: specify a reducer
    job.setReducerClass(Reducer4.class);

    // TODO: specify output types
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // TODO: specify input and output DIRECTORIES (not files)
    FileInputFormat.setInputPaths(job, new Path("hdfs://localhost:9000/input/exp4_postlab"));
    FileOutputFormat.setOutputPath(job, new Path("hdfs://localhost:9000/output/exp4_postlab"));

    long start = System.nanoTime();
    System.out.println("Execution started at: "+Long.toString(start)+" : "+java.time.LocalTime.now());

    System.out.println("MapReduce job successful "+job.waitForCompletion(true));

    long end = System.nanoTime();
    System.out.println("Execution started at: "+Long.toString(end)+" : "+java.time.LocalTime.now());
    long elapsed = end-start;
    System.out.println("Time elapsed: "+Long.toString(elapsed)+"ns");
  }

}
```

Execution started at: 1542792734619500 : 11:42:58.288
MapReduce job successful true
Execution started at: 1542801765783800 : 11:43:07.277
Time elapsed: 9031164300ns