Exp-02: Develop a MapReduce program to implement matrix multiplications

Step 1: Set Up Hadoop (Single Node)

1. **Install Java:** Ensure Java is installed (java -version).
2. **Download Hadoop:** Get the latest version of Hadoop from Apache Hadoop Official Website.
3. **Extract and Configure:**
   - Extract Hadoop to a directory (/usr/local/hadoop).
   - Edit configuration files (core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml) as needed.
4. **Format HDFS:**

   hdfs namenode -format
5. **Start Hadoop Services:** start-all.sh OR

   start-dfs.sh

   start-yarn.sh
6. **Check the services are running and up :** jps

Step 2: Set the MapRed classpath to the dfs :

Type the following command in terminal and hit enter.

mapred classpath.

Copy the output of above and paste in the below command.

export CLASSPATH="paste here class path output"

hit enter, classpath successfully set.

Step 3: Editing the mapreduce java program. (use gedit or nano **)**

1. Type the following command in terminal
   gedit MatrixMultiplication.java
2. copy the below java code and save the file :

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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;

import java.io.IOException;
import java.util.ArrayList;

public class MatrixMultiplication {

    public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] tokens = value.toString().split(",");
            String matrixName = tokens[0];
            int row = Integer.parseInt(tokens[1]);
            int col = Integer.parseInt(tokens[2]);
            int val = Integer.parseInt(tokens[3]);

            if (matrixName.equals("A")) {
                for (int k = 0; k < 2; k++) {  // Assuming B has 2 columns
                    context.write(new Text(row + "," + k), new Text("A," + col + "," + val));
                }
            } else if (matrixName.equals("B")) {
                for (int i = 0; i < 2; i++) {  // Assuming A has 2 rows
                    context.write(new Text(i + "," + col), new Text("B," + row + "," + val));
                }
            }
        }
    }

    public static class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
            ArrayList<int[]> aElements = new ArrayList<>();
            ArrayList<int[]> bElements = new ArrayList<>();

            for (Text val : values) {
                String[] tokens = val.toString().split(",");
                int[] pair = {Integer.parseInt(tokens[1]), Integer.parseInt(tokens[2])};

                if (tokens[0].equals("A")) {
                    aElements.add(pair);
                } else {
                    bElements.add(pair);
                }
            }

            int sum = 0;
            for (int[] a : aElements) {
                for (int[] b : bElements) {
```

```
            if (a[0] == b[0]) {  // Multiply only if column index matches row index
              sum += a[1] * b[1];
            }
          }
        }

        context.write(key, new IntWritable(sum));
      }
    }

    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Matrix Multiplication");
      job.setJarByClass(MatrixMultiplication.class);
      job.setMapperClass(MatrixMapper.class);
      job.setReducerClass(MatrixReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
      FileOutputFormat.setOutputPath(job, new Path(args[1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

3. Compile and execute the code
   javac MatrixMultiplication.java
4. Now go to the **FILES** , there we can see java code file and 3 setup files (Driver class, mapper class and reducer class)

Step 4 : Creating the jar file.

1. Create new folder and name it as MM, move all 3 setup files into MM folder (home/MM/…)
2. Go back to terminal , type & execute the below command to create jar file
   jar -cvf MM.jar C MM /.
   Added manifest – means jar file created successfully. Go to FILE and check MM.jar file is created.

Step 5: Prepare Input Data File.

- gedit matrix.txt enter the values to the text file and save.
- Store matrices as text files in HDFS. **hdfs dfs -put matrix.txt /Input**
- Format: **Matrix (m x n):**

A,i,j,value

A,0,0,5

A,0,1,5

A,1,0,5

A,1,1,5

Format: **Matrix B (n x p):**

B,j,k,value

B,0,0,5

B,0,1,5

B,1,0,5

B,1,1,5

Step 7: Run the MapReduce Program

hadoop jar MM.jar MatrixMultiplication /Input /Output

Step 8: View the Output

hdfs dfs -ls /Output

hdfs dfs -cat /Output/part-r-00000

The output will show the resulting matrix in the format:

0,0 50

0,1 50

1,0 50

1,1 50

Refer youtube: https://www.youtube.com/watch?v=hBR4a0InHzw

https://medium.com/@nafthalivm/mapreduce-program-to-implement-matrix-multiplication-aefa14481b0c