EXPERIMENT - 5

Aim:

Develop a MapReduce program to perform Matrix Multiplication on two matrices (A and B) to produce a resultant matrix C.

Theory:

Matrix multiplication is a binary operation that produces a matrix from two matrices. For a matrix A of size m×n and matrix B of size n×p, the result C is an m×p matrix. The value of a cell C[i][k] is the dot product of the i-th row of A and the k-th column of B. In MapReduce, this requires restructuring the data so that the Reducer receives all necessary elements from Row 'i' of Matrix A and Column 'k' of Matrix B to calculate the single value for cell C[i][k].

Description:

The input file contains data in the format: 'MatrixName, Row, Col, Value'. The Mapper reads these entries. If it reads a value from Matrix A(i, j), it emits (i, k) as the key for all possible k. If it reads a value from Matrix B(j, k), it emits (i, k) as the key for all possible i. The Reducer receives a list of values for a specific cell (i, k) containing inputs from both A and B. It separates them, performs the multiplication of matching indices, and sums the results.

Code:

```
import java.io.IOException;
import java.util.HashMap;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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 MatrixMultiplication {

    // 1. Mapper Class
    public static class MatrixMapper extends Mapper<LongWritable, Text, Text,
Text> {
        // Assuming Matrix A is m*n and B is n*p
        // Let's assume m=2, n=2, p=2 for this simplified experiment
        int m = 2;
        int p = 2;

        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
```

```
            String line = value.toString();
            String[] parts = line.split(","); // Format: Matrix,Row,Col,Value

            if (parts[0].equals("A")) {
                // For A(i,j), emit key(i, k) for all k
                for (int k = 0; k < p; k++) {
                    String outputKey = parts[1] + "," + k;
                    // Value format: "A,j,val"
                    String outputValue = "A," + parts[2] + "," + parts[3];
                    context.write(new Text(outputKey), new
Text(outputValue));
                }
            } else if (parts[0].equals("B")) {
                // For B(j,k), emit key(i, k) for all i
                for (int i = 0; i < m; i++) {
                    String outputKey = i + "," + parts[2];
                    // Value format: "B,j,val"
                    String outputValue = "B," + parts[1] + "," + parts[3];
                    context.write(new Text(outputKey), new
Text(outputValue));
                }
            }
        }
    }

    // 2. Reducer Class
    public static class MatrixReducer extends Reducer<Text, Text, Text,
IntWritable> {

        public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
            String[] value;
            HashMap<Integer, Integer> hashA = new HashMap<>();
            HashMap<Integer, Integer> hashB = new HashMap<>();

            for (Text val : values) {
                value = val.toString().split(",");
                if (value[0].equals("A")) {
                    hashA.put(Integer.parseInt(value[1]),
Integer.parseInt(value[2]));
                } else {
                    hashB.put(Integer.parseInt(value[1]),
Integer.parseInt(value[2]));
                }
            }

            int n = 2; // Dimension n
            int result = 0;

            for (int j = 0; j < n; j++) {
                int a_val = hashA.containsKey(j) ? hashA.get(j) : 0;
                int b_val = hashB.containsKey(j) ? hashB.get(j) : 0;
                result += a_val * b_val;
```

```
            }

            if (result != 0) {
                context.write(key, new IntWritable(result));
            }
        }
    }

    // 3. Driver Method
    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.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Input:

A text file named matrix_input.txt (Format: Matrix-Name, Row, Col, Value):

```
root@DESKTOP-0MNJIUH:~/matrixmul# hdfs dfs -cat /user/root/matrix/input/matrix.txt
A 0 0 1
A 0 1 2
A 0 2 3
A 1 0 4
A 1 1 5
A 1 2 6
A 2 0 7
A 2 1 8
A 2 2 9
B 0 0 9
B 0 1 8
B 0 2 7
B 1 0 6
B 1 1 5
B 1 2 4
B 2 0 3
B 2 1 2
B 2 2 1
```

Output:

The content of the output file (Key: Row,Col Value: Result):

```
root@DESKTOP-0MNJIUH:~/matrixmul# hdfs dfs -cat /user/root/matrix/output/part-r-00000
0,0     30
0,1     24
0,2     18
1,0     84
1,1     69
1,2     54
2,0     138
2,1     114
2,2     90
```

Commands:

Description	Command
Create input directory:	hdfs dfs -mkdir /matrix_in
Upload input file:	hdfs dfs -put matrix_input.txt /matrix_in
Compile Java program:	javac -classpath $(hadoop classpath) -d . MatrixMultiplication.java
Create JAR file:	jar -cvf matrix.jar *.class
Run Hadoop Job:	hadoop jar matrix.jar MatrixMultiplication /matrix_in /matrix_out
View Output:	hdfs dfs -cat /matrix_out/part-r-00000

Conclusion:

In this experiment, we successfully implemented Matrix Multiplication using MapReduce. We learned how to design a Key-Value strategy where the Mapper replicates data to send rows and columns to the appropriate Reducers, and the Reducer performs the dot product calculation.