Here's a step-by-step guide to implement **Matrix Multiplication using MapReduce in Hadoop (Java)**:

**🔧 Step 1: Understand the Input Format**

Assume you want to compute **C = A × B**, where:

* A is an m x n matrix.
* B is an n x p matrix.
* Result matrix C is m x p.

We'll represent matrices in the following **text input format** for Hadoop:

matrix\_name i j value

**Example:**

**matrixA.txt:**

A 0 0 1

A 0 1 2

A 1 0 3

A 1 1 4

**matrixB.txt:**

B 0 0 5

B 0 1 6

B 1 0 7

B 1 1 8

**📁 Step 2: Set Up Directory Structure in Hadoop Project**

MatrixMultiplication/

├── src/

│ ├── MatrixMultiplyMapper.java

│ ├── MatrixMultiplyReducer.java

│ └── MatrixMultiplicationDriver.java

├── matrixA.txt

├── matrixB.txt

└── build/

**👨‍💻 Step 3: Write Java Code**

**🔹 MatrixMultiplyMapper.java**

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

import java.io.IOException;

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

public void map(LongWritable key, Text value, Context context)

throws IOException, InterruptedException {

String[] tokens = value.toString().split("\\s+");

String matrixName = tokens[0];

int i = Integer.parseInt(tokens[1]);

int j = Integer.parseInt(tokens[2]);

int val = Integer.parseInt(tokens[3]);

if (matrixName.equals("A")) {

for (int k = 0; k < 10; k++) // assuming B has 10 columns

context.write(new Text(i + "," + k), new Text("A," + j + "," + val));

} else {

for (int k = 0; k < 10; k++) // assuming A has 10 rows

context.write(new Text(k + "," + j), new Text("B," + i + "," + val));

}

}

}

**🔹 MatrixMultiplyReducer.java**

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

import java.io.IOException;

import java.util.\*;

public class MatrixMultiplyReducer extends Reducer<Text, Text, Text, IntWritable> {

public void reduce(Text key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

Map<Integer, Integer> amap = new HashMap<>();

Map<Integer, Integer> bmap = new HashMap<>();

for (Text val : values) {

String[] tokens = val.toString().split(",");

if (tokens[0].equals("A")) {

amap.put(Integer.parseInt(tokens[1]), Integer.parseInt(tokens[2]));

} else {

bmap.put(Integer.parseInt(tokens[1]), Integer.parseInt(tokens[2]));

}

}

int sum = 0;

for (int k : amap.keySet()) {

if (bmap.containsKey(k)) {

sum += amap.get(k) \* bmap.get(k);

}

}

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

}

}

**🔹 MatrixMultiplicationDriver.java**

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

public class MatrixMultiplicationDriver {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Matrix Multiplication");

job.setJarByClass(MatrixMultiplicationDriver.class);

job.setMapperClass(MatrixMultiplyMapper.class);

job.setReducerClass(MatrixMultiplyReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0])); // path to A and B combined

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

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

}

}

**🚀 Step 4: Compile and Package**

mkdir build

javac -classpath `hadoop classpath` -d build src/\*.java

jar -cvf matrixmult.jar -C build/ .

**📤 Step 5: Run on Hadoop**

hadoop fs -mkdir /matrix

hadoop fs -put matrixA.txt /matrix/

hadoop fs -put matrixB.txt /matrix/

hadoop jar matrixmult.jar MatrixMultiplicationDriver /matrix /matrix\_output

**📥 Step 6: Check Output**

hadoop fs -cat /matrix\_output/part-r-00000

Would you like a GitHub-ready project structure or sample matrices with expected outputs too?