|  |  |
| --- | --- |
| EXP NO: 3 | Implement of Matrix Multiplication with Hadoop Map Reduce |
|  |
| DATE: |  |

**AIM: -**

**BACKGROUND THEORY: -**

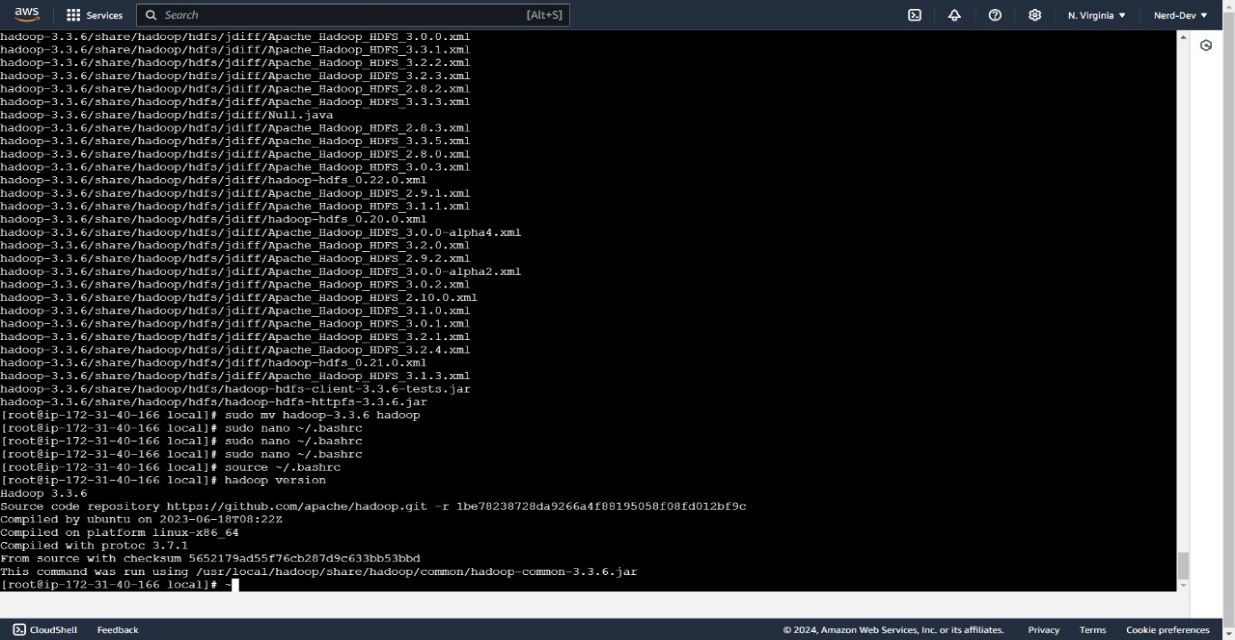
# **PROCEDURE: -**

* Switch to superuser mode using sudo su.
* Input Format:
  + We'll have two input files representing matrices A and B.
  + Matrix A (m x n): Split by rows.
  + Matrix B (n x p): Split by columns.
* Map Step:
  + For each element in matrix A (i, k, A[i][k]), emit the product to the intermediate key (i, j) where j is the column index in matrix B.
  + For each element in matrix B (k, j, B[k][j]), emit the product to the intermediate key (i, j) for all rows i of matrix A.
* Reduce Step:
  + For each intermediate key (i, j), sum the products of the corresponding values to calculate the result matrix C (i, j).
* Output Format:
  + Output matrix C, where each line is of the form i, j, C[i][j].

# **CODING: -**

* sudo su
* MatrixMapper.java
  + import java.io.IOException;
  + import org.apache.hadoop.io.IntWritable;
  + import org.apache.hadoop.io.Text;
  + import org.apache.hadoop.mapreduce.Mapper;
  + public class MatrixMapper extends Mapper<Object, Text, Text, Text> {
  + @Override
  + public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
  + String[] line = value.toString().split(",");
  + String matrixName = line[0]; // A or B
  + int i = Integer.parseInt(line[1]);
  + int j = Integer.parseInt(line[2]);
  + int valueOfElement = Integer.parseInt(line[3]);
  + if (matrixName.equals("A")) {
  + // Emit for all columns of B
  + for (int k = 0; k < context.getConfiguration().getInt("p", 0); k++) {
  + context.write(new Text(i + "," + k), new Text("A," + j + "," + valueOfElement));
  + }
  + } else {
  + // Emit for all rows of A
  + for (int k = 0; k < context.getConfiguration().getInt("m", 0); k++) {
  + context.write(new Text(k + "," + j), new Text("B," + i + "," + valueOfElement));
  + }
  + }
  + }
  + }
* MatrixReducer.java
  + import java.io.IOException;
  + import java.util.HashMap;
  + import java.util.Map;
  + import org.apache.hadoop.io.IntWritable;
  + import org.apache.hadoop.io.Text;
  + import org.apache.hadoop.mapreduce.Reducer;
  + public class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {
  + @Override
  + public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
  + Map<Integer, Integer> mapA = new HashMap<>();
  + Map<Integer, Integer> mapB = new HashMap<>();
  + for (Text val : values) {
  + String[] parts = val.toString().split(",");
  + if (parts[0].equals("A")) {
  + mapA.put(Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
  + } else {
  + mapB.put(Integer.parseInt(parts[1]), Integer.parseInt(parts[2]));
  + }
  + }
  + int result = 0;
  + for (Integer k : mapA.keySet()) {
  + if (mapB.containsKey(k)) {
  + result += mapA.get(k) \* mapB.get(k);
  + }
  + }
  + context.write(key, new IntWritable(result));
  + }
  + }
* MatrixMultiplication.java
  + 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.lib.input.FileInputFormat;
  + import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
  + public class MatrixMultiplication {
  + public static void main(String[] args) throws Exception {
  + Configuration conf = new Configuration();
  + // Dimensions of the matrices A (m x n) and B (n x p)
  + conf.setInt("m", 3); // Rows of A
  + conf.setInt("n", 2); // Columns of A and Rows of B
  + conf.setInt("p", 3); // Columns of B
  + 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(IntWritable.class);
  + FileInputFormat.addInputPath(job, new Path(args[0]));
  + FileOutputFormat.setOutputPath(job, new Path(args[1]));
  + System.exit(job.waitForCompletion(true) ? 0 : 1);
  + }
  + }
* Steps to Run the Code
  + hdfs dfs -put matrixA.txt /input/
  + hdfs dfs -put matrixB.txt /input/
  + hadoop com.sun.tools.javac.Main MatrixMultiplication.java
  + jar cf matrixmultiplication.jar MatrixMultiplication\*.class
  + hadoop jar matrixmultiplication.jar MatrixMultiplication /input/ /output/
  + hdfs dfs -cat /output/part-r-00000

# **OUTPUT: -**



# Matrix Multiplication Execution steps Using Map Reduce in HDFS - YouTube