Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

Big Data Analytics/Odd Sem 2023-23/Experiment 2A

Name: Subrato Tapaswi	**Class/Roll No.: D16AD/60**	**Grade:**

Title of Experiment: To implement the following programs using Map Reduce:

a. Word Count
b. matrix-vector multiplication

Objective of Experiment:

a. **Word Count using MapReduce:**
Implement the Word Count program using the MapReduce paradigm to efficiently calculate the frequency of each word in a given text corpus.

b. **Matrix-Vector Multiplication using MapReduce:**
Develop a MapReduce solution for performing matrix-vector multiplication, aiming to efficiently compute the product of a matrix and a vector by distributing the computation across a cluster of nodes.

Outcome of Experiment:

Thus, we implemented both programs using Map Reduce in Hadoop

Problem Statement:

a. **Word Count using MapReduce:**
Develop a MapReduce program to analyze a large text dataset and determine the frequency of each unique word present in the corpus. The program should take advantage of parallel processing to efficiently handle the computation and provide an accurate count of word occurrences.

b. **Matrix-Vector Multiplication using MapReduce:**
Create a MapReduce solution to perform matrix-vector multiplication. Given a matrix and a vector, the program should distribute the computation across multiple nodes, effectively calculating the product and generating the resulting vector.

Description / Theory:

Hadoop MapReduce:

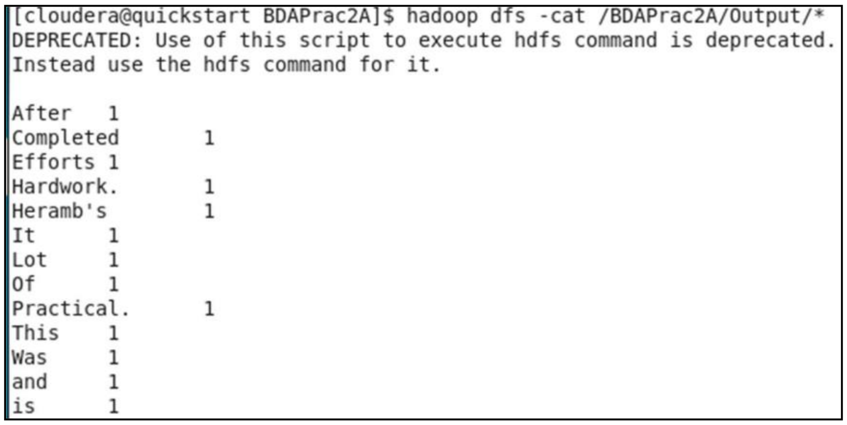
Hadoop MapReduce is a programming model and processing framework designed to handle large-scale data processing tasks across clusters of computers. It's used to efficiently process and analyze vast amounts of data in parallel, making it suitable for tasks like data transformation, aggregation, and more. Here's a concise explanation of how Hadoop MapReduce works:

Program:

A. To implement the Word Count

```
import java.io.IOException; import
java.util.StringTokenizer;

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;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);private
    Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());while
      (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {private
    IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,Context
                       context
                       ) throws IOException, InterruptedException {int sum
      = 0;
      for (IntWritable val : values) {sum
        += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.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);
  }
}
```

Output:



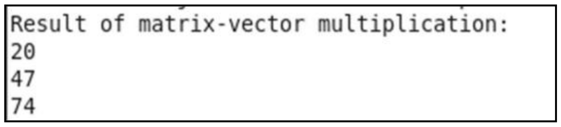
```
[cloudera@quickstart BDAPrac2A]$ hadoop dfs -cat /BDAPrac2A/Output/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

After   1
Completed       1
Efforts 1
Hardwork.       1
Heramb's        1
It      1
Lot     1
Of      1
Practical.      1
This    1
Was     1
and     1
is      1
```

B. Matrix-Vector multiplication

Program:

```
public class MatrixVectorMultiplication {public
    static void main(String[] args) {
        // Define the matrix and vector
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[] vector = {2, 3, 4};

        // Check if matrix and vector dimensions are compatibleint
        matrixRows = matrix.length;
        int matrixCols = matrix[0].length;int
        vectorSize = vector.length;

        if (matrixCols != vectorSize) {
            System.out.println("Matrix and vector dimensions are not compatible formultiplication.");
            return;
        }

        // Perform matrix-vector multiplicationint[]
        result = new int[matrixRows];
        for (int i = 0; i < matrixRows; i++) {

                for (int j = 0; j < matrixCols; j++) {
                result[i] += matrix[i][j] * vector[j];
            }
        }


        // Display the result
        System.out.println("Result of matrix-vector multiplication:");for (int i =
        0; i < matrixRows; i++) {
            System.out.println(result[i]);
        }
    }
}
```

Output:



```
Result of matrix-vector multiplication:
20
47
74
```

Results and Discussions:

MapReduce is a parallel processing model for large-scale data.

Word Count:

<u>Result</u>*:* Efficiently counts word occurrences in texts.
<u>Discussion</u>: Map phase splits text, emits (word, 1). Reduce phase aggregate counts. Scales well for basic tasks.

Matrix-Vector Multiplication:

<u>Result:</u> Computes matrix-vector product.
<u>Discussion:</u> Mappers process matrix rows to emit (col, partial) pairs. Reduce combines partials for results. Shows distributed linear algebra.

Overall Discussion:

MapReduce simplifies parallel processing, automating distribution and fault tolerance. While powerful, newer models like Apache Spark offer more features and speed