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 :** Prasad Jawale | **Class/RollNo.:** D16AD 20 | **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:**
Created 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 for distributed and parallel processing of large datasets in the Hadoop ecosystem.

**Mapper:** In the MapReduce paradigm, data processing begins with a Mapper. The input data is divided into smaller chunks or blocks, and each Mapper processes a portion of the data. The Mapper applies a user-defined map function to each data record and produces a set of key-value pairs as intermediate results. These key-value pairs are often used to group and sort the data for further processing.

**Shuffling and Sorting:** After the mapping phase, the framework performs shuffling and sorting. This step groups and sorts the intermediate key-value pairs by their keys, ensuring that all values associated with the same key are grouped together. This is essential for the subsequent Reduce phase.

**Reducer:** Once the data is shuffled and sorted, the Reducer phase begins. Reducers apply a user-defined reduce function to the grouped and sorted key-value pairs, producing the final output. The output of the Reducer is typically written to a storage system like Hadoop Distributed File System (HDFS).

**2. Distributed Processing:**
Hadoop MapReduce is designed for distributed processing, meaning it can distribute the data and computation across a cluster of machines. Each machine in the cluster can run Mappers and Reducers in parallel, allowing for the efficient processing of large datasets.

**3. Fault Tolerance:**
Hadoop MapReduce is fault-tolerant. If a node in the cluster fails during processing, the framework automatically reroutes tasks to healthy nodes, ensuring that the job can continue without data loss.

**4. Scalability:**

MapReduce scales horizontally, which means you can add more machines to the cluster as the dataset or processing requirements grow. This scalability makes it suitable for handling big data workloads.

**5. Batch Processing:**

Hadoop MapReduce is primarily designed for batch processing. It is well-suited for tasks like log processing, data aggregation, and ETL (Extract, Transform, Load) operations.

**6. Limitations:**

While Hadoop MapReduce is powerful for certain types of batch processing tasks, it may not be the most efficient choice for iterative algorithms and real-time data processing. For these use cases, other frameworks like Apache Spark have gained popularity due to their in-memory processing capabilities and support for diverse workloads.

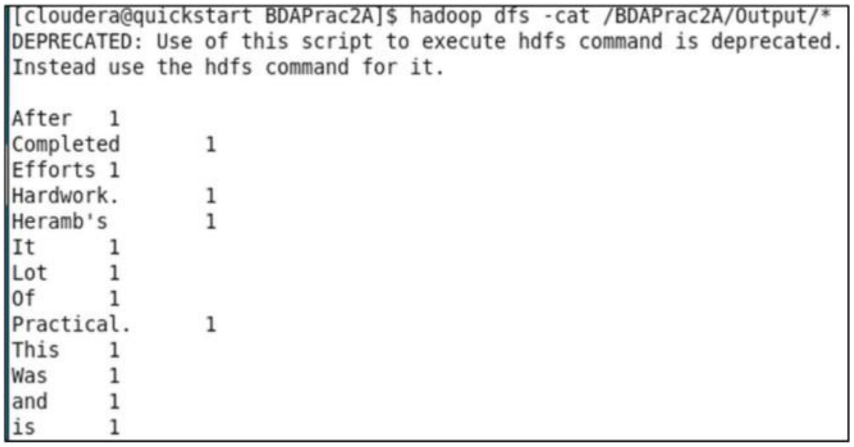
Hadoop MapReduce is a foundational component of the Hadoop ecosystem, providing a framework for distributed and batch-oriented data processing. It has been instrumental in enabling organizations to handle and extract insights from large-scale datasets, but it has also faced competition from newer, more versatile big data processing frameworks like Apache Spark.

**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

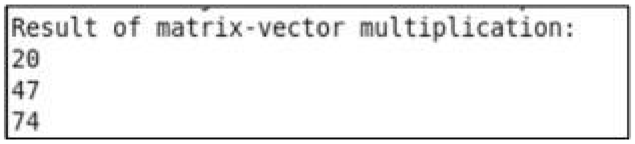


```
[cloudera@quickstart BDAPrac2A]$ hadoop dfs -cat /BDAPrac2A/Output/*
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

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**
**Programs:**

```
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 compatible
        int 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 for
multiplication.");
            return;
        }

        // Perform matrix-vector multiplication
        int[] 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:**

**Result:** Efficiently count words occurrences in texts.
**Discussions:** Map phase splits text, emits . Reduce phase aggregate counts. Scales well for basic tasks.

**Matrix-Vector Multiplication:**

**Result**: Computes matrix-vector product.
**Discussion:** Mappers process matrix rows to emit pairs. Reduce combines partials for results. Shows distributed linear algebra.

**Overall Discussion:**
MapReduce simplified parallel processing, automating distribution and fault tolerance. While powerful, newer models like Apache Spark offer more features and speed.