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

Artificial Intelligence and Data Science Department

BDA/Odd Sem 2023-24/Experiment 2a

Name : Manav Pahilwani	Class/Roll No. : D16AD/ 37	Grade :

Title of Experiment : To implement the following programs using Map Reduce:
a. Word Count
b. Matrix – Vector multiplication

Theory:

Hadoop Map Reduce:

It is a programming model and processing framework designed for processing and generating large datasets that can be distributed across clusters of commodity hardware. It's a core component of the Apache Hadoop ecosystem, which is an open-source framework for distributed storage and processing of big data.

The following are the key concepts and components of Hadoop Map Reduce:

Map Phase: In this phase, input data is divided into smaller chunks, and each chunk is processed independently. The processing is done by user-defined "map" functions, which take a set of key-value pairs as input and produce intermediate key-value pairs as output. The map phase runs in parallel across the cluster.

Shuffle and Sort Phase: After the map phase, the intermediate key-value pairs are grouped by their keys and sorted. This phase ensures that all values for a given key are grouped together and ordered properly, which is essential for the reduce phase.

Reduce Phase: In this phase, user-defined "reduce" functions take the sorted and grouped intermediate key-value pairs as input and perform further processing to produce the final output. The reduce phase also runs in parallel, and each reducer processes a subset of the keys.

Job Tracker and Task Tracker: These are components of the Hadoop framework responsible for managing the Map Reduce jobs. The Job Tracker coordinates job execution, while Task Trackers run individual map and reduce tasks on worker nodes.

Fault Tolerance: Hadoop Map Reduce is designed to be fault-tolerant. If a node fails during processing, the framework automatically redistributes the work to other nodes, ensuring that the job can continue without data loss.

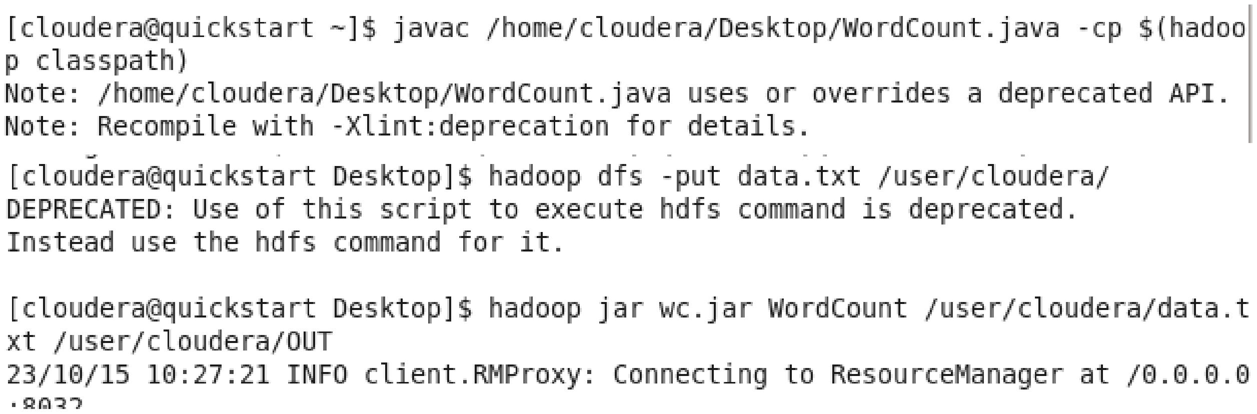
Hadoop Map Reduce is particularly well-suited for batch processing of large datasets, where the processing tasks can be parallelized and distributed across a cluster of machines. However, it has some limitations, such as relatively high latency for real-time or interactive processing, which has led to the development of other processing frameworks within the Hadoop ecosystem, like Apache Spark, for different use cases.

Program:

- 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 {
```

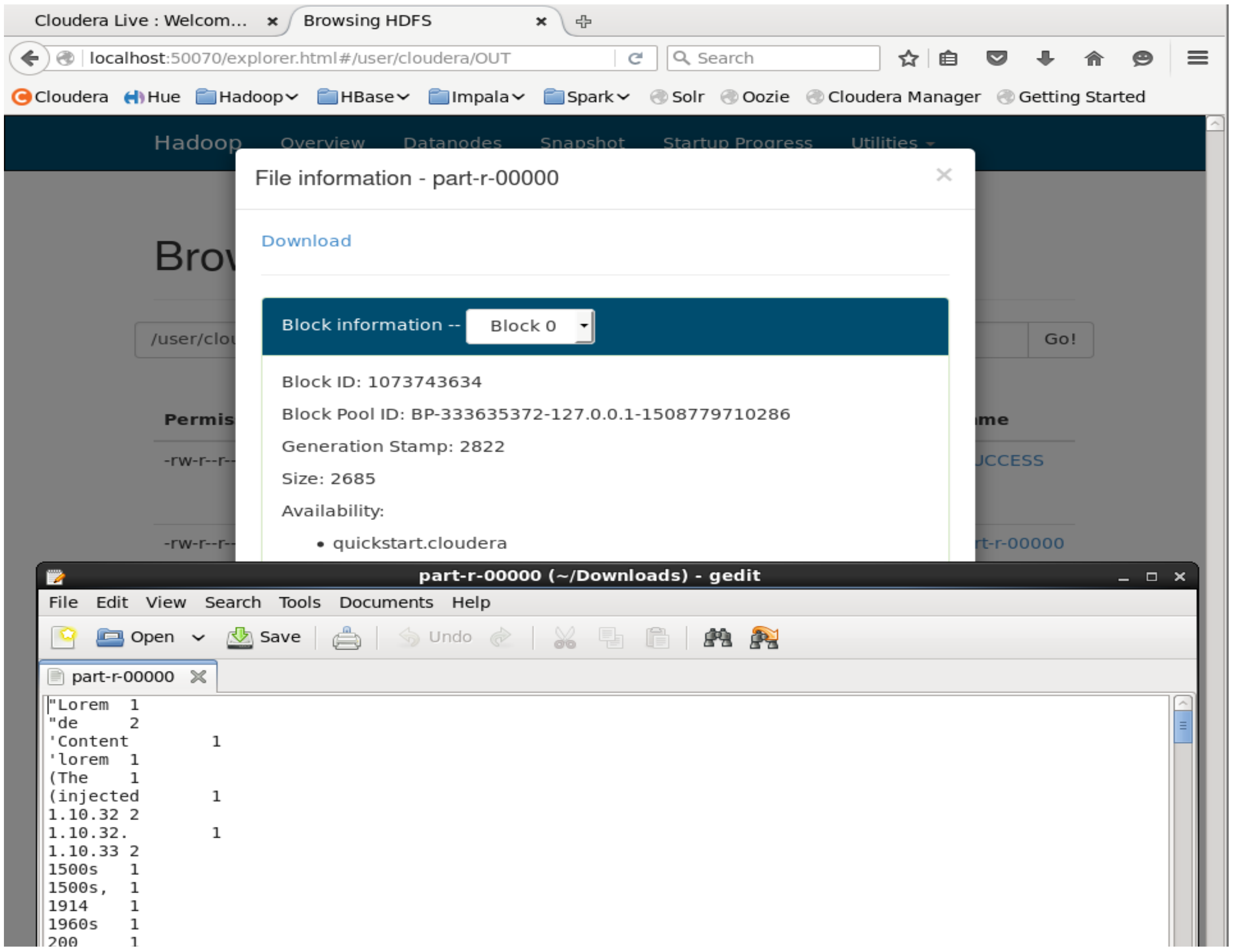
```
 StringTokenizer itr = new StringTokenizer(value.toString());
 while (itr.hasMoreTokens()) {
 word.set(itr.nextToken());
 context.write(word, one);
 }
 }
 }
public static class IntSumReducer
 extends Reducer<Text,IntWritable,Text,IntWritable> {
 private IntWritable result = new IntWritable();
 public 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);
 }
}
```



```
[cloudera@quickstart ~]$ javac /home/cloudera/Desktop/WordCount.java -cp $(hadoo
p classpath)
Note: /home/cloudera/Desktop/WordCount.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

[cloudera@quickstart Desktop]$ hadoop dfs -put data.txt /user/cloudera/
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

[cloudera@quickstart Desktop]$ hadoop jar wc.jar WordCount /user/cloudera/data.t
xt /user/cloudera/OUT
23/10/15 10:27:21 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
```



File information - part-r-00000

Download

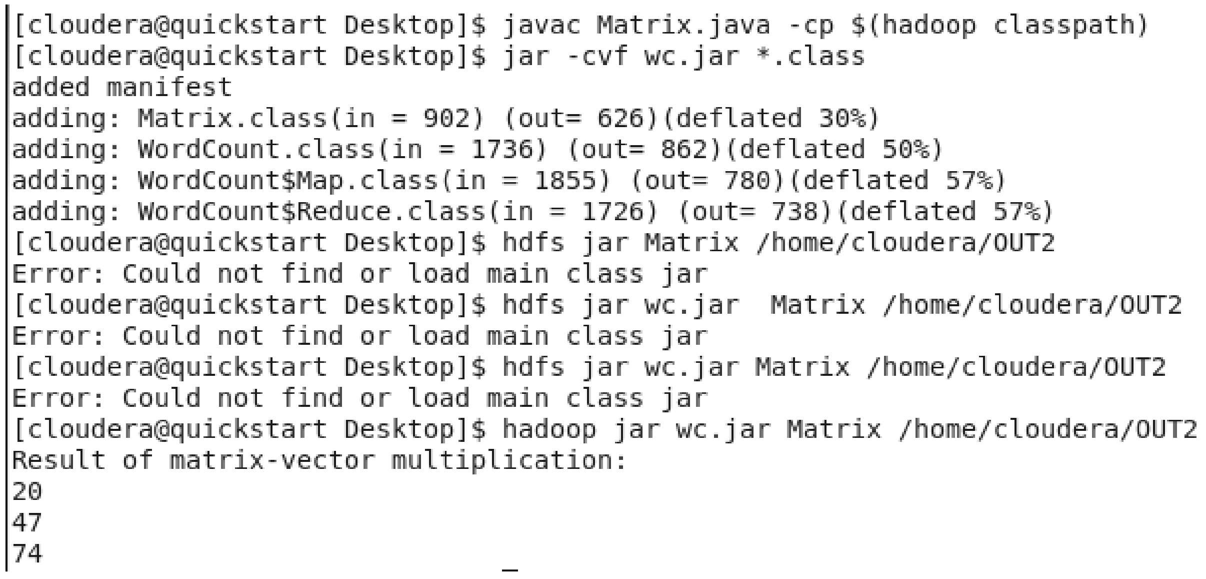
Block information -- Block 0

Block ID: 1073743634
Block Pool ID: BP-333635372-127.0.0.1-1508779710286
Generation Stamp: 2822
Size: 2685
Availability:
- quickstart.cloudera

```
"Lorem  1
"de     2
'Content        1
'lorem  1
(The    1
(injected       1
1.10.32 2
1.10.32.        1
1.10.33 2
1500s   1
1500s,  1
1914    1
1960s   1
200     1
```

- Matrix Vector Multiplication
  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]);
 }
 }
}
```



```
[cloudera@quickstart Desktop]$ javac Matrix.java -cp $(hadoop classpath)
[cloudera@quickstart Desktop]$ jar -cvf wc.jar *.class
added manifest
adding: Matrix.class(in = 902) (out= 626)(deflated 30%)
adding: WordCount.class(in = 1736) (out= 862)(deflated 50%)
adding: WordCount$Map.class(in = 1855) (out= 780)(deflated 57%)
adding: WordCount$Reduce.class(in = 1726) (out= 738)(deflated 57%)
[cloudera@quickstart Desktop]$ hdfs jar Matrix /home/cloudera/OUT2
Error: Could not find or load main class jar
[cloudera@quickstart Desktop]$ hdfs jar wc.jar  Matrix /home/cloudera/OUT2
Error: Could not find or load main class jar
[cloudera@quickstart Desktop]$ hdfs jar wc.jar Matrix /home/cloudera/OUT2
Error: Could not find or load main class jar
[cloudera@quickstart Desktop]$ hadoop jar wc.jar Matrix /home/cloudera/OUT2
Result of matrix-vector multiplication:
20
47
74
```

Results and Discussions :

Map Reduce is a parallel processing model for large – scale data. Map Reduce simplifies parallel processing, automating distribution and fault tolerance. While powerful, newer models like Apache Spark offer more features and speed.