Program – 3

AIM: Run a basic Word Count MapReduce program to understand MapReduce paradigm: Count words in a given file. View the output file. Calculate the execution time

About HDFS

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

Reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Below are the steps for MapReduce data flow:

- **Step 1:** One block is processed by one **mapper** at a time. In the mapper, a developer can specify his own business logic as per the requirements. In this manner, Map runs on all the nodes of the cluster and process the data blocks in parallel.
- **Step 2:** Output of Mapper also known as intermediate output is written to the local disk. An output of mapper is not stored on HDFS as this is temporary data and **writing on HDFS** will create unnecessary many copies.
- **Step 3:** Output of mapper is shuffled to **reducer** node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling/copying is a physical movement of data which is done over the network.
- **Step 4:** Once all the mappers are finished and their output is shuffled on reducer nodes then this intermediate output is merged & sorted. Which is then provided as input to reduce phase.
- **Step 5:** Reduce is the second phase of processing where the user can specify his own custom business logic as per the requirements. An input to a reducer is provided from all the mappers. An output of reducer is the final output, which is written on HDFS.

Program – 3

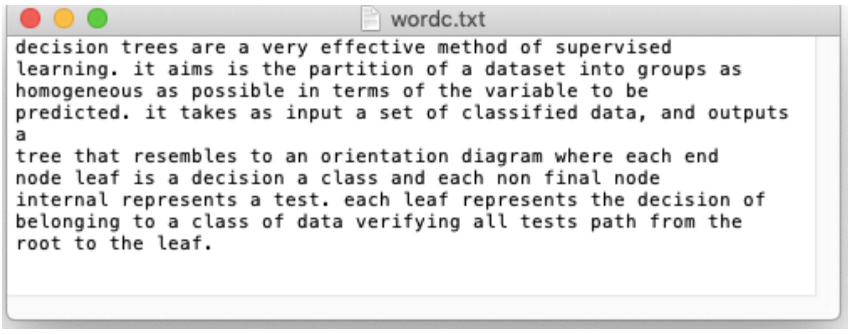
Code

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class wordCount {

 public static class Map extends Mapper<LongWritable, Text, Text,
IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while  (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> {

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

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
```

Program – 3

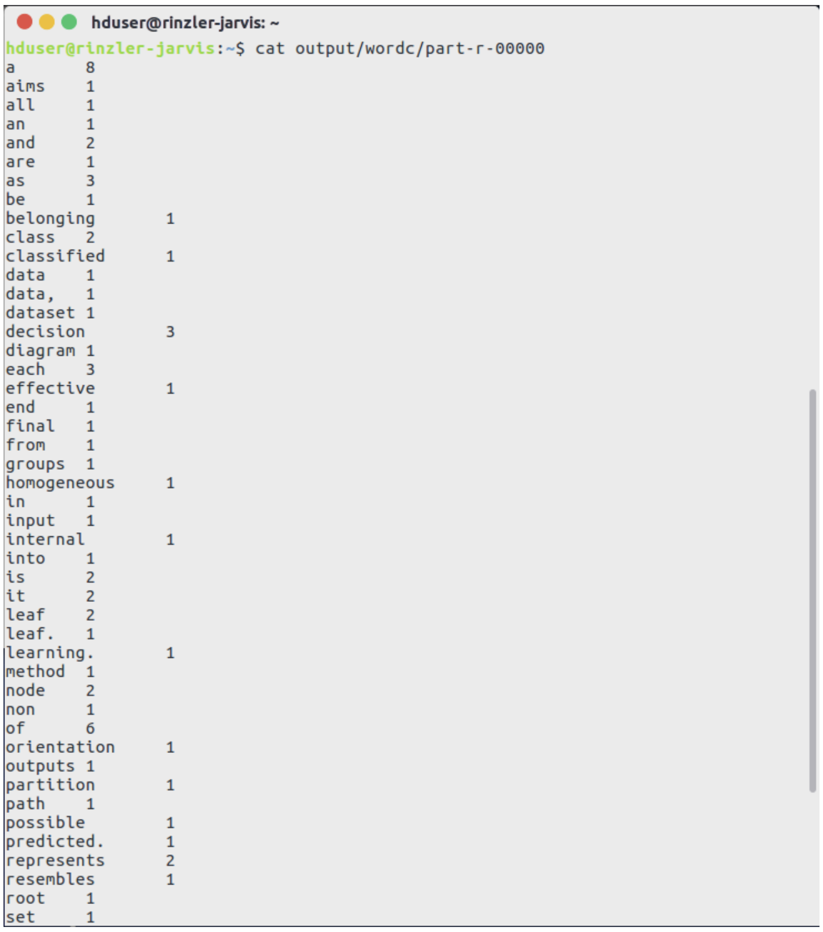
```java

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job.waitForCompletion(true);
  }
}
```

Output



```
decision trees are a very effective method of supervised
learning. it aims is the partition of a dataset into groups as
homogeneous as possible in terms of the variable to be
predicted. it takes as input a set of classified data, and outputs
a
tree that resembles to an orientation diagram where each end
node leaf is a decision a class and each non final node
internal represents a test. each leaf represents the decision of
belonging to a class of data verifying all tests path from the
root to the leaf.
```

input text data



```
hduser@rinzler-jarvis:~$ cat output/wordc/part-r-00000
a       8
aims    1
all     1
an      1
and     2
are     1
as      3
be      1
belonging       1
class   2
classified      1
data    1
data,   1
dataset 1
decision        3
diagram 1
each    3
effective       1
end     1
final   1
from    1
groups  1
homogeneous     1
in      1
input   1
internal        1
into    1
is      2
it      2
leaf    2
leaf.   1
learning.       1
method  1
node    2
non     1
of      6
orientation     1
outputs 1
partition       1
path    1
possible        1
predicted.      1
represents      2
resembles       1
root    1
set     1
```

output

Findings and Learnings:

1. We learned how map-reduce works.
2. We learned how to code using hadoop in Java.
3. We learned how input and output in hadoop works.