Experiment No. 03

Semester	B.E. Semester VII – Computer Engineering
Subject	Big Data Analysis Lab
Lab Professor In-charge	Dr. Umesh Kulkarni
Subject Professor In-charge	Prof. Pankaj Vanwari
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

Title: Develop and execute a simple MapReduce program to count word occurrences in a text file.

Theory:

MapReduce Overview: MapReduce is a programming model used for processing large data sets in a parallel, distributed manner. It consists of two main phases:

- Map Phase: Processes input data, usually by splitting it into key-value pairs.
- Reduce Phase: Aggregates and processes the output from the Map phase to produce the final result.

The word count problem is a classic example of how MapReduce can be used to count the occurrences of each word in a large text file.

Architecture of MapReduce for Word Count:

1. Input Split: The input file is divided into fixed-size chunks or splits. Each split is processed independently by the Mapper.
2. Mapper: Takes each line of input, splits it into words, and outputs a key-value pair (word, 1) for each word.
3. Shuffle and Sort: The framework automatically sorts the key-value pairs by key (word).
4. Reducer: Receives all key-value pairs with the same key (word) and sums up the values (counts) for each word.
5. Output: The Reducer outputs the word and its final count.

Steps for Developing and Executing a Word Count MapReduce Program:

Step 1 : Set Up the Hadoop Environment

Start the Hadoop services and verify the hadoop services are running properly:

```
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> start-dfs.cmd
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> start-yarn.cmd
starting yarn daemons
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> jps
10256 NameNode
30676 NodeManager
7108 ResourceManager
1724 DataNode
22860 Jps
```

Step 2 : Prepare the Input File (input.txt)

```
Hello Hadoop
Hadoop is a big data framework
This is for BDA Prac 3
```

Step 3 : Upload the input file to HDFS

```
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hdfs dfs -mkdir "/bdaPrac3"
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hdfs dfs -put "C:\Users\chait\OneDrive\Documents\Lone\BDAPrac\wordCount\input.
txt"  "/bdaPrac3"
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hdfs dfs -cat "/bdaPrac3/input.txt"
Hello Hadoop
Hadoop is a big data framework
This is for BDA Prac 3
```

Step 4 : Develop the MapReduce Program

Mapper Class (WordCountMapper.java):

This class splits each line into words and outputs a key-value pair of each word with a count of 1.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

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

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
```

```
        String line = value.toString();
        String[] words = line.split("\\s+");
        for (String w : words) {
            word.set(w);
            context.write(word, one);
        }
    }
}
```

Reducer Class (WordCountReducer.java):

This class sums up all occurrences of each word and outputs the word along with its final count.

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

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

Driver Class (WordCountDriver.java):

This class sets up the MapReduce job configuration.

```
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 WordCountDriver {

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

Step 5 : Compile and Package the Program

Compile the MapReduce program and package it into a JAR file.

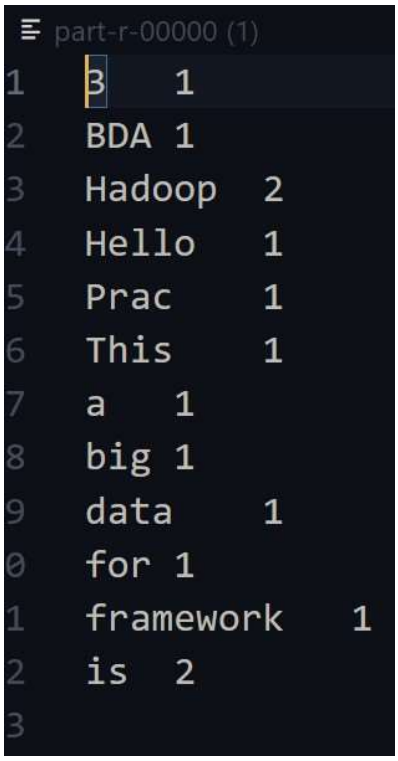
```
PS C:\Users\chait\OneDrive\Documents\Lone\BDAPrac\wordCount> javac -classpath $(hadoop classpath) -d wordcount
_classes WordCountMapper.java WordCountReducer.java WordCountDriver.java
PS C:\Users\chait\OneDrive\Documents\Lone\BDAPrac\wordCount> jar -cvf wordcount.jar -C wordcount_classes/ .
added manifest
adding: WordCountDriver.class(in = 1374) (out= 750)(deflated 45%)
adding: WordCountMapper.class(in = 1888) (out= 787)(deflated 58%)
adding: WordCountReducer.class(in = 1602) (out= 669)(deflated 58%)
```

Step 6 : Execute the MapReduce Job

```
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hadoop jar "C:\Users\chait\OneDrive\Documents\Lone\BDAPrac\wordCount\wordcount.jar" WordCountDriver "/bdaPrac3/input.txt" "/bdaPrac3/output"
2024-09-18 02:02:40,750 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2024-09-18 02:02:41,429 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2024-09-18 02:02:41,442 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/chait/.staging/job_1726600828893_0004
2024-09-18 02:02:41,624 INFO input.FileInputFormat: Total input files to process : 1
2024-09-18 02:02:42,091 INFO mapreduce.JobSubmitter: number of splits:1
2024-09-18 02:02:42,219 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1726600828893_0004
2024-09-18 02:02:42,221 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-09-18 02:02:42,375 INFO conf.Configuration: resource-types.xml not found
2024-09-18 02:02:42,375 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2024-09-18 02:02:42,452 INFO impl.YarnClientImpl: Submitted application application_1726600828893_0004
2024-09-18 02:02:42,481 INFO mapreduce.Job: The url to track the job: http://Chaitanya:8088/proxy/application_1726600828893_0004/
2024-09-18 02:02:42,482 INFO mapreduce.Job: Running job: job_1726600828893_0004
2024-09-18 02:02:49,609 INFO mapreduce.Job: Job job_1726600828893_0004 running in uber mode : false
2024-09-18 02:02:49,611 INFO mapreduce.Job:  map 0% reduce 0%
2024-09-18 02:02:55,709 INFO mapreduce.Job:  map 100% reduce 0%
2024-09-18 02:03:00,771 INFO mapreduce.Job:  map 100% reduce 100%
2024-09-18 02:03:00,775 INFO mapreduce.Job: Job job_1726600828893_0004 completed successfully
```

Step 7 : Check The Output

```
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hdfs dfs -ls "/bdaPrac3/output"
Found 2 items
-rw-r--r--   1 chait supergroup          0 2024-09-18 02:02 /bdaPrac3/output/_SUCCESS
-rw-r--r--   1 chait supergroup         81 2024-09-18 02:02 /bdaPrac3/output/part-r-00000
PS D:\hadoop-extracted\hadoop-3.2.4\sbin> hdfs dfs -cat "/bdaPrac3/output/part-r-00000"
3       1
BDA     1
Hadoop  2
Hello   1
Prac    1
This    1
a       1
big     1
data    1
for     1
framework       1
is      2
```



Conclusion :

This experiment demonstrates how to use the MapReduce programming model to solve the classic word count problem in Hadoop. By dividing the problem into Map and Reduce tasks, large datasets can be processed in a distributed and parallel fashion, allowing efficient data processing even for massive files.