

LAB EXERCISE 3

Aim: Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

- Find the number of occurrences of each word appearing in the input file(s)
- Performing a MapReduce Job for word search count (look for specific keywords in a file)

Theory:

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 are merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Procedure:

Step1) Upload input.txt file on hdfs which contain some sentences with repeated words

```
>>> hadoop fs -put "Input.txt file path" /input
```

Step 2) Open IntelliJ and add required dependencies in "pom.xml" file.

```
<dependencies>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>3.3.3</version>
  </dependency>
  <dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>3.3.3</version>
  </dependency>
</dependencies>
```

Step 3) After adding dependencies add 3 java file in org.example folder named

1. WC_Runner
2. WC_Reducer
3. WC_Mapper

Step 4) Open IntelliJ terminal and write the below command,

```
>>>hadoop jar target/wordcount-1.0-SNAPSHOT.jar org.ankit.WC_Runner "hdfs  
input.txt file path" /output
```

For WC_Mapper.java:

```
package org.priyanshu;  
  
import java.io.IOException;  
import  
java.util.StringTokenizer;  
import org.apache.hadoop.io.IntWritable;  
import  
org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import  
org.apache.hadoop.mapred.MapReduceBase;  
import org.apache.hadoop.mapred.Mapper;  
import  
org.apache.hadoop.mapred.OutputCollector;  
import org.apache.hadoop.mapred.Reporter;  
public class WC_Mapper extends MapReduceBase implements  
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, OutputCollector<Text,IntWritable>  
        output, Reporter reporter) throws IOException{  
        String line = value.toString();  
        StringTokenizer tokenizer = new  
        StringTokenizer(line); while  
        (tokenizer.hasMoreTokens()){  
            word.set(tokenizer.nextToken(  
                )); output.collect(word, one);  
        }  
    }  
}
```

For WC_Reducer.java:

```
package org.priyanshu;

import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapred.MapReduceBase;
import
org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WC_Reducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException
{
    int sum=0;
    while (values.hasNext()) {
        sum+=values.next().get();
    }

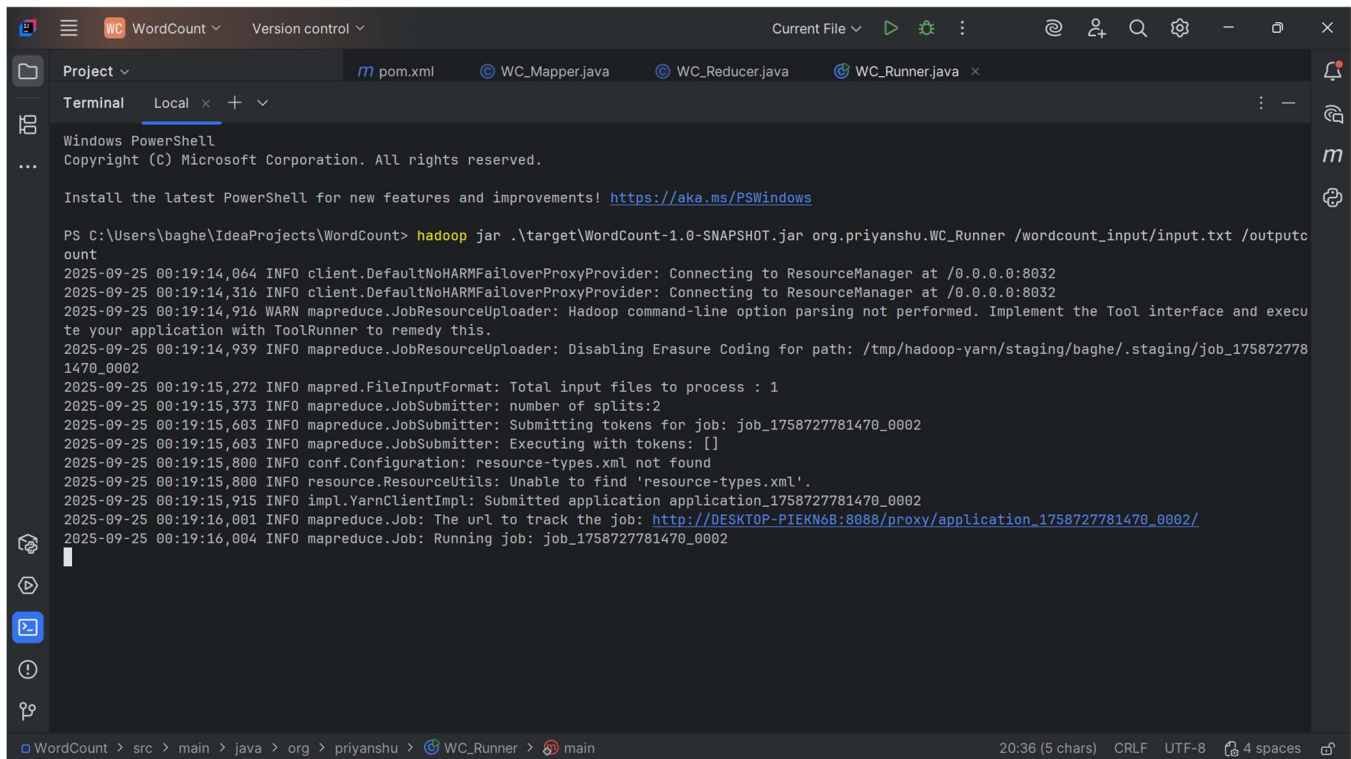
    output.collect(key,new IntWritable(sum));
}
}
```

For WC_Runner.java:

```
package org.priyanshu;

import java.io.IOException;
import
org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapred.FileInputFormat;
import
org.apache.hadoop.mapred.FileOutputFormat
; import
org.apache.hadoop.mapred.JobClient; import
org.apache.hadoop.mapred.JobConf;
import
org.apache.hadoop.mapred.TextInputFormat;
import
org.apache.hadoop.mapred.TextOutputFormat;
public class WC_Runner {
    public static void main(String[] args) throws IOException{
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
        FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new
        Path(args[1])); JobClient.runJob(conf);
    }
}
```

Running WC_Runner:



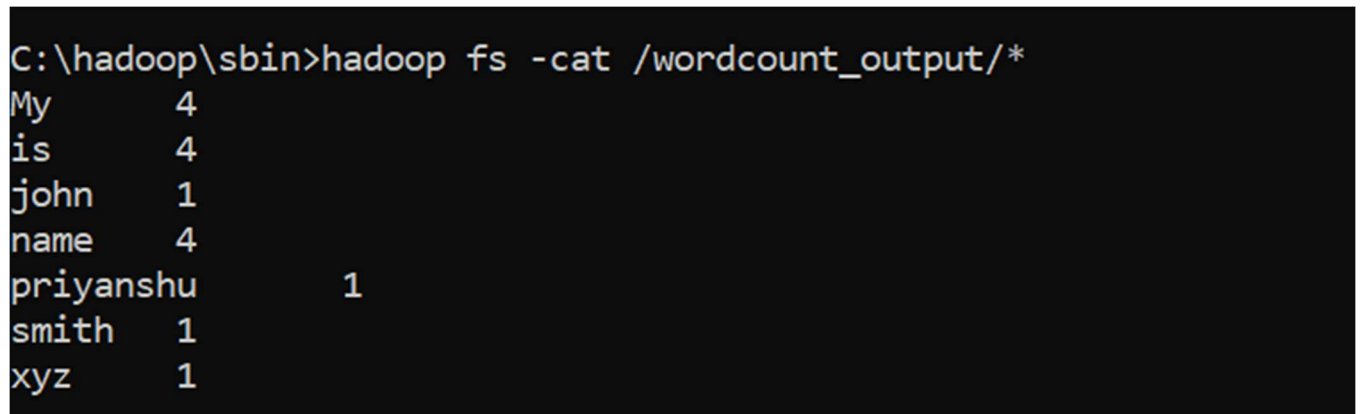
The screenshot shows an IDE window with a terminal running a Hadoop job. The terminal output includes log messages from the client, resource manager, and mapreduce components. The job is successfully executed, and the output is written to the file system.

```
PS C:\Users\baghe\IdeaProjects\WordCount> hadoop jar .\target\WordCount-1.0-SNAPSHOT.jar org.priyanshu.WC_Runner /wordcount_input/input.txt /outputcount

2025-09-25 00:19:14,064 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-09-25 00:19:14,316 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
2025-09-25 00:19:14,916 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2025-09-25 00:19:14,939 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/baghe/.staging/job_1758727781470_0002
2025-09-25 00:19:15,272 INFO mapred.FileInputFormat: Total input files to process : 1
2025-09-25 00:19:15,373 INFO mapreduce.JobSubmitter: number of splits:2
2025-09-25 00:19:15,603 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1758727781470_0002
2025-09-25 00:19:15,603 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-09-25 00:19:15,800 INFO conf.Configuration: resource-types.xml not found
2025-09-25 00:19:15,800 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-09-25 00:19:15,915 INFO impl.YarnClientImpl: Submitted application application_1758727781470_0002
2025-09-25 00:19:16,001 INFO mapreduce.Job: The url to track the job: http://DESKTOP-PIEKN6B:8088/proxy/application_1758727781470_0002/
2025-09-25 00:19:16,004 INFO mapreduce.Job: Running job: job_1758727781470_0002
```

Snapshot No. 6 (Running of WordCount Code)

Output:



The screenshot shows a terminal window with the command `hadoop fs -cat /wordcount_output/*` and its output, which lists words and their counts.

```
C:\hadoop\sbin>hadoop fs -cat /wordcount_output/*
My      4
is      4
john    1
name    4
priyanshu      1
smith    1
xyz      1
```

Snapshot No. 7 (Output of WordCount Code)

Conclusion:

In this, we learnt about the map-reduce and its function in Hadoop and how map breaks the data and reduce combine the generated output.