Group B- Big Data Analytics

Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map-Reduce framework on local-standalone set-up.

Execute following commands on Terminal

java -version

To check the version of Java installed on your system, you can use the command java -version.

su - hadoop

The command `su - hadoop` is used to switch the current user to the user named "hadoop" in a Unix-based operating system. This is typically used in environments where multiple users have access to a system, and "hadoop" might be a user account associated with the Hadoop distributed computing framework.

When you run this command, you will be prompted to enter the password for the "hadoop" user account. If the password is entered correctly, you will then be logged in as the "hadoop" user, inheriting its environment settings and permissions.

cd hadoop

hadoop version

To check the version of Hadoop installed on your system, you can use the `hadoop version` command. This command will display the installed version of Hadoop along with some additional information.

nano data1.txt

put following text into data1.txt

```
HDFS is a storage unit of Hadoop
MapReduce is a processing tool of Hadoop
```

press control + s then control x

start-all.sh

The `start-all.sh` script is typically used in Hadoop environments to start all the Hadoop daemons simultaneously.

hdfs dfs -mkdir /test_wc

The command `hdfs dfs -mkdir /test_wc` is used to create a directory named "test_wc" in the Hadoop Distributed File System (HDFS).

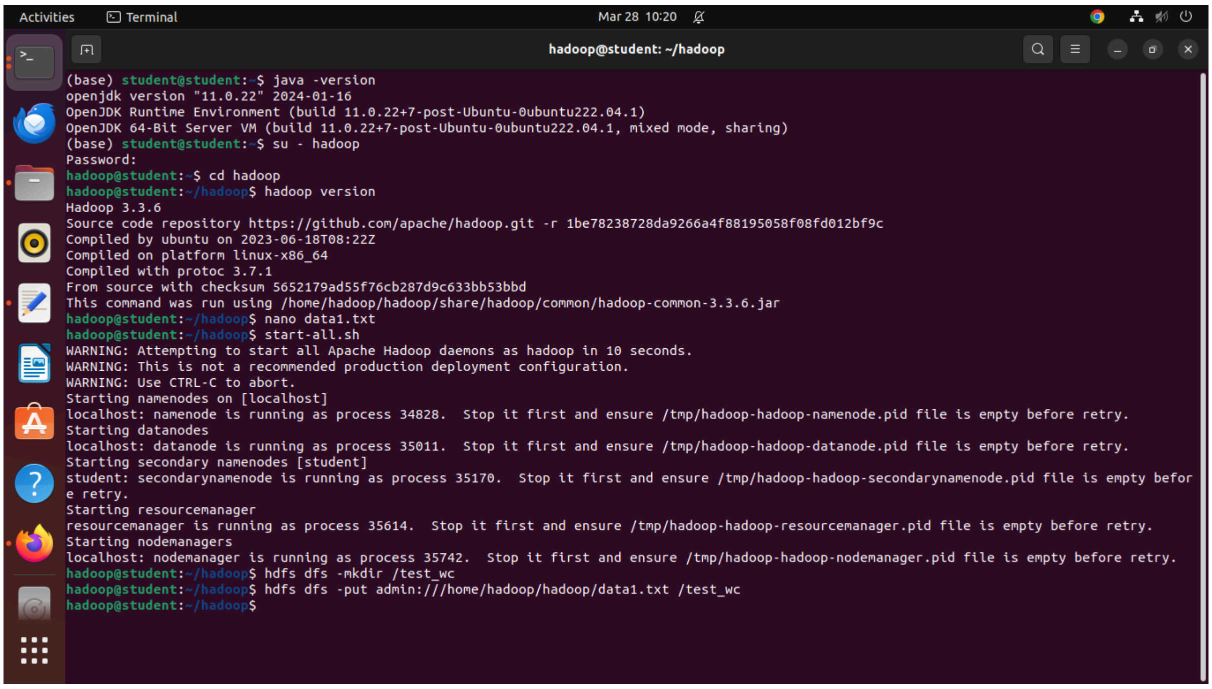
Here's what each part of the command does:

- `hdfs`: This command is used to interact with the Hadoop Distributed File System (HDFS).
- `dfs`: This sub-command specifies that the operation is related to the Hadoop Distributed File System.
- `-mkdir`: This option specifies that you want to create a directory.
- `/test_wc`: This is the path of the directory you want to create. The leading forward slash (/) indicates that the directory should be created in the root directory of HDFS.

So, when you run `hdfs dfs -mkdir /test_wc`, it creates a directory named "test_wc" in the root directory of HDFS.

hdfs dfs -put admin:///home/hadoop/hadoop/data1.txt /test_wc
Or
hdfs dfs -put /home/hadoop/hadoop/data1.txt /test_wc

data1.txt will be copied into the /test_wc directory in HDFS.



```
(base) student@student:~$ java -version
openjdk version "11.0.22" 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1, mixed mode, sharing)
(base) student@student:~$ su - hadoop
Password:
hadoop@student:~$ cd hadoop
hadoop@student:~/hadoop$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /home/hadoop/hadoop/share/hadoop/common/hadoop-common-3.3.6.jar
hadoop@student:~/hadoop$ nano data1.txt
hadoop@student:~/hadoop$ start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as hadoop in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
localhost: namenode is running as process 34828.  Stop it first and ensure /tmp/hadoop-hadoop-namenode.pid file is empty before retry.
Starting datanodes
localhost: datanode is running as process 35011.  Stop it first and ensure /tmp/hadoop-hadoop-datanode.pid file is empty before retry.
Starting secondary namenodes [student]
student: secondarynamenode is running as process 35170.  Stop it first and ensure /tmp/hadoop-hadoop-secondarynamenode.pid file is empty befor
e retry.
Starting resourcemanager
resourcemanager is running as process 35614.  Stop it first and ensure /tmp/hadoop-hadoop-resourcemanager.pid file is empty before retry.
Starting nodemanagers
localhost: nodemanager is running as process 35742.  Stop it first and ensure /tmp/hadoop-hadoop-nodemanager.pid file is empty before retry.
hadoop@student:~/hadoop$ hdfs dfs -mkdir /test_wc
hadoop@student:~/hadoop$ hdfs dfs -put admin:///home/hadoop/hadoop/data1.txt /test_wc
hadoop@student:~/hadoop$
```

Ifconfig
Copy the ip address

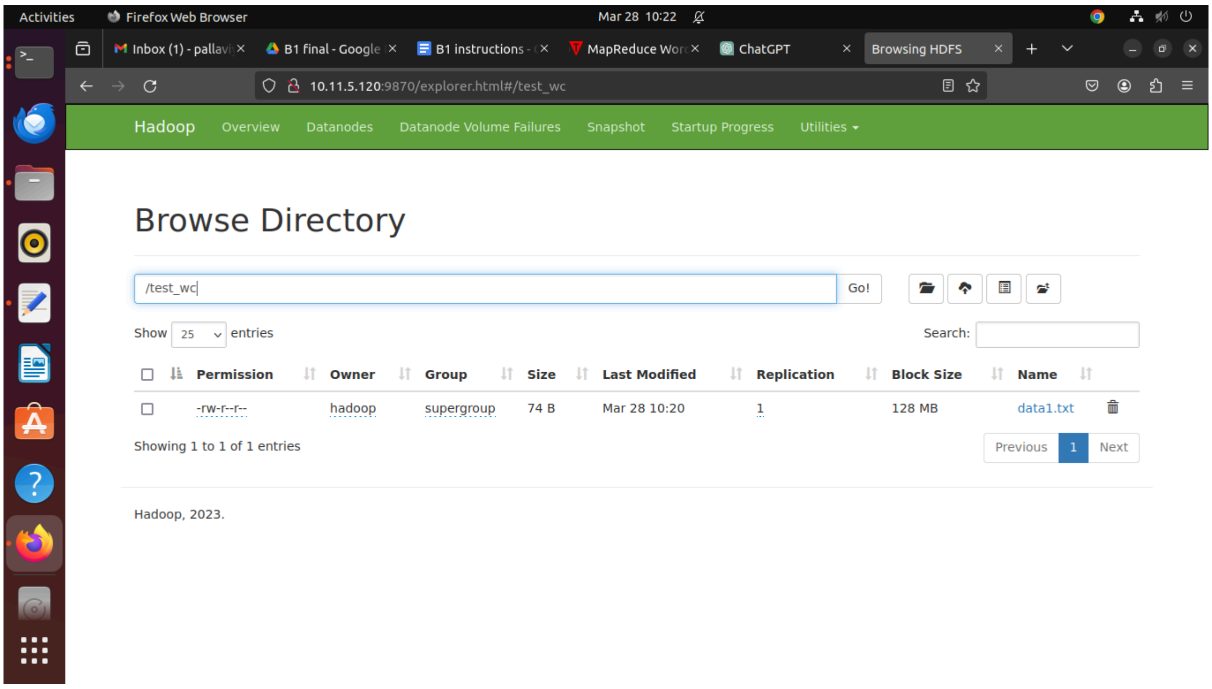
put following URL in firefox

Ip address followed by :9870

go to utilities

browse the file system

type /test_wc



Browse Directory

/test_wc Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	74 B	Mar 28 10:20	1	128 MB	data1.txt

Showing 1 to 1 of 1 entries Previous 1 Next

Hadoop, 2023.

nano WC_Mapper.java

```
package com.javatpoint;
```

/* This line specifies the package name where this Java class belongs. Packages are used for organizing classes into namespaces.*/

```
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;
```

/* These lines import necessary Java and Hadoop libraries that are required for the functionalities used in this class. For instance, `java.io.IOException` is imported for handling input/output exceptions, `java.util.StringTokenizer` is used to tokenize strings, and the `org.apache.hadoop.io` packages contain classes for various data types used in Hadoop MapReduce jobs. */

```
        public class WC_Mapper extends MapReduceBase implements
Mapper<LongWritable,Text,Text,IntWritable>{
```

/* This line declares a Java class named `WC_Mapper`. It extends `MapReduceBase`, which is a Hadoop class used as a base class for MapReduce mapper and reducer classes. It also implements the `Mapper` interface, specifying the input and output key-value types for the mapper. Here, `LongWritable` represents the input key type (offset of a line in the input file), `Text` represents the input value type (a line of text), `Text` represents the output key type (a word), and `IntWritable` represents the output value type (count of occurrences of the word). */

```
        private final static IntWritable one = new IntWritable(1);
```

/* This line declares a constant variable named `one` of type `IntWritable`, initialized with the value `1`. It is used to represent the count of each word, initialized to 1. */

```
        private Text word = new Text();
```

/* This line declares a variable named `word` of type `Text`. It is used to store each word extracted from the input text during the mapping process. */

```
        public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable>
output, Reporter reporter) throws IOException{
```

/* This line defines the `map` method required by the `Mapper` interface. It takes four parameters: `key` (representing the offset of a line in the input file), `value` (representing a line of text), `output` (used to collect output key-value pairs), and `reporter` (used for reporting progress and status). It throws `IOException` to handle input/output exceptions. */

```
        String line = value.toString();
```

/* This line converts the `Text` object `value` (representing a line of text) into a Java string named `line`. */

```
        StringTokenizer  tokenizer = new StringTokenizer(line);
```

/* This line creates a `StringTokenizer` object named `tokenizer`, which is used to tokenize the `line` into individual words based on whitespace. */

```
        while (tokenizer.hasMoreTokens()){
                word.set(tokenizer.nextToken());
                output.collect(word, one);
```

/* This block of code iterates over each tokenized word using the `StringTokenizer`. For each word, it sets the `word` variable to the current word using `word.set()`, and then it collects the key-value pair `(word, one)` using the `OutputCollector`. This effectively emits each word with a count of 1, which is later aggregated in the reducer phase to calculate the total count of each word. */

```
        }
        }

        }
```

control+s then control+x

nano WC_Reducer.java

```
        package com.javatpoint;
        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 {
```

/* This line defines the `reduce` method, which is a part of the reducer class in a Hadoop MapReduce program.

- `Text key`: This parameter represents the key for this particular invocation of the `reduce` method. In a word count example like this, it represents a unique word.
- `Iterator<IntWritable> values`: This parameter represents an iterator over the list of values associated with the `key`. In this case, it iterates over the counts of occurrences of the word represented by the `key`.

- `OutputCollector<Text, IntWritable> output`: This parameter is used to collect the output key-value pairs produced by the reducer. The reducer aggregates the values for each key (word) and emits the final key-value pairs.
- `Reporter reporter`: This parameter is used for reporting progress and status of the reducer job to the Hadoop framework.
- `throws IOException`: This method may throw an `IOException` in case of input/output errors. */

```
int sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
```

/* This loop iterates through all the values associated with the given key. For each value, it adds the integer value retrieved by `get()` method of `IntWritable` to the `sum` variable. This effectively calculates the total count of occurrences of the word represented by the `key`. */

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

/* After summing up all the counts for the current word, this line emits the final key-value pair. The key remains the same (representing the word), while the value is the total count of occurrences (`sum`). This key-value pair is collected using the `output` object, which is an instance of `OutputCollector`. This output will be passed on to the Hadoop framework to be written to the output file. */

```
}
}
```

Nano WC_Runner.java

```
package com.javatpoint;

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 {
```

/ * This line declares a Java class named `WC_Runner`, which serves as the entry point for running the MapReduce job. */

```
    public static void main(String[] args) throws IOException{
```

/* This line defines the `main` method, which is the starting point of execution for the Java program. It takes an array of strings `args` as input arguments and may throw an `IOException`. */

```
        JobConf conf = new JobConf(WC_Runner.class);
```

/* This line creates a new instance of `JobConf`, which is a configuration class for a MapReduce job. The constructor takes the class name of the job as an argument (`WC_Runner.class` in this case). */

```
        conf.setJobName("WordCount");
```

/* This line sets the name of the MapReduce job to "WordCount" using the `setJobName` method of the `JobConf` object `conf`. */

```
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
```

/* These lines set the output key and value classes for the MapReduce job. In this case, the output key is of type `Text` (representing words) and the output value is of type `IntWritable` (representing counts). */

```
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
```

/* These lines specify the mapper, combiner, and reducer classes for the MapReduce job. `WC_Mapper.class` is set as the mapper class, `WC_Reducer.class` is set as both the combiner and the reducer class. This indicates that the same reducer logic will be applied as the combiner logic for intermediate aggregation. */

```
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
```

/* These lines specify the mapper, combiner, and reducer classes for the MapReduce job. `WC_Mapper.class` is set as the mapper class, `WC_Reducer.class` is set as both the combiner and the reducer class. This indicates that the same reducer logic will be applied as the combiner logic for intermediate aggregation.. */

```
                FileInputFormat.setInputPaths(conf,new Path(args[0]));
                FileOutputFormat.setOutputPath(conf,new Path(args[1]));
```

/* These lines specify the input and output paths for the MapReduce job. The input path is taken from the first argument (`args[0]`) provided in the command-line arguments, and the output path is taken from the second argument (`args[1]`). Both paths are converted to `Path` objects using the `Path` class constructor. */

```
                JobClient.runJob(conf);
```

/* Finally, this line runs the MapReduce job with the configuration specified in the `conf` object using the `runJob` method of `JobClient`. This initiates the execution of the MapReduce job. */

```
        }
        }
```

```
javac -classpath "$(hadoop classpath)" -d . WC_Mapper.java WC_Reducer.java
WC_Runner.java
```

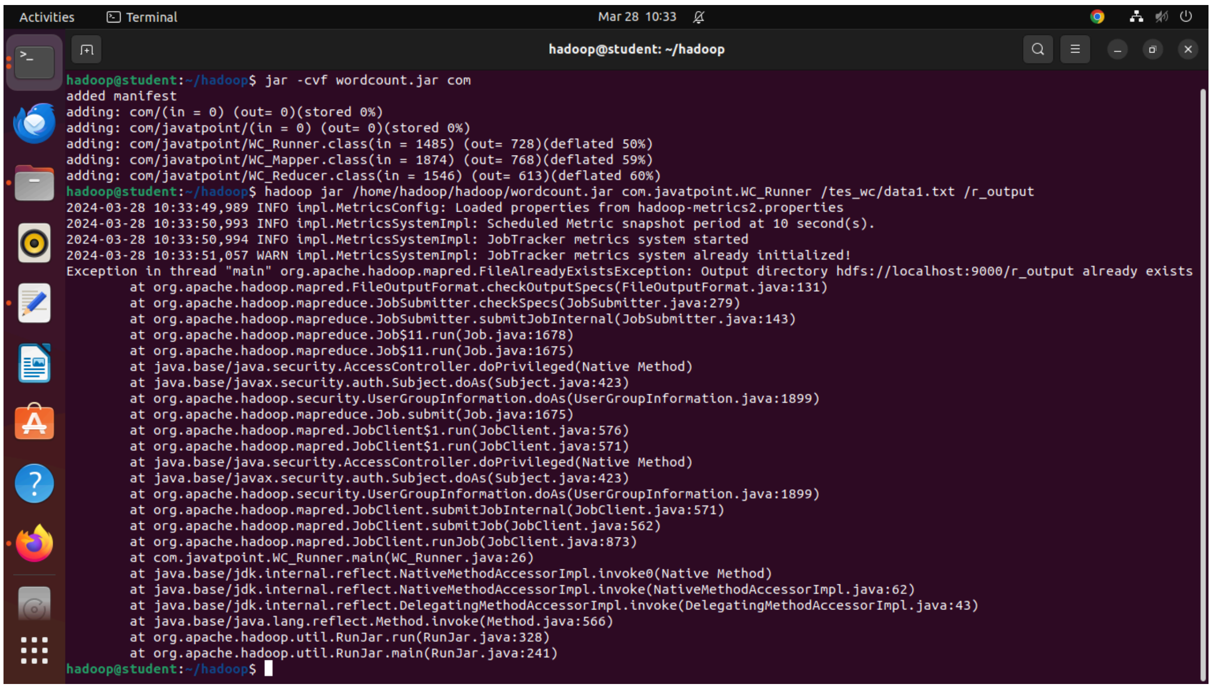
/* This command compiles the three Java files (`WC_Mapper.java`, `WC_Reducer.java`, and `WC_Runner.java`) using the Hadoop classpath for resolving dependencies and places the compiled `.class` files in the current directory. */

```
jar -cvf wordcount.jar com
```

/* By running this command, a JAR file named `wordcount.jar` will be created, containing all the contents of the `com` package directory and its subdirectories. This JAR file can then be used to distribute and execute the Hadoop MapReduce application. */

```
hadoop jar /home/hadoop/hadoop/wordcount.jar com.javatpoint.WC_Runner
/test_wc/data1.txt /r_output
```

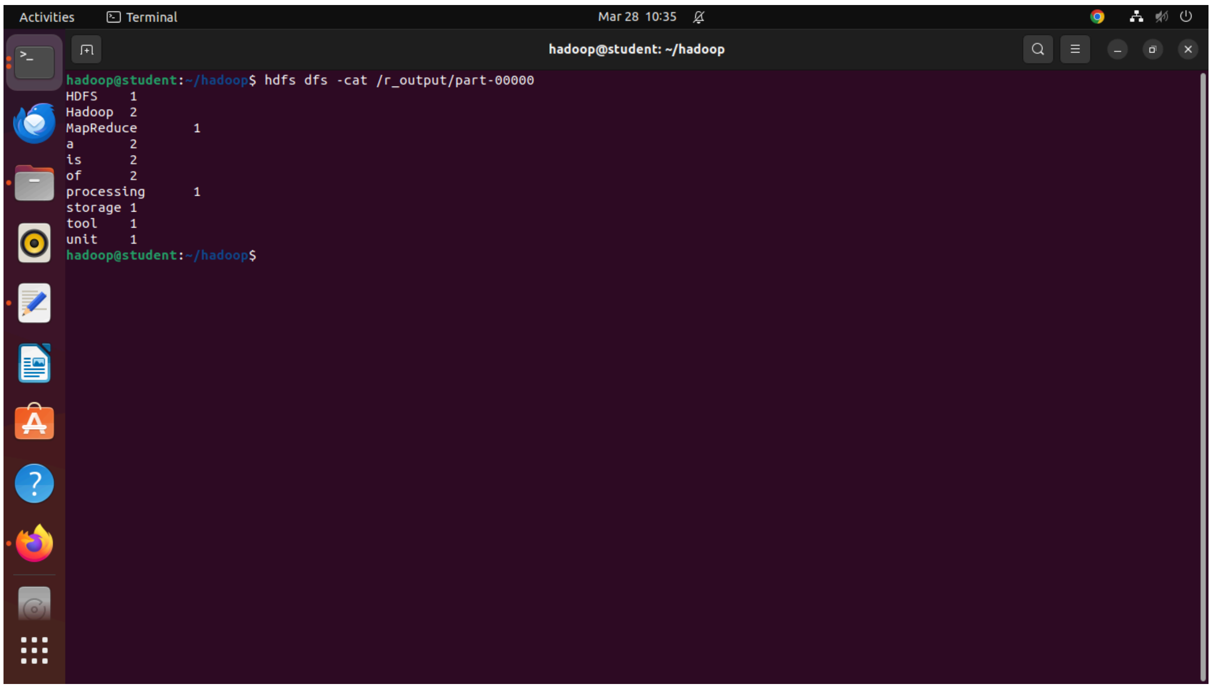
/* This command executes the Hadoop MapReduce job defined in the JAR file `/home/hadoop/hadoop/wordcount.jar`, using the `com.javatpoint.WC_Runner` class as the main entry point. It takes `/tes_wc/data1.txt` as input and writes the output to `/r_output`. */



```
hadoop@student:~/hadoop$ jar -cvf wordcount.jar com
added manifest
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/javatpoint/(in = 0) (out= 0)(stored 0%)
adding: com/javatpoint/WC_Runner.class(in = 1485) (out= 728)(deflated 50%)
adding: com/javatpoint/WC_Mapper.class(in = 1874) (out= 768)(deflated 59%)
adding: com/javatpoint/WC_Reducer.class(in = 1546) (out= 613)(deflated 60%)
hadoop@student:~/hadoop$ hadoop jar /home/hadoop/hadoop/wordcount.jar com.javatpoint.WC_Runner /tes_wc/data1.txt /r_output
2024-03-28 10:33:49,989 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2024-03-28 10:33:50,993 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2024-03-28 10:33:50,994 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2024-03-28 10:33:51,057 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://localhost:9000/r_output already exists
        at org.apache.hadoop.mapred.FileOutputFormat.checkOutputSpecs(FileOutputFormat.java:131)
        at org.apache.hadoop.mapreduce.JobSubmitter.checkSpecs(JobSubmitter.java:279)
        at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:143)
        at org.apache.hadoop.mapreduce.Job$11.run(Job.java:1678)
        at org.apache.hadoop.mapreduce.Job$11.run(Job.java:1675)
        at java.base/java.security.AccessController.doPrivileged(Native Method)
        at java.base/javax.security.auth.Subject.doAs(Subject.java:423)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1899)
        at org.apache.hadoop.mapreduce.Job.submit(Job.java:1675)
        at org.apache.hadoop.mapred.JobClient$1.run(JobClient.java:576)
        at org.apache.hadoop.mapred.JobClient$1.run(JobClient.java:571)
        at java.base/java.security.AccessController.doPrivileged(Native Method)
        at java.base/javax.security.auth.Subject.doAs(Subject.java:423)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1899)
        at org.apache.hadoop.mapred.JobClient.submitJobInternal(JobClient.java:571)
        at org.apache.hadoop.mapred.JobClient.submitJob(JobClient.java:562)
        at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:873)
        at com.javatpoint.WC_Runner.main(WC_Runner.java:26)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:566)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:328)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:241)
hadoop@student:~/hadoop$
```

hdfs dfs -cat /r_output/part-00000

/* when you run `hdfs dfs -cat /r_output/part-00000`, you'll see the contents of the `/r_output/part-00000` file displayed in the terminal. This file should contain the output of your MapReduce job, which, in the case of a word count program, would likely consist of word-count pairs. */



```
hadoop@student:~/hadoop$ hdfs dfs -cat /r_output/part-00000
HDFS    1
Hadoop  2
MapReduce       1
a       2
is      2
of      2
processing      1
storage 1
tool    1
unit    1
hadoop@student:~/hadoop$
```