BIG DATA ANALYTICS

LAB ASSIGNMENT 4

Mahesh Pachare

FINAL YEAR B.TECH IT

191080054

AIM: Write a Map Reduce code to find top five trending hashtags on twitter.

THEORY:

MapReduce is a parallel processing framework that is used to process large amounts of data across a cluster of computers. It was originally developed by Google and has since become a popular tool for big data processing.

The MapReduce programming model is based on two main operations: Map and Reduce. The Map operation takes in a dataset and maps each element in the dataset to a set of intermediate key-value pairs. The Reduce operation takes the intermediate key-value pairs and aggregates them into a final set of outputs. The map and reduce operations are performed in parallel across a large number of nodes in the cluster, allowing for efficient and scalable processing of large datasets.

MapReduce Architecture

MapReduce process has the following phases:

- Input Splits
- Mapping
- Shuffling and Sorting
- Reducing

Input Splits

MapReduce splits the input into smaller chunks called input splits, representing a block of work with a single mapper task.

Mapping

The input data is processed and divided into smaller segments in the mapper phase, where the number of mappers is equal to the number of input splits.

RecordReader produces a key-value pair of the input splits using TextFormat, which Reducer later uses as input. The mapper then processes these key-value pairs using coding logic to produce an output of the same form.

Shuffling

In the shuffling phase, the output of the mapper phase is passed to the reducer phase by removing duplicate values and grouping the values. The output remains in the form of keys and values in the mapper phase. Since shuffling can begin even before the mapper phase is complete, it saves time.

Sorting

Sorting is performed simultaneously with shuffling. The Sorting phase involves merging and sorting the output generated by the mapper. The intermediate key-value pairs are sorted by key before starting the reducer phase, and the values can take any order. Sorting by value is done by secondary sorting.

Reducing

In the reducer phase, the intermediate values from the shuffling phase are reduced to produce a single output value that summarizes the entire dataset. HDFS is then used to store the final output.

Limitations of MapReduce

MapReduce also faces some limitations, and they are as follows:

- ➔ MapReduce is a low-level programming model which involves a lot of writing code.
- ➔ The batch-based processing nature of MapReduce makes it unsuitable for real-time processing.
- ➔ It does not support data pipelining or overlapping of Map and Reduce phases.

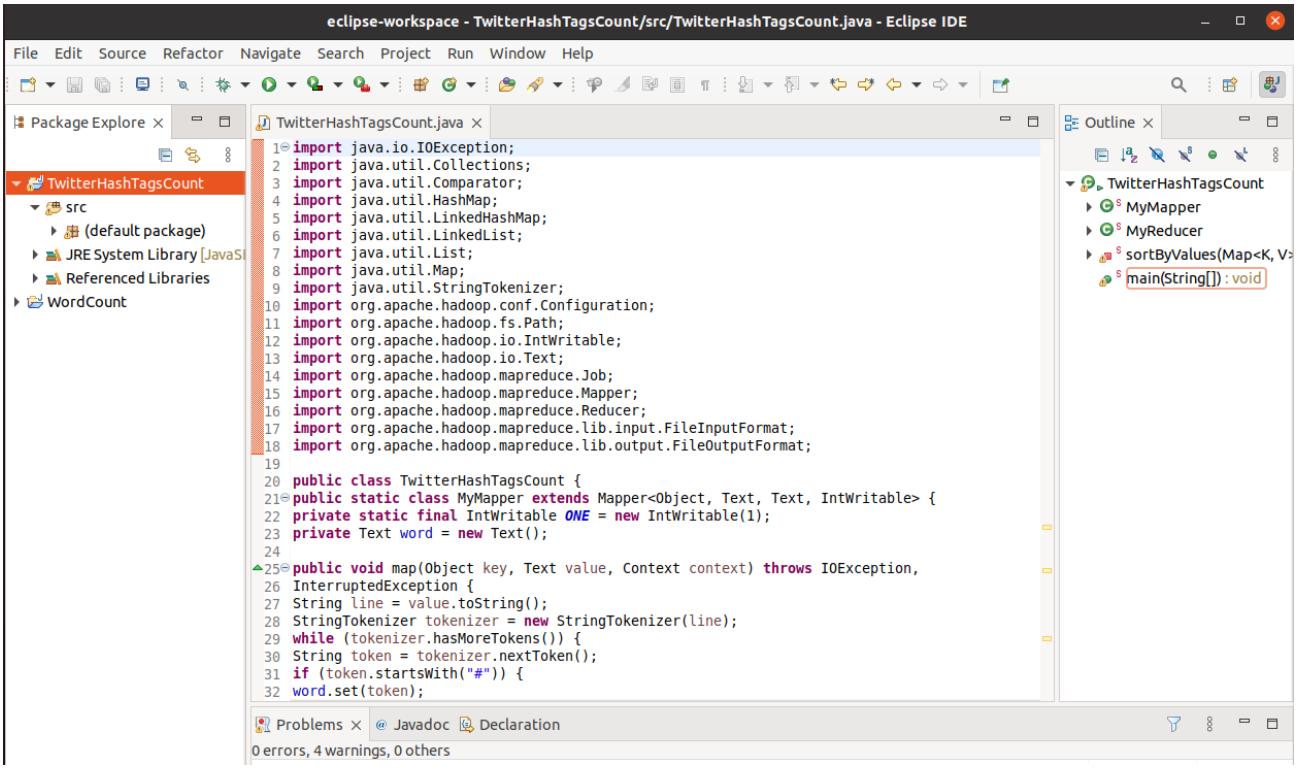
- ➔ Task initialization, coordination, monitoring, and scheduling take up a large chunk of MapReduce's execution time and reduce its performance.
- ➔ MapReduce cannot cache the intermediate data in memory, thereby diminishing Hadoop's performance.

Tweet Dataset Link :
https://www.kaggle.com/code/ludovicocuoghi/twitter-sentiment-analysis-with-bert-roberta/input?select=Corona_NLP_test.csv

PROCEDURE:

After setting up hadoop multinode cluster in experiment 2, now we are running the map reduce code to find the top five twitter hashtags from the dataset.
The dataset have used COVID-19 dataset for sentiment analysis, which contains tweets along with other details such as location of the person, date on which tweet was made and so on.

1. Start the eclipse and create a folder for the java project.
2. Import the necessary jar files from hadoop
3. Write the code, where in the mapper class the input text is tokenized and checked for hashtags, and if a hashtag is found, it is emitted as a key-value pair with a count as 1. The reducer class aggregates the input key value pairs by summing the counts of the same hashtag.



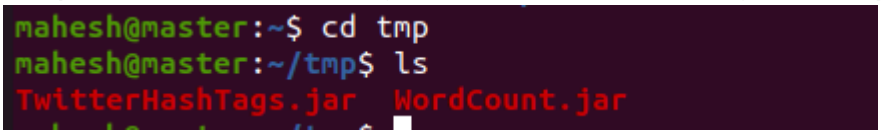
Code:

```
package twitterHashTags;

import java.io.IOException;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
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 TwitterHashTagsCount {
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
private static final IntWritable ONE = new IntWritable(1);
private Text word = new Text();
```

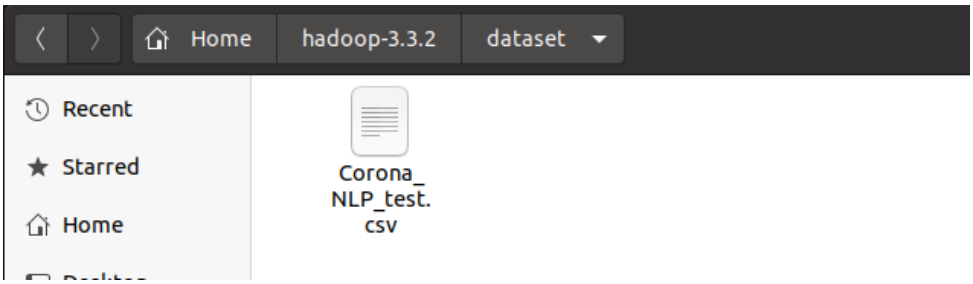
```
public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
String token = tokenizer.nextToken();
if (token.startsWith("#")) {
word.set(token);
context.write(word, ONE);
}
}
}

}
public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
private Map<Text, IntWritable> countMap = new HashMap<>();
@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
countMap.put(new Text(key), new IntWritable(sum));
}
@Override
protected void cleanup(Context context) throws IOException,
InterruptedException {
Map<Text, IntWritable> sortedMap = sortByValues(countMap);
int counter = 0;
for (Text key : sortedMap.keySet()) {
if (counter++ == 5) {
break;
}
context.write(key, sortedMap.get(key));
}
}
}
private static <K extends Comparable, V extends Comparable> Map<K, V>
sortByValues(Map<K, V> map) {
List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());

Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {
@Override
public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {
return o2.getValue().compareTo(o1.getValue());
}
});
Map<K, V> sortedMap = new LinkedHashMap<K, V>();
for (Map.Entry<K, V> entry : entries) {
sortedMap.put(entry.getKey(), entry.getValue());
}
return sortedMap;
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf);
job.setJarByClass(TwitterHashTagsCount.class);
```

```
job.setMapperClass(MyMapper.class);
job.setReducerClass(MyReducer.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);
}
}
```

4. Export the code as a jar file to run it using hadoop.



5. Download the data from Kaggle and place it in hadoop directory in a new folder Dataset.

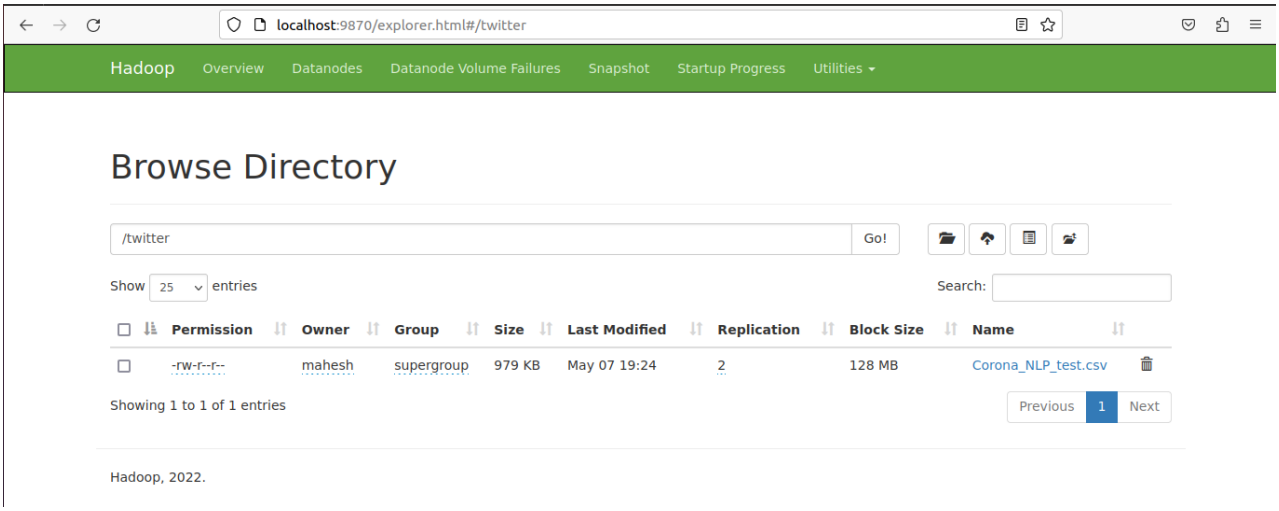


6. Starting multinode cluster of hadoop.

7. Copying the data from “/usr/local/hadoop/Dataset” folder to “/Data” folder of hadoop using hadoop dfs command in hadoop directory.

```
mahesh@master:~/hadoop-3.3.2$ hadoop dfs -copyFromLocal dataset /twitter
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

mahesh@master:~/hadoop-3.3.2$ hadoop dfs -ls /twitter
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

Found 1 items
-rw-r--r--   2 mahesh supergroup    1002494 2023-05-07 19:24 /twitter/Corona_NLP_test.csv
```

8. Viewing the data in hadoop directory using web interface.

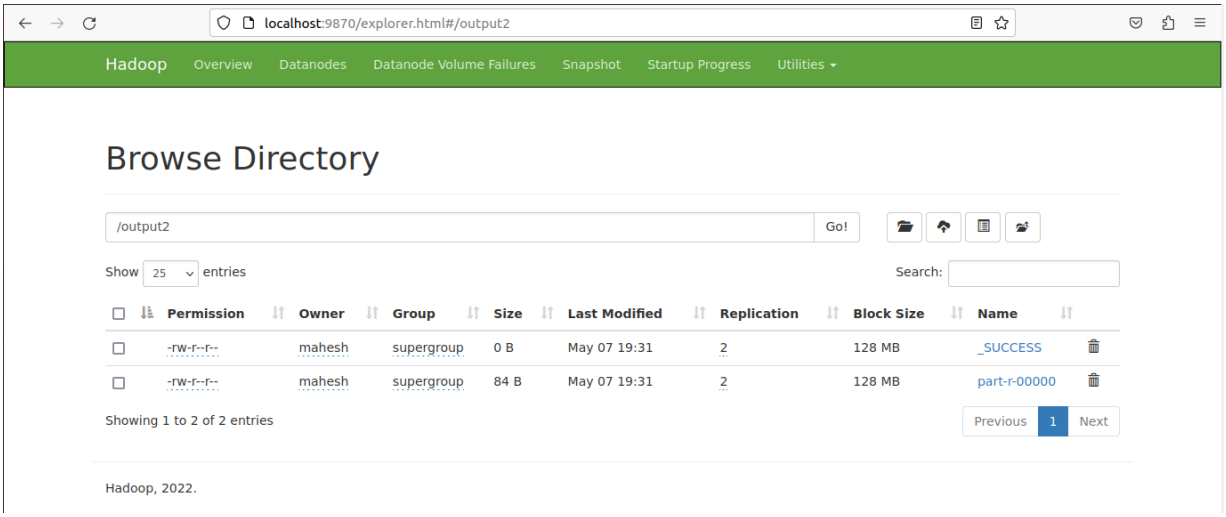


9. Running the mapReduce task by supplying the Word Count jar file created above and providing input and output directories.

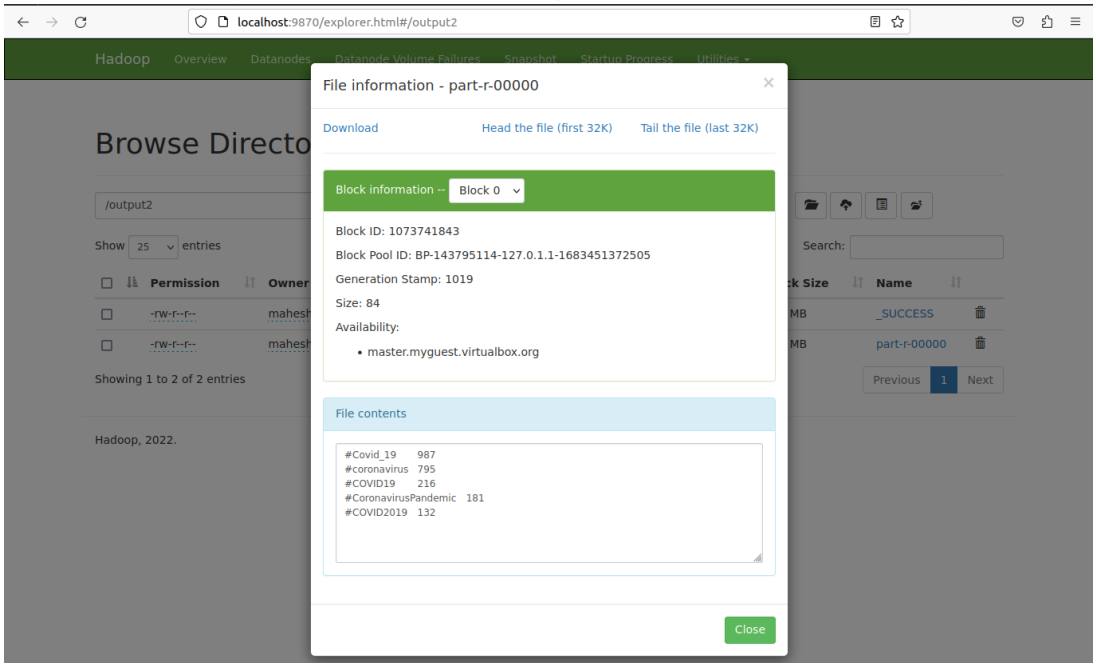
```
mahesh@master:~/hadoop-3.3.2$ hadoop jar /home/mahesh/tmp/TwitterHashTags.jar TwitterHashTagsCount /twitter /output2
2023-05-07 19:29:25,671 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at master/127.0.1.1:8032
2023-05-07 19:29:33,552 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute
 your application with ToolRunner to remedy this.
2023-05-07 19:29:33,673 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/mahesh/.staging/job_1683462450
968_0002
2023-05-07 19:29:35,736 INFO input.FileInputFormat: Total input files to process : 1
2023-05-07 19:29:37,340 INFO mapreduce.JobSubmitter: number of splits:1
2023-05-07 19:29:38,792 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1683462450968_0002
2023-05-07 19:29:38,800 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-05-07 19:29:47,863 INFO conf.Configuration: resource-types.xml not found
2023-05-07 19:29:47,884 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-05-07 19:29:51,130 INFO impl.YarnClientImpl: Submitted application application_1683462450968_0002
2023-05-07 19:29:51,394 INFO mapreduce.Job: The url to track the job: http://master.myguest.virtualbox.org:8088/proxy/application_1683462450968_0002/
2023-05-07 19:29:51,397 INFO mapreduce.Job: Running job: job_1683462450968_0002
2023-05-07 19:31:48,682 INFO mapreduce.Job: Job job_1683462450968_0002 running in uber mode : false
2023-05-07 19:31:48,690 INFO mapreduce.Job:  map 0% reduce 0%
2023-05-07 19:31:54,819 INFO mapreduce.Job:  map 100% reduce 0%
2023-05-07 19:31:59,870 INFO mapreduce.Job:  map 100% reduce 100%
2023-05-07 19:32:00,903 INFO mapreduce.Job: Job job_1683462450968_0002 completed successfully
2023-05-07 19:32:01,072 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=151381
                FILE: Number of bytes written=852447
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1002605
                HDFS: Number of bytes written=84
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3219
                Total time spent by all reduces in occupied slots (ms)=2813
                Total time spent by all map tasks (ms)=3219
                Total time spent by all reduce tasks (ms)=2813
                Total vcore-milliseconds taken by all map tasks=3219
```

```
                Total time spent by all maps in occupied slots (ms)=3219
                Total time spent by all reduces in occupied slots (ms)=2813
                Total time spent by all map tasks (ms)=3219
                Total time spent by all reduce tasks (ms)=2813
                Total vcore-milliseconds taken by all map tasks=3219
                Total vcore-milliseconds taken by all reduce tasks=2813
                Total megabyte-milliseconds taken by all map tasks=3296256
                Total megabyte-milliseconds taken by all reduce tasks=2880512
        Map-Reduce Framework
                Map input records=12392
                Map output records=7618
                Map output bytes=136139
                Map output materialized bytes=151381
                Input split bytes=111
                Combine input records=0
                Combine output records=0
                Reduce input groups=2503
                Reduce shuffle bytes=151381
                Reduce input records=7618
                Reduce output records=5
                Spilled Records=15236
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=104
                CPU time spent (ms)=1400
                Physical memory (bytes) snapshot=481652736
                Virtual memory (bytes) snapshot=5065596928
                Total committed heap usage (bytes)=378535936
                Peak Map Physical memory (bytes)=293601280
                Peak Map Virtual memory (bytes)=2530172928
                Peak Reduce Physical memory (bytes)=188051456
                Peak Reduce Virtual memory (bytes)=2535424000
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1002494
        File Output Format Counters
                Bytes Written=84
mahesh@master:~/hadoop-3.3.2$ ls
```

10. After completing the Map Reduce job viewing the output in the web interface which displays the occurrence of each word in the provided input directory.



11. Viewing the actual five top twitter hashtags in logs.



CONCLUSION:

In this experiment, map reduce code for finding the top five most occurring twitter hashtags is written and, a jar file is created, a dataset is used and then map reduce job is executed on hadoop multinode cluster.