BI and Big Data Analytics

Assignment Word Count

BSc Information Systems Level 3

Name: Tencamah Reean
Student ID: 1711359

1.0 Introduction

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring the m and re-executes the failed tasks.

1.1 Hadoop Map Reduce

MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage .

Map stage- The text from the input text file is tokenized into words to form a key value pair with all the words present in the input text file. The key is the word from the input file and value is '1'.

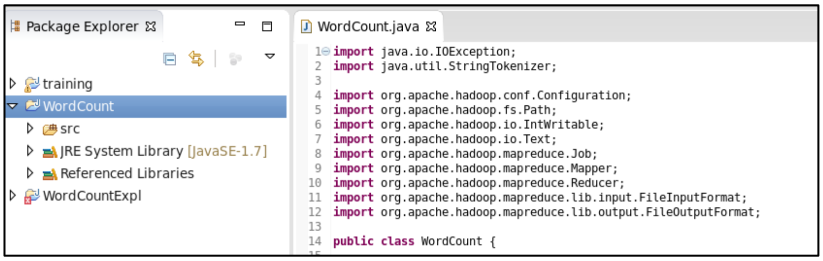
Shuffle stage- The key-value pairs generated in the map phase are taken as input and then sorted in alphabetical order and map output from Mapper to Reducer.

Reducer stage- It takes the output of shuffle phase as input and then reduces the key-value pairs to unique keys with values added up.

2.0 Create WordCount Example in Hadoop MapReduce using Java Project with Eclipse

Steps:

1. Create java project and named it **WordCount**
2. Add the required Hadoop jars in the project.
3. Create **WordCount class** in the project.
4. Import the required libraries in **WordCount.java**



```java
import java.io.IOException;
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 WordCount {
```

Figure 1: import libraries in WordCount.java

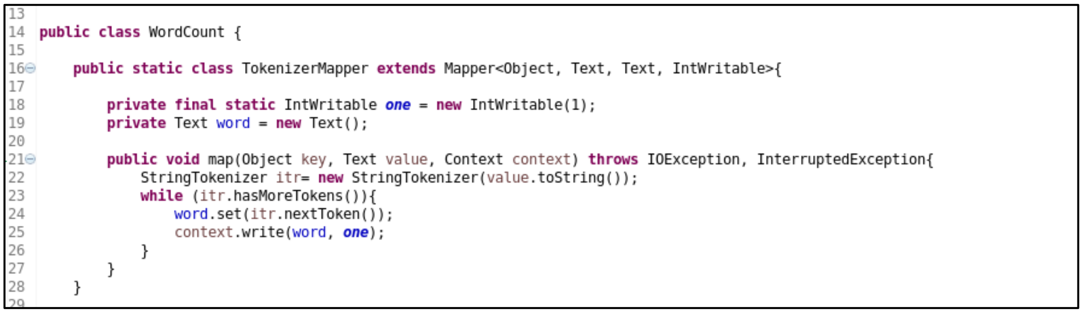
5. Create **TokenizeMapper class** which extends **Mapper** (super class) where the objects are passed in it.

    Mapper <Object, Text, Text, IntWritable>,

    Object, Text -> Input

    Text, IntWritable (value 1) -> Output

6. Create **map** method within TokenizeMapper class which named the object address as a **key** and actual content as a **value** and it throws an IOException. Then convert the block of content to string and break them into a word.



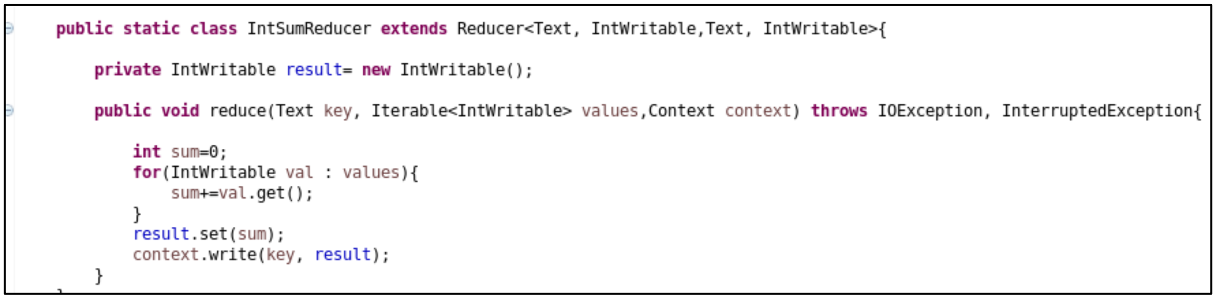
```
public class WordCount {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

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

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

Figure 2: Tokenize Mapper Class

7. Create **IntSumReducer class** which extends **Reducer** (input will be output of the Mapper, output will be the word and its number of occurrences).
8. Create a method **reduce** which take Text as a key and iterate IntWritable and take it as the values.

    Iterable Function:
    Example: input: hello word hello word
    Output before iterate: hello 1 word 1 hello 1 word 1
    Output after iterate: hello 2 word 2

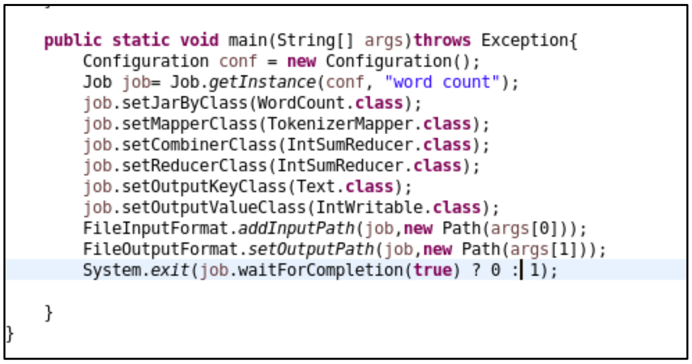
    And create a for loop which iterate through each and every value and sums up the values and stored it in variable result (IntWritable).



```
    public static class IntSumReducer extends Reducer<Text, IntWritable,Text, IntWritable>{

        private IntWritable result= new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,Context context) throws IOException, InterruptedException{

            int sum=0;
            for(IntWritable val : values){
                sum+=val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
```

Figure 3: IntSumReducer class

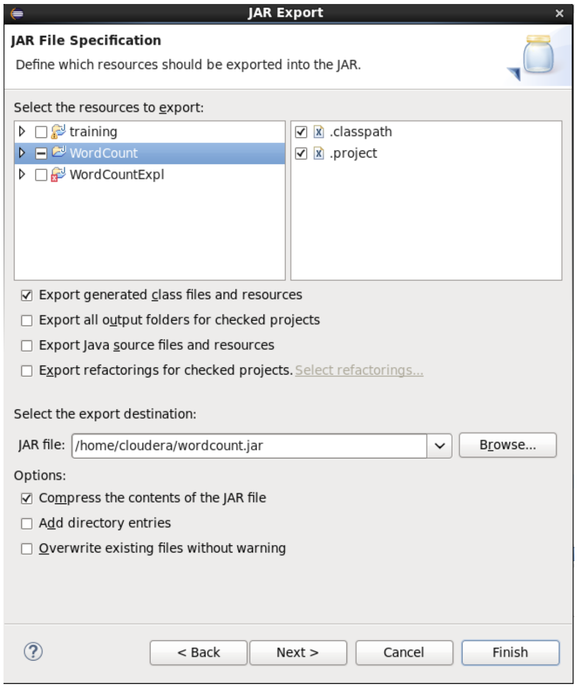
9. Create the **main** method for the Wordcount. Create an object conf which take the class name '**WordCount'** of the driver class and set it as a jar in a job. Then set the mapper class, reducer class, output key and output value respectively. Set the input and output path to the job and save the job.



```
    public static void main(String[] args)throws Exception{
        Configuration conf = new Configuration();
        Job job= Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.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);

    }
}
```

Figure 4: main class

10. Export the WordCount project as a jar file. Location: /home/cloudera/



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- □ training
- ▣ WordCount
- □ WordCountExpl

- ☑ .classpath
- ☑ .project

☑ Export generated class files and resources

□ Export all output folders for checked projects

□ Export Java source files and resources

□ Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: /home/cloudera/wordcount.jar

Options:

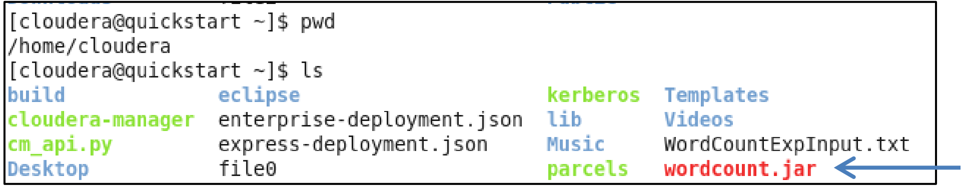
☑ Compress the contents of the JAR file

□ Add directory entries

□ Overwrite existing files without warning

Figure 5: Export file as a jar

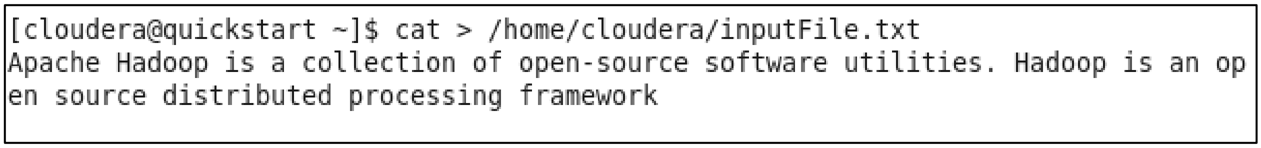
11. Open terminal, verify the file in the location.



```
[cloudera@quickstart ~]$ pwd
/home/cloudera
[cloudera@quickstart ~]$ ls
build                   eclipse                       kerberos  Templates
cloudera-manager        enterprise-deployment.json    lib       Videos
cm_api.py               express-deployment.json       Music     WordCountExpInput.txt
Desktop                 file0                         parcels   wordcount.jar
```

Figure 6: verify file

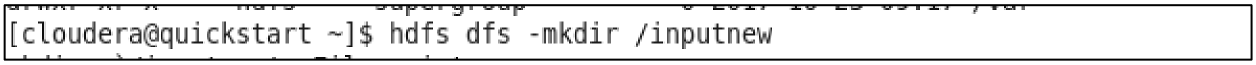
12. Create an input file for MapReduce Program and save in location /home/cloudera/



```
[cloudera@quickstart ~]$ cat > /home/cloudera/inputFile.txt
Apache Hadoop is a collection of open-source software utilities. Hadoop is an op
en source distributed processing framework
```

Figure 7: create input file

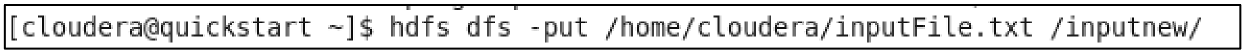
13. Create input directory in hdfs using command *hdfs dfs –mkdir /inputnew*.



```
[cloudera@quickstart ~]$ hdfs dfs -mkdir /inputnew
```

Figure 8: create input directory

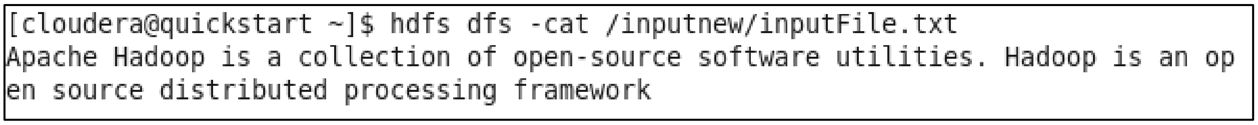
14. Move the inputFile.txt to hdfs.



```
[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/inputFile.txt /inputnew/
```

Figure 9: Move file to hdfs

15. View the file in hdfs.



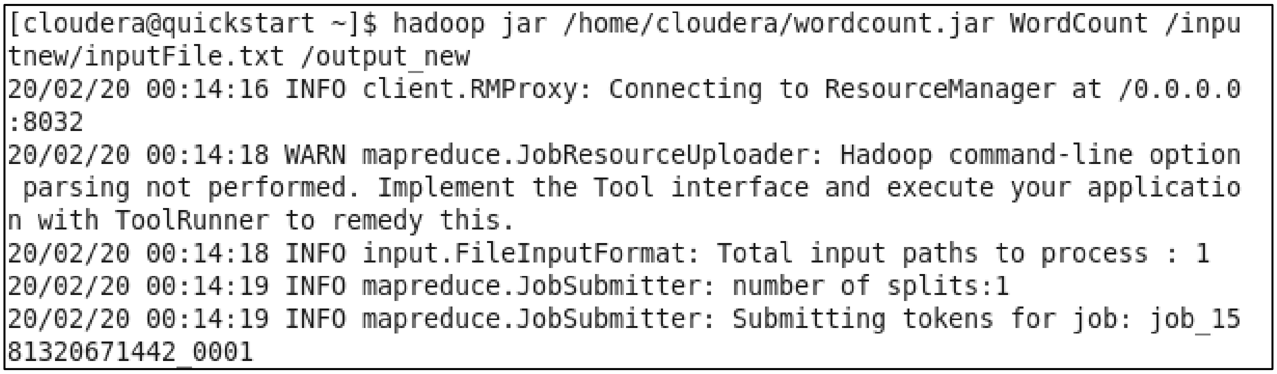
```
[cloudera@quickstart ~]$ hdfs dfs -cat /inputnew/inputFile.txt
Apache Hadoop is a collection of open-source software utilities. Hadoop is an op
en source distributed processing framework
```

Figure 10: View file in hdfs

16. Run MapReduce program.

    Syntax: *hadoop jar /home/cloudera/worcount.jar WordCount /inputnew/inputFile.txt /output_new*

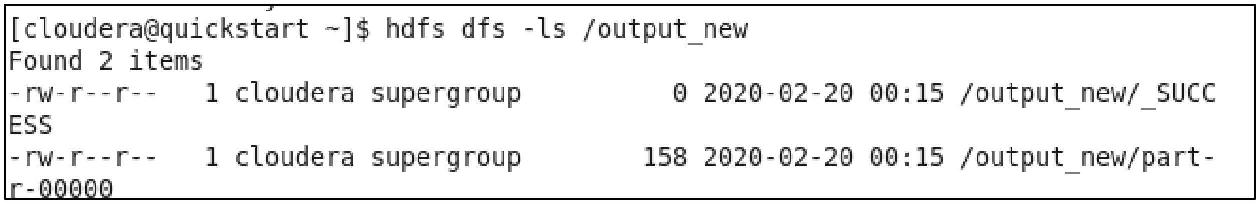
    Output directory: output_new



```
[cloudera@quickstart ~]$ hadoop jar /home/cloudera/wordcount.jar WordCount /inpu
tnew/inputFile.txt /output_new
20/02/20 00:14:16 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
20/02/20 00:14:18 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your applicatio
n with ToolRunner to remedy this.
20/02/20 00:14:18 INFO input.FileInputFormat: Total input paths to process : 1
20/02/20 00:14:19 INFO mapreduce.JobSubmitter: number of splits:1
20/02/20 00:14:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_15
81320671442_0001
```

Figure 11: Run map reduce progarm

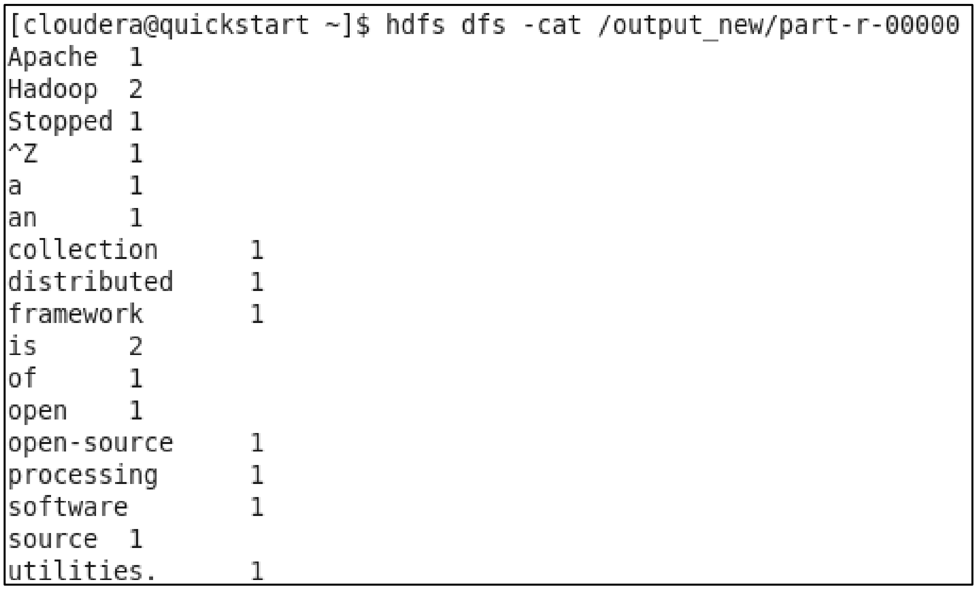
17. View the output_new directory



```
[cloudera@quickstart ~]$ hdfs dfs -ls /output_new
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2020-02-20 00:15 /output_new/_SUCC
ESS
-rw-r--r--   1 cloudera supergroup        158 2020-02-20 00:15 /output_new/part-
r-00000
```

Figure 12: View output_new directory

18. Finally, view the output file.
    Syntax: *hdfs dfs –cat/output_new/part-r-00000*



```
[cloudera@quickstart ~]$ hdfs dfs -cat /output_new/part-r-00000
Apache  1
Hadoop  2
Stopped 1
^Z      1
a       1
an      1
collection      1
distributed     1
framework       1
is      2
of      1
open    1
open-source     1
processing      1
software        1
source  1
utilities.      1
```

Figure 13: View output file