Experiment No: 3

Name: Alston Fernandes **Roll No: 19**

Topic:	Implementing distinct word count using MapReduce.
Prerequisite:	Basic Java programming, Hadoop and MapReduce knowledge is required
Mapping With COs:	CSL702.2
Objective:	To write the program for mapper class, reducer class and driver class to find the number of distinct words present in the input file.
Outcomes:	Students will be able to write the program for mapper class, reducer class and driver class to find the number of distinct words present in the input file and be able to produce part-r file as output.
Instructions:	This experiment is a compulsory experiment. All the students are required to perform this experiment individually.

Deliverables:	1. Give the details of the input file for this program. **input.txt** dog cat deer car cat car dog boy dog car welcome all all is well well done cat deer deer is car cat dog car done cat is dog well deer dog cat deer car cat car dog boy dog car welcome all all is well well done cat deer deer is car cat dog car done cat is dog well deer dog cat deer

	car cat car dog boy dog car welcome all all is well well done cat deer deer is car cat dog car done cat is dog well deer dog cat deer car cat car dog boy dog car welcome all all is well well done cat deer deer is car cat dog car done cat is dog well deer dog cat deer car cat car dog boy dog car

```
welcome all
all is well
well
done
cat deer
deer is car
cat dog
car done
cat is dog
well deer
dog cat
deer
car
cat car
dog
boy dog car
welcome all
all is well
well
done
cat deer
deer is car
cat dog
car done
cat is dog
well deer
```

2. Write the code of mapper, reducer and driver.

```
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;
import org.apache.hadoop.util.GenericOptionsParser;


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);
       }
       }
  }


  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);
      }
  }


  public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      String[] otherArgs = new GenericOptionsParser(conf,
args).getRemainingArgs();
      if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...]
<out>");
      System.exit(2);
      }
      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);
      for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new
Path(otherArgs[i]));
      }
      FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
      System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

3. Output for this program.(Snapshots)

Starting hadoop

```
dbit@complab3:~$ hdfs namenode -format
23/09/10 12:17:03 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = complab3/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.7
STARTUP_MSG:   classpath = /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/common/lib/sl
share/hadoop/common/lib/hamcrest-core-1.3.jar:/usr/local/hadoop/share/hadoop/common/lib/jackson-mapp
/hadoop/common/lib/log4j-1.2.17.jar:/usr/local/hadoop/share/hadoop/common/lib/paranamer-2.3.jar:/usr
```

```
dbit@complab3:~$ start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
dbit@localhost's password:
localhost: starting namenode, logging to /usr/local/hadoop/logs/hadoop-dbit-namenode-complab3.out
dbit@localhost's password:
localhost: starting datanode, logging to /usr/local/hadoop/logs/hadoop-dbit-datanode-complab3.out
Starting secondary namenodes [0.0.0.0]
dbit@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-dbit-secondarynamenode-complab3.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-dbit-resourcemanager-complab3.out
dbit@localhost's password:
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-dbit-nodemanager-complab3.out
dbit@complab3:~$
```

Put file into HDFS

```
dbit@complab3:~$ hadoop fs -mkdir /WordCountExp
dbit@complab3:~$ hadoop fs -put Desktop/WordCount/input.txt /WordCountExp
dbit@complab3:~$
```

Compile file

```
dbit@complab3:~$ javac -classpath ${HADOOP_CLASSPATH} -d Desktop/WordCount/classes/ Desktop/WordCount/WordCount.java
dbit@complab3:~$ cd Desktop/WordCount/
dbit@complab3:~/Desktop/WordCount$ jar -cvf exp.jar classes/
added manifest
adding: classes/(in = 0) (out= 0)(stored 0%)
adding: classes/WordCount.class(in = 1927) (out= 1052)(deflated 45%)
adding: classes/WordCount$TokenizerMapper.class(in = 1752) (out= 764)(deflated 56%)
adding: classes/WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
dbit@complab3:~/Desktop/WordCount$
```

Pass the file to hadoop MapReducer

```
dbit@complab3:~/Desktop/WordCount$ hadoop jar exp.jar WordCount /WordCountExp /WordCountExp/Out
23/09/10 20:18:21 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/09/10 20:18:23 INFO input.FileInputFormat: Total input paths to process : 1
23/09/10 20:18:23 INFO mapreduce.JobSubmitter: number of splits:1
23/09/10 20:18:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1694356176649_0001
23/09/10 20:18:29 INFO impl.YarnClientImpl: Submitted application application_1694356176649_0001
23/09/10 20:18:29 INFO mapreduce.Job: The url to track the job: http://complab3:8088/proxy/application_1694356176649_0001/
23/09/10 20:18:29 INFO mapreduce.Job: Running job: job_1694356176649_0001
23/09/10 20:18:52 INFO mapreduce.Job: Job job_1694356176649_0001 running in uber mode : false
23/09/10 20:18:52 INFO mapreduce.Job:  map 0% reduce 0%
23/09/10 20:18:59 INFO mapreduce.Job:  map 100% reduce 0%
```

```
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=4991
        Total time spent by all reduces in occupied slots (ms)=4382
        Total time spent by all map tasks (ms)=4991
        Total time spent by all reduce tasks (ms)=4382
        Total vcore-milliseconds taken by all map tasks=4991
        Total vcore-milliseconds taken by all reduce tasks=4382
        Total megabyte-milliseconds taken by all map tasks=5110784
        Total megabyte-milliseconds taken by all reduce tasks=4487168
    Map-Reduce Framework
        Map input records=16
        Map output records=18
        Map output bytes=153
        Map output materialized bytes=152
        Input split bytes=116
        Combine input records=18
        Combine output records=14
        Reduce input groups=14
        Reduce shuffle bytes=152
        Reduce input records=14
        Reduce output records=14
        Spilled Records=28
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=175
        CPU time spent (ms)=1890
        Physical memory (bytes) snapshot=431947776
        Virtual memory (bytes) snapshot=3835953152
        Total committed heap usage (bytes)=295174144
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
```

Output

```
dbit@complab3:~/Desktop/WordCount$ hadoop fs -cat /WordCountExp/Output/part-r-00000
dog     30
cat     30
car     30
deer    24
well    18
is      18
done    12
all     12
welcome 6
boy     6
dbit@complab3:~/Desktop/WordCount$
```

Conclusion:	Thus we are able to execute the Mapreduce program to count distinct words in the input file using hadoop pseudo distributed mode.
References:	https://learnomate.org/steps-to-resolve-when-datanode-services-is-not-starting/ https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm

Don Bosco Institute of Technology

Department of Computer Engineering

Academic year – 2022-2023

Big Data Analytics

Assessment Rubric for Experiment No.: 03

Performance Date :
Submission Date :

Title of Experiment : Implementing distinct word count using MapReduce.

Year and Semester : IV^{rh} Year and VIIth Semester
Batch :
Name of Student :
Roll No. :

Performance	**Poor**	**Satisfactory**	**Good**	**Excellent**	
	1 point	2 points	3 points	4 points	
Results and Documentations	**Poor**	**Satisfactory**	**Good**	**Excellent**	
	1 point	2 points	3 points	4 points	
Viva	**Poor**	**Satisfactory**	**Good**	**Excellent**	
	1 point	2 points	3 points	4 points	
Timely Submission	**Submission beyond 7 days of the deadline**	**Late submission till 7 days**	**Submission on time**		
	1 points	2 points	3 points		

Faculty Incharge : Ms. Sana Shaikh