**Name - Ankita Adhik Kanse. Roll no. - 747(M.Sc.DSAI-1) Div - H**

**Subject - Big Data Technology.**

**Practical No. 02 -**

Character Counting using MapReduce and Java

**Aim - To Execute Character Count Program in MapReduce Hadoop**

* In MapReduce char count example, we find out the frequency of each character. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of a Key-value pair.

## Prerequisite

* Java Installation - Check whether the Java is installed or not using the following command.  
  java -version
* Hadoop Installation - Check whether the Hadoop is installed or not using the following command.  
  hadoop version

**1. Implementation steps of CharCount problem using Hadoop MapReduce in cloudera (in Eclipse): using java**

Open virtual box and then start cloudera quickstart.

**Step 1:**

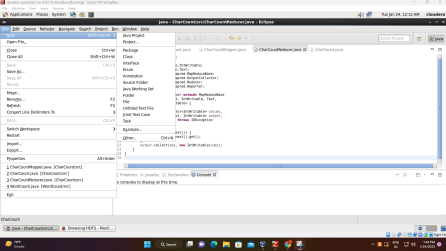
First Open Eclipse -> then select File -> New -> Java Project ->Name it CharCount -> then select use an execution environment -> choose JavaSE-1.8 then next

**Step 2:**

Adding the Hadoop libraries to the project Click on Libraries -> Add External JARs Click on

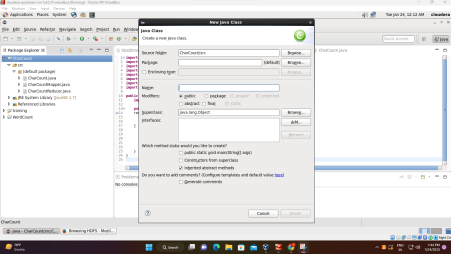
File System -> usr -> lib -> hadoop Select all the libraries (JAR Files) -> click OK Click on

Add External jars, -> client -> select all jar files -> ok -> Finish



**Step 3:**

Right Click on Project CharCount. Using new class - Create Three Java Classes into the project. Name them CharCount(having the main function), CharCountMapper, CharCountReducer.



**Project :- CharCount**

Code:CharCountMapper.java

import java.io.IOException;

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 CharCountMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{

public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException{

String line = value.toString();

String tokenizer[] = line.split("");

for(String SingleChar : tokenizer)

{

Text charKey = new Text(SingleChar);

IntWritable One = new IntWritable(1);

output.collect(charKey, One);

}

}

}

Code : CharCountReducer.java

import java.io.IOException;

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 CharCountMapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{

public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,

Reporter reporter) throws IOException{

String line = value.toString();

String tokenizer[] = line.split("");

for(String SingleChar : tokenizer)

{

Text charKey = new Text(SingleChar);

IntWritable One = new IntWritable(1);

output.collect(charKey, One);

}

}

}

Code: CharCount.java

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 CharCount {

public static void main(String[] args)

throws IOException

{

JobConf conf = new JobConf(CharCount.class);

conf.setJobName("CharCount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(CharCountMapper.class);

conf.setCombinerClass(CharCountReducer.class);

conf.setReducerClass(CharCountReducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

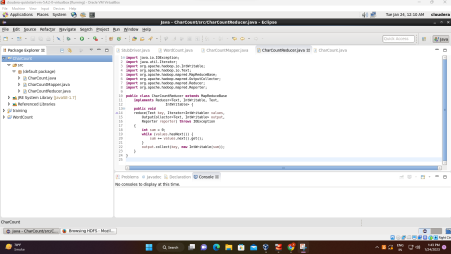
FileInputFormat.setInputPaths(conf,new Path(args[0]));

FileOutputFormat.setOutputPath(conf,new Path(args[1]));

JobClient.runJob(conf);

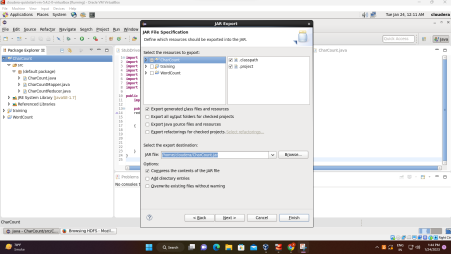
}

}



**Step 4:**

Now export the project as a jar file. Right-click on CharCount choose Export. and go to Java -> JAR file click -> Next and choose your export destination then click -> Next. Choose Main Class as CharCount by clicking -> Browse and then click -> Finish -> Ok.



Now the Jar file is successfully created and saved at /cloudera directory with the name CharCount.jar

**Step 5:**

Create a simple text file and add some data to it.

Create an input file named charcounterdata.txt and input the following text.

I am a student of RJ College.

RJ College is one of the good colleges in Mumbai.

Create an input directory named charInputDir in hadoop HDFS.

Copy file charcounterdata.txt in charInputDir

Execute the jar file and store the default output file in the charOutputDir directory.

| Command | JarFileName | Class Name  Input File OutputDir  including  main() |  |
| --- | --- | --- | --- |

| hadoop jar | /home/cloud era/charCou ntDriver.jar | charCountD  /charInputDi  river  r  /charOutput  /charcounter  Dir  data.txt |  |
| --- | --- | --- | --- |

Syntax

hadoop jar jarFileName ClassNameofMainMethod InputFile OutputDir Executable command ��

hadoop jar /home/cloudera/charCountDriver.jar charCountDriver /charInputDir /charcounterdata.txt /charOutputDir

Using Terminal (Java):-

[cloudera@quickstart ~]$ gedit charcounterdata.txt

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

[cloudera@quickstart ~]$ hdfs dfs -ls /

Found 11 items

drwxr-xr-x - cloudera supergroup 0 2023-01-23 19:51 /charinputdir drwxr-xr-x - hbase supergroup 0 2022-11-11 19:06 /hbase

drwxr-xr-x - cloudera supergroup 0 2023-01-20 23:25 /input

drwxr-xr-x - cloudera supergroup 0 2023-01-23 19:25 /inputdir

drwxr-xr-x - cloudera supergroup 0 2023-01-16 20:25 /myhdfsdir

drwxr-xr-x - cloudera supergroup 0 2023-01-22 22:50 /op1

drwxr-xr-x - cloudera supergroup 0 2023-01-22 23:01 /outputdir

drwxr-xr-x - solr solr 0 2015-06-09 03:38 /solr

drwxrwxrwx - hdfs supergroup 0 2022-11-11 19:50 /tmp

drwxr-xr-x - hdfs supergroup 0 2022-11-07 00:22 /user

drwxr-xr-x - hdfs supergroup 0 2015-06-09 03:36 /var

[cloudera@quickstart ~]$ hdfs dfs -put /home/cloudera/charcounterdata.txt /charinputdir/ [cloudera@quickstart ~]$ hadoop jar /home/cloudera/CharCount.jar CharCount /charinputdir/charcounterdata.txt /charoutputdir

23/01/23 20:00:30 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032 23/01/23 20:00:30 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032 23/01/23 20:00:31 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

23/01/23 20:00:31 INFO mapred.FileInputFormat: Total input paths to process : 1 23/01/23 20:00:31 INFO mapreduce.JobSubmitter: number of splits:2

23/01/23 20:00:31 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1668222340280\_0006

23/01/23 20:00:31 INFO impl.YarnClientImpl: Submitted application

application\_1668222340280\_0006

23/01/23 20:00:31 INFO mapreduce.Job: The url to track the job:

http://quickstart.cloudera:8088/proxy/application\_1668222340280\_0006/ 23/01/23 20:00:31 INFO mapreduce.Job: Running job: job\_1668222340280\_0006 23/01/23 20:00:37 INFO mapreduce.Job: Job job\_1668222340280\_0006 running in uber mode : false

23/01/23 20:00:37 INFO mapreduce.Job: map 0% reduce 0%

23/01/23 20:00:46 INFO mapreduce.Job: map 50% reduce 0%

23/01/23 20:00:47 INFO mapreduce.Job: map 100% reduce 0%

23/01/23 20:00:52 INFO mapreduce.Job: map 100% reduce 100%

23/01/23 20:00:52 INFO mapreduce.Job: Job job\_1668222340280\_0006 completed successfully

23/01/23 20:00:52 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=204

FILE: Number of bytes written=332321

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=355

HDFS: Number of bytes written=96

HDFS: Number of read operations=9

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=2

Launched reduce tasks=1

Data-local map tasks=2

Total time spent by all maps in occupied slots (ms)=12225

Total time spent by all reduces in occupied slots (ms)=3506

Total time spent by all map tasks (ms)=12225

Total time spent by all reduce tasks (ms)=3506

Total vcore-seconds taken by all map tasks=12225

Total vcore-seconds taken by all reduce tasks=3506

Total megabyte-seconds taken by all map tasks=12518400

Total megabyte-seconds taken by all reduce tasks=3590144

Map-Reduce Framework

Map input records=4

Map output records=82

Map output bytes=488

Map output materialized bytes=210

Input split bytes=232

Combine input records=82

Combine output records=25

Reduce input groups=24

Reduce shuffle bytes=210

Reduce input records=25

Reduce output records=24

Spilled Records=50

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=176

CPU time spent (ms)=1110

Physical memory (bytes) snapshot=537112576

Virtual memory (bytes) snapshot=4507594752

Total committed heap usage (bytes)=391979008

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=123

File Output Format Counters

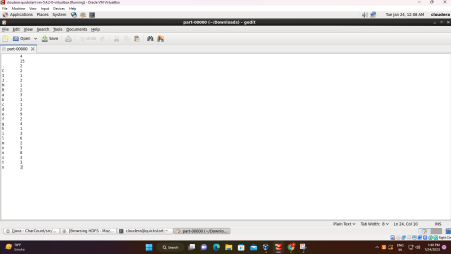
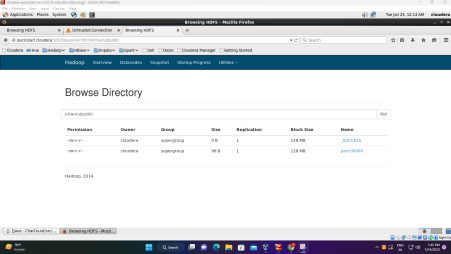
Bytes Written=96

The file can be accessed using a browser.

Browse the Directory by

Hadoop->HDFS Namenode->Utilities ->Browse the file system Now downloading the part-r-00000 file.

Inside the part-r-00000 file we will have output as we are getting after executing.

It will give the count of number of times each character has occurred as output.

**2. Hadoop Streaming Using Python – Word Count Problem**

Using Python :-

**Step1:-**

Create a simple text file and add some data to it.

Create an input file named charcounterdata.txt and input the following text.

I am a student of RJ College.

RJ College is one of the good colleges in Mumbai.

**Step 2:-**

Create a mapper.py file that implements the mapper logic. It will read the data from STDIN and will split the lines into words, and will generate an output of each word with its individual count.

Copy the below code to the mapper.py file

#!/usr/bin/env python

# import sys because we need to read and write data to STDIN and STDOUT

import sys

# reading entire line from STDIN (standard input)

for line in sys.stdin:

# to remove leading and trailing whitespace

line = line.strip()

# split the line into words

words = line.split()

# we are looping over the words array and printing the word # with the count of 1 to the STDOUT

for word in words:

# write the results to STDOUT (standard output);

# what we output here will be the input for the

# Reduce step, i.e. the input for reducer.py

print '%s\t%s' % (word, 1)

**Step 3:-**

Create a reducer.py file that implements the reducer logic. It will read the output of mapper.py from STDIN(standard input) and will aggregate the occurrence of each word and will write the final output to STDOUT.

#!/usr/bin/env python

from operator import itemgetter

import sys

current\_word = None

current\_count = 0

word = None

# read the entire line from STDIN

for line in sys.stdin:

# remove leading and trailing whitespace

line = line.strip()

# splitting the data on the basis of tab we have provided in mapper.py

word, count = line.split('\t', 1)

# convert count (currently a string) to int

try:

count = int(count)

except ValueError:

# count was not a number, so silently

# ignore/discard this line

continue

# this IF-switch only works because Hadoop sorts map output # by key (here: word) before it is passed to the reducer if current\_word == word:

current\_count += count

else:

if current\_word:

# write result to STDOUT

print '%s\t%s' % (current\_word, current\_count)

current\_count = count

current\_word = word

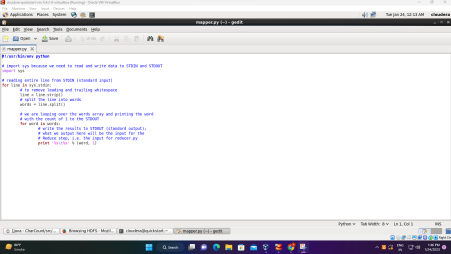
# do not forget to output the last word if needed!

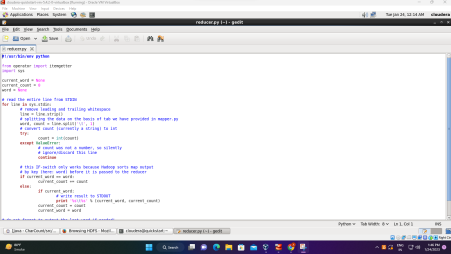
if current\_word == word:

print '%s\t%s' % (current\_word, current\_count)

Terminal:-

[cloudera@quickstart ~]$ gedit mapper.py

[cloudera@quickstart ~]$ gedit reducer.py



Now let’s check our reducer code reducer.py with mapper.py is it working properly or not with the help of the below command.

[cloudera@quickstart ~]$ cat charcounterdata.txt | python mapper.py | sort -k1,1 | python reducer.py

a 1

am 1

College 1

College. 1

colleges 1

good 1

I 1

in 1

is 1

Mumbai. 1

of 2

one 1

RJ 2

student 1

the 1

