## HU Extension School E-63 Big Data Analytics

## Assignment 03

### Handed out: 02/12/2016 Due by 11:30PM on Friday, 02/19/2016

Download and Install VMWare Workstation or Fusion product from HU VMWare store.

**Problem 1)** Modify attached class WordCount.java so that its result excludes the following stop words:

|  |  |  |
| --- | --- | --- |
| I  a  about  an  are  as  at  be  by  com  for  from how | in  is  it  of  on  or  that the  this to  was  what  when where | who  will  with the www |

as well as special characters (dashes, parentases, etc). Stop word lists could be much lomger than this. You do not have to be extremely thoroughful. You would like to get a more or less clean list of ordinary words with the numbers of their occurances. Do not fret. Be reasonable. Perform analysis on te text of James Joyce’s Ulysis.

I took WordCount.java.

I kept the Reducer class (IntSumReducer) as it is.

I modifed the Mapper class (TokenizerMapper) implementation to:

// **validate the input**

if (value == null) {

// this means the line read is a blank/empty line.

// In that case, there is no tokenization/analysis

// that we can do on it

return;

}

StringTokenizer itr = new StringTokenizer(value.toString());

while (itr.hasMoreTokens()) {

String str = itr.nextToken();

if (str == null) {

continue;

}

// will **exclude special characters** from the final output

str = str.replaceAll("[^A-Za-z0-9]", "");

if (str.isEmpty()) {

continue;

}

// will **exclude stop words** from the final output

if (stopWordsList.contains(str.toLowerCase())) {

continue;

}

word.set(str);

context.write(word, one);

}

I defined **stopwords** list as:

// defines stopwords to be excluded from final output

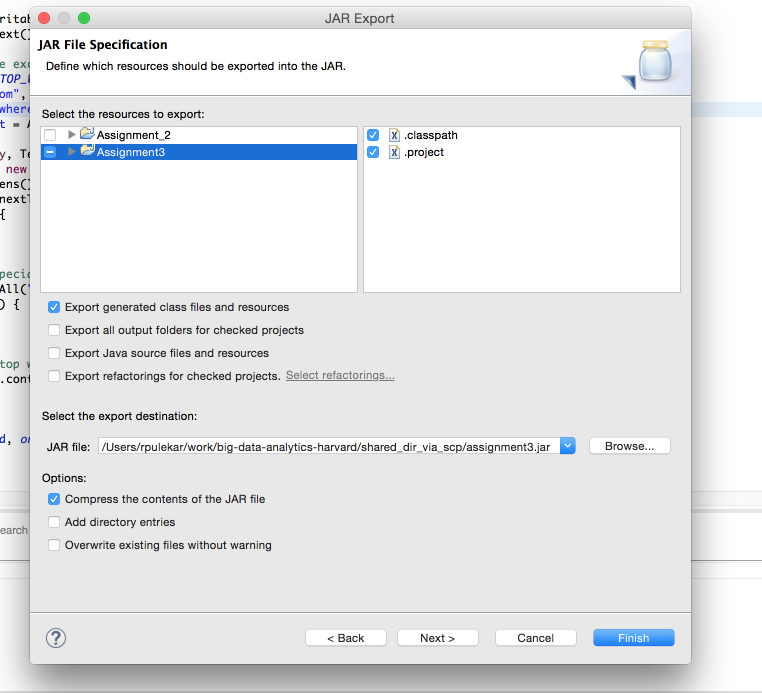
private static String[] STOP\_WORDS = new String[] { "i", "a", "about", "an", "are", "as", "at", "be", "by",

"com", "for", "from", "how", "in", "is", "it", "of", "on", "or", "that", "the", "this", "to", "was",

"what", "when", "where", "who", "will", "with", "the", "www" };

List<String> stopWordsList = Arrays.<String>asList(STOP\_WORDS);

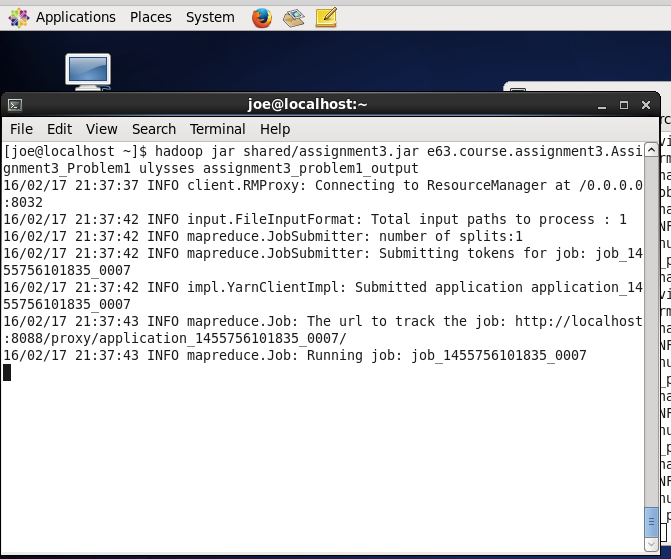
Then I exported the jar file



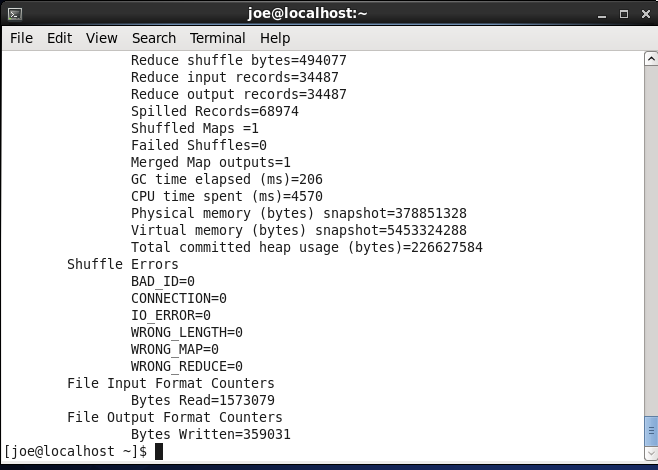
Copied the jar file over to the VM:



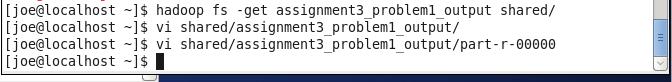
Ran the mapreduce job on the VM:



It ran successfully:



Copied the mapreduce job output from hdfs to CentOS file system:



Top 10 records of output:

0 2

001 5

002 1

003 2

004 3

007 1

010 1

016 1

0175 1

020 1

B

Botton 10 records of output:

zivio 1

zmellz 1

zodiac 2

zodiacal 2

zoe 1

zones 1

zoo 1

zoological 1

zouaves 1

zrads 3

Deliverables:

Complete program (*Assignment3\_Problem1.java*) ,

complete output (*assignment3\_problem1\_output.txt*)

are provided in assignment submision.

**Problem 2)** Write another MapReduce program which would read the “word count” output of the previous job and order the results by the declaining number of occurances.

For this, I set the input format as:

job.setInputFormatClass(KeyValueTextInputFormat.**class**);

I implemented the map function as:

public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

// **validate** the input

if (key == null) {

// this means the line read is a blank/empty line.

// In that case, there is no tokenization/analysis

// that we can do on it

return;

}

try {

// the below String.valueOf function means that even if value is

// null, the wordCount will be considered as 0.

wordCount.set(Integer.parseInt(String.valueOf(value)));

} catch (NumberFormatException nfe) {

wordCount.set(0);

}

// the below is done so that wordcount becomes the key and the word

// itself becomes the value

context.write(wordCount, key);

}

Then I reversed key and value in reducer:

public void reduce(IntWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

for (Text value : values) {

**// this inverts key and value**

context.write(value, key);

}

}

I did not set the combiner, only set the mapper and reducer:

job.setMapperClass(Problem2Mapper.class);

job.setReducerClass(Problem2Reducer.class);

I created a custom DescendingKeyComparator class and used it as the SortComparatorClass:

job.setSortComparatorClass(DescendingKeyComparator.**class**);

public static class DescendingKeyComparator extends WritableComparator {

protected DescendingKeyComparator() {

super(IntWritable.class, true);

}

@SuppressWarnings("rawtypes")

@Override

public int compare(WritableComparable w1, WritableComparable w2) {

IntWritable key1 = (IntWritable) w1;

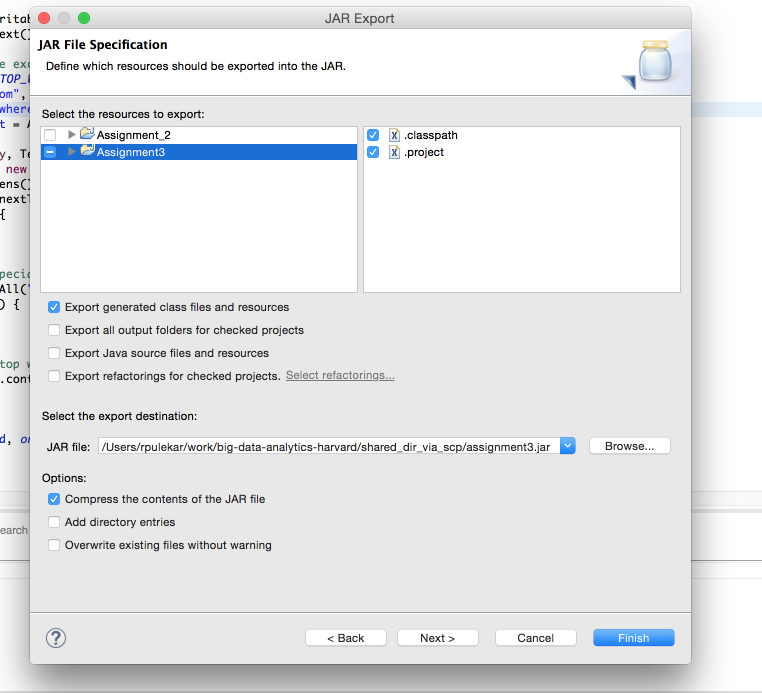
IntWritable key2 = (IntWritable) w2;

return -1 \* key1.compareTo(key2);

}

}

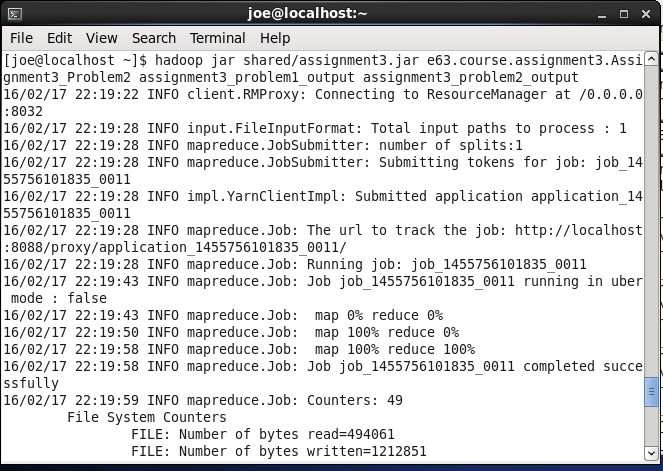
Then I exported the jar file



Copied the jar file over to the VM:



Ran the mapreduce job on the VM:



Fetched the file from hdfs to CentOS file system:



First 10 lines of output (first column is the word, second column is its’s number of occurences):

and 6691

his 3064

he 2880

her 1686

you 1666

him 1505

said 1201

all 1154

He 1153

me 922

Last 10 lines of output (first column is the word, second column is its’s number of occurences):

harelip 1

harebell 1

peccaminous 1

peccato 1

hardon 1

pecked 1

hardihood 1

hardhumped 1

hardheaded 1

Envel 1

Thus you can see that we get output that shows all words by declining number of occurences.

Deliverables:

Complete program (*Assignment3\_Problem2.java*) ,

complete output (*assignment3\_problem2\_output.txt*)

are provided in assignment submision.

**Problem 3)** Create a program that will find out how many words appear only once, how many twice, three times, four times and so on in James Joyce’s Ulysis

For this program I have considerd input to be the output of Program 1.

I set the input format as:

job.setInputFormatClass(KeyValueTextInputFormat.class);

I implemented the mapper class to have:

public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

try {

// the below String.valueOf function means that even if value is

// null, the wordCount will be considered as 0.

wordCount.set(Integer.parseInt(String.valueOf(value)));

} catch (NumberFormatException nfe) {

wordCount.set(0);

}

// the below is done so that wordcount becomes the key and the word

// itself becomes the value

context.write(wordCount, one);

}

For te

I implemented the reducer class to have:

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

// this will count the number of words for a particular word-count

int numberOfWordsCounter = 0;

for (IntWritable val : values) {

numberOfWordsCounter = numberOfWordsCounter + val.get();

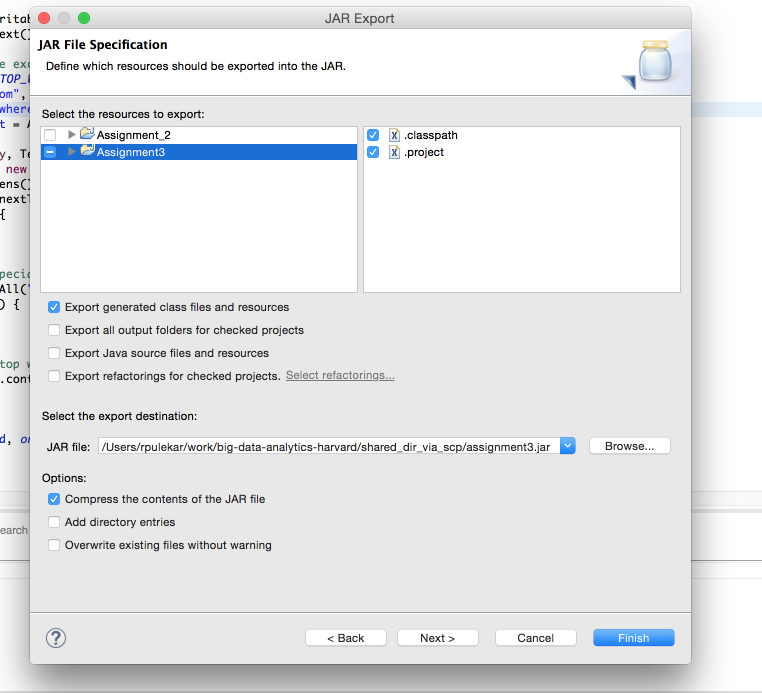
}

numberOfWords.set(numberOfWordsCounter);

// here key=word-count. value=number Of Words which have that word //count

context.write(key, numberOfWords);

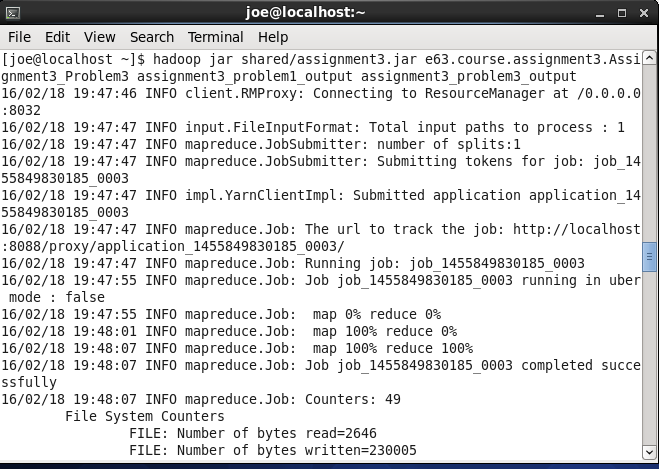
}

Then I exported the jar file

Copied the jar file over to the VM:



Ran the mapreduce job on the VM:

****

Fetched the file from hdfs into CentOS file system:

****

First 10 lines of output

(wordcount, number\_of\_words\_that\_have\_that\_wordcount)

1 19693

2 5415

3 2437

4 1433

5 958

6 694

7 517

8 377

9 313

10 252

Last 10 lines of output

(wordcount, number\_of\_words\_that\_have\_that\_wordcount)

922 1

1153 1

1154 1

1201 1

1505 1

1666 1

1686 1

2880 1

3064 1

6691 1

Deliverables:

Complete program (*Assignment3\_Problem3.java*) ,

complete output (*assignment3\_problem3\_output.txt*)

are provided in assignment submision.

**Problem 4)** Combine operations of two MapReduce programs in Problems 1 and 3 above into a single program with chained MapReduce jobs.

I created Problem1Mapper which is exactly same as Mapper class of problem1:

private static class Problem1Mapper extends Mapper<Object, Text, Text, IntWritable> {

....

}

Created Problem1Reducer which is exactly same as Reducer class of problem1:

private static class Problem1Reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

….

}

Created Problem3Mapper which is exactly same as Mapper class of problem3:

private static class Problem3Mapper extends Mapper<Text, Text, IntWritable, IntWritable> {

....

}

Created Problem3Reducer which is exactly same as Reducer class of problem3:

private static class Problem3Reducer extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

....

}

Then I created a temp dir for communication between the 2 jobs:

Path temp\_dir = new Path("assignment3\_problem4\_temp\_dir");

Then I created the jobs with input and output directories as:

Job problem1Job = createProblem1Job(conf, input, temp\_dir);

Job problem3Job = createProblem3Job(conf, temp\_dir, out);

Added code to delete the temporary directory:

private void cleanup(Path temp, Configuration conf) throws IOException {

FileSystem fs = temp.getFileSystem(conf);

fs.delete(temp, true);

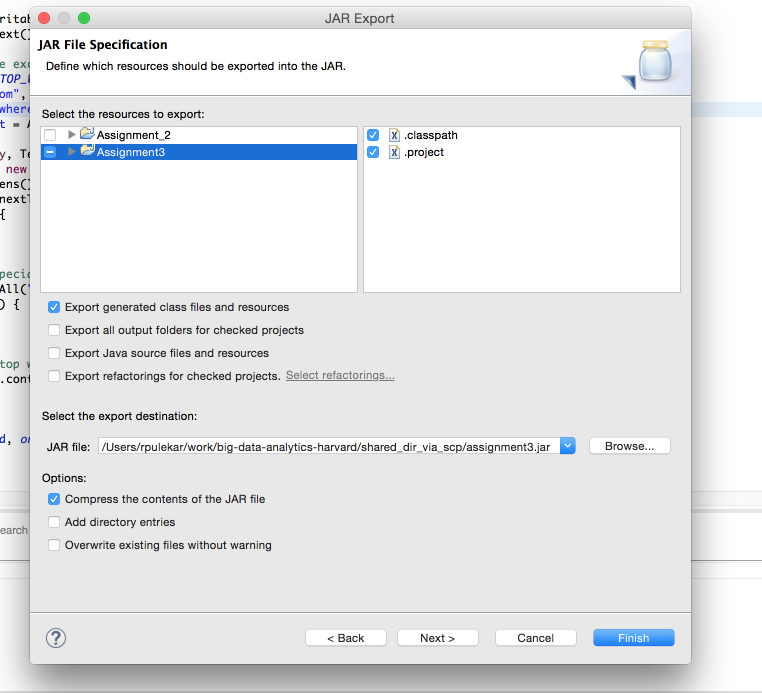
}

Then, wrote code that will run the jobs:

problem1Job.waitForCompletion(true);

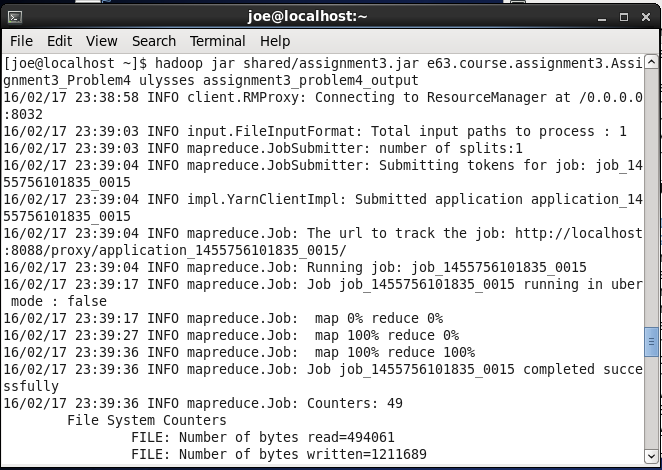
boolean success = problem3Job.waitForCompletion(true);

Then I exported the jar file



Copied the jar file over to the VM:

Ran the mapreduce job on the VM:

****

Fetched the file from hdfs into CentOS file system:

****

First 10 lines of output (wordcount, number\_of\_words\_that\_have\_that\_wordcount):

1 19693

2 5415

3 2437

4 1433

5 958

6 694

7 517

8 377

9 313

10 252

Last 10 lines of output (wordcount, number\_of\_words\_that\_have\_that\_wordcount):

922 1

1153 1

1154 1

1201 1

1505 1

1666 1

1686 1

2880 1

3064 1

6691 1

Output of this program (with 2 chained jobs) is same as the output of Problem 3.

Deliverables:

Complete program (*Assignment3\_Problem4.java*) ,

complete output (*assignment3\_problem4\_output.txt*)

are provided in assignment submision.

**Problem 5)** Move attached Inverter.java class from old MapReduce API to the new API. Demonstrate that new and old clas produce the same result. Use patent data set to demonstrate your work.

To do this, I modified the mapper function to reflect the signature and functionality:

public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

context.write(value, key);

}

Then I modified the reducer class to reflect the signature and implementation:

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

String csv = "";

for (Text value : values) {

if (csv.length() > 0)

csv += ",";

csv += value.toString();

}

context.write(key, new Text(csv));

}

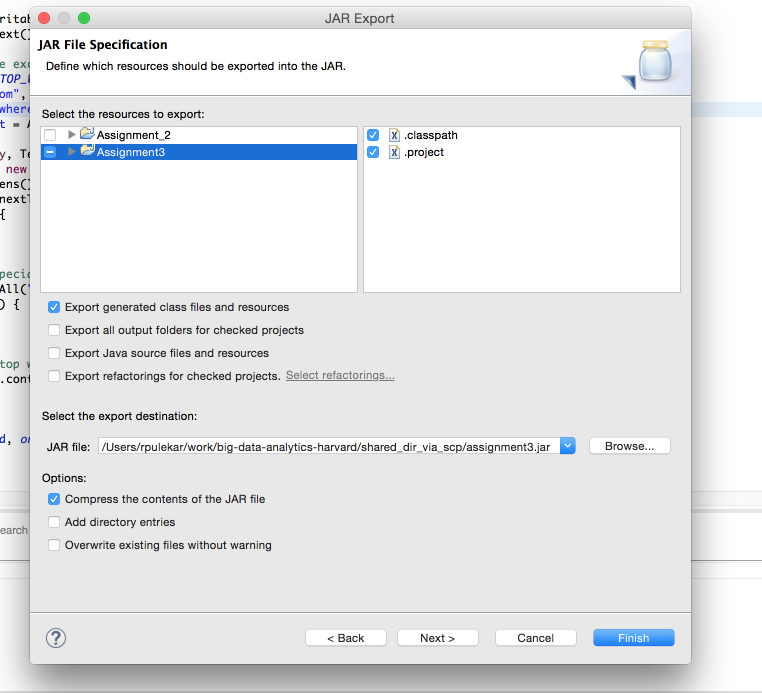
I used Job instead of JobConf:

Job job = Job.getInstance(conf, "Assignment3\_Problem5NewAPI");

Then I used the Job api to start the job:

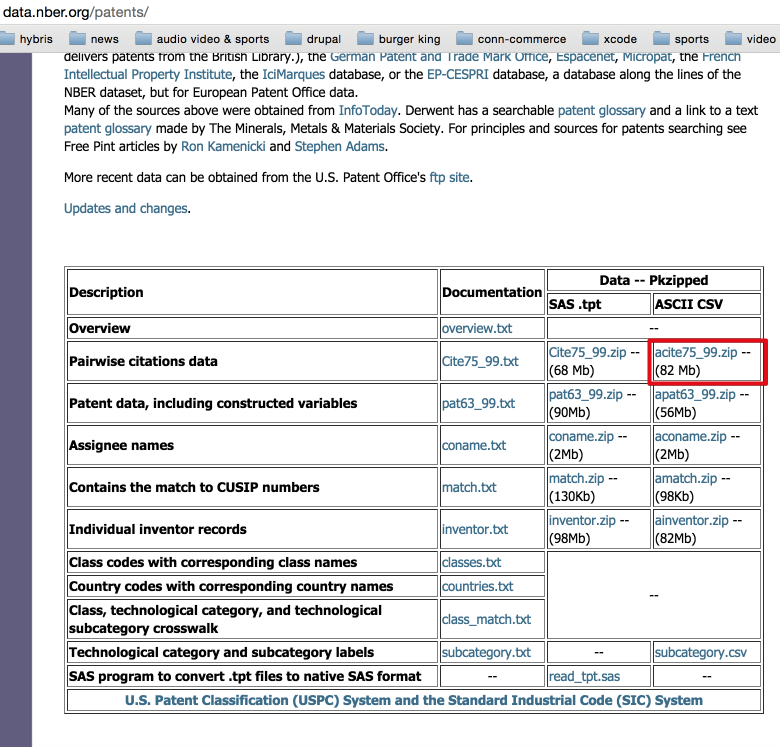
job.waitForCompletion(true);

Then I exported the jar file



Copied the jar file over to the VM:

Downloaded the patente citation data:



First 10 lines of patent citation data:

"CITING","CITED"

3858241,956203

3858241,1324234

3858241,3398406

3858241,3557384

3858241,3634889

3858242,1515701

3858242,3319261

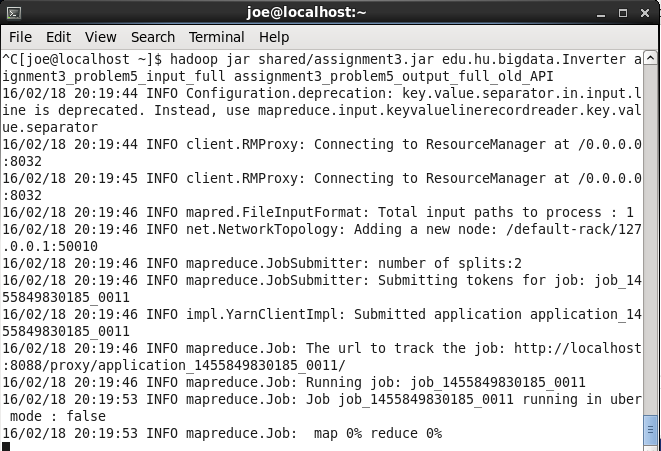
3858242,3668705

3858242,3707004

Loaded the **patent citation** input file on hdfs:



Then I ran the Inverter mapreduce job (**old API**) on the VM:



Fetched the output file from hdfs into CentOS file system:



First 10 lines of output (from old API):

"CITED" "CITING"

1 3964859,4647229

10000 4539112

100000 5031388

1000006 4714284

1000007 4766693

1000011 5033339

1000017 3908629

1000026 4043055

1000033 4190903,4975983

Last 10 lines of output (from old API)

999961 5878901,5738381,5782495,4171117,4262874,5048788,4871140,4832301,4437639

999965 5052613

999968 3916735

999971 3965843

999972 4038129

999973 4900344,5427610

999974 5464105,4560073,4728158

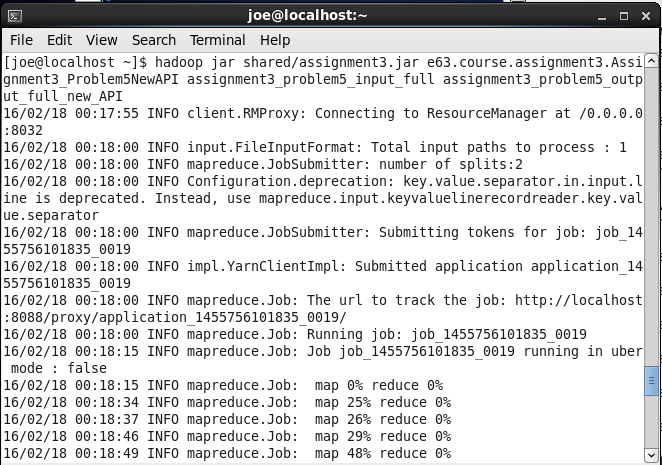
999977 4092587

999978 3915443

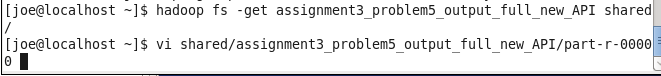
999983 5143114,5394715,5806555

Then I ran the Inverter program with new API

(this program name being Assignment3\_Problem5NewAPI ):



Fetched the output file from hdfs into CentOS file system:



First 10 lines of output (**new API**):

"CITED" "CITING"

1 3964859,4647229

10000 4539112

100000 5031388

1000006 4714284

1000007 4766693

1000011 5033339

1000017 3908629

1000026 4043055

1000033 4190903,4975983

Last 10 files of output (**new API**):

999961 5782495,5738381,5878901,4171117,4262874,5048788,4871140,4832301,4437639

999965 5052613

999968 3916735

999971 3965843

999972 4038129

999973 4900344,5427610

999974 5464105,4560073,4728158

999977 4092587

999978 3915443

999983 5143114,5394715,5806555

So, basically both outputs (output from Inverter with old API and output from Inverter with new API) are the same.

Deliverables:

Complete program (*Assignment3\_Problem5NewAPI.java*) ,

output from Inverter program with old API (*assignment3\_problem5\_output\_full\_old\_API.txt*) ,

output from Assignment3\_Problem5NewAPI.java which has new API (*assignment3\_problem5\_output\_full\_new\_API.txt*)

are provided in the assignment submission.

If you have your own working VM with installed CDH5.5.1, do this assignment on that VM. If, for what ever reason, you do not posses a working VM with CDH5.5.1, please be free to download Clouder’s Getting started VM and do you assignment on that VM.

Capture all steps of your implementation with comments indicating what is it you are accomplishing with every step in an MS Word document. Upload to the class site. Please upload your working Java files as well. Please post comments and questions to the class Discussion Board on the Canvas site.