Massive Data Processing

Assignment_2 WANG Wanqing

Undergraduate background: Computer Science and Technology

Pre-processing the input (10)

In this pre-process, the intuition is to use file “pg100.txt” and the file produced in last assignment which is “stopwords.txt”.

And the process is to 1) remove all stop words and special characters, just keep numbers and alphabets, then keep each unique word in each line without empty; 2) store the number of records on HDFS; 3) Ascendingly order the tokens with frequency and store them on HDFS.

For this pre-processing part, the main intuition is to create a new project, then implement 2 classes that one is implemented to make the word count, the other is to remove all stop words and count the frequency.

To finish the pre-processing part, the file “stopwords.txt” will be used to remove the stop words from the file “pg100.txt”. So, the 2 files need to be input into Hadoop HDFS for the next usage.

The code shows here in terminal:

Create folders in the project and put the files need to be used into HDFS

```
mkdir workspace/Assignment_similarity/input
mkdir workspace/Assignment_similarity/output
```

After creating the input and output folders

```
hadoop fs -put workspace/Assignment_similarity/input/pg100.txt input
Hadoop fs -put workspace/Assignment_similarity/input/stopwords.txt input
```

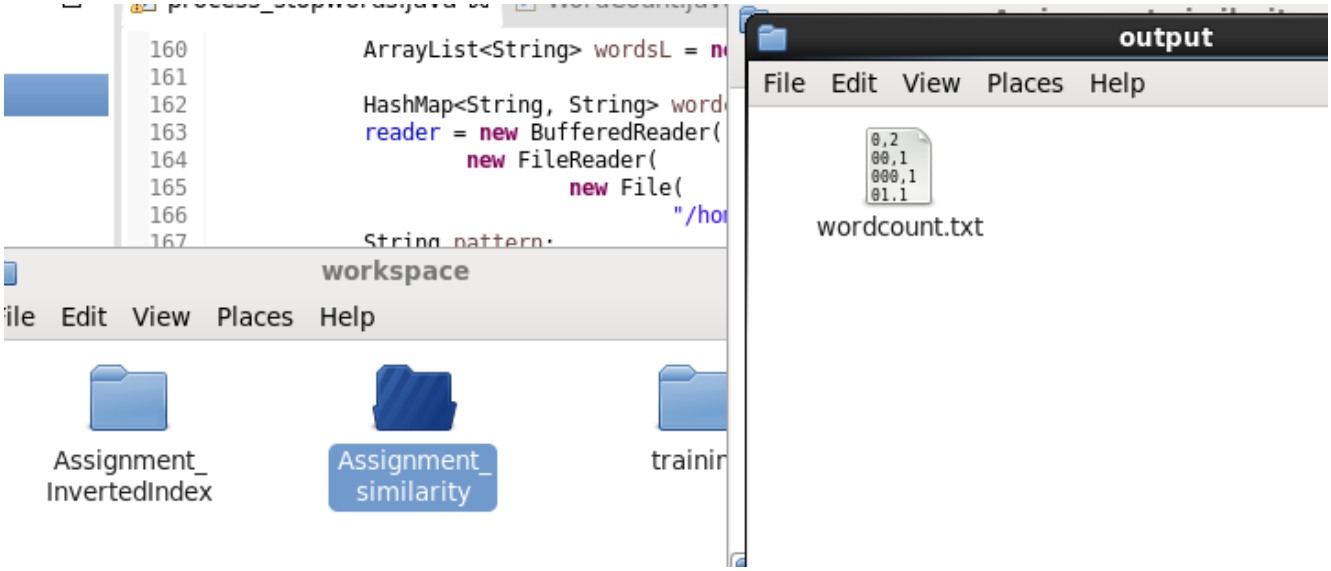
The files pg100.txt and stopwords.txt are now in HDFS. After implementing 2 classes in Eclipse, export the jar file of the project named similarity.jar. We can now run the project in terminal.

```
Hadoop jar similarity.jar mdpassignment. similarity.WordCount input/pg100.txt output
hadoop fs -getmerge output workspace/Assignment_similarity/output/wordcount.txt
```

```
        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=5589889
        File Output Format Counters
                Bytes Written=248047
[cloudera@quickstart ~]$ hadoop fs -getmerge output workspace/Assignment similarity/output/wordcount.txt
```

The reason why we applied WordCount class is to use the output “wordcount.txt” which contains the words and numbers we can get to use in the next step.

In the second line of commands, “getmerge” is used to merge all the pieces in the memory produced by the reducers to be one text file. We can see the text file “wordcount.txt”.



After finishing the wordcount part and the implementation of process_stopwords class, we can apply that in terminal to Hadoop system.

```
Hadoop jar similarity.jar mdpassignment. similarity.process_stopwords input/pg100.txt
output -skip input/stopwords.txt
hadoop fs -getmerge output workspace/Assignment_similarity/output/tokensinline_gf.txt
```

```
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        mdpassignment.similarity.process_stopwords$CUSTOM_COUNTER
                NB_LINES=114665
        File Input Format Counters
                Bytes Read=5589889
        File Output Format Counters
                Bytes Written=5978524
[cloudera@quickstart ~]$
```

the reason why we applied skip is to prevent the data in the file we do not need to make the program crash. After that, we merged all pieces into text file named tokensinline_gf.txt.

todensinline_gf.txt

```
0, ebook_17, complete_248, shakespeare_272, works_284, gutenberg_326, project_331, william_354
81, shakespeare_272, william_354
104, anyone_7, anywhere_8, ebook_17, cost_51, use_562
170, restrictions_2, whatsoever_17, copy_29, almost_163, away_864, give_1353
240, included_5, license_26, terms_97, under_304, re_305, gutenberg_326, project_331, use_562
309, online_4, www_11, org_17, ebook_17, gutenberg_326
359, details_1, copyrighted_2, ebook_17, below_58, gutenberg_326, project_331
427, guidelines_1, file_22, copyright_244, follow_338, please_395
497, title_92, complete_248, shakespeare_272, works_284, william_354
549, author_18, shakespeare_272, william_354
580, september_1, posting_5, ebook_17, date_26, 100_5, 1_357, 2011_1
626, january_3, release_10, date_26, 1994_1
657, language_37, english_167
680, ebook_17, start_37, complete_248, shakespeare_272, works_284, gutenberg_326, project_331, william_354
771, produced_4, future_20, inc_224, library_233, world_904
846, presented_18, file_22, etext_244, gutenberg_326, project_331, 100th_1
912, cooperation_1, presented_18, inc_224, library_233, world_904
978, cdroms_1, future_20, library_233, shakespeare_272, gutenberg_326, project_331
1044, releases_1, etexts_3, placed_11, domain_12, public_66, often_125
```

**It took really a long time to run this class. It should be the complex operations in reducers that caused that.*

Logged in as: dr.who

MapReduce Job job_1489183217268_0002

Job Overview	
Job Name:	process_stopwords
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Mar 10 14:54:44 PST 2017
Started:	Fri Mar 10 14:54:52 PST 2017
Finished:	Fri Mar 10 15:16:56 PST 2017
Elapsed:	22mins, 4sec
Diagnostics:	
Average Map Time	9sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	21mins, 43sec

The format of the output that is asked is word_global frequency, in each line, the word is in that format.

Set-similarity joins (90)

In this part, the intuition is to compare the pairs of documents (d1, d2) that are similar (sim(d1, d2) >= t), given a similarity function *sim* and a similarity threshold t. While what is different in this assignment is to replace documents with lines produced by pre-processing part, the tokensinline_gf.txt file.

With these instructions:

- each output line of the pre-processing job is a unique document (line number is the document id),
- documents are represented as sets of words,
- *sim*(d1, d2) = Jaccard(d1, d2) = |d1 ∩ d2| / |d1 U d2|,
- t = 0.8.

Approaches

A naïve approach is to perform all pairwise comparisons between documents and then output only the pairs that are similar. Another approach is to group together documents that have at least one common word and then perform comparisons only between those pairs. Remember the inverted index from the previous assignment... A third approach is to perform indexing, skipping some of the words of each document. For example, when the similarity threshold is 1 (i.e., two documents are considered similar, only if they match 100%), then indexing only a single word for each document would be enough, to skip comparisons between some non-similar documents and guarantee that all similar documents will be compared. Likewise, [Chaudhuri et al. 2006] proves that it is adequate to index only the first |d| - [t •|d|] + 1 words of each document d, without missing any similar documents (known as the prefix-filtering principle), where t is the Jaccard similarity threshold and |d| is the number of words in d. You are asked to implement the first and the last approach and report your conclusions, along with the execution times and the number of performed comparisons.

Following the approaches and instructions above, we continue launching question(a)

a)

In this part, following what we saw in the instructions and the example, we could easily find that the format of data we are going to use is just the words and lines without global frequency after words. So it is needed to run the process_stopwords class again while this time some lines need to be modified to get the result without global frequency following words.

Exactly what we need to do is just add replaceAll after writing code. *In line 215*

```
context.write(key, new Text(wordswithcountsortedSB.toString().replaceAll("#\\d+", "")));
```

To prevent mistaken, copy the process_stopwords.java and rename it as process_stopwords_justwords.java with modifying the code above.

[\\ d] + number appears one or more times, match the number.

The meaning of this sentence is to replace the string with the number of empty, that is to remove all the numbers

Run again:

```
Hadoop jar similarity.jar mdpassignment. similarity.process_stopwords_justwords
input/pg100.txt output -skip input/stopwords.txt
hadoop fs -getmerge output workspace/Assignment_similarity/output/tokensinline.txt
```

Logged in as: dr.who

MapReduce Job job_1489404220090_0001

Job Overview

Job Name:	process_stopwords
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Mar 13 04:31:42 PDT 2017
Started:	Mon Mar 13 04:31:52 PDT 2017
Finished:	Mon Mar 13 04:54:09 PDT 2017
Elapsed:	22mins, 16sec
Diagnostics:	
Average Map Time	10sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	21mins, 54sec

The format of text file “tokensinline.txt” shows below.

tokensinline.txt

```
0,ebook complete shakespeare works gutenberg project william by of the
81,shakespeare william
104,anyone anywhere ebook cost use at no this with for is of and the
170,restrictions whatsoever copy almost away give may or no it you
240,included license terms under re gutenberg project use it of the
309,online www org ebook gutenberg at or this with
359,details copyrighted ebook below gutenberg project this is a
427,guidelines file copyright follow please this in the
497,title complete shakespeare works william of the
549,author shakespeare william
580,2011 september 100 posting ebook date 1
626,1994 january release date
657,language english
680,ebook start complete shakespeare works gutenberg project william this of
771,produced future inc library world their from by of the
846,100th presented file etext gutenberg project by this is and the
912,cooperation presented inc library world their from with is in
978,cdroms future library shakespeare gutenberg project of and the
1044,releases etexts placed domain public often are not in that the
1112,shakespeare
1127,implications certain read etext copyright has should this you
1195,version complete works william electronic this of the
```

Thinking through the process to compare the similarity of documents(here lines), we might come up with some problems need to be solved first.

1) The formula we have is *sim*(d1, d2) = Jaccard(d1, d2) = |d1 ∩ d2| / |d1 U d2|, while d1 and d2 are not ordered. So when we implement the configuration to run the program, the process will be duplicated. For example with line1 and line2, the program cannot see them as ordered, so these 2 lines might be compared twice, sim(line1, line2) and sim(line2, line1). While the result will be the same.
2) After thinking about the duplicated problem we might face with, we still need to think about how to make a structure to store the documents(lines here) with the format of key-values. While it is kind of easy

to solve this problem, as what we`ve learnt from the fundamental of programming in java, python or others, structures that cannot be modified in keys are dictionary and tuple(cannot be modified in anything). So we can think about what to do with the lines.

To run this program, we need to input the file “tokensinline.txt” into HDFS input so that the mapreduce can find and use it in the program.

```
Hadoop fs -put workspace/Assignment_similarity/input/stopwords.txt input
```

The important part to compare the similarity shows here:

```
public double jaccardsim(TreeSet<String> s1, TreeSet<String> s2) {

        if (s1.size() < s2.size()) {
            TreeSet<String> s1bis = s1;
            s1bis.retainAll(s2);
            int inter = s1bis.size();
            s1.addAll(s2);
            int union = s1.size();
            return (double) inter / union;
        } else {
            TreeSet<String> s1bis = s2;
            s1bis.retainAll(s1);
            int inter = s1bis.size();
            s2.addAll(s1);
            int union = s2.size();
            return (double) inter / union;
        }

    }
```

After implementing the codes in the class.

```
hadoop jar similarity.jar mdpassignment.similarity.similarity input/tokensinline.txt
output
hadoop fs -getmerge output workspace/similarity/output/similarity.txt

hadoop fs -getmerge com_sim.txt workspace/StringSimilarityJoins/output/com_sim.txt
```

****In this part, something happened when I ran the program. Every time I execute the program, it shows Java heap space error. So I modify the configuration of Hadoop system by doing following measures:*

1. *get the permission to modify the file “mapred-site.xml” by implementing:*

```
cd /usr/lib/Hadoop-0.20-mapreduce/conf
chmod a+w mapred-site.xml
```

2. *modify virtual memory in this file by adding:*

```
<property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx2048m</value>
</property>
```

I have tried with the command below while it did not work.
`export HADOOP_OPTS="-Xmx2048m”`

In the meantime, I added some memory control on reduce process and task timeout control

```
<property>
    <name>mapreduce.reduce.java.opts</name>
    <value>-Xmx2048m</value>
</property>
<property>
    <name>mapred.task.timeout</name>
    <value>800000000</value>
</property>
```

3. *After doing those above, the problem about out of memory was over. When I was going to run the program with terminal, a new problem occurred: The file tokensinline.txt contains too many lines which cause the program crashed. So I picked up the top 800 lines to test the program.*

After solving these questions:

The mapreduce process:
The number of performed comparisons is 319600.

```
        WRONG_MAP=0
        WRONG_REDUCE=0
mdpassignment.similarity.similarity$CUSTOM_COUNTER
        com_sim=319600
File Input Format Counters
        Bytes Read=37575
File Output Format Counters
        Bytes Written=248
```

The result similarity.txt shows below:

similarity.txt

```
(680, 0), 0.8
(1195, 2228), 1.0
(1195, 7082), 1.0
(2286, 1255), 0.9
(1255, 7142), 1.0
(7209, 1322), 1.0
(7278, 1391), 1.0
(7343, 1456), 1.0
(7406, 1519), 1.0
(1575, 7462), 1.0
(7529, 1642), 1.0
(2175, 6998), 1.0
(2228, 7082), 1.0
(2286, 7142), 0.9
```

Logs in YARN: The execution time of similarity with top 800 lines is **2mins, 17sec**.

Logged in as: dr.who

MapReduce Job job_1489416105189_0004

Job Overview

Job Name:	similarity
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Mar 13 07:54:22 PDT 2017
Started:	Mon Mar 13 07:54:29 PDT 2017
Finished:	Mon Mar 13 07:56:47 PDT 2017
Elapsed:	2mins, 17sec
Diagnostics:	
Average Map Time	11sec
Average Shuffle Time	5sec
Average Merge Time	0sec
Average Reduce Time	1mins, 55sec

b)

In this part, first we need to do is define the number |d| - ⌈t |d|⌉ + 1 for each line in the input file, in order to keep only the first |d| - ⌈t |d|⌉ + 1 words of each document. Then we can output the inverted index of words.

```
String[] wordsL = value.toString().split(" ");
        long numberofwordstokeep = Math.round(wordsL.length
                - (wordsL.length * 0.8) + 1);
        String[] wordstokeepL = Arrays.copyOfRange(wordsL, 0,
                (int) numberofwordstokeep);

        for (String wordtokeep : wordstokeepL) {
            word.set(wordtokeep);
            context.write(word, key);
        }
```

Then the rest part nearly the same as what we implemented in question a).

```
hadoop jar similarity.jar mdpassignment.similarity.inverted_index_sim
input/tokensinline.txt output
hadoop fs -getmerge output workspace/similarity/output/inverted_index_sim.txt

hadoop fs -getmerge inverted_index_com_sim.txt workspace/StringSimilarityJoins/output/
inverted_index_com_sim.txt
```

The mapreduce process:
The number of performed comparisons is 1453.

```
                WRONG_MAP=0
                WRONG_REDUCE=0
        mdpassignment.similarity.inverted_index_sim$CUSTOM_COUNTER
                inverted_index_com_sim=1453
        File Input Format Counters
                Bytes Read=37575
        File Output Format Counters
                Bytes Written=694
```

The result similarity.txt shows below:

similarity.txt | inverted_index_sim.txt

```
(7343, 1456), 1.0
(2286, 7142), 0.9
(2286, 1255), 0.9
(7142, 1255), 1.0
(2286, 1255), 0.9
(2286, 7142), 0.9
(1255, 7142), 1.0
(7406, 1519), 1.0
(1322, 7209), 1.0
(7529, 1642), 1.0
(7209, 1322), 1.0
(1575, 7462), 1.0
(2228, 7082), 1.0
(2228, 1195), 1.0
(680, 0), 0.8
(7082, 1195), 1.0
(1456, 7343), 1.0
(7529, 1642), 1.0
(0, 680), 0.8
(7209, 1322), 1.0
(7142, 2286), 0.9
```

Logs in YARN: The execution time of inverted_index_sim with top 800 lines is **22sec**.

Logged in as: dr.who

MapReduce Job job_1489416105189_0005

Job Overview

Job Name:	inverted_index_sim
User Name:	cloudera
Queue:	root.cloudera
State:	SUCCEEDED
Uberized:	false
Submitted:	Mon Mar 13 08:00:40 PDT 2017
Started:	Mon Mar 13 08:00:47 PDT 2017
Finished:	Mon Mar 13 08:01:09 PDT 2017
Elapsed:	22sec
Diagnostics:	
Average Map Time	6sec
Average Shuffle Time	9sec
Average Merge Time	0sec
Average Reduce Time	1sec

c)

The execution time of similarity with top 800 lines is **2mins, 17sec**.
The number of performed comparisons is **319600**.

The execution time of inverted_index_sim with top 800 lines is **22sec**.
The number of performed comparisons is **1453**.

Obviously, we can find that the method which implements the inverted index is much faster than the original method. The reason of this is that the similarity with inverted index is intended to reduce the amount of comparison, which is a computationally intensive step. And it can really save a lot of time as we can see the execution time with huge difference from the figures. In this assignment, we just picked up 800 top lines from the file tokensinline.txt. As the amount of lines gets larger, the huger gap of execution time we will get between these two methods.

The reason why we have difference in the amount of comparisons is that the second method only indexes some words for each document, which is the less frequent words in the corpus.