Cloud Programming HW02

Ming-Chang Chiu 100060007

May 18, 2015

1 Introduction

In this assignment, we are required to implement a basic search engine using inverted indexing and tf-idf theory. Inverted indexing is to build a table which allows us to retrieve the desired information by searching keywords. tf-idf theory is meant to calculate the importance of the keyword for us to rank the output prioirties.

2 System Details: Algorithms

The system contains two parts, inverted indexing, and information retrieval. In the first part, I have four phases:

Mapper: In Mapper, since the MR framework will give me each line and the offset in the input file, I filter out the special notations and the make the output as (key,value) = (word:filename, 1:offset)

Partitioner: In this phase, I simply make the output of the starting letter of key range form 'A' to 'G' or 'a' to 'g' to part-00000 under my output directory and the rest in part-00001.

Combiner: Since I have word of one kind and the file it reside in as key, 1:offset as value, I can calculate how many words of one kind in a file and their offsets. Therefore, my output is (word, file:term count: offset1:offset2....) in this phase.

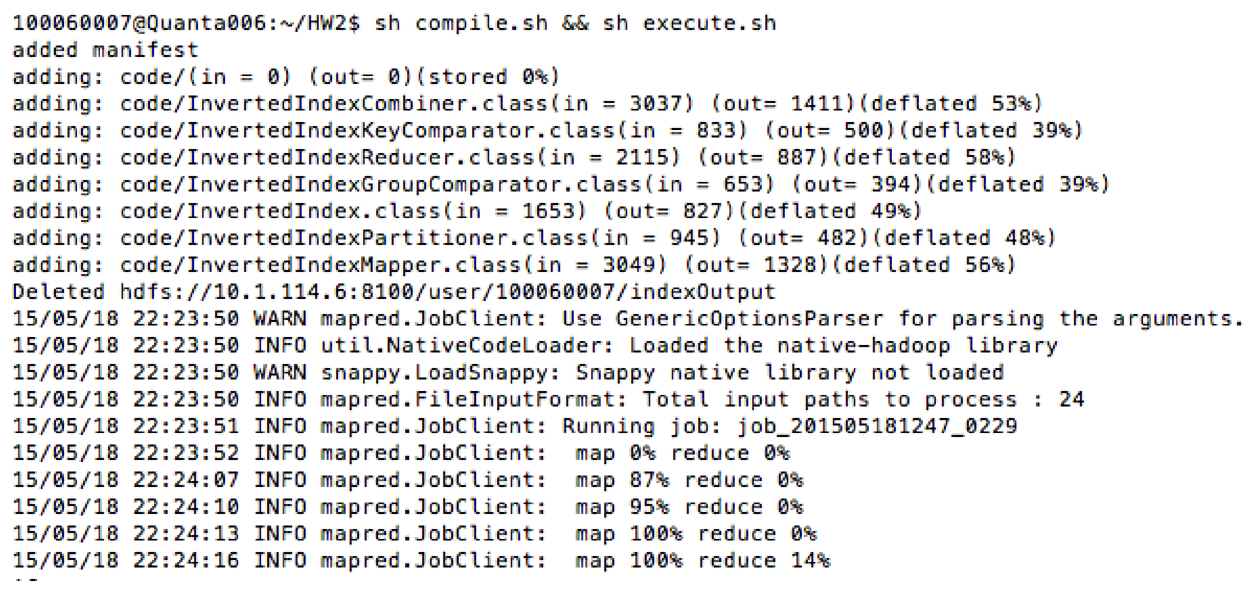
Reducer: The MR frame work will group keys of the same word for me so I simply concatenate the values and then calculate the document numbers. So, my final output is (word:document count, filename1:term count1:offset1:offset2...; filename2....;......).

For the second part, I have not much to say about it. I just use HDFS API to retrieve the table stored in my output directory and then search for the result. The interesting part is to google the HDFS API to manipulate the HDFS files, and the calculate the tf-idf for the keywords. Finally the program should print the line segments by parsing the inverted index table. But I cannot think of any dazzling algorithm or fancy implementation in sum.

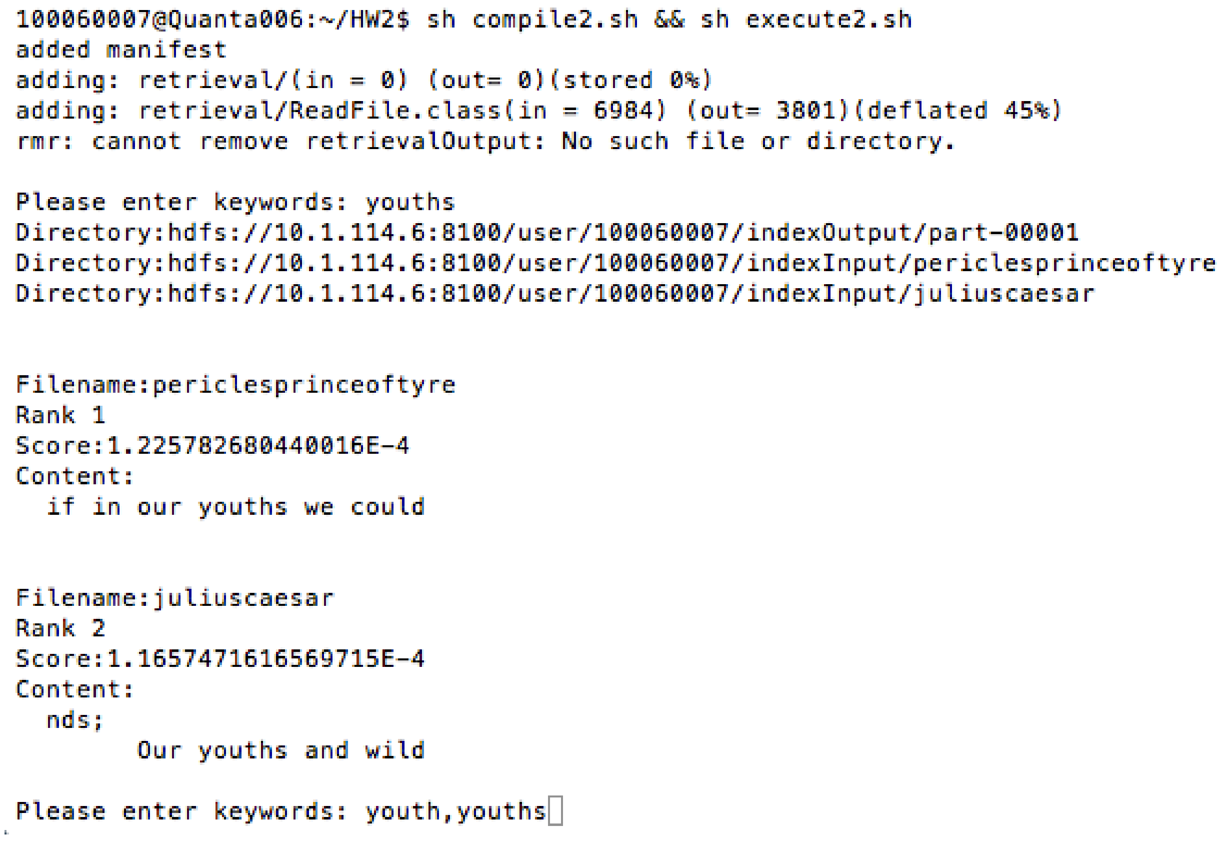
3 Usage

In my directory, there are multiple .sh files in it, compile.sh, compile2.sh, execute.sh, and execute2.sh. To build the inverted index, first user should make sure there are input files on HDFS, then type ”sh compile.sh && sh execute.sh” in the command line, and this command shall run a hadoop job named ”inverted index” and them output my inverted index on the screen. Notice that first letters of words starting from ’a’ ’g’, ’A’ ’G’ are stored in /indexOutput/part-00000, and the rest are in /indexOutput/part-00001.

To retrieve the information, simply type ”sh compile2.sh && sh execute2.sh” in the command line, the command line will then ask you to type in keywords you want to search. After you enter the keywords, no matter uppercase or lowercase, the program will print out the results of the first keyword ranked by tf-idf and then those of latter keywords in the same way. Notice that each keyword MUST be separated by a comma(i.e. ’,’)with no other extra redundant letters. I did not do much work on preventing illegal input. Easier said than done, a screen shot would be more helpful:



```
100060007@Quanta006:~/HW2$ sh compile.sh && sh execute.sh
added manifest
adding: code/(in = 0) (out= 0)(stored 0%)
adding: code/InvertedIndexCombiner.class(in = 3037) (out= 1411)(deflated 53%)
adding: code/InvertedIndexKeyComparator.class(in = 833) (out= 500)(deflated 39%)
adding: code/InvertedIndexReducer.class(in = 2115) (out= 887)(deflated 58%)
adding: code/InvertedIndexGroupComparator.class(in = 653) (out= 394)(deflated 39%)
adding: code/InvertedIndex.class(in = 1653) (out= 827)(deflated 49%)
adding: code/InvertedIndexPartitioner.class(in = 945) (out= 482)(deflated 48%)
adding: code/InvertedIndexMapper.class(in = 3049) (out= 1328)(deflated 56%)
Deleted hdfs://10.1.114.6:8100/user/100060007/indexOutput
15/05/18 22:23:50 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
15/05/18 22:23:50 INFO util.NativeCodeLoader: Loaded the native-hadoop library
15/05/18 22:23:50 WARN snappy.LoadSnappy: Snappy native library not loaded
15/05/18 22:23:50 INFO mapred.FileInputFormat: Total input paths to process : 24
15/05/18 22:23:51 INFO mapred.JobClient: Running job: job_201505181247_0229
15/05/18 22:23:52 INFO mapred.JobClient:  map 0% reduce 0%
15/05/18 22:24:07 INFO mapred.JobClient:  map 87% reduce 0%
15/05/18 22:24:10 INFO mapred.JobClient:  map 95% reduce 0%
15/05/18 22:24:13 INFO mapred.JobClient:  map 100% reduce 0%
15/05/18 22:24:16 INFO mapred.JobClient:  map 100% reduce 14%
```



```
100060007@Quanta006:~/HW2$ sh compile2.sh && sh execute2.sh
added manifest
adding: retrieval/(in = 0) (out= 0)(stored 0%)
adding: retrieval/ReadFile.class(in = 6984) (out= 3801)(deflated 45%)
rmr: cannot remove retrievalOutput: No such file or directory.

Please enter keywords: youths
Directory:hdfs://10.1.114.6:8100/user/100060007/indexOutput/part-00001
Directory:hdfs://10.1.114.6:8100/user/100060007/indexInput/periclesprinceoftyre
Directory:hdfs://10.1.114.6:8100/user/100060007/indexInput/juliuscaesar


Filename:periclesprinceoftyre
Rank 1
Score:1.225782680440016E-4
Content:
  if in our youths we could


Filename:juliuscaesar
Rank 2
Score:1.1657471616569715E-4
Content:
  nds;
        Our youths and wild

Please enter keywords: youth,youths
```

Notice: In order to run correctly on each directory, one should check those 4 .sh files for the directory names. And one should modify the retrieval/ReadFile.java for the HDFS paths. The above description is based on my working directory on Quanta006.

4 Questions

How many phases I use in part1? Is there any other way to do it? Whats the pros and c

I use four phases in part1, Mapper, Partitioner, Combiner, Reducer. Using partitioner to discriminate words from different starting letters is easier for me to debug, for instance, if my keyword is ”youth,” then I can go to part-00001 to examine my inverted index table. Using Combiner is to count the term count can be more efficient.

Whats the most difficult part in your implementation? Whats your extension?

I think it is part1, building full inverted index table. I have to fully understand the structure of inverted index and every detail about MapReduce API like Reporter, TextInputFormat and so on. I was not familiar with Java so it took me even more time to process the String, Array, Set, etc, in part1. To me, the tricky part is to come up with a the flow chart, like, what is the output of Mapper, what should be done in Combiner.

I did a basic action on preventing user to enter word that is not in the table.

How do you filter those useless notation? I simply use tokenizer's built-in function, I provide delimiters and let the Java API to help me.

If we need to search these special notations, how to modify your filter? If we are required to find those special notations, I can use Pattern and Matcher APIs provided by Java to find them and locate them, it is not very hard to implement.