CS410 MP3---MapReduce

Due: April 10th 11:59 PM

Introduction

This is an individual assignment. Your task will be: - complete the implementation of the MapReduce indexer, build index on a Hadoop cloud-computing cluster, and run retrieval experiments using a basic TF-IDF retriever of a simple Java retrieval toolkit on a given retrieval test collection to obtain baseline retrieval accuracy. - In this assignment, we will use this server: *sp17-cs410-xxx.cs.illinois.edu*

Complete the implementation of a simple retrieval toolkit based on Hadoop

The purpose of this task is to give you hands-on experience with implementing a simple IR system using MapReduce/Hadoop. As the data sets in many applications keep growing, parallel indexing using a framework such as MapReduce/Hadoop is becoming increasingly important for both research in IR and development of practical applications. Through this assignment, you will become familiar with the Hadoop File System (HDFS), learn how to create an inverted index using MapReduce, learn how to run a standard retrieval model by using an inverted index stored in HDFS. You are provided with an almost-complete simple retrieval toolkit and your first task is to complete the implementation of this toolkit. Starting with such a toolkit would minimize your work on coding and allow you to focus on learning the most essential knowledge about this topic. This simple retrieval toolkit has all the essential components of a retrieval system except that three of the Java source files have a few lines of code missing, and your task is to fill in those a few lines of code for each to make it really work. Here's what you need to do exactly:

First, do the following to set up the toolkit.

Login to your VM account using the following command

```
ssh YourID@sp17-cs410-YourNumber.cs.illinois.edu
```

If you have problems connecting to the server, consider using this VPN. Create a directory under your home directory for finishing this assignment, and name it "cs410" by doing the following:

```
mkdir cs410
```

Enter the "cs410" directory and download the toolkit from the path "/data/simir.tar.gz" by doing the following:

```
hadoop fs -copyToLocal /data/simir.tar.gz
tar -zxvf simir.tar.gz
```

You should see the "assign4" directory under "cs410/", and if you enter "assign4", you will see three sub-directories: (1) assign4/src: all the Java source files and Perl scripts for evaluation; (2) assign4/obj: directory to put all the object code (i.e., .classes); (3) assign4/exp: the directory which you will use for storing and analyzing retrieval results. You will be mostly working in the directory "assign4".

Second, take a look at the following two provided data files in the HDFS: - Document collection (about 500MB news articles): */data/docsrc/apsrc.txt* - Queries (47 TREC queries): */data/query/query.txt*

To view these files, use the following commands:

```
hadoop fs -cat /data/docsrc/apsrc.txt | more
hadoop fs -cat /data/query/query.txt | more
```

You will see that both are formatted such that each document (or query) occupies one separate line with the document ID (or query ID) at the beginning, followed by a sequence of terms. All the terms were stemmed with a Porter stemmer. Third, study the four Java source files in the "src" subdirectory to understand how the toolkit works: - **InvertedIndex.java**: this is the MapReduce-based inverted indexer. It takes as input all the source files in an HDFS directory and creates a "raw" inverted index file where each line is of the form "term1 doc1 termfreq1 doc2 termfreq2 .... " (indicating that term1 occurs in doc1 termfreq1 times, in doc2 termfreq2 times, etc). This is a raw inverted index because the term frequencies are stored in ASCII form (which need to be parsed into integers at run time) and we do not yet have a lexicon that can tell us the starting position for each term. We will generate the final inverted index by using another program (i.e., IndexGeneration) - **IndexGeneration.java**: this program takes the output from InvertedIndex as input and generates a final inverted index, which includes two files: "IndexFileName.lex" and "IndexFileName.pos", where "IndexFileName" is any given index file name. "IndexFileName.lex" is the lexicon, which is a table with 5 columns, each tuple is of the form (termID, documentFrequency,countInCollection,postingStartPosition,postingSpan). This lexicon allows us to quickly read in all the posting entries for a given query term stored in the posting file "IndexFileName.pos". All the output files are HDFS files. - **ComputeDocLen.java**: all effective retrieval functions need information about document lengths, which are provided through a file generated by this program. Similar to IndexGeneration.java, this program also takes the output from InvertedIndex (i.e., the raw inverted index) as input. It then uses MapReduce to generate a document length lexicon with each entry containing a pair (docID, docLength). This document length file should be eventually named as "IndexFileName.dlen" and put together with the ".lex" and ".pos" files created by IndexGeneration to form a complete inverted index. - **Retrieval.java**: this program takes as input (1) a complete inverted index (i.e., XX.lex, XX.pos, and XX.dlen), specified by "XX", (2) a query file (which may contain multiple queries each at a line), and optionally (3) a

numerical value for the retrieval parameter. The meaning and the valid range of the retrieval parameter depend on the actual retrieval function implemented. It then generates as standard output a ranked list of documents for each query (up to 1000 results for each query).

All the code is fairly well documented so if you just read through the code carefully, you can understand how it works.

Fourth, fill in the missing lines in InvertedIndex.java, IndexGeneration.java and Retrieval.java. Each file only has a few lines missing, so once you understand how the code works, it won't take long to fill in the missing statements. If you aren't familiar with notations of Java, you may need to look up relevant functions/classes using this website to understand how to use a particular function. The places where you need to add missing statements are all marked with comments of the following format, so it's easy for you to spot them.

1. Complete the file InvertedIndex.java

Start with the file InvertedIndex.java. After you finish this file, do the following to test the file:

Create a directory for this assignment in HDFS by using the following command if not already created:

```
hadoop fs -mkdir /home/YourID/cs410
```

Further create a subdirectory to store the inverted index that you will build.

```
hadoop fs -mkdir /home/YourID/cs410/index
```

Going to the "assign4" directory. All the following commands should be run while you are in "assign4". Compile the java files using the following command (in one line)

```
javac -classpath
/data/hadoop-common-2.7.3.jar:/data/hadoop-mapreduce-client-core-2.7.3.jar:/data/ha
doop-annotations-2.7.3.jar -d obj src/InvertedIndex.java
```

Generate a Java Archive (.jar) file using the following command

```
jar -cvf simir.jar -C obj ./
```

Execute InvertedIndex using the following command

```
hadoop jar simir.jar InvertedIndex /data/docsrc/ /home/YourID/cs410/tmp1
```

This will probably take about 10 minutes. The program would put the output (i.e., the raw inverted index file) in /home/YourID/cs410/tmp1/. Note that if the directory /home/YourID/cs410/tmp1/ already exists, you should delete it before running InvertedIndex, or you should specify a directory that doesn't already exist as output. The output is usually written to a file named as "part-00000".

To verify whether your program generates the results correctly, you may use a toy test data to test your program (You can't use this dataset for evaluation and submission):

```
hadoop jar simir.jar InvertedIndex /data/testsrc/ /home/YourID/cs410/tmp1
```

If you write the result to the same place (e.g. /home/YourID/cs410/tmp1), you need to first remove the results generated.

Once your InvertedIndex works well, you can compile and run the program ComputeDocLen to generate a document length file. Again, make sure that you are in "assign4". Do the following:

```
javac -classpath
/data/hadoop-common-2.7.3.jar:/data/hadoop-mapreduce-client-core-2.7.3.jar:/data/ha
doop-annotations-2.7.3.jar -d obj src/ComputeDocLen.java
```

```
jar -cvf simir.jar -C obj ./
```

```
hadoop jar simir.jar ComputeDocLen /home/YourID/cs410/tmp1/part-00000
/home/YourID/cs410/tmp2
```

This would generate a document length file and put it in /home/YourID/cs410/tmp2/part-00000. You may take a look at the file to see if it looks right and then copy it to the final index directory

```
hadoop fs -cp /home/YourID/cs410/tmp2/part-00000 /home/YourID/cs410/index/ind.dlen
```

2. Complete the file IndexGeneration.java

Next, work on IndexGeneration.java and add the missing statements. Go to the "assign4" directory and test your implementation by doing:

```
javac -classpath
/data/hadoop-common-2.7.3.jar:/data/hadoop-mapreduce-client-core-2.7.3.jar:/data/ha
doop-annotations-2.7.3.jar -d obj src/IndexGeneration.java
```

```
jar -cvf simir.jar -C obj ./
```

```
hadoop jar simir.jar IndexGeneration /home/YourID/cs410/tmp1/part-00000
/home/YourID/cs410/index/ind
```

Again, you may want to test your program by using the toy data set first to make sure it works correctly. Once your IndexGeneration program works correctly, it would generate "ind.lex" and "ind.pos" and put them in the "cs410/index" directory, which together with "ind.dlen" that you generated earlier form the complete inverted index for the provided news data set.

3. Complete the file Retrieval.java & Experiment with the retrieval toolkit

Finally, experiment with the retrieval toolkit based on Retrieval.java. Go to the "assign4" directory and do:

```
javac -classpath
/data/hadoop-common-2.7.3.jar:/data/hadoop-mapreduce-client-core-2.7.3.jar:/data/ha
doop-annotations-2.7.3.jar -d obj src/Retrieval.java
```

```
jar -cvf simir.jar -C obj ./
```

```
hadoop jar simir.jar Retrieval /home/YourID/cs410/index/ind  /data/query/query.txt
> exp/result
```

Note that when you compile Retrieval.java, you would see the following warning. This isn't a problem, so please ignore it.

```
Note: src/Retrieval.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

If your implementation is correct, you should get retrieval results in the file "result" under the "exp" sub-directory. Note that the retrieval results are now in your local directory (i.e., it's not an HDFS file). You can view the file normally by using, e.g., "more exp/result". The results are a sequence of tuples of the form "queryID docID score".

Now, go to the directory "exp" and evaluate this result by doing the following

```
perl ../src/ireval.pl  -j qrel -o pr < result
```

This would generate a TREC-style evaluation result and store it in the file "pr" in the directory "exp".

From the file pr, you can extract the Mean Average Precision (MAP), precision at 10 documents, and other measures. Specifically, the pr file will have such results for each of the queries, and in the end, the average of all the figures over all the queries. The MAP over all the queries is the most important measure, and it's reported in the line "Set average (non-interpolated) precision = ...". Since the toolkit only implemented a naive TF-IDF retrieval model, its retrieval accuracy is not very good. But you should be able to get a MAP above 0.04 if your implementation is correct.

4. Submission

Please submit a PDF with the code snippets you implemented for the retrieval toolkit and the evaluation result and pack all the following files into one single zip or tar file to Compass by midnight of the due date:

3 completed Java source files (i.e., InvertedIndex.java, IndexGeneration.java and Retrieval.java) and the precison file of the basic TF-IDF retrieval function as implemented in the original toolkit (i.e., the file generated by "ireval.pl").