CPSC 5330 — Spring 2024
Week 3 Lab: Term Frequency Using MapReduce and Hive

Part 0, Style Points (5 points)

(These are easy points to get, just by following instructions and handing in good clean work!)

Was the output clean and well presented? Was the text and explanations well written? Were all the files there and named according to instructions?

Part 1, Term Frequency (35 points)

Computing "term frequency" is the first step in a pipeline that indexes documents for the purpose of keyword query searching. The search will be over a corpus of text documents (books, web pages, etc.) and each document has an ID. Term frequency is a triple

```
(document_id, term, frequency_pct)
```

Our documents will be books we like we used in Lab 1, and each file contains the text for a single book. You are already familiar with the concept of a term: taking a word (a string not containing whitespace), removing all characters except letters, and converting letters to lower case. There are many ways of converting a word to a term; for this lab we will use the same rules we used in the Hadoop Streaming word count examples.

The concept of "document ID" is new. For our application we will use the file name as the document ID – if the input file was /data/textcorpora/shakespere-hamlet.txt for example, we would use shakespere-hamlet as the document_id. Code below will show you how a mapper can get the document ID for the line it is working on.

The last element of the tuple is `frequency_pct`, which is the number of occurrences of the term in the document divided by the total number of terms in the document.

You will get to `(document_id, term, frequency_pct)` in three steps

1. A map-reduce job reads the input document and produces tuples of the form `(document_id, term, count)`
2. A second map-reduce job reads *the output of the first map-reduce job* and produces tuples of the form `(document_id, count)`
3. A Hive script joins these two data sets on document ID and produces tuples of the form `(document_id, term, frequency_pct)`

We will use streaming Hadoop, and mappers and reducers written in Python.

Here is what's new and not new about this problem

1. The first map-reduce job from document to `(document_id, term, count)` looks like the usual word count (with term cleaning) except the mapper needs to get a document_id for the line it is processing. The code for this mapper is provided for you below.
2. Also for the first map-reduce job, one thing that is different from the word count you have seen is that the key for the reducer is not `document_id` (the first element in the tuple generated by the mapper), rather it is both `document_id` and `term`. In other words, you want Hadoop to group on two fields instead of one -- both document_id and term. You can control that in your call to Hadoop Streaming, by adding this flag to the call to Hadoop:
   `-D stream.num.map.output.key.fields=2`
3. The second map-reduce job transforms `(document_id, term, count)` tuples to `(document_id, count)` tuples, which is very simple. The only complication is that you can't use the `run-hadoop-streaming` script in lab 2 as is, because that script assumes its input will come from `/input` and its output will go to `/output`. You want the input for the second map-reduce job to be the output of the first map-reduce job. You will write customized scripts for both map-reduce jobs that will be based on `run-hadoop-streaming`, but your scripts will not call `run-hadoop-streaming`.
4. The third step is a simple version of what we did in the class Demo in week 3 to compute average population. Your third step does a join, then does a division. First you create a Hive table for the output of the first map-reduce job `(document_id, term, term_count)` and a Hive table for the output of the second map-reduce job `(document_id, doc_count)`. Then you join the two tables on `document_id` and compute `term_frequency = term_count/doc_count` which produce tuples of the form `(document, term, frequency_pct)`.

How To Structure Your Work

All of your code will be in a folder named `term-frequency`. You will have two scripts named `run-term-count-doc-and-term` and `run-term-count-doc` to run streaming map reduce, and two folders named `term-count-doc-and-term` and `term-count-doc`, which will contain the mapper and reducer for the two map reduce jobs.

You will have a Hive script `tf.hive` that will create the Hive tables, link them to the output of the two map reduce jobs, do the SELECT statement to compute term frequency, and export the results to the local filesystem.

Finally you will have a shell script named `run-tf` which will run the map reduce jobs, then run the Hive script, then clean up.

Hints and assumptions

- Directories `term-count-doc-and-term` and `term-count-doc` each contain a mapper and reducer file
- Scripts `run-term-count-doc-and-term` and `run-term-count-doc`, both will be shell scripts with a single call to Hadoop. They will be edited versions of `run-hadoop-streaming`, but will not take any command-line parameters.

- A Hive script `tf.hive` that will create Hive tables from the output of the two map reduce outputs, and will run a Hive SQL query with output to the local filesystem with the tuples `(document_id, term, frequency)`. These fields should be comma separated.
- The main script will be a shell script named `run-term-frequency`, which will
    - Run the two map reduce jobs
    - Run the Hive script
    - Move the Hive output part files into a single text file `tf.txt`
    - Clean up all temporary files and directories on the local filesystem and on HDFS
- Hint: in the class demo we ran Hive in an interactive mode where one (server) process ran `hiveserver2` then another (client) process used `beeline` to make Hive queries interactively. For this lab we want to run Hive in "batch" mode, and so it will be easier not to run hiveserver2, and instead just use a shell command like `hive < tf.hive`

Summary: your solution directory `term-frequency` contains:

- Directories `term-count-doc-and-term` and `term-count-doc` with files `mapper` and `reducer`
- File `tf.hive`
- File `run-term-frequency`

When we run that script you may assume

- The working directory is your solution directory `term-frequency`
- HDFS, Hadoop, and Hive services are all running
- The document collection is already in HDFS in /input/textcorpora
- There are no other files or directories in HDFS

After running your script the only change should be that the file `tf.txt` exists in the working directory, and contains one comma-separated line for each document_id/term pair.

Hints:

- The only way you will know for sure that your solution is fully working is to
    - Create a new EC2 instance using the Week 3 AMI Image
    - Copy your solution directory to the instance
    - Run the startup script on the image
    - Create the HDFS directory `/input` and copy the text corpora into `/input/textcorpora`
    - `cd` to `term-frequency`
    - Run the script `run-tf`
    - Run the command (below) to create the `abhor.txt` file

- If those steps run without error and you can verify that the numbers in `tf.txt` and `abhor.txt` are accurate, you are done. Otherwise you aren't done.

Submission

A `zip` file with:

1. The directory `term-frequency`
2. A file `abhor.txt` which you will generate after running your solution, using the command
   ```
   grep ,abhor, tf.txt > abhor.txt
   ```
3. A retrospective report in a file `retrospective.pdf` – a reflection on the assignment, with the following components
   a. Your name
   b. How much time you spent on the assignment
   c. If parts of the assignments are not fully working, which parts and what the problem(s) are
   d. Were there aspects of the assignment that were particularly challenging? Particularly confusing?
   e. What were the main learning take-aways from this lab – that is, did it introduce particular concepts or techniques that might help you as an analyst or engineer in the future?

Mapper Code for `term-count-doc-and-term`

```
#!/usr/bin/python3

# In:  string w/ document or line from document
# Out:  (docid, term, 1)

import sys
import re
import os

def termify(word):
    regex = re.compile('[^a-z]')
    return regex.sub('', word.lower())

for line in sys.stdin:
     docid = os.path.splitext( \
                 os.path.basename(os.getenv('map_input_file')))[0]
      for term in map(termify, line.strip().split()):
        if term:
            print('%s\t%s\t%s' % (docid, term, 1))
```