Lab 12

Objectives:

- Get hands-on experience running MapReduce and gain a deeper understanding of the MapReduce paradigm.
- Become more familiar with Apache Spark and get hands on experience with running Spark on a local installation.
- Learn how to apply the MapReduce paradigm to Spark by implementing certain problems/algorithms in Spark.

Setup

Please follow the guide in starter file to setup the environment. Also, please pull the files from the repository.

NOTE: some files are managed by Git LFS, make sure you have pulled the correct files.

Exercises

Exercise 1

Please finish the environment setup, exercise 0, and exercise 1 in the guide.

Checkoff

- Show TA that you have finished the above exercises.

Exercise 2: How many documents does each word appear in?

Earlier, we used the `—END.OF.DOCUMENT—` token to split a text file into multiple documents. The sample files included in this lab are also split into documents. For example, `billOfRights.txt` is split into 10 documents (one for each amendment). For this exercise we want to count how many documents each word appears in. For example, `"Amendment"` should appear in all 10 documents of `billOfRights.txt`.

Open `perWordDocumentCount.py`. It currently contains code that will execute the same functionality as `wordCount.py`. Modify it to count the number of documents containing each word rather than the number of times each word occurs in the input and to sort that output in alphabetical order.

To help you with understanding the code, we have added some comments, but you will also need to take a look at the list of Spark transformations for a more detailed explanation of the methods that can be used in Spark. There are methods that you can use to help sort an output or remove duplicate items. To help with distinguishing when a word appears in a document, you will want to make use of the document ID as well – this is mentioned in the comments of `flatMapFunc`. Just because we gave you an outline doesn't mean you need to stick to it, feel free to add/remove transformations as you see fit. You're also encouraged to rename functions to more useful titles.

You can test `perWordDocumentCount.py` (with results in `spark-wc-out-perWordDocumentCount/part-00000`) with the following command:

```
$ spark-submit perWordDocumentCount.py seqFiles/billOfRights.txt.seq
```

You should also try it on the other sequence files you have to look for some interesting results.

Checkoff

- Explain your modifications to `perWordDocumentCount.py` to your TA.
- Show your output from `billOfRights` to the TA. In particular, what values did you get for "Amendment", "the", and "arms"? Do these values make sense? You can confirm your results by looking through the `billOfRights.txt` file.

Exercise 3: Full Text Index Creation

Next, for each word and document in which that word appears at least once, we want to generate a list of index into the document for EACH appearance of the word, where an index is defined as the number of words since the beginning of the document (with the first word being index 0). Also make sure the output is sorted alphabetically by the word. Your output should have lines that look like the following (minor line formatting details don't matter):

```
(word1  document1-id, word# word# ...)
(word1  document2-id, word# word# ...)
. . .
(word2  document1-id, word# word# ...)
(word2  document3-id, word# word# ...)
. . .
```

Notice that there will be a line of output for EACH document in which that word appears and EACH word and document pair should only have ONE list of indices. Remember that you need to also keep track of the document ID as well.

For example, given a document with the text `With great power comes great responsibility`, the word `With` appears at index 0 while the word `great` appears at index 1 and 4, and the output would look like:

```
('comes doc_somerandomnumbers', 3)
('great doc_somerandomnumbers', 1 4)
('power doc_somerandomnumbers', 2)
('responsibility doc_somerandomnumbers', 5)
('With doc_somerandomnumbers', 0)
```

The file you should edit to do this task is `createIndices.py`. For this exercise, you may not need all the functions we have provided. If a function is not used, feel free to remove the method that is trying to call it. Make sure your output for this is sorted as well (just like in the previous exercise).

You can test by running the script with `spark-submit`:

```
$ spark-submit createIndices.py seqFiles/billOfRights.txt.seq
```

The results are stored in `spark-wc-out-createIndices/part-00000`. The output from running this will be a large file. In order to more easily look at its contents, you can use the commands `cat`, `head`, `more`, and `grep`:

```
$ head -25 OUTPUTFILE       # view the first 25 lines of output
$ cat OUTPUTFILE | more     # scroll through output one screen at a time (use Space)
$ cat OUTPUTFILE | grep the # output only lines containing 'the' (case-sensitive)
```

Make sure to verify your output. Open `billOfRights.txt` and pick a few words. Manually count a few of their word indices and make sure they all appear in your output file.

Checkoff

- Explain your code in `createIndices.py` to your TA. Next, run:

  ```
  $ spark-submit createIndices.py seqFiles/complete-works-mark-twain.txt.seq
  ```

- Show your TA the first page of your output for the word "Mark" in `complete-works-mark-twain.txt.seq` to verify correct output. You can do this by running:

  ```
  $ cat spark-wc-out-createIndices/part-00000 | grep Mark | less
  ```

Exercise 4: What's the most popular word?

Use Spark to determine what the most popular non-article word is in the Bill of Rights. (Articles are the words "a", "an", and "the", so ignore those) We have copied over the code from `wordCount.py` into a new script `mostPopular.py` since it is a good starting point.

Hint: After the `reduceByKey` operation has been run, you can still apply additional map operations to the data. Looking at the arguments for `sortByKey` may save you a lot of scrolling as well.

To test your code, run:

```
$ spark-submit mostPopular.py seqFiles/billOfRights.txt.seq
```

The results are stored in `spark-wc-out-mostPopular/part-00000`. As a fun exercise, try doing this on the book you downloaded in Exercise 0!

Checkoff

- Explain your code to the TA and what the most popular non-article word is.

Schwertfeger, Sören <`soerensch` AT `shanghaitech.edu.cn`>
Chundong Wang <`wangchd` AT `shanghaitech.edu.cn`>
Last modified: 2020-05-31