Lab 11

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

Pull the lab files from the lab starter repository with

```
$ git pull starter master
```

Be aware, this lab is only going to work on Hive machines.

You will also be working with Spark (in Python!), so you may need to brush up a bit on your Python!!!! To be able to run Spark, you must create a virtual environment using the correct version of Python. This can be done as such:

```
$ conda create --name lab11env python=2.7
```

Respond to the prompt to install packages with "y" (with no quotes). You can (and should) ignore any warnings about conda being out of date. These will take about 30 seconds to install. Finally, run the following command to activate the virtual environment:

```
$ source activate lab11env
```

This will put you in a virtual environment needed for this lab. **Please remember that if you exit the virtual environment and want to return to work on the lab, you must re-run `source activate lab11env` first for Spark to work.**

In addition, when logged into the Hive machines, run the following command in your Terminal:

```
$ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

If you do not do this, Spark will throw an error about being unable to find the correct Java version.

Background Information

In lecture we've exposed you to cluster computing (in particular, the MapReduce framework), how it is set up and executed, but now it's time to get some hands-on experience running programs with a cluster computing framework!

In this lab, we will be introducing you to a cluster computing framework called Spark. Spark was developed right at Berkeley before being donated to the Apache Software Foundation in 2013. We will be writing Python code to run in Spark to give us some practice in writing Map and Reduce routines.

Spark has its own website, so you are free to try to install it onto your local machines, although it may be easier to ssh into the lab computers to complete this lab.

Avoid Global Variables

When using Spark, avoid using global variables! This defeats the purpose of having multiple tasks running in parallel and creates a bottleneck when multiple tasks try to access the same global variable. As a result, most algorithms will be implemented without the use of global variables.

Documentation, and Additional Resources

- A quickstart programming guide for Spark (click the Python tab to see the Python code) is available here!
- The version of Spark we will be using will be 1.1.0 and the link to the API documentation is available here (Note that the docs likely say a different version, but the API should be compatible).

Exercises

Note: Different exercises may be solvable or needed to be solved by reconsidering how map(), flat_map() and reduce() are implemented and called and in which order, so keep this in mind when calling whichever you must use.

The following exercises use sample input files, which can be found in `seqFiles/`.

1. `billOfRights.txt.seq` – the first 10 Amendments of the US constitution split into separate documents (a very small input)
2. `complete-works-mark-twain.txt.seq` – The Complete Works of Mark Twain (a medium-sized input)

Notice the `.seq` extension, which signifies a sequence file that is readable by Spark. These are NOT human-readable. Spark supports other input formats, but you will not need to worry about that for this lab.

The human-readable text file version of each is included in `textFiles/` so you can open those to get a sense of the contents of each file.

Although an exercise may not explicitly ask you to use it, we recommend testing your code on the `billOfRights` data set first in order to verify correct behavior and help you debug.

Reminder: this lab will only work on Hive machines and requires you to first activate a python virtual environment, as described above.

Exercise 0: Generating an Input File for Spark

In this lab, we'll be working heavily with textual data. We have some pre-generated datasets as indicated above, but it's always more fun to use a dataset that you find interesting. This section of the lab will walk you through generating your own dataset using works from Project Gutenberg (a database of public-domain literary works).

Step 1: Head over to Project Gutenberg, pick a work of your choosing, and download the "Plain Text UTF–8" version into your lab directory.

Step 2: Open up the file you downloaded in your favorite text editor and insert `–END.OF.DOCUMENT–` by itself on a new line wherever you want Spark to split the input file into separate `(key, value)` pairs. The importer we're using will assign an arbitrary key (like `doc_xyz`) and the value will be the contents of our input file between two `–END.OF.DOCUMENT–` markers. You'll want to break the work into reasonably-sized chunks, but don't spend too much time on this part (chapters/sections within a single work or individual works in a body of works are good splitting points).

Step 3: Now, we're going to run our Importer to generate a `.seq` file that we can pass into the Spark programs we'll write. The importer is actually a MapReduce program, written using Hadoop! You can take a look at `Importer.java` if you want, but the implementation details aren't important for this part of the lab. You can generate your input file like so:

```
$ make generate-input myinput=YOUR_FILE_FROM_STEP_2.txt
```

Your generated `.seq` file can now be found in the `seqFiles/` directory in your lab directory. When you complete other exercise in this lab, run them on your downloaded file as well and investigate the results.

Exercise 1: Running Word Count

For this exercise you will use the already-completed `wordCount.py`. Open the file and take a look. Make sure you understand what the file is attempting to do.

You can run it on the `billOfRights` text with the following command:

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

Where `spark-submit` takes a python file describing a series of map and reduce steps, distributes that files between different worker processors (often across many physical computers, but just across local processors for this lab), and provides the `.seq` file as an input to that python file.

In this case, the command will run `wordCount.py` over `billOfRights.txt.seq`. Your output should be visible in `spark-wc-out-wordCount/part-00000`.

Next, try your the code on the larger input file `complete-works-mark-twain.txt.seq`.

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

Your output for this command will be located in the same `spark-wc-out-wordCount/part-00000` file (overwriting the previous results). Search through the file for a word like `'the'` to get a better understanding of the output.

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.

Check-off

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.

Check-off

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 word is in the Bill of Rights by generating a list of `(count, word)` tuples in decsending order. 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!

Check-off

Run `./submit` in the `lab11` folder, then submit your code to the **Lab Autograder** assignment. **Remember to commit the `outputs` folder!**. Below are some questions to consider:

Exercise 2:

- Explain your modifications to `perWordDocumentCount.py`

Exercise 3:

- Explain your modifications to `createIndices.py`

Exercise 4:

- Explain your code to the TA