Parallel and Distributed Computing
Lab3
MapReduce and Web Indexing

In this lab you will build a simple MapReduce process using Spark to build an index of web page references.

Web page reference index

It is fairly easy for automated web spiders to go out and collect link information from webpages. In particular, we have collected information about the links on a webpage that go to other webpages. This information will be stored in text files (see below). Each line in a file will have the format:

www.sourceURL.com www.link1.com www.link2.com

That is, the first string on the line is the URL location of a webpage. Each of the following strings on that same line are locations of links from that webpage. Note that for a given page, there could be any number of outgoing links (say, 0 to 100). There will be a large number of these lines as each webpage will have its own line. This information was gathered directly from webpages without much processing. There might even been several input files containing link this link information.

It is your job in this lab to turn this into more useable information. In particular, an index of webpage references. That is, for a given webpage, we wish to find out which other webpages referenced it. This type of index has many uses such as helping to determine the relevance of a given webpage for results in search engines. Thus, for each webpage we want to have information in the form:

www.webPage.com www.refFrom1.com www.refFrom2.com

Again, there may be any number of incoming reference links to the given page.

Spark MapReduce

To accomplish the task above, you will use a Spark implementation of MapReduce. You should use the WordCount.py example as your guide.

You will have to do a mapping which pairs each target URL as the first item in the pair with the URL of a page which links to it as the second element. You will have to pick from the reduction/group functions Spark provides to perform the reduction.

You should collect() your results after the MapReduce has completed and print them.

To get full credit, you should sort both the target URLs and the referencing URLs (see example below).

Input Files

Log into DataBricks and start a new cluster for the lab, along with a lab3 Notebook.

Next we will need to load our local text files into the DBFS (Data Bricks File System). The DBFS is the distributed filesystem that Spark uses within the databricks system. In order for our Spark code to access the files on each worker node, they will have to exist in the DBFS. There are several ways to load data into DataBricks, but the way I want you to do it is the following.

You are given 2 sets of data. A short set that you should use for development/testing purposes. And then a full set that has 100k lines of web page url data. Start with the short and once you have it working, then run the full set. The short set has 2 files of data, the full has 4 files.

Click on the DataBricks button in the top-left. Drag your files to the landing pad box in the middle of the screen. Do not create any tables from these files. This will place the files in the folder dbfs:/FileStore/tables/. You can see the contents of the filesystem via the DataBricks button and the Import & Explore Data link. This takes you to the Create Table page. And even though we are not going to create any new tables for this lab, you can switch to the DBFS view. You can then expand FileStore and tables to see the newly loaded files.

We will want a copy of these files to a different location for use in our code. I want you to do the rest of the dbfs file manipulation with the dbutils. Please put these commands into their own Notebook cmd. Your code will go in another cmd.

First you should make a directory to store your data files. You will make two of these eventually, one for the short and one for the full. Use the command:

dbutils.fs.mkdirs("yourDirectoryPathHere")

I would suggest “FileStore/tables/lab3short” as a good place to put the testing files.

Once you have created the directory, you should move or copy the files to that location. Do this with the mv or cp command. For example:

dbutils.fs.cp("fromFilePath", "toDirectory")

Note that you cannot use wildcards (*) in the filePath, so please write Python code with a loop that creates the filePaths for the given files. That is, don’t cut-paste the line 4 times to move the 4 files in the full set. Write code to automate this process.

Note that if you put things in the incorrect locations, etc. you may need to remove files from the dbms system. You can use the rm command to do this:

dbutils.fs.rm("yourFilePathHere")

There are other commands such as ls for directory listing too, but the ones given above should be enough for this lab. Don't erase your commands as they will need to be turned in as part of the lab.

Running on the cluster

Note that when Spark reads in input files with sc.textFile(), if you give it the directory name rather than a specific file name, it will automatically read in every file in that directory. This is why we wanted separate directories for the Short and Full data.

The Short version is perfect for doing debugging as it only contains a handful of webpages whereas the Full version contains 100k. For testing purposes, you should be able to look through the Short version and determine if your results are correct or not. But to help you get started, I've provided what your result line should look like for the example5 site. Note that the first item in the tuple is the string of the URL and the second item is a Python List that contains all the references. Your results should be in the same format.

```
('www.example5.com', ['www.example1.com', 'www.example20.com', 'www.example3.com'])
```

Note that each list of reference URLs should be in sorted order. The above example at first may not seem like sorted order, but it is sorted alphabetically when both letters and numbers are included (i.e. 20.com comes before 3.com since 2 comes before 3). Your overall list should also be sorted by target URLs. In this example, www.example5.com is the target URL.

Submission

You are to attach 3 documents to Canvas as your submission. The first two are your Python source code (PY) and DataBricks archive (DBC) files. Make sure to have your name in a comment at the top of your source code. Third is a document that has a cut-paste of your results from running your code on the Short input files as well as the Full input files. Note for the Short, you should print out the result of collect() since it is a small number. For the Full, print out both the count() for the final result as well as take(10) to grab the first 10 results.