# Introduction to Data Science (CAP4770/CAP5771) Lab III

Instructor: Dr. Daisy Zhe Wang

Oct 1, 2015

*Department of Computer and Information Science and Engineering*

*University of Florida*

# PageRank Implementation on Amazon Elastic MapReduce

# 1 PageRank

One of the biggest changes in our lives in the decade was the availability of efficient and accurate web search. Google is the first search engine which is able to defeat the spammers who had made search almost useless. The technology innovation behind Google is called PageRank. This project is to implement the PageRank to find the most important Wikipedia pages on the provided Wikipedia dataset using AWS Elastic MapReduce(EMR).

## 1.1 Definition of PageRank

PageRank is a function that assigns a real number to each page in the Web. The intent is that the higher the PageRank of a page, the more important it is. The equation is as follows:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)} \tag{1}$$

Where $d = 0.85$, $p_1, p_2, ..., p_N$ are the pages under consideration, $M(p_i)$ is the set of pages that link to $p_i$, $L(p_j)$ is the number of outbound links on page $p_j$, and $N$ is the total number of pages. In our project, we do not use teleport to deal with the sink node for simplicity. At the initial point, each page is initialized with PageRank $1/N$ and the sum of the PageRank is 1. But the sum will gradually decrease with the iterations due to the PageRank leaking in the sink nodes.

# 2 MapReduce and Amazon Elastic MapReduce

## 2.1 MapReduce

MapReduce is a framework for processing parallelizable problems across huge datasets using a large number of computers.

- Map step: The master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. The worker node processes the smaller problem, and passes the answer back to its master node.
- Reduce step: The master node then collects the answers to all the sub-problems and combines them in some way to form the output the answer to the problem it was originally trying to solve.

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel. Similarly, a set of reducers can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. Another way to look at MapReduce is as a 3-step parallel and distributed computation:

$$< K1,V1 > -map- < K2,V2 > -shuffle- < K2, alistofV2 > -reduce- < K3,V3 > \tag{2}$$

1. Prepare the *Map*() input the MapReduce system designates Map processors, assigns the $K1$ input key value each processor would work on, and provides that processor with all the input data associated with that key value. Run the user-provided *Map*() code *Map*() is run exactly once for each $K1$ key value, generating output organized by key values $K2$.
2. Shuffle the Map output to the Reduce processors the MapReduce system designates Reduce processors, assigns the $K2$ key value each processor would work on, and provides that processor with all the Map-generated data associated with that key value.
3. Run the user-provided *Reduce*() code *Reduce*() is run exactly once for each $K2$ key value produced by the Map step. Produce the final output the MapReduce system collects all the Reduce output, and sorts it by $K2$ to produce the final outcome.

## 2.2 Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) is a web service that makes it easy to quickly and cost-effectively process vast amounts of data. Amazon EMR uses Hadoop, an open source framework, to distribute your data and processing across a resizable cluster of Amazon EC2 instances.

*How to redeem and view AWS credits?* Sign into your account. In the upper right corner, click on the arrow next to your name and go to Billing & Cost Management. Next, in your Dashboard menu on the left, click on Credits and once you are there, you will be able to see all the relevant info such as the remaining balance, applicable products and services, and expiration date.

# 3 PageRank Implementation on Amazon EMR

The PageRank algorithm *must* be implemented using Hadoop 2.4.0 which is corresponding to the AWS EMR AMI version 3.9(Hadoop 2.4.0). Your implementation might follow these steps.

1. Write a MapReduce job that extracts wikilinks and also remove all the red links. This will be a dataset that contains, for each article, a list of links going out of that article, as well as the articles title and PageRank. This is the data structure that all of the PageRank steps will operate on. The input for this job should be the Wikipedia dataset, and the job should look at the XML text for each article, extract title and wikilinks. A single page would be represented as follows (with some fields and attributes omitted):

   ```
   <page> <title>Title</title>…
   <text …>
     Body of page
     [[link text]]
   </text>
   </page>
   ```

   Your task is to extract the **Title** and the **wikilinks** from these articles. Within the text of the page, a wikilink is indicated by [[link text]]. In the simplest kind of link, the link text is the name of another page (with any spaces replaced with underscores). A more complicated kind gives different text to appear as well as the name of the page: [[page name|text to appear]]. How to differentiate wikilinks to other types of links? Please refer to the http://en.Wikipedia.org/wiki/Help:Wikilinks#Wikilinks.

There two formats of wikilinks as described in the above Wikipedia page. Let me repeat here:

(a) [[abc]] is seen as "abc" in text and links to page "abc". We need to extract "abc" out.
(b) [[a|b]] is labelled "b" on this page but links to page "a". We need to extract "a" out.

But "abc" in (a) and "a" in (b) may be a red link which basically mean there is no page with title "abc" and "a" in the wiki dataset. You need to exclude all red links in this job.

Note: we need to replace all the empty space ' ' in the Title and links(title of other pages) with '_'. No other processing is needed in this project.

2. Write a MapReduce job to generate the outlink adjacency graph.

3. Write a MapReduce job to compute the total number of pages denoted as $N$ in the equation 1.

4. Write a MapReduce job that performs PageRank for exactly *8 iterations*. This job should take as input a link graph, update the PageRank of each page using the propagation formula, and output a new link graph. This process repeats until the maximum number of iterations is reached.

5. Write a MapReduce job that takes your final link graph and prints out a human readable list of article names and PageRank scores, sorted in descending order of PageRank. Hint: You will need to use a single reduce task, so that all of the data is sorted by the same process.

# 4 Input and Output

You must exactly follow the input and output format described in the following subsections. If you fail to follow the format, your program may not be able to parse the input file and the TAs' grading script may not be able to interpret your output.

## 4.1 Input

You can use the input file in s3://uf-dsr-courses-ids/data/enwiki-latest-pages-articles.xml. The input file is made public readable, so you have the permission to read it. We will provide correct output files too. (Please note that for grading, we will use a different input file)

## 4.2 Output

### 4.2.1 produce outlink graph

You should produce a outlink graph file named *PageRank.outlink.out* as described below:
pageTitle outlink1 outlink2 outlink3 ... outlinkn.

The first the entry is the page title and it is followed by the all the outlinks in that page. The order of outlinks doesn't matter. Tab is used to seprate the data column.

### 4.2.2 produce the total number of pages

You should have a file named *PageRank.n.out* to record the total number of pages(Nodes) *N* in the outlink graph. the output format of the n.out is simply one line:
N=your result

### 4.2.3 produce the PageRank

You should have two output files PageRank.iter1.out and PageRank.iter8.out under the path s3://your-bucket/results/. As the names suggested, PageRank.iter1.out is the PageRank result at the end of iteration 1(the first iteration). PageRank.iter8.out is the PageRank result at the end of eighth iteration(last iteration). Each file should have the Wikipedia title and its corresponding PageRank in the descending order of PageRank. The two columns are separated by a tab. Please *only* print out the entries with PageRank$\geq \frac{5}{N}$. The sample output is as follows:

```
Main Page	0.1
index.html	0.08
GOOGLE	0.06
Wiki	0.05
```

### 4.2.4 Input & Output

In sum, the S3 buckets directory layout can be described as follows:

```
Bucket-name: uf-dsr-courses-ids
    data/enwiki-latest-pages-articles.xml (the input data)
your-bucket-name
    results/PageRank.outlink.out
    results/PageRank.n.out
    results/PageRank.iter1.out (output file for iteration 1)
    results/PageRank.iter8.out (output file for iteration 8)
    logs/ (the job log direcotry)
    job/PageRank.jar (your job jar)
    tmp/ (temporary files, you might or might not need it)
```

Your bucket name must be passed as an argument to the program. The TAs will create a new bucket with the same directory layout as the *your-bucket-name*. Before run your code, the TAs's grading script will remove all the data in the directory and upload your jar file to the job/PageRank.jar. After your program finished, the TAs' grading script will match against your outputs to the correct outputs and give a grade based on the correctness.

## 4.3 Project Submission

You project submission should include source code, a job jar and a project report. Your project submission layout should be as follows:

```
PageRank.tar
        report.txt
        PageRank/PageRank.java
        PageRank/*.java (other java files)

        PageRank.jar
```

You should tar all you java code files, jar file and the txt report in a single tar file using this command on Linux/SunOS:

```
tar cvf PageRank.tar report.txt PageRank/*.java PageRank.jar
```

### 4.3.1 Project Source code

You may prefer to develop and debug your code in your local machine. We recommend you to use Eclipse for developing and debugging locally.

Instructions to compile and run your MapReduce code via command line:

1. compile:

```
javac -classpath $HADOOP_HOME/share/hadoop/common/hadoop-common-*.jar:
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-*.jar:
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-*.jar   -d
PageRank PageRank.java
```

2. create jar:

```
jar -cvf PageRank.jar -C PageRank/ .
```

3. run in local:

```
hadoop jar PageRank.jar PageRank.PageRank input output
```

# 5 Automatic Testing Script

The TAs will use their accounts to test your code on the AWS EMR. You only need to submit your source code, jar file and report to the Sakai. The TAs will run the following script to grade your program on the AWS. **Please do your own implementation. We will be**

**checking similarities between the solutions.** Sample input and output files will be provided. The grading test cases will not be disclosed until the grades are released.

```
aws emr create-cluster --name "PageRank" --ami-version 3.9 --instance-type
m3.xlarge --instance-count 3
--use-default-roles --ec2-attributes KeyName=myKey
--steps Type=CUSTOM_JAR, Name=" PageRank", ActionOnFailure=CONTINUE, MainClass=
PageRank.PageRank,Jar=pathtojarfile,
Args=[Input_XML_File_Path, pathtooutputbucket]
```

What do the above command line options mean? click me

# 6 Grading Guidelines:

This project *must* be implemented using Java. The project which violates this policy will be graded below 60. It is your responsibility to make sure you follow the input and output format. Your project will be graded 0 if either your program cannot read the input files provided by the TAs or the TAs' autograder cannot parse your output files.

| **Components** | outlink graph | calculate N | PageRank Iter1 | PageRank Iter8 |
|---|---|---|---|---|
| **Grade**(%) | 40 | 30 | 30 | 30(bonus) |

# 7 Late Submission Policy:

Late submissions are not encouraged but we rather have you submit a working version than submit a version that is broken or worse not submit at all. Every late submission will be penalized 20% for each day late for up to a maximum of 3 days from the due date.

# 8 Resources

## 8.1 Tutorials

You can have a look at the following tutorials that might be helpful in doing this project:

1. http://en.Wikipedia.org/wiki/PageRank
2. https://aws.amazon.com/training/intro_series/
3. http://infolab.stanford.edu/~ullman/mmds/ch5.pdf
4. http://en.Wikipedia.org/wiki/MapReduce
5. http://aws.amazon.com/elasticmapreduce/
6. http://en.Wikipedia.org/wiki/Help:Wikilinks#Wikilinks

## 8.2 Questions and Answers

All the questions should be asked through Piazza. The TAs should answer your questions *ASAP*. Please let the TAs know *ASAP* if you find any ambiguity or any issue which may lead to non-unique outputs through Piazza. Bonus points are possible!