CS 6240: Assignment 3

Goal: Implement PageRank in MapReduce to explore the behavior of an iterative graph algorithm.

This homework is to be completed individually (i.e., no teams). You have to create all deliverables yourself from scratch. In particular, it is not allowed to copy someone else's code or text and modify it. (If you use publicly available code/text, you need to cite the source in your code and report!)

Please submit your solution through Blackboard by the due date shown online. For late submissions you will lose one percentage point per hour after the deadline. This HW is worth 100 points and accounts for 15% of your overall homework score. To encourage early work, you will receive a 10-point bonus if you submit your solution on or before the early submission deadline stated on Blackboard. (Notice that your total score cannot exceed 100 points, but the extra points would compensate for any deductions.) Always package all your solution files, including the report, into a single standard **ZIP** file. Make sure your report is a **PDF** file.

To enable the graders to run your solution, make sure your project includes a standard Makefile with the same top-level targets (e.g., *alone* and *cloud*) as the one Joe presented in class (see the Extra Material folder in the Syllabus and Course Resources section). You may simply copy Joe's Makefile and modify the variable settings in the beginning as necessary. For this Makefile to work on your machine, you need Maven and make sure that the Maven plugins and dependencies in the pom.xml file are correct. Notice that in order to use the Makefile to execute your job elegantly on the cloud as shown by Joe, you also need to set up the AWS CLI on your machine. (If you are familiar with Gradle, you may also use it instead. However, we do not provide examples for Gradle.)

As with all software projects, you must include a README file briefly describing all of the steps necessary to build and execute both the standalone and AWS Elastic MapReduce (EMR) versions of your program. This description should include the build commands, and fully describe the execution steps. This README will also be graded.

PageRank in MapReduce

For this assignment, we are going to apply PageRank to Wikipedia articles in order to find the most referenced ones. In practice this requires several non-trivial pre-processing steps. For example, the Wikipedia dumps might be compressed in a format not (well-) supported by Hadoop. In principle it is easy to find hyperlinks in HTML, doing so requires some experience with document parsers. As some links are not interesting, an understanding of the domain document schema is necessary but takes a significant amount of time. Dealing with such obstacles is part of the real data analysis experience, but since we want you to focus on parallel data processing algorithms, we decided to help you get started more quickly.

We transformed the original Wikipedia 2006 data dump into Hadoop-friendly bz2-compressed files and made them available at:

https://drive.google.com/drive/folders/1IIySfwwyvup2cy2bP4BfFaTYoFUSbWIK?usp=sharing

Use the simple 2006 English dataset for local development: wikipedia-simple-html.bz2

Use the four files comprising the full 2006 English dataset for final evaluation on EMR: wikipedia--html.?.bz2

.bqz2 File Format

The bz2 compression format works well for parallel computation, because it reduces size while remaining "splittable", meaning that it can be processed in chunks in parallel like uncompressed data. Other compression formats such as zip require centralized decompression. Each line of the bz2 file is formatted to contain the name of the Wikipedia page, a colon (:), and then the full contents of the page on a single line. (Note that there may be a header and/or footer—not shown below—bracketing the sequence of pages.)

```
<page-name0>:<wikipedia file html contents0>
<page-name1>:<wikipedia file html contents1>
...
```

Wikipedia HTML Format

You need to convert the Wikipedia data into an adjacency-list based graph representation, using the original page names as the node IDs. The distracting nuisance components (e.g., file path prefix and .html suffix) must be removed. The Wikipedia files are in XHTML so they may be parsed by either an HTML or XML parser. An example parser will be made available together with this assignment to help you get started. Feel free to use and modify it. It is important to only keep hyperlinks from within the bodyContent div tag of the document and ignore all others.

For example:

```
<!DOCTYPE ...>
<html>
... ignore all <a href="url"> links in this part of page ...
    <div id="bodyContent">
    ... KEEP all <a href="URL"> links in this section, including any nested in sub <div> tags ...
    </div>
... ignore all <a href="url"> links in this part of page ...
</html>
```

From each anchor tag href attribute within the body content div, you must parse the referenced page name. Strip off any path information from the beginning as well as the .html suffix from the end, leaving only the page name. Also, discard any page names containing a tilde (~) character as these are well-connected but uninteresting to the results.

A SAX XML parser, also using regular expressions, is provided as a usable example. You may incorporate it into your preprocessing stage, however it is not warranted to be bug free. Basically, it is an

implementation that filters and keeps relevant links from a Wikipedia page, per the above specification. The parser should work fine, but is not guaranteed to be 100% correct. Check its output on some carefully selected examples to see if it performs satisfactorily. Report any parser errors and possible fixes on the discussion board for participation credit. (Only the first one to report a bug or fix will receive credit for it.)

Tip: It is very important to see the data; we strongly recommend you write a Java program that reads lines from a provided bz2 file and then pretty-prints a few of these html pages to allow you to see their structure. Then incorporate your parser into this standalone program to confirm that it keeps only the required URLs and strips them to only their name. Taking these incremental steps will save time in the long term and allow you to verify the accuracy of the parser.

In order to implement PageRank, you will need to issue an iterative series of jobs. The first job will receive lines from a bz2 file chunk in the format shown above and perform the pre-processing. The next set of jobs will successively refine the ranks of the pages 10 times. The final job will output the top-100 highest ranked pages, sorted from highest to lowest, along with their calculated ranks.

The specifications are intended to describe the high-level problem. It is impossible to document every step you will take and issue that you will encounter processing this big data. You must look at and analyze the data yourself and use your judgement to guide your design and implementation choices. (See tip above.) The following are suggested steps.

Pre-Processing Summary (complete in week 1)

1. Build a standalone program to parse input files and display them in human-readable form (optional but strongly recommended).
2. Incorporate parser to find relevant links and strip URLs to only the page name.
3. Discard pages with names containing ~ as well as links containing these characters.
4. Incorporate the remaining pages and links to create a graph in adjacency list representation.

Note: You might, and probably will, encounter surprises in the data. For example, a page might point to another page that is not in the collection. Similarly, some page might occur multiple times, possibly with different content. To not get derailed by such issues, think carefully about any assumptions you make about the data and then include code that handles exceptions or flags violations of those assumptions. If you believe that some data cleaning cannot be done in the same job that is doing the pre-processing, you may add another job. **Document the data issues you find and your solutions in your report.**

Overall Workflow Summary

Design all steps of the workflow as MapReduce jobs that are executed in sequence from a single driver program:

1. Pre-processing Job: As stated above, turn input Wikipedia data into a graph represented as adjacency lists. Make sure that only the page names are used as node and link IDs; strip off any path information from the beginning as well as the .html suffix from the end, leaving only the page name, and discard any pages and links containing a tilde (~) character.

2. PageRank Job: run 10 iterations of PageRank. Set the random surfer so that s/he follows a link with probability 0.85, and jumps to a random page with probability 0.15. Try to find a way to estimate how much the PageRank values are converging. (start in the middle of week 1, complete in week 2)
3. Top-k Job: From the output of the last PageRank iteration, find the 100 pages with the highest PageRank and output them, along with their ranks, from highest to lowest. (complete in week 2)

For PageRank, make sure your program sets the **initial** PageRank values to 1/numberOfPages and handles dangling nodes. If a page appears in a link, but the page itself does not exist in the data, you can treat it as a dangling node. This means that you do not need to remove the links pointing to it and that it needs to be accounted for in numberOfPages.

Report

Write a brief report about your findings, using the following structure.

Header

This should provide information like class number, HW number, and your name. **Also include a link to your CCIS Github repository for this homework.**

Design Discussion (15 points total)

Show the pseudo-code for pre-processing, PageRank and Top-k computation. For the parser, you do not need to show details and can instead simply state it like a blackbox function, e.g., as parse(…). Add comment lines to explain any modifications you may have made to the provided parser. If your pseudo-code is identical to material shown in a course module, you do not need to copy it. Instead, just refer to the exact location in the module where you found it. (Note: make sure the pseudo-code matches your actual program.) (15 points)

Performance Comparison (20 points total)

Run your program in Elastic MapReduce (EMR) on the four provided bz2 files, which comprise the full English Wikipedia data set from 2006, using the following two configurations:

- 6 m4.large machines (1 master and 5 workers)
- 11 m4.large machines (1 master and 10 workers)

Report for both configurations (i) pre-processing time, (ii) time to run ten iterations of PageRank, and (iii) time to find the top-100 pages. There should be 2*3=6 time values. (6 points)

Report for both configurations the amount of data transferred from Mappers to Reducers, and from Reducers to HDFS, separately for each iteration of the PageRank computation. Does it change with the cluster size? Does it change over time? There should be 2*2*10=40 numbers. (5 points)

Critically evaluate the runtime results by comparing them against what you had expected to see and discuss your findings. Make sure you address the following question: Which of the computation phases

showed a good speedup? If a phase seems to show fairly poor speedup, briefly discuss possible reasons—make sure you provide concrete evidence, e.g., numbers from the log file or analytical arguments based on the algorithm's properties. (4 points)

Report the top-100 Wikipedia pages with the highest PageRanks, along with their rank values and sorted from highest to lowest, for both the simple and full datasets. Do they seem reasonable based on your intuition about important information on Wikipedia? (5 points)

Deliverables

Submit the following in a **single zip file**:

1. The report as discussed above. (1 PDF file)
2. The syslog files for a successful EMR run for both system configurations. (5 points)
3. *Final* output files from EMR execution, i.e., only the top-100 pages and their PageRank values on the full data set. (5 points)

Make sure the following is easy to find in your **CCIS Github** repository:

4. The source code of your programs, including an easily-configurable Makefile that builds your programs for local execution. **Make sure your code is clean and well-documented. Messy and hard-to-read code will result in point loss.** In addition to correctness, efficiency is also a criterion for the code grade. (55 points)