CS 6240: Assignment 4

Goal: Implement PageRank in Spark to compare the behavior of an iterative graph algorithm in Spark vs. Hadoop MapReduce.

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.

Please name your submitted ZIP file "**CS6240_first_last_HW_#**", where "first" is your first name, "last" your last name, and "#" the number of the HW assignment. For each program submission, include complete source code, build scripts, and small output files. Do not include input data, output data over 1 MB, or any sort of binaries such as JAR or class files.

To enable the graders to run your solution, make sure you include 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 start with unzipping the original submission, include the build commands, and fully describe the execution steps. This README will also be graded.

PageRank in Spark

We solve the same problem from Assignment 3, but this time in Spark. Your Spark program needs to be written in **Scala**, but can call the input parser (written in Java) from Assignment 3 as a named function passed to a Scala command. Overall, the Spark program should perform the following steps:

1. Read the bz2-compressed input.

2. Call the input parser from Assignment 3 on each line of this input to create the graph.
3. Run 10 iterations of PageRank on the graph. The PageRank algorithm has to be written in Scala. As before, it has to deal with dangling nodes.
4. Output the top-100 pages with the highest PageRank and their PageRank values, in decreasing order of PageRank.

When designing your Spark program, think carefully about using RDDs, pair RDDs or DataSets. If in doubt, try different versions and see what works best. Also think carefully about persisting data in memory and how to best separate data that does not change from data that does. However, you do *not* need to explore balanced min cut or similar algorithms that attempt to find a graph partitioning that minimizes the number of edges between the partitions in order to minimize network traffic during PageRank computation iterations.

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.

Program Discussion (20 points total)

Describe briefly how each step of your program is transforming the data. Be precise, e.g., by showing the structure of the input and output as a table. (10 points)

For each step, state if the dependency is narrow (no shuffling) or wide (shuffling). How many stages does your Spark have? (10 points)

Performance Comparison (12 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 the Spark execution time. For comparison, also include the total execution time (from pre-processing to top-k) of the corresponding Hadoop executions from Assignment 3. (8 points)

Discuss which system is faster and briefly explain what could be the main reason for this performance difference. (4 points)

Deliverables

1. The report as discussed above. (1 PDF file)

2. The source code of your Spark program, 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. (60 points)
3. The syslog (or similar) files for a successful EMR run for both system configurations. (4 points)
4. *Final* output files from EMR execution only, i.e., only the top-100 pages and their PageRank values. (4 points)