CS 6240: Assignment 1

Goals: Set up your environment for developing and running Hadoop MapReduce and Spark Scala programs. Test your setup by writing and executing a Word-Count inspired program.

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

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. Demo examples for MapReduce and Spark are included in the assignment material. 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 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 the steps necessary to build and execute both the standalone and the 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, and you will be able to reuse it on all this semester's assignments with little modification (assuming you keep your project layout the same).

You have about 2 weeks to work on this assignment. Section headers include recommended timings to help you schedule your work. Of course, the earlier you work on this, the better.

Set Up CCIS Github (Week 1)

Find the **CCIS Github** (not the public Github!) server and create two projects for this assignment: one for MapReduce, the other for Spark. We recommend using an IDE like Eclipse with the corresponding Github plugin to pull and push your code updates. Make sure you do the following:

- Set all your projects for this course so that they are private in general, but accessible to the TAs and instructor. We will post our CCIS logins on Blackboard.
- Make sure you commit and push changes regularly. As a rule of thumb, the "delta" between consecutive snapshots of your source code should be equivalent to about 2 hours' worth of

intensive coding. Committing large, complete chunks of code that look like you just copied from someone else will result in point loss.

Sign Up for an AWS Account (Week 1)

Before doing anything else, take a look at a document Joe created to help you get started with an efficient industry-style setup:
https://docs.google.com/document/d/1-UjNVFasTSzhAaqLtKSmeie6JZMinhtVEqCEZwUkxeE/edit?usp=sharing

It might be a little outdated, but it gives you a good idea about important recommended steps and best practice. The instructions below outline this same approach, but without the details.

Go to Amazon Web Services (AWS) and create an account. You should be able to find out how to do this on your own. Enter the requested information and you are ready to go. Make sure you choose the right geographic region and apply for the education credit. Then set up access to AWS from the commandline (CLI). There are instructions on AWS for how to do this. Note: it may be necessary to run an EMR job manually from the Web GUI one time in order for AWS to create the default security roles and to determine your region's subnet-id.

Note: If you do not want to use the Amazon cloud, you can alternatively work with any equivalent cloud environment, e.g., those hosted by other companies, Northeastern's Discovery cluster, or the Mass Open Cloud. However, we are not able to provide custom instructions or support for those environments.

Set Up the Local Development Environment (Week 1)

We recommend using Linux for MapReduce and Spark development. MacOS should also work fine, but we had problems with Hadoop on Windows. If your computer is a Windows machine, you can run Linux in a virtual machine. We tested Oracle VirtualBox and VMware Workstation Player: install the virtual machine player (free) and create a virtual machine running Linux, e.g., Ubuntu (free). (If you are using a virtual machine, then you need to apply the following steps to the virtual machine.)

On Amazon AWS, you can run your jobs using EMR or plain EC2. With EMR, it is easier to set up a virtual cluster for the assignments in this course. EC2 offers lower hourly rates per machine. We generally recommend EMR for this course and will provide setup help and instructions only for EMR. However, you are welcome to work with plain EC2—but you need to deal with it on your own. (There are scripts on the Web that make it relatively easy to fire up a cluster of MapReduce or Spark nodes on EC2.)

Check the versions of Hadoop MapReduce and Spark available on EMR. To avoid incompatibility issues, make sure your local development environment matches one of them, at least in terms of the main version number(s) (e.g., all 2.7.* should be compatible with each other). We recommend you install the software mentioned in Joe's README file, but that is not required. E.g., some people may prefer Cloudera's distributions or others. After installing, set up your project like in Joe's demo examples. Then

you should be ready to run it from an IDE. To run from the commandline, edit the Makefile as described in the README.

To summarize, you can install the MapReduce and Spark environment of your choice, as long as it allows you to create projects that look like Joe's demo examples. You need to be able to pull and push to CCIS Github, and instructor and TAs should be able to run your project following the instructions in your README—only having to customize the variable settings at the top of the Makefile.

Hint for those new to Maven and Gradle: In your IDE, create a Maven project. This makes it simple to build "fat jars," which recursively include dependent jars used in your program. There are many online tutorials for installing Maven and for creating Maven projects via archetypes. These projects can be imported into your IDE or built from a shell. Sometimes you may need to modify the pom.xml file slightly. Use the discussion board to get help when you are stuck with Maven; or talk to our friendly TAs.

Write a Grouping and Aggregation Program (Week 2)

Look carefully at the Word Count examples that come with Hadoop MapReduce and Spark. You can also find them in Joe's demo zip files. The Word Count program extracts all "words" from a line of text, then groups by the word and aggregates all counts associated with the same word. Your task is to write a very similar program, but for the Twitter follower dataset at
http://socialcomputing.asu.edu/datasets/Twitter

Before designing your algorithm and program, become familiar with the data by reading the information on their Web page. Make sure you know the direction of the follower edges. Then write a program that determines the number of followers for each user. Output this result as plain text, each line containing userID and follower count:

userID1, number_of_followers_this_user_has
userID2, number_of_followers_this_user_has

The output does not need to be sorted. We leave it as a bonus challenge (no extra points) to output the result in order of the number of followers, so that the user with most followers appears at the top.

Hint: The structure of the program should be similar to Word Count. Instead of words, group by user IDs. When deciding about your program and its execution, take (estimated) input and output size into account. How big will the output for the Twitter data be?

Start with the MapReduce program and get it to run in the IDE on your development machine. Notice that you will need to provide an input directory containing text files and a path to an output directory (that directory should not exist). Once the program runs fine, look at it closely and see what Map and Reduce are doing. Use the debugging perspective, set breakpoints in the Map and Reduce functions, then experiment by stepping through the code to see how it is processing the input file. Make sure you work with a small data sample. Then work on the Spark Scala program and find out how to ask Spark to tell you about the underlying execution of the program.

WARNING: Leaving large data on S3 and using EMR will cost money. Read carefully what Amazon charges and estimate how much you would pay per hour of computation time before running a job. Use only the smallest available general-purpose machine instances. And remember to test your code as much as possible on your development machine. When testing on AWS, first work with small data samples.

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.**

Map-Reduce Implementation (20 points)

Show the pseudo-code for the Twitter-follower count program implementation in MapReduce. Look at the online modules and your lecture notes for examples. Remember, pseudo-code captures the essence of the algorithm and avoids wordy syntax. Do not just copy-and-paste source code! Briefly discuss the main idea of your solution.

Spark Scala Implementation (30 points total)

Show the pseudo-code for the Twitter-follower count program implementation in Spark Scala. Since many Scala functions are similar to pseudo-code, you may copy-and-paste them here whenever appropriate. (20 points)

Report the Scala function(s) you used to ask Spark to report the execution details of your program. Copy-and-paste the reported execution information into the report. (It should tell you about the implementation of the aggregation, if aggregation happened before shuffling, if there is shuffling at all, etc.) (10 points)

Speedup Measurements (20 points total)

Run both the MapReduce and the Spark version of your program on AWS on the full Twitter edges dataset, using the following two configurations:

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

Look for the log files that tell you about the job execution timing, and how many bytes of data were moved around. Make sure you keep them stored on S3 for at least 3 weeks. Use them to answer the following questions.

Report the running time of each program on each cluster. Repeat the time measurements one more time, each time starting the program from scratch. Report all 2 programs * 2 clusters * 2 independent runs = 8 running times you measured. (8 points)

Report the speedup of the MapReduce program and the speedup of the Spark program. (2 points)

Report for both clusters the amount of data transferred to the Mappers, from Mappers to Reducers, and from Reducers to output. There should be 2*3 = 6 numbers. (3 points)

Is the speedup near-optimal? Argue, briefly why or why not your program is expected to have good speedup. Make sure you discuss (i) *how many tasks* were executed in each stage and (ii) if there is a part of your program that is *inherently sequential* (see discussion of Amdahl's Law in the module.) (7 points)

Deliverables

Submit the following in a **single standard ZIP file** (**not** rar, gzip, tar etc!):

1. The report as discussed above. (1 PDF file)
2. Log files describing the overall job execution for one successful run of the MapReduce program on the full input on AWS *on each cluster size*. (syslog or similar, 2 points)
3. Log files describing the overall job execution for one successful run of the Spark program on the full input on AWS *on each cluster size*. (similar to MapReduce syslog, but may be in another log file, 2 points)
4. All output files produced by those same successful runs on the full input on AWS (4 sets of part-r-… or similar files). If any of those files exceeds 10 MB, do not include it. Instead, leave it in a special directory on Github, and include a pointer to that directory in the report. (4 points)

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

5. The MapReduce project, including source code and build scripts. (11 points)
6. The Spark Scala project, including source code and build scripts. (11 points)

IMPORTANT: Please ensure that your code is properly documented. In particular, there should be comments concisely explaining the role/purpose of a class. Similarly, if you use carefully selected keys or custom Partitioners, make sure you explain their purpose (what data will be co-located in a Reduce call; does input to a Reduce function have a certain order that is exploited by the function, etc.). But do not over-comment! For example, a line like "SUM += val" does not need a comment. As a rule of thumb, you want to add a brief comment for a block of code performing some non-trivial step of the computation. You also need to add a brief comment about the role of any major data structure you introduce.