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 must 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., *local* and *aws*) as the one presented in class. Demo examples for MapReduce and Spark are included in the assignment material. You may simply copy the provided Makefile and modify the variable settings in the beginning. 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. (If you are familiar with other major dependency-management software, you may use that instead. However, we only provide examples for Maven.) Notice that to use the Makefile for executing your job in the cloud, you also need to set up the AWS CLI on your machine.

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. The earlier you work on this, the better.

Set Up CCIS Github (Week 1)

Find the **CCIS Github** (if you do not have a CCIS account, you may alternatively use the public Github github.com) service and create two projects for this assignment: one for MapReduce, the other for Spark. We recommend using an IDE such as 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 the relevant TA/instructor information on the discussion board.

- 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.

AWS Account and Development Environment Setup (Week 1)

Before doing anything else, look at this document:

https://docs.google.com/document/d/1KnaJbyDnJpOz0uxLI3haUwmoo7YtTxPXbFuTUC46tT0/edit?usp=sharing

It might be a little outdated, but it gives you a good idea about important recommended steps and best practice.

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.

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 setup 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 slightly 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 should make it relatively easy for advanced users 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 MapReduce 2.8.* should be compatible with each other). Instead of the software mentioned in the demo README files, you may use pre-packaged environments or virtual machines, e.g., as offered by Cloudera. Whatever you choose, the goal is for you to be able to develop, debug, and test your MapReduce (Java) and Spark (Scala) programs locally, then run them on EMR. Remember that to run from the commandline, you must edit the variables in the Makefile as described in the demo README.

In the end, your projects on Github should look similar in structure to the examples MR-demo and Spark-demo. Instructor and TAs must 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 dependency management software in general: In your IDE, create a Maven project. This makes it easy 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 provided 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 program in the two demos. It 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 understand the direction of the follower edges. Then write a program that takes *edges.csv* as input and counts the number of incoming edges for each user ID. This corresponds to the **number of followers for each user who has at least one follower**. Output this result as plain text, each line containing userID and follower count, e.g.:

userID1, number_of_followers_this_user_has
userID2, number_of_followers_this_user_has

The output may be partitioned into multiple files. It does not need to be sorted, and each line may contain other formatting characters, e.g., "(userID1, number_of_followers_this_user_has)," which may be added by default by MapReduce or Spark output writers.

Bonus challenge 1 (2 extra points): Output the result in order of the number of followers, so that the user with most followers appears at the top.

Bonus challenge 2 (2 extra points): Also output users who have no followers. Think about how to find them, then try to implement your idea.

The bonus challenges are intended for students who already completed all required tasks of the HW. Including bonus points, the final HW score cannot exceed 100 points.

Hint: The structure of the program should be similar to Word Count. You may start with the provided demos and simply modify them. Instead of words, group by the appropriate user ID of an edge. 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 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 what the RDD *toDebugString* method does. Also look at the *explain* method of DataSet in the Spark Scala API.

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/cheapest 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. Make sure TAs and instructor have access to it.**

Map-Reduce Implementation (30 points total)

Show the pseudo-code for your Twitter-follower count program implementation in MapReduce. Look at the online modules and your lecture notes for examples, in particular the Word Count program. Pseudo-code captures the essence of the algorithm and avoids wordy syntax. Do *not* just copy-and-paste source code! (20 points)

Briefly discuss the main idea of your solution. Example: "My program reads the input line by line. The map function parses a line to extract … For each … it outputs … These … are grouped by … and then the reduce function computes … for each group." (10 points)

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)

You may implement your program using RDDs (RDD, pair RDD), or the more advanced DataSet. If you use (pair) RDDs, let's assume you called the RDD with the final result *countsRDD*. This is the one that you output in the end, using countsRDD.saveAsTextFile(args(1)). Right before the save-statement, add a call to countsRDD.toDebugString and make sure this information is written to your log file, e.g., using *logger.info(countsRDD.toDebugString)*. Copy-and-paste the lines produced by this toDebugString call into your report. If you use DataSet, then do the same for the explain function of DataSet instead. (10 points)

Running Time Measurements (12 points total)

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

- 6 *cheap* machine instances (1 master and 5 workers)

Look for the cheapest machine instance you can select on EMR. In the past these were usually general-purpose machines of type m[?].[*], e.g., m1.small or m5.xlarge. (For some instance types, possibly those from a previous generation, you may need to submit a subnet-id when creating the cluster.)

Look for the log files that tell you about the job execution timing, and how many bytes of data were moved around. This should be syslog for MapReduce and stderr for Spark. Use them to answer the following questions.

Measure the running time of each program. Repeat the time measurements one more time, starting each program from scratch as a new job. Report all 2 programs * 2 independent runs = 4 running times you measured. (4 points)

Report the amount of data transferred to the Mappers, from Mappers to Reducers, and from Reducers to output. There should be 3 numbers. (3 points)

Argue briefly, why or why not your MapReduce 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.) (3 points)

Copy to your Github the syslog (MapReduce) and stderr (Spark) log files of the runs you are reporting the measurements for. Check that the log is not truncated—there might be multiple pieces for large log files! Include a link to each log file/directory (4 links total) in the report. Similarly, copy the output produced (all parts of it) to your Github and include the links to the output directories (4 links total) in the report. (2 points)

Deliverables

IMPORTANT: The submission time of your solution is the latest timestamp of any of the deliverables included. For the PDF it is the time reported by Blackboard; for the files on Github it is the time the files were pushed to Github, according to Github. We recommend the following approach:

1. Push all files to github and make sure everything is there (see deliverables below).
2. Submit the report on Blackboard. Make sure you hit "submit," not just "save." Open the submitted file to verify everything is okay.
3. Do not make any more changes in the project on Github.

Submit the report as a **PDF** file (**not** txt, doc, docx, tex etc!) on Blackboard. To simplify the grading process, please name this file yourFirstName_yourLastName_HW#.zip. Here yourFirstName and yourLastName are your first and last name, as shown in Blackboard; the # character should be replaced by the HW

number. So, if you are Amy Smith submitting the solution for HW 7, your solution file should be named Amy_Smith_HW7.pdf:

1. The report as discussed above. Make sure it includes the links to the project, log and output files as described above. (1 PDF file)

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

2. Log file for the overall job execution for each successful run of the MapReduce program on the full input on AWS for which you report numbers. (syslog or similar, 2 points)
3. Log file for the overall job execution for each successful run of the Spark program on the full input on AWS for which you report numbers. (similar to MapReduce syslog, but will probably be in stderr, 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). (4 points)
5. The MapReduce project, including source code, build scripts etc. (10 points)
6. The Spark Scala project, including source code, build scripts etc. (10 points)

Note: If you cannot get your program to run on AWS, then you can instead include the log files and output from execution on your local machine for partial credit.

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.