CS 6240: Assignment 1

**Goals**: Set up your environment for developing and running the Hadoop MapReduce program. 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! No more than 3 or 4 isolated lines can be copied from publicly available code.)

Please submit your solution through a Khoury Linux submit script by the due date shown online. (We will explain in class and online how to use the submit scripts.) This HW is worth 100 points and accounts for 15% of your overall homework score.

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, but not needed) 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** (‘make’ for Java) 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)

We will not use Github in this version of the course.

# AWS Account and Development Environment Setup (Week 1)

Before doing anything else, look at this document:

[https://docs.google.com/document/d/1HhPyexZ\_ESdTE8\_g3igUCZiDgyZnZUZHEIMHaSSfFR4/edit?usp=sharing](https://docs.google.com/document/d/1HhPyexZ_ESdTE8_g3igUCZiDgyZnZUZHEIMHaSSfFR4/edit%22%20%5Cl%20%22heading%3Dh.kk1yabz6vk95)

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 later for 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.

You will also need a Khoury account, anyway, to use the Khoury Linux submit scripts. You may request a Khoury account here: <https://my.ccs.neu.edu/account/apply>

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 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) 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 should look similar in structure to the examples MR-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.

# 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.

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.

**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.

## Map-Reduce Implementation (70 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! (45 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.” (25 points)

## Running Time 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 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. (8 points)

Report the amount of data transferred to the Mappers, from Mappers to Reducers, and from Reducers to output. There should be 3 numbers. (6 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.) (6 points)

Copy to yourown submit directory the syslog (MapReduce) 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 submit directory and include the links to the output directories (4 links total) in the report. (When you submit your assignment using the submit script, you will first use Linux tar to create a HW1.tar.gz file. This will then be submitted using the Khoury Linux submit scripts provided.)

# Deliverables

**IMPORTANT**: The submission time of your solution is the latest timestamp of the tar file that you submit.

1. **NOTE:** The submission mechanism this semester will be based on a shell script on the Khoury Linux system. Further details about submission will be announced later, including what files to submit.

Submit the report as a **PDF** file (**not** txt, doc, docx, tex etc!) inside your submit directory that you tar. To simplify the grading process, please name this file yourFirstName\_yourLastName\_HW#.zip. Here yourFirstName and yourLastName are your first and last name; 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 your tar file includes the project, log and output files as described above. (1 PDF file)

**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.