PROGRAMMING ASSIGNMENT 2: MAPREDUCE ON CLOUDLAB

COSC560

April 7, 2019

Clara Nguyen
Rachel Offutt
University of Tennessee EECS

Contents

0.1 Project Summary

In this assignment, you will program the Map Reduce parallel data processing system on the Cloudlab cloud computing platform. This will allow you gain practical experience on Map Reduce programming and learn the performance implications of parallel data processing. The following basic goals must be fulfilled:

- Cloudlab setup
- Create Cloudlab cluster on Hadoop
- Build the MapReduce code for the Reverse-indexer
- Run the MapReduce program on the Hadoop cluster
- Query the inverted index

0.2 Project Specific Requirements

0.2.1 Identifying and removing stop words

One issue is that some words are so common that their presence in an inverted index is "noise," that is they can obfuscate the more interesting properties of a document. Such words are called “stop words.” For this part of the assignment, write a word count Map Reduce function to perform a word count over a corpus of text files and to identify stop words. It is up to you to choose a reasonable threshold (word count frequency) for stop words, but make sure you provide adequate justification and explanation of your choice. A parser will group words by attributes which are not relevant to their meaning (e.g.,"hello","Hello", and "HELLO" are all the same word), so it is up to you to define "scrub" however you wish; some suggestions include case-insensitivity, etc. It is not required that you treat “run” and “ran” as the same word, but your parser should handle case insensitivity. Once you have written your code, then run your code and collect the word counts for submission with all your Mapper and Reducer files.

0.2.2 Building the Inverted Index

For this portion of the assignment, you will design a MapReduce-based algorithm to calculate the inverted index. To this end, you are to create a full inverted index, which

maps words to their document ID + line number in the document. Note that your final inverted index should not contain the words identified in Step 1. The format of your MapReduce output (i.e.,the inverted index) must be simple enough to be machine-parseable; it is not impossible to imagine your index being one of many data structures used in a search engine's indexing pipeline. Your submitted indexer should be able to run successfully on one or multiple input txt files, where "successfully" means it should run to completion without errors or exceptions, and generate the correct word->DocID mapping. You are required to submit all relevant Mapper and Reducer Java files, in addition to any supporting code or utilities.

0.2.3 Query the Inverted Index

Write a query program on top of your full inverted file index that accepts a user-specified query (one or more words) and returns not only the document IDs but also the locations in the form of line numbers. The query program can be local: it does not need to handle the task using Map-Reduce framework again. It is not required that your query program to return text snippets from the original text files.

0.3 Project Design Choices

0.3.1 Language Choice

We chose to implement this project in Python because data parsing, especially word parsing, is exponentially easier in Python and one of our team members has extensive experience in working with big data and text parsing in Python.

0.3.2 Configuration

We created a script file to run the mapReduce functions to allow for ease of use for the user. The user provides a list of input arguments in the form of file names they would like to run on.

0.3.3 Setup of Output File

The output file is set up in terms of word, then file it appears in, then the lines it appears on. We structured our output file this way to make querying easier.

0.3.4 Stop Words

To parse our data, we see if there are over 1000 instances of a word and if the word is five letters or less. The reasoning behind our design decision is because when using the complete works of Shakespeare, we opened up the total count of words, and found that generally words that appeared more than 1000 times were likely to be stop words, and the 5 character cutoff is to ensure that the majority of the names are not cut off as well.

0.4 Running the Project

NOTE: A recording of this procedure can be viewed on Asciinema at the following link: `https://asciinema.org/a/x8wRWlMvEh5Y8aBImklUvLzYZ`

1. Step-by-Step Procedure:

   (a) Set up a Cloudlab instance with the default Hadoop configuration (by gary).

   (b) Have SSH RSA keys setup so you can copy to the Cloudlab instance.

   (c) On your local machine, open up terminal and go to the directory that contains copy.sh from this project.

   (d) Run ./copy.sh. It takes 2 arguments:

   - username@address. This is your username and Hadoop cluster address.
   - Path to private RSA key. This is used to authenticate you to use scp.

   (e) SSH into the Hadoop Cluster. The copy script from Step (d) created a directory in your home directory called mapreduce, which has everything.

   (f) Go into ~/mapreduce. To simply run Hadoop on files, run the following:

   ```
   ./execute.sh data/100-0.txt data/test.txt
   ```

   It automates the entire procedure of copying files over to HDFS and doing the Mapreduce for you.

   (g) The py/query.py script will run automatically after completion. Type in a few words!

   (h) After the script's completion, you will see a results .txt file created in the local directory. To run the query script on this file, run the following:

   ```
   python3 py/query.py results.txt
   ```

2. Screenshots of commands running:

Figure 1: An example of the run script.

0.5 Implementation Screenshots

NOTE: We have already demoed to the TA for this course and have proven that our implementation of the MapReduce on Hadoop is correct and in working order.

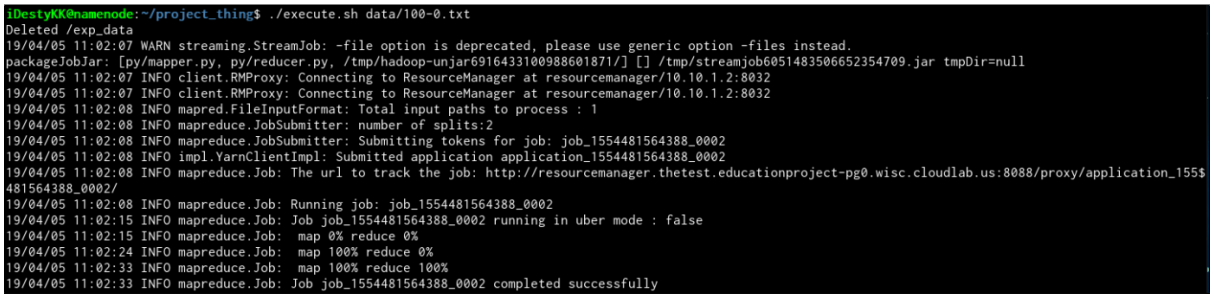

Figure 2: Proof of program running with one text input file

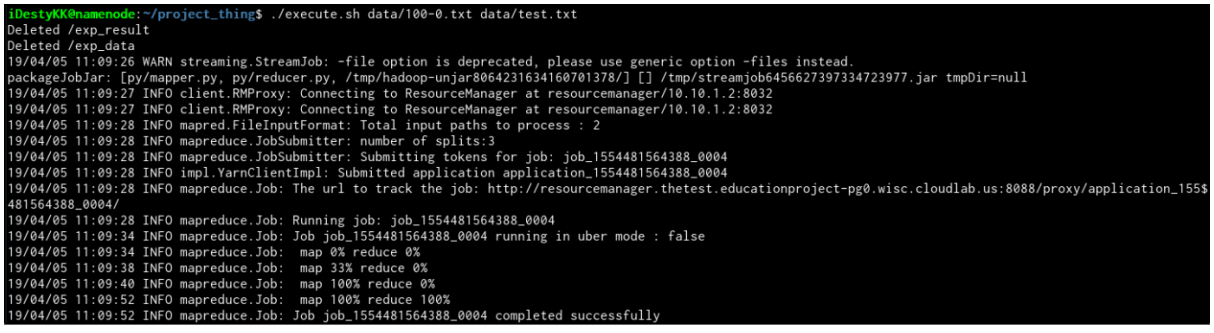

Figure 3: Proof of program running with multiple text input files

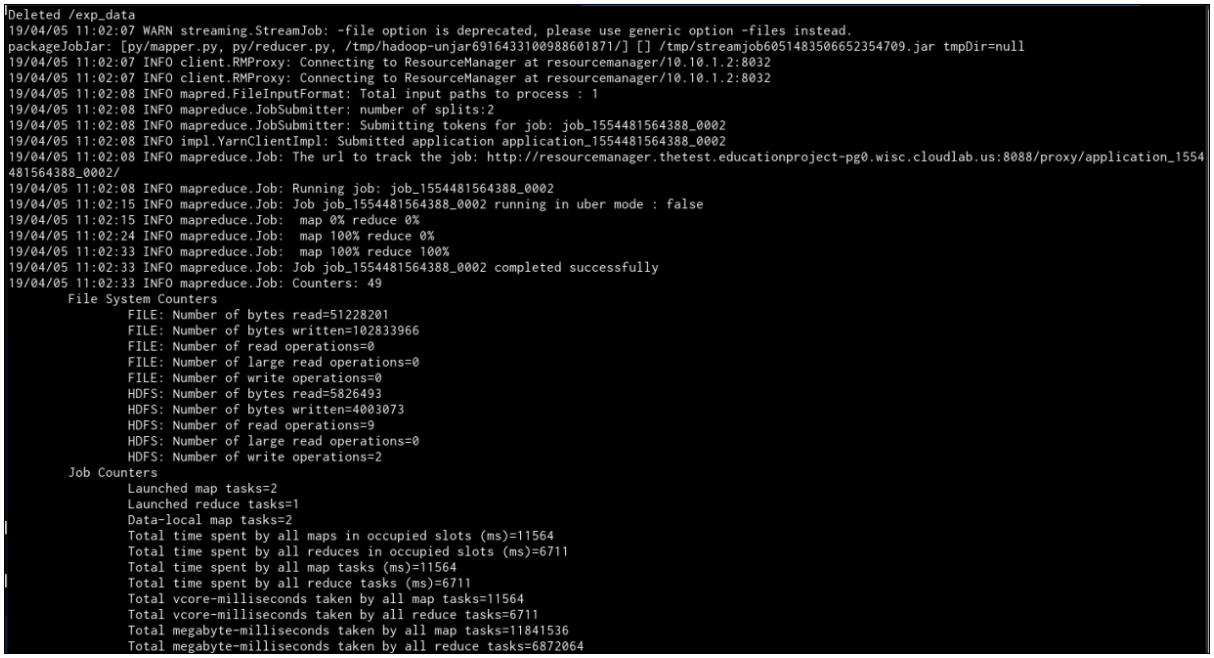

Figure 4: Proof of program completion

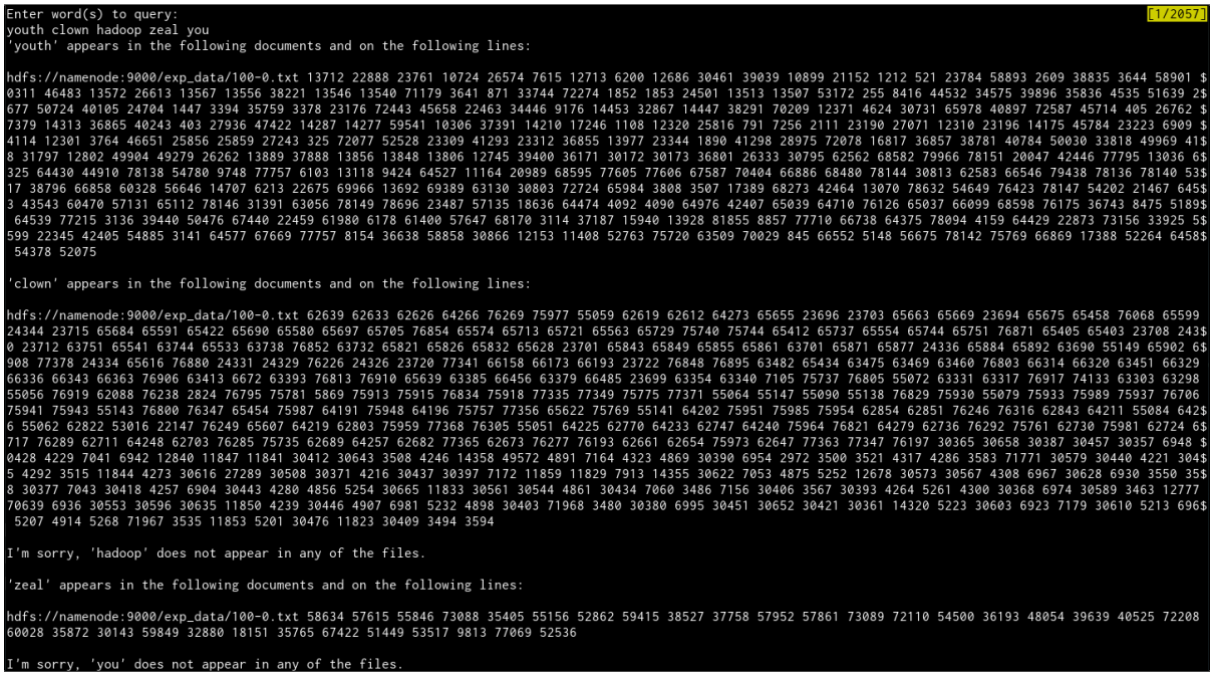

Figure 5: Query function showing line display for words in line, not in file, and the removal of stop words.