Nick Petty

CAP 6315 Social Networks and Big Data Analytics

Homework 4

**Question 1 (0.5 pts): Please use your own language to describe the following concepts:**

MapReduce (including Map and Reduce): a programming model, framework, or implementation for processing and generating large data sets using a distributed computing system. It is composed of two main functions, Map, which filters and sorts key-value pairs into a usable form, and Reduce, which aggregates the key-value pairs to create intermediate output. Additionally, the data set must be read by a scanner process, moved from Mappers to Reducers by intermediate steps called shuffling and sorting, and retrieved from the Reducers by a reader process that produces usable output.

Combiner: a less powerful Reducer-like component of MapReduce that is used with a Mapper to optimize intermediate values before moving them on to Reducers. They lower network traffic and lighten the workload of the Reducers.

Hadoop Distributed File System: a program designed to store and process large sets of data using clusters of computers. The program started from research at Google, but is now open-source and managed by Apache. Also called HDFS, it is programmed in Java and built to be fault-tolerant, easy to use, and inexpensive.

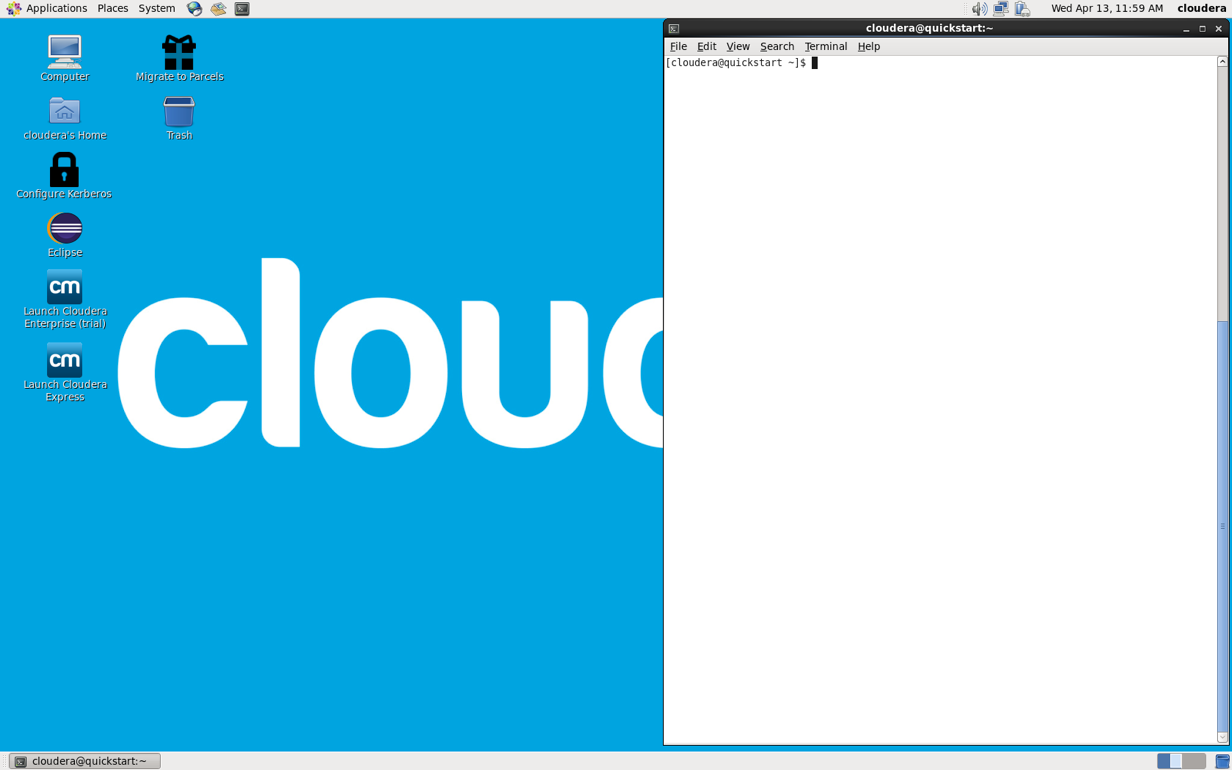
Hadoop Name Node: the control center and single point of failure of an HDFS implementation. It tracks the locations and attributes of all files in a cluster and provides the interface to add/copy/move/delete files on the file system. If the NameNode fails, the entire system is offline.

Hadoop Data Node: a server in an HDFS cluster that stores files/data. The DataNode receives its directions for file management from the NameNode, other DataNodes, or applications after they have been directed by the NameNode. DataNodes work together to coordinate data replication and create redundancy in the file system, and periodically update the NameNode on their contents.

**Question 2 (1.5 pts) Hadoop Installation:** Please follow the “MapReduce Programming Platform Installation Instruction” posted in the Blackboard (in the “Lectures” folder) to install Hadoop on your computer. Please report following major steps (capturing screenshots)

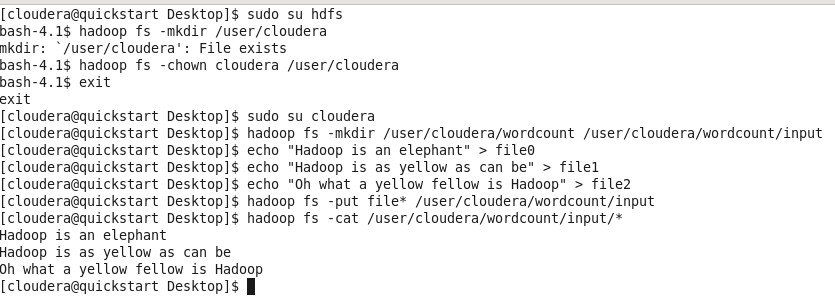
* **Part I:** Cloudera MapReduce Installation (0.5 pt)
* **Part II:** First MapReduce Job Task (0.5 pt)
* Pleases report the WordCounttask outputs **(**0.5 pt**)**

Part I results in the screenshot below. Virtual machine created with VirtualBox, terminal opened.

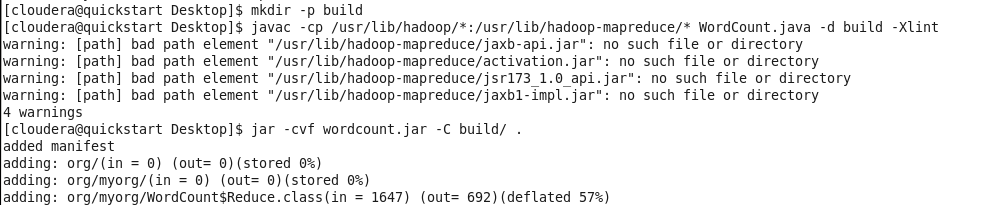


Part II results in the screenshots below.

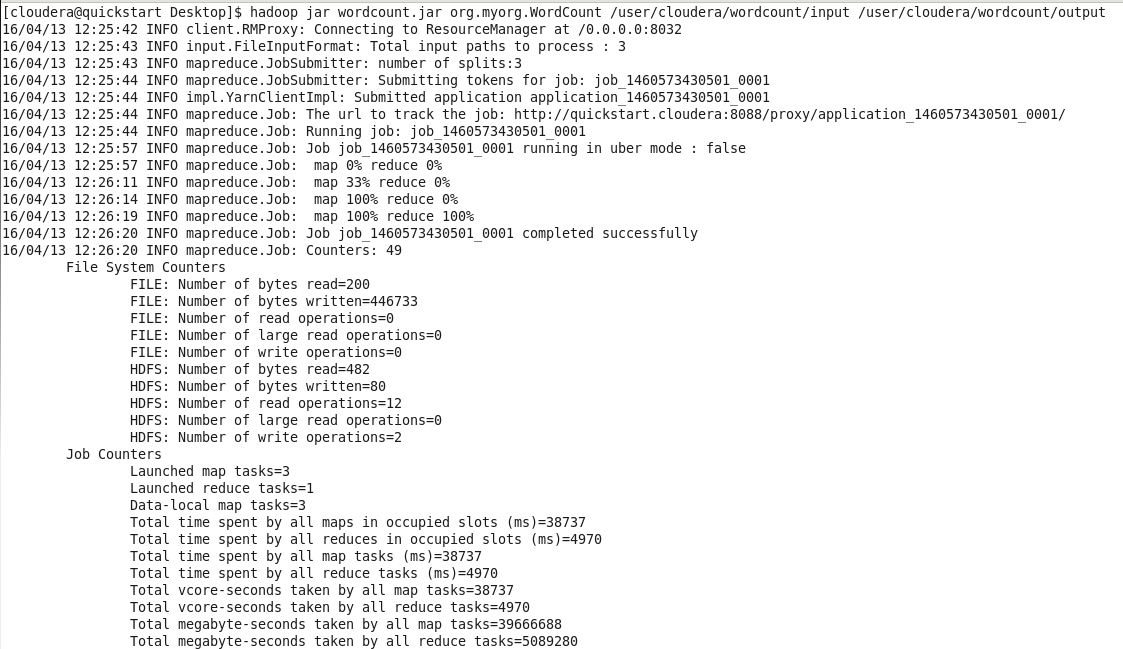
Steps 1 and 2:

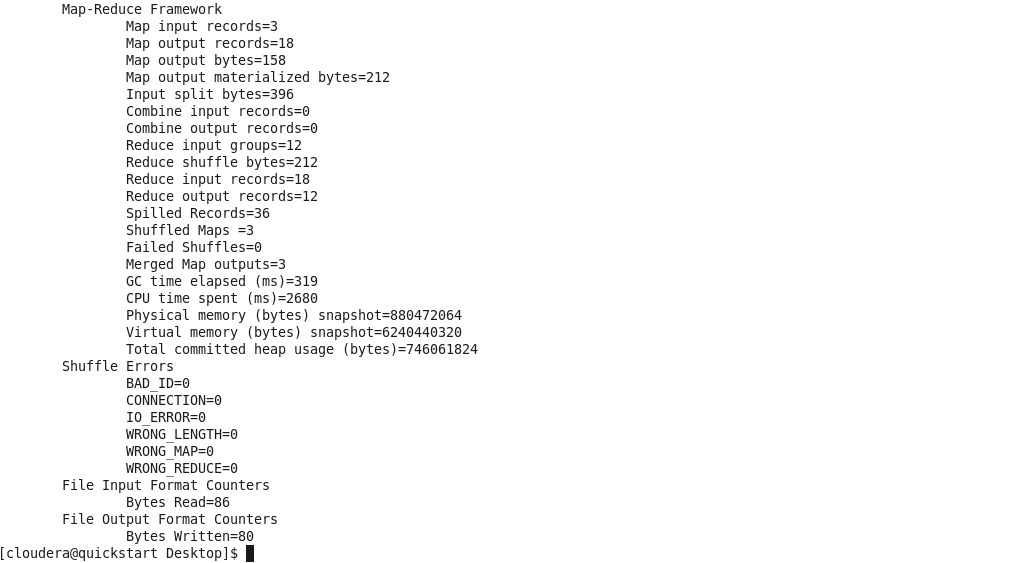


Steps 3 and 4:

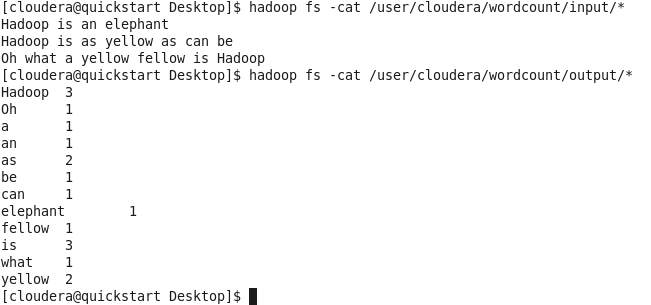


Output from running .jar file:





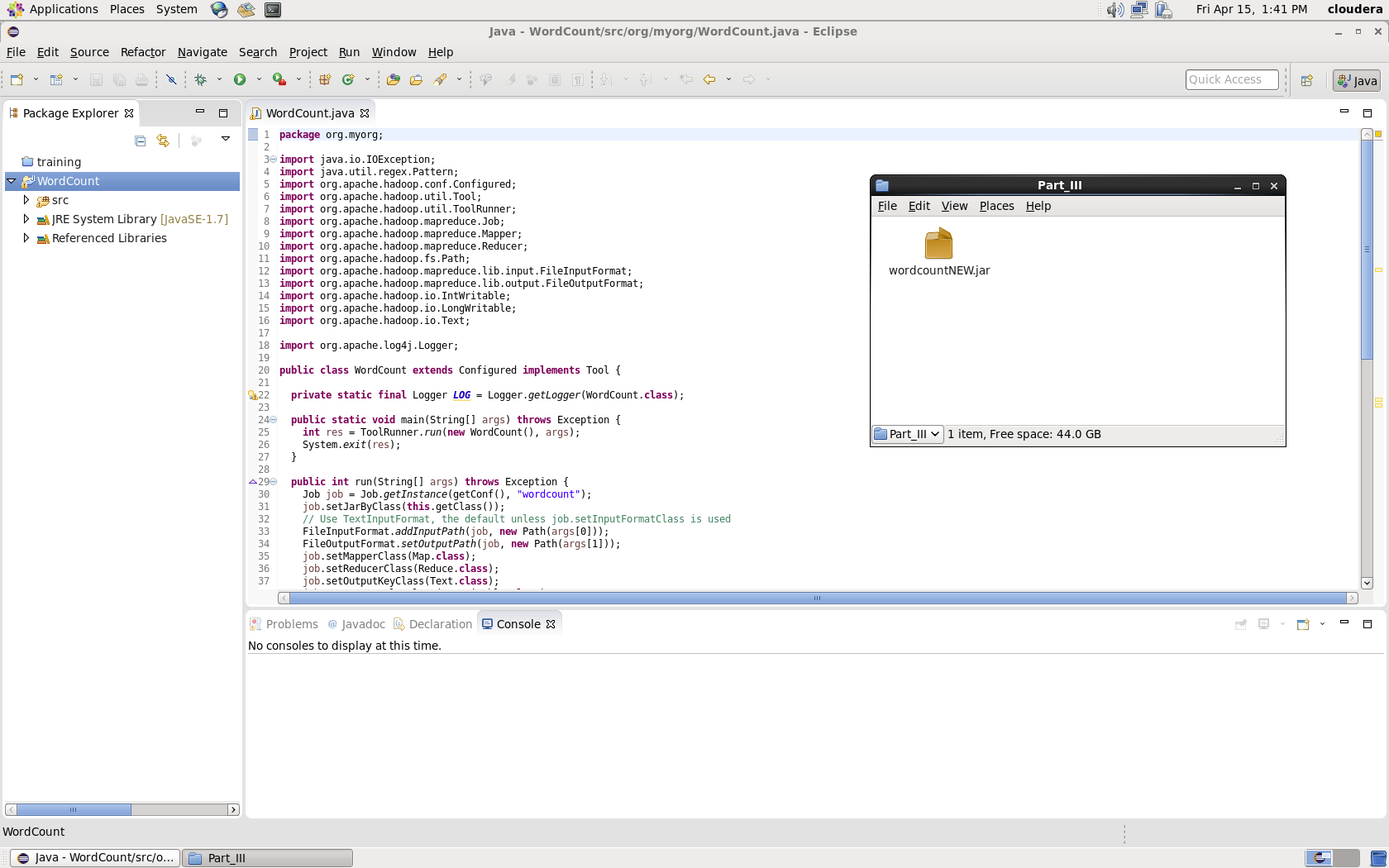
Step 5 complete, wordcount program run with provided inputs:



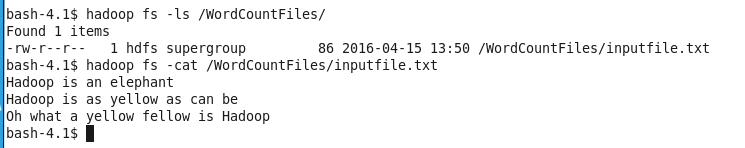
**Question 3 (2 pts)** **Eclipse Hadoop project development:** Please follow the installation instruction Part III (**Part III:** Eclipse MapReduce programming platform) to create a WordCount Eclipse Project. You can use WordCount.java file downloaded from the Cloudera website (please refer to the instruction for details). After that, please report the following major steps (capturing screenshots)

* Report that you have created an Eclipse WordCount project (0.5 pt)
* Report that you can compile the WordCount project and output JAR file (0.5 pt)
* Run WordCound.jar as a MapReduce task, and report the output (1 pt)

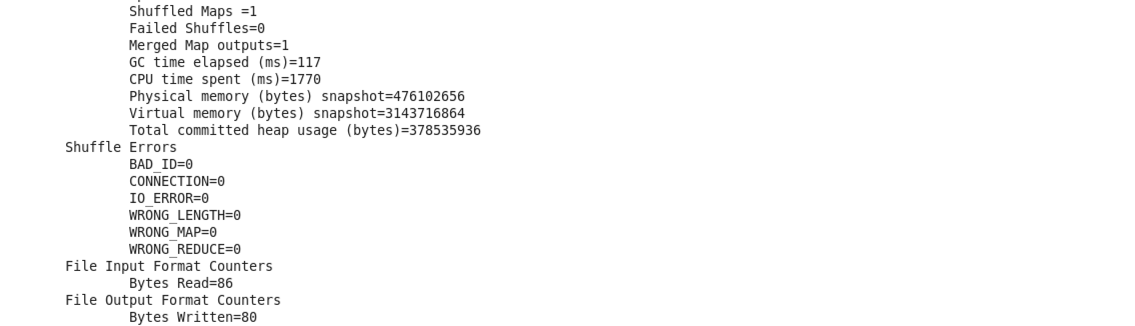
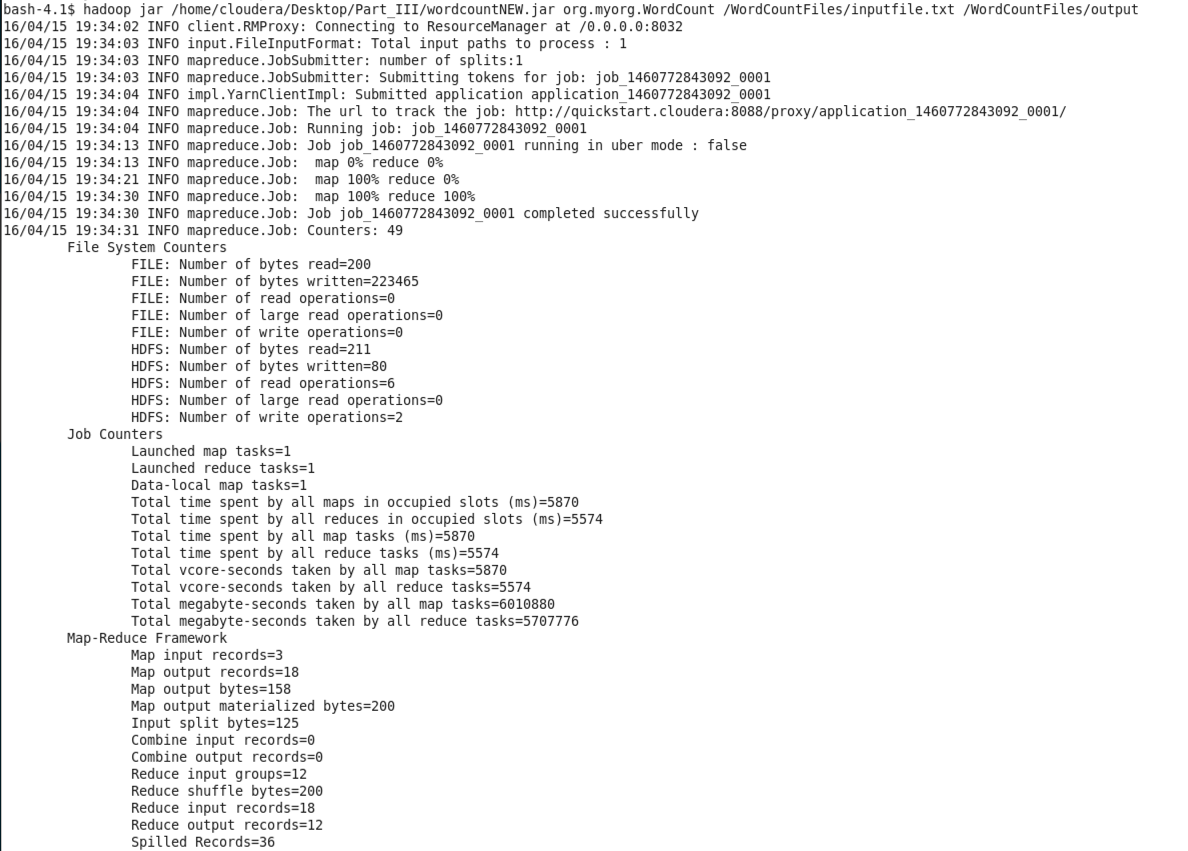
1. & 2. The WordCount project is created in Eclipse and the .jar file is output (called wordcountNEW):



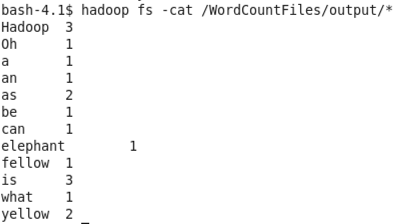
Input file to be used with wordcountNEW.jar is moved into the HDFS:



3. The MapReduce task is run via wordcountNEW.jar:



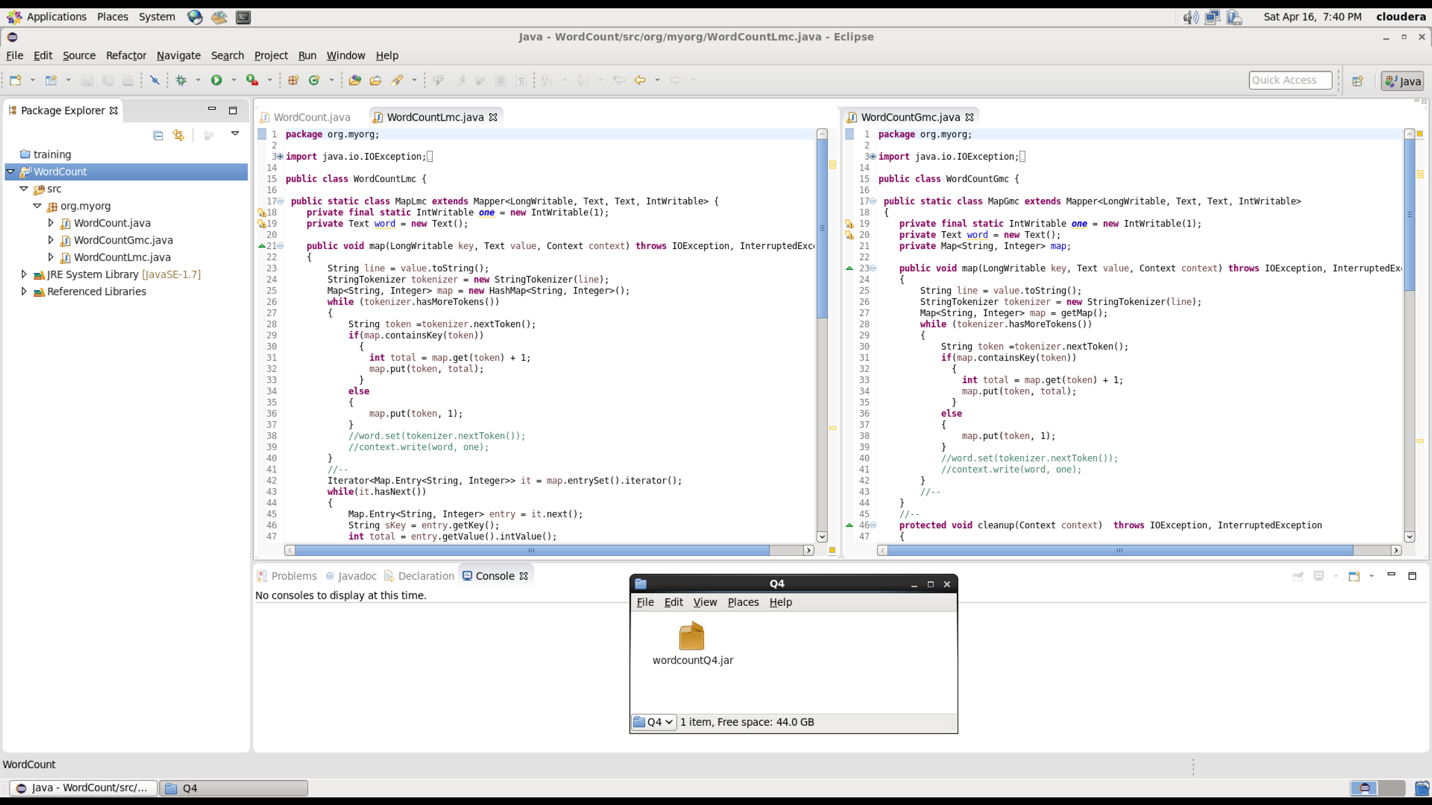
The final results are output:



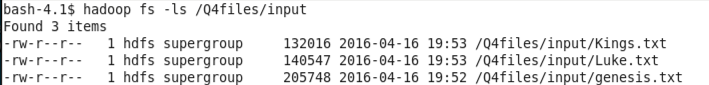
**Question 4 (3 pts) Mapper functions:** In the attached “WordCountLmc.java” and “WordCountGmc.java”, the mapper functions create associate array to maintain key-value pair status. The difference is that WordCountLmc uses local in-mapper-combing, and WordCountGmc uses global in-mapper-combing.

* Please modify your Hadoop Project in Question 3, to create a new project “WordCountLmc”, which uses local in-mapper-combing to count word frequency. Please use “genesis.txt”, “luke.txt”, and “kings.txt” as input (place all three files in a folder named “input”), and report the running results of the project. (1 pt)
* Please modify your Hadoop Project in Question 3, to create a new project “WordCountGmc”, which uses global in-mapper-combing to count word frequency. Please use “genesis.txt”, “luke.txt”, and “kings.txt” as input (place all three files in a folder named “input”), and report the running results of the project. (1 pt)
* Please compare running results from three MapReduce Tasks, WordCount, WordCountLmc, and WordCountGmc. Analyze and report the differences (i.e. Explain the benefits of in-mapper-combining, and explain how local and global in-mapper-combing achieve the efficiency gain) (1 pt)

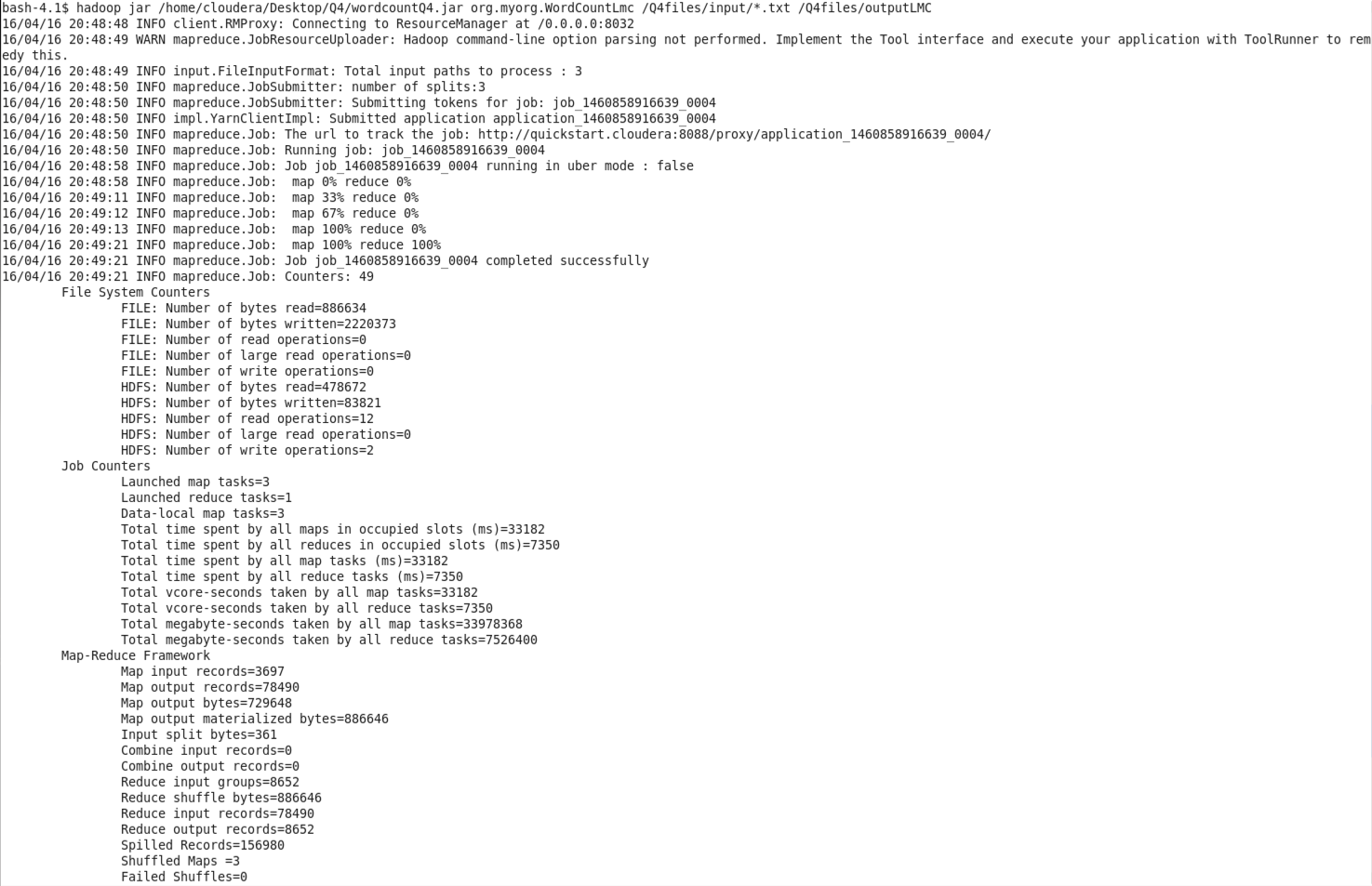
1. The WordCountLmc.java and WordCountGmc.java code is added to the project and a new .jar file is created:



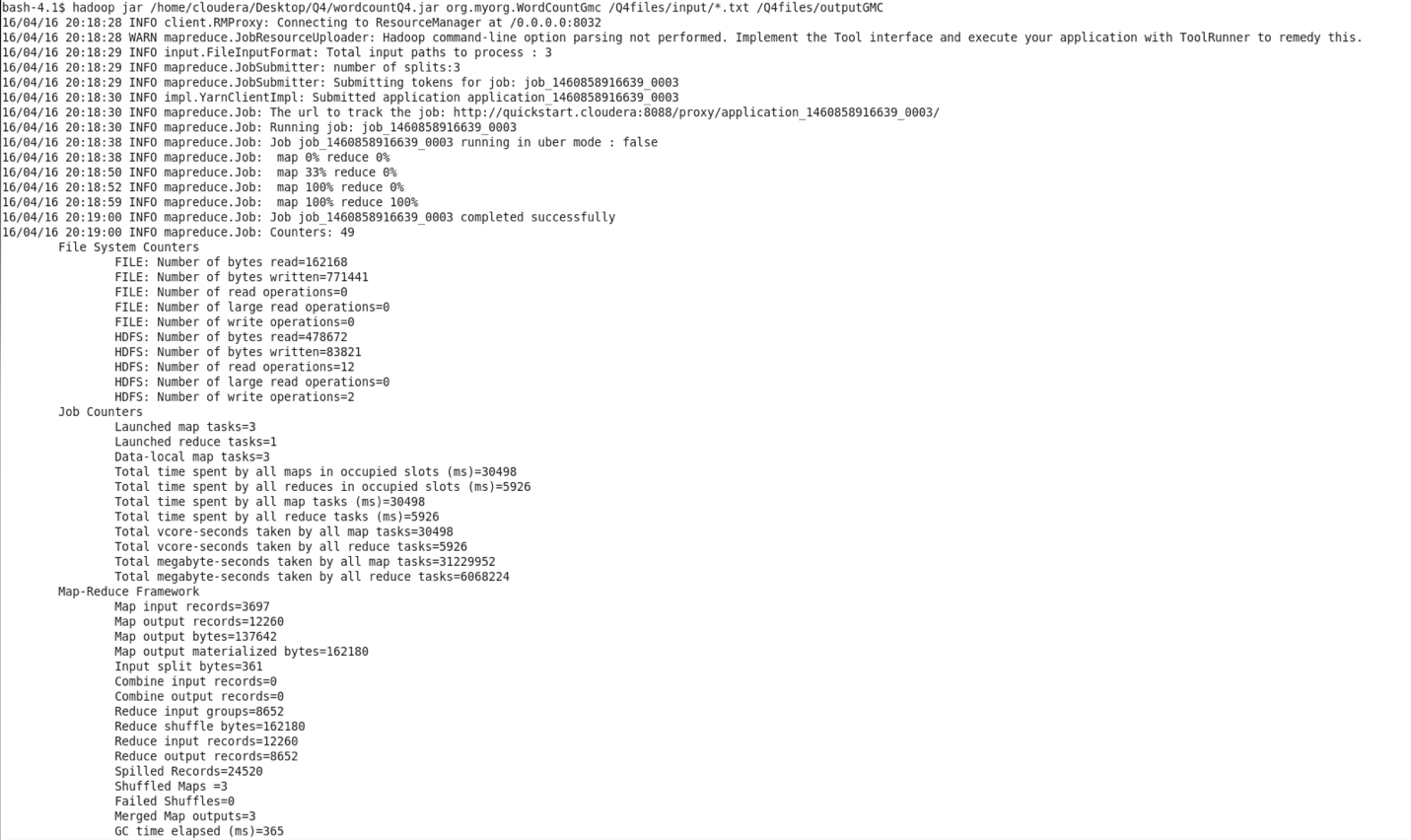
The text files are copied into the HDFS:



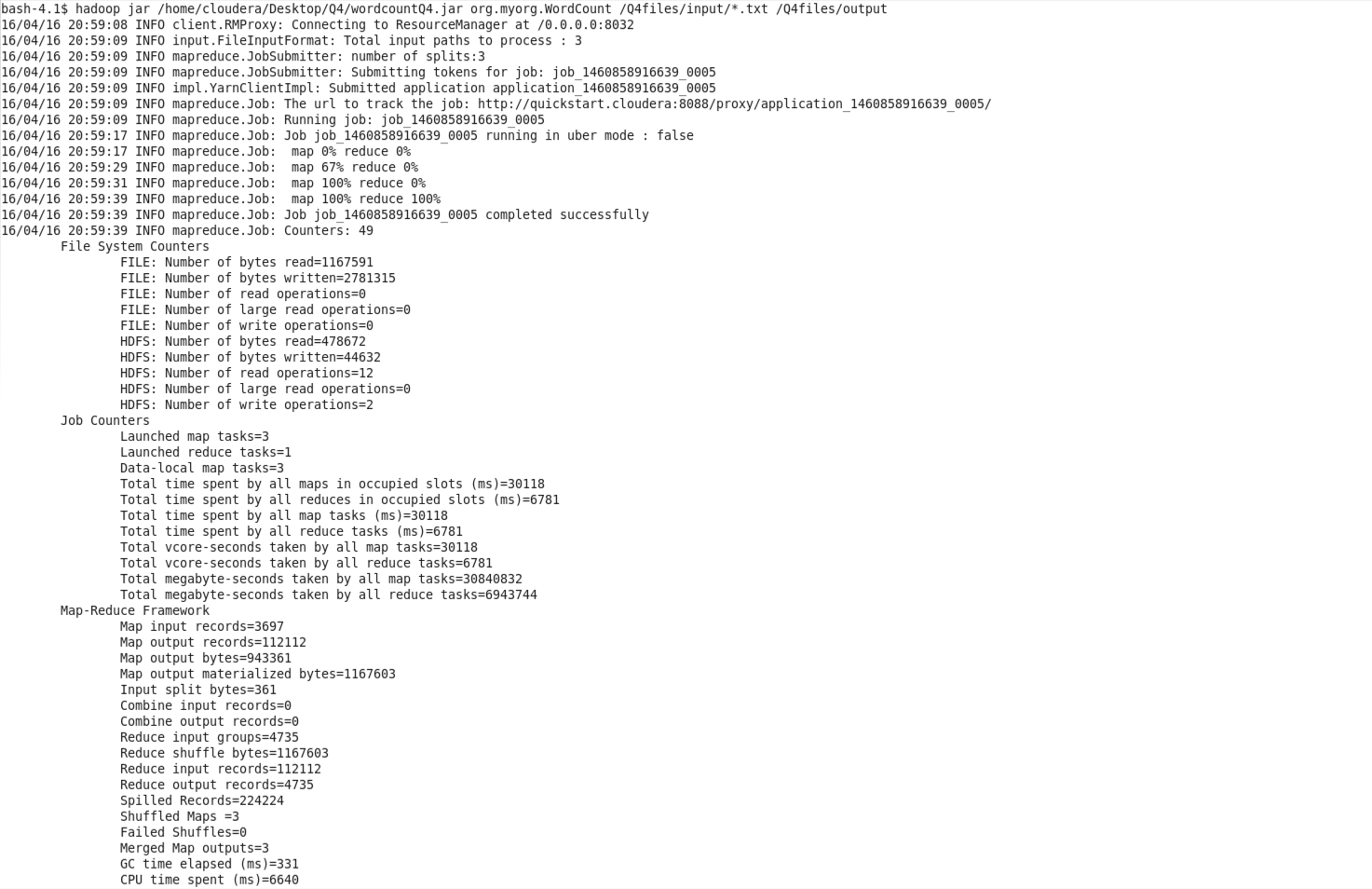
1. The WordCountLmc.java program is run. Output is too large to display in the console, run results:



2. The WordCountGmc.java program is run. Again, output is too large to display in the console, run results:



3. WordCount.java is run on the current dataset for comparison:



Without a combiner, each token will emit a key-value pair for the reducers to sum, which is a major inefficiency in the MapReduce program. By using a local in-mapper combiner, an associative array is created inside the map method to collect and count tokens. This means that the reducers will not have to sum tokens, but only words, which can be a much smaller task. Further optimization can be made by moving the associative array out of the map method and into the mapper cleanup method. In this case, the mapper emits the key-value word counts only once – upon its task completion. With a large number of mappers and reducers, this greatly reduces shuffling overhead, but could create a space issue if the array grows too large.

In my program execution, the most observable differences between the three implementations was in the file system read & write space, and CPU time, as shown below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Read bytes | Written bytes | CPU time |
| No optimization | 1,167,591 | 2,781,315 | 6,640 |
| Local | 886,634 | 2,220,373 | 6,280 |
| Global | 162,168 | 771,441 | 5,900 |

By using in-mapper combining, the efficiency improved somewhat for the local version, and substantially for the global version. This is due to the lower frequency of emissions shuffled to the reducers.

**Question 5 (1.5 pt)** **Bigram Counting MapReduce Task:** Given a sentence, a bigram denotes a unit consists of two consecutive words of the sentence. For example, given a sentence “I am a student at FAU”, there are five bigrams: (I am), (am a) (a student) (student at) (at FAU). Bigrams are used to preserve the context information in the sentence.

Please deign a MapReduce task, which takes “genesis.txt” as the input, and count the frequency of all bigrams (excluding punctuations).

* Please submit your java file [1.0 pt]
* Please use “genesis.txt” as input, and report the final counting results [0.5 pt]

1. The java file is attached to the homework submission on BlackBoard. The provided WordCount.java file was modified slightly to count Bigrams. Rather than iterating through the file and emitting a key-value for each word, the loop saves the previous word and emits a key-value when a bigram is found. This uses the provided regular expression for splitting the text and the provided document for input. As such, punctuation is not excluded from the bigram count.

2. The output text file is attached to the homework submission on BlackBoard as well. This file is too long to add to this document, but the run logs are shown below.

