CS562 Assignment 1

Alexandros Kornilakis

October 2018

1 Introduction

At this assignment we are called to implement a number of simple tasks (find stopwords, create an inverted index) on Apache Hadoop Platform. To implement these tasks, I used the standard Java bindings in a testing environment. This environment consisted of a vagrant VM with CDH and a host machine with AMD FX-6300 and 8GB RAM.

2 Exercise 1 : Frequent Terms and Stop Words

To run the code for this exercise, go to exercise1 directory and run make. You probably have to modify the username variable in the makefile to match the right one (cloudera).

The main method is in MRManager.java file where the job configurations hold. The mapper and reducer for the wordcount are at CountMap and CountReduce classes respectively, while the corresponding classes for Stopwords are StopwordsMap and StopwordsReduce. In order to sort the output by value, I created a custom Writable class for words as well as custom Partitioner and GroupingComparator.

The sorted stopwords list is at the file stopwords.csv. However, the 10 most frequent words sorted by their frequency (desc) are:

the	157207
and	124391
a	83592
of	80834
to	72891
i	58296
it	51992
in	48977
that	41497
was	39729

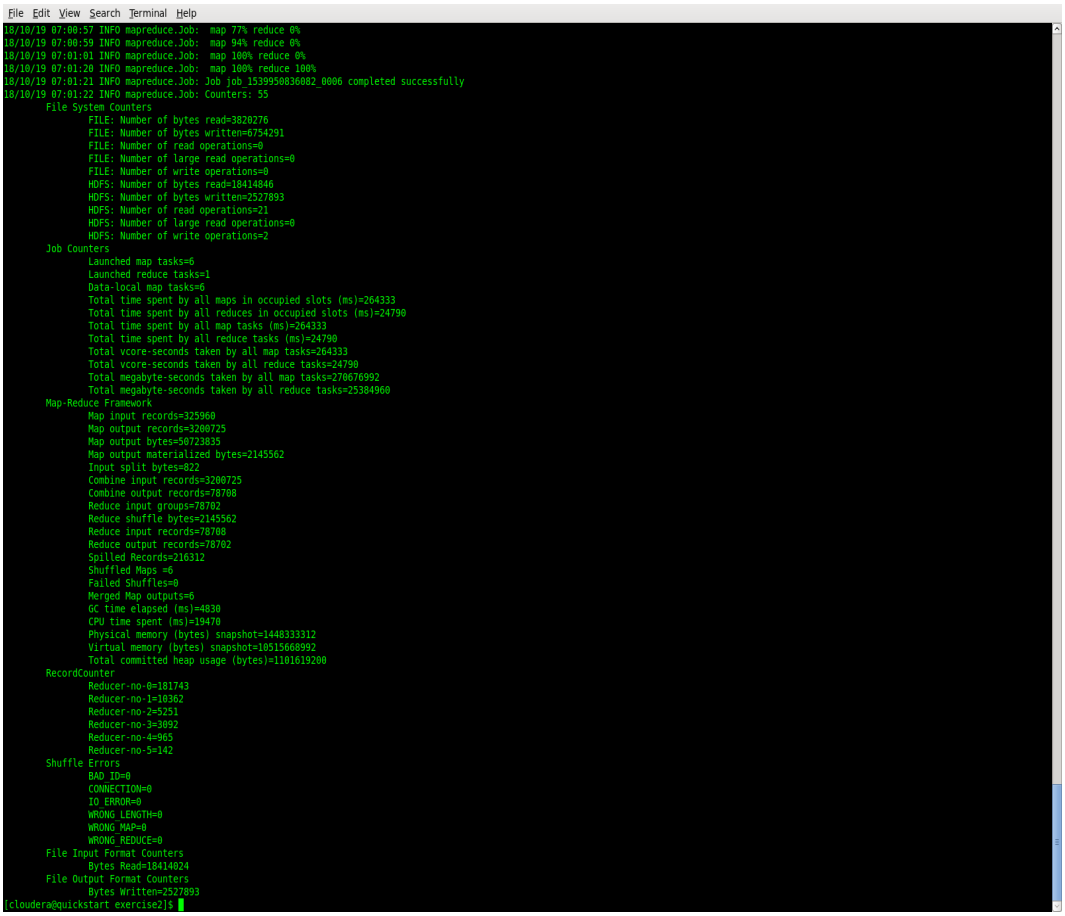
```
18/10/18 06:59:41 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=924937
                FILE: Number of bytes written=3081257
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=666867
                HDFS: Number of bytes written=549
                HDFS: Number of read operations=33
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=69287
        Job Counters
                Launched map tasks=10
                Launched reduce tasks=1
                Data-local map tasks=10
                Total time spent by all maps in occupied slots (ms)=197239
                Total time spent by all reduces in occupied slots (ms)=660683
                Total time spent by all map tasks (ms)=197239
                Total time spent by all reduce tasks (ms)=660683
                Total vcore-seconds taken by all map tasks=197239
                Total vcore-seconds taken by all reduce tasks=660683
                Total megabyte-seconds taken by all map tasks=201972736
                Total megabyte-seconds taken by all reduce tasks=676539392
        Map-Reduce Framework
                Map input records=69285
                Map output records=69285
                Map output bytes=786361
                Map output materialized bytes=924991
                Input split bytes=1390
                Combine input records=0
                Combine output records=0
                Reduce input groups=69285
                Reduce shuffle bytes=924991
                Reduce input records=69285
                Reduce output records=10
                Spilled Records=138570
                Shuffled Maps =10
                Failed Shuffles=0
                Merged Map outputs=10
                GC time elapsed (ms)=23687
                CPU time spent (ms)=475280
                Physical memory (bytes) snapshot=2403602432
                Virtual memory (bytes) snapshot=16518885376
                Total committed heap usage (bytes)=1716887552
        RecordCounter
                Reducer-no-0=69285
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
```

Figure 1: Screenshot of ex1 output

3 Exercise 2

3.1 a) Measuring the Performance on Map Reduce

1. 10 reducers and no Combiner. Execution time : 19470ms

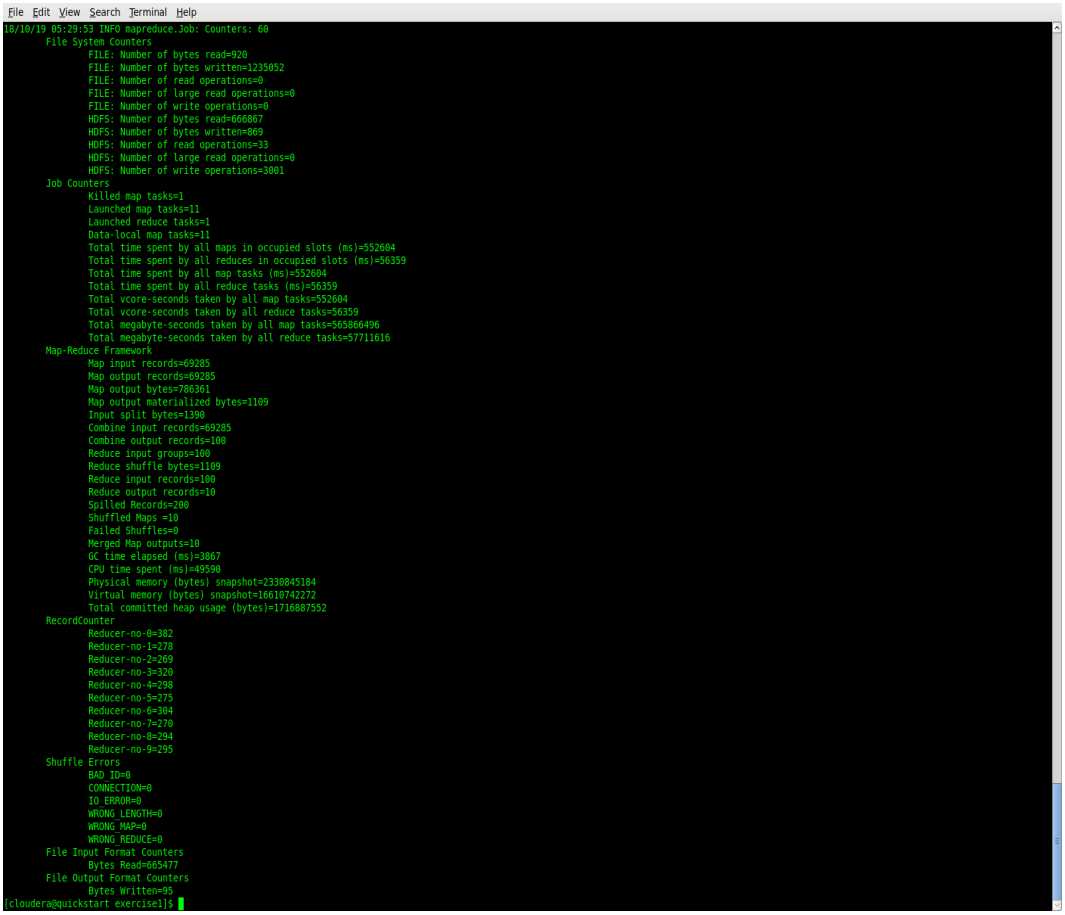


2. 10 reducers and a Combiner. Execution time : 52580ms. The combiner reduces the number of data that are send to the reducer and thus provides a speedup. However, this may not always be the case. The use of combiner enforce an additional shuffle and sort phase and thus creates a trade-off. Furthermore, the use of 10 reducers for a single node machine imposes a significant latency due to the increased I/O requirements of the data trans-

```
                Map output records=69285
                Map output bytes=786361
                Map output materialized bytes=1075
                Input split bytes=1390
                Combine input records=69285
                Combine output records=100
                Reduce input groups=100
                Reduce shuffle bytes=1075
                Reduce input records=100
                Reduce output records=10
                Spilled Records=200
                Shuffled Maps =10
                Failed Shuffles=0
                Merged Map outputs=10
                GC time elapsed (ms)=4900
                CPU time spent (ms)=52580
                Physical memory (bytes) snapshot=2212003840
                Virtual memory (bytes) snapshot=16551325696
                Total committed heap usage (bytes)=1716887552
        RecordCounter
                Reducer-no-0=378
                Reducer-no-1=269
                Reducer-no-2=269
                Reducer-no-3=301
                Reducer-no-4=288
                Reducer-no-5=238
                Reducer-no-6=296
                Reducer-no-7=275
                Reducer-no-8=269
                Reducer-no-9=281
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=665477
        File Output Format Counters
                Bytes Written=91
[cloudera@quickstart exercise1]$
```

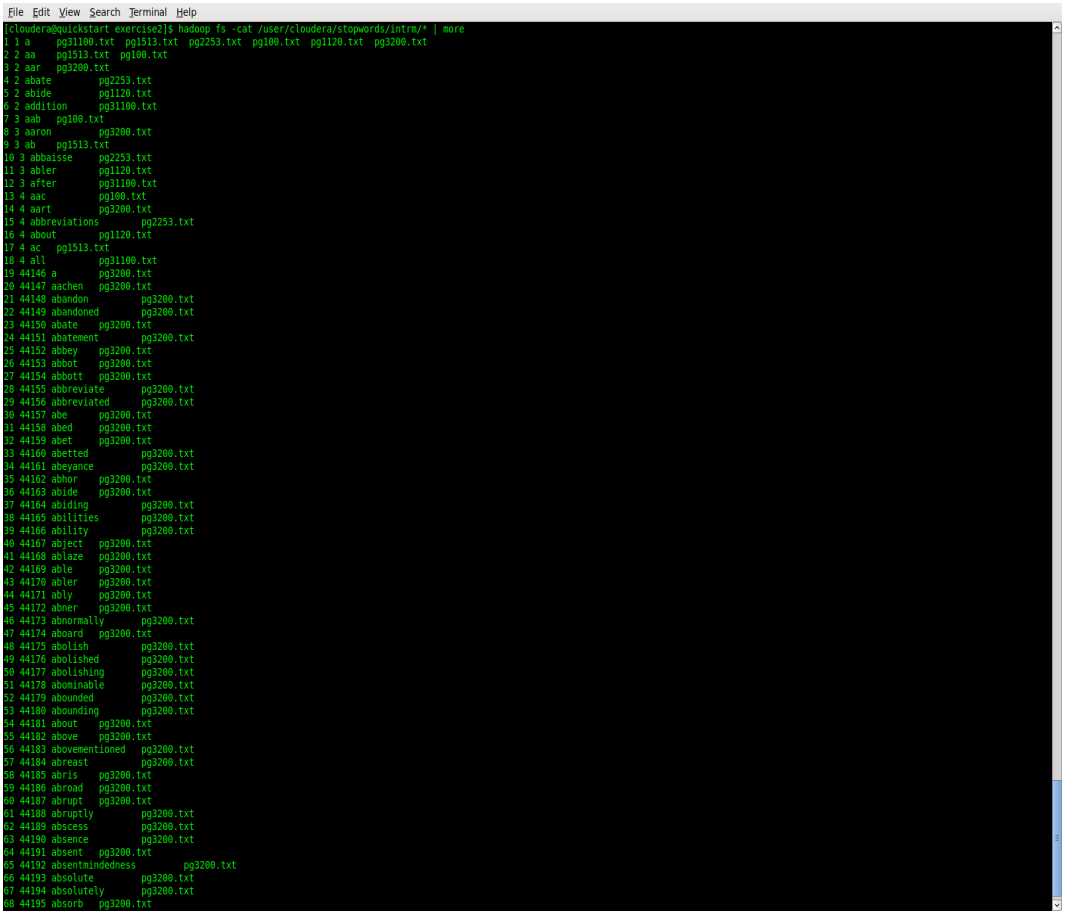
fer.

3. 10 reducers, a Combiner and compressing the intermediate results. For compression I used the snappy codec. Execution time : 49590ms. I've noticed a decreased latency due to the compression. For larger file sizes, we could observe vastly improvements.



3.2 b) A variation of an inverted index.

1. At the following screenshot we can observe the first entries of the requested invented index.



2. The counter that reveals the number of unique words is named "reduce output records". I defined my own counter and the value of that counter was 78702. As a result, this is the number of unique words. T

4 Exercise 3. An extension of an inverted index

At this exercise we had to extend the inverted index in order to keep the frequency of each word in each document. It is obvious that that index is similar to the previous one. An example entry in our output set: 39 ability pg3200.txt144 pg1120.txt2 pg2253.txt3