## SG3 Lab 5 Report

The purpose of this lab was to write MapReduce programs in Spark and Hadoop to count the top 10 frequency of words in a group of documents. For our documents, we chose a crime big dataset, since it's relevant to our project NightBot, which is a security robot for the elderly. Our data set consisted of 3 CSV files with the crime that occurred and the day it occurred to satisfy the group of documents requirement. The idea behind finding the top 10 frequency of words is to gather some interesting statistics of the most common crimes that occur and when they occur so that NightBot can tailor it's security settings towards them.

First, we used Hadoop to write a MapReduce algorithm in Java. The algorithm used maps to map the words to a count of 1. The reducers then combined all of the same words into words and their frequencies using the intermediate output from the mappers and then sorted them to obtain the top 10 most frequently used words from our group of documents. We have the output here (the output file is included on the git repo as well):

none 5927

theft 4563

arrest 3211

of 2031

larceny 1998

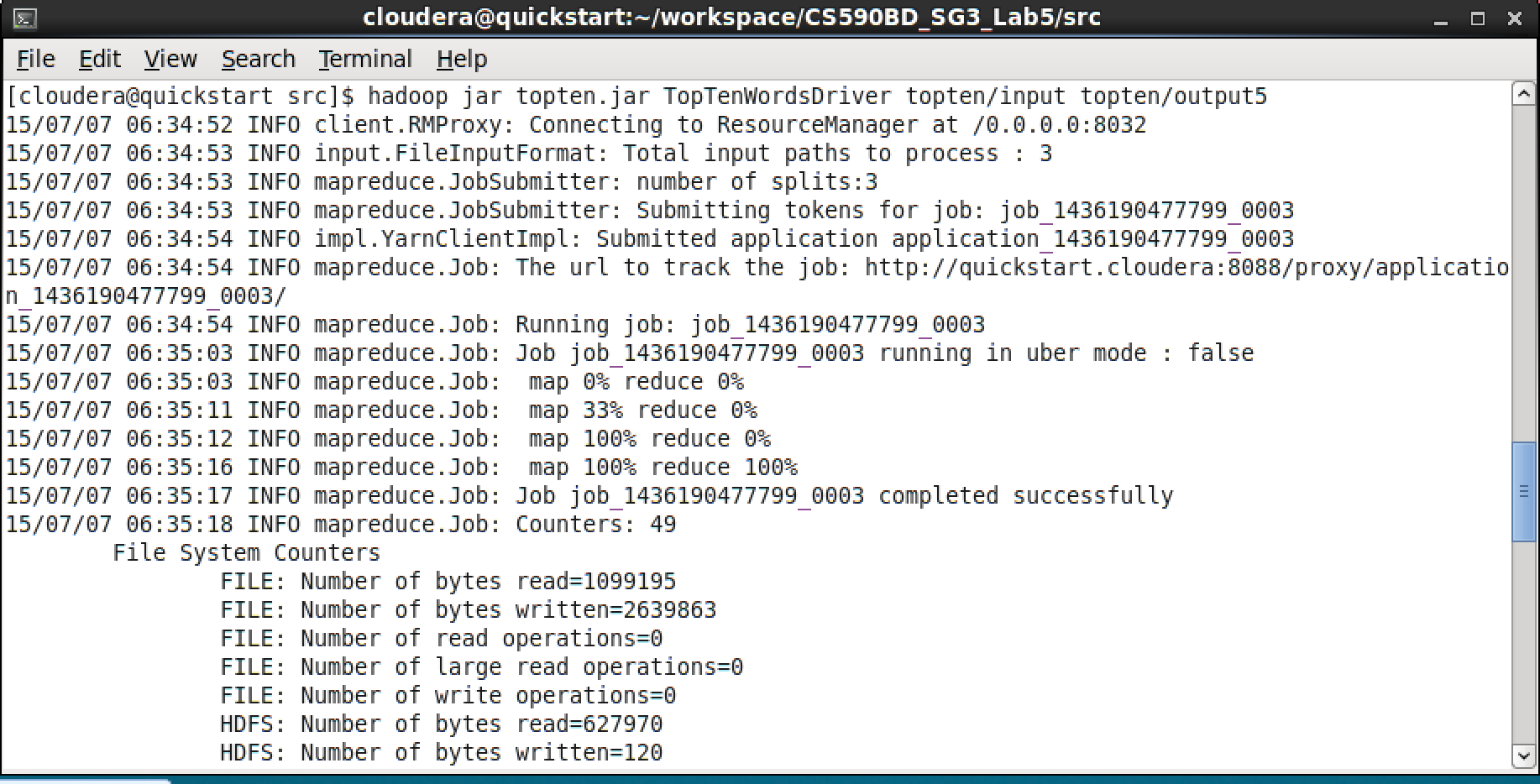
booked 1995

southern 1731

friday 1679

saturday 1607

thursday 1499



Second, we wrote a Spark program to functionally do the same thing, but written using Scala.

Here is our algorithm using Spark and Scala, note how the code is much shorter than Hadoop:

val dataFile = "hdfs://quickstart.cloudera:8020/user/cloudera/topten/input/"

val conf = new SparkConf().setAppName("TopTenWords Application")

val sc = new SparkContext(conf)

// split each document into words

val tokenized = sc.textFile(args(0)).flatMap(\_.replaceAll("[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']", " ").split("\\s+"))

// count the occurrence of each word

val wordCounts = tokenized.map((\_, 1)).reduceByKey(\_ + \_)

// filter out words with less than threshold occurrences

//val filtered = wordCounts.filter(\_.\_2 >= threshold)

val filtered = wordCounts.sortBy(\_.\_2, false).take(10)

val results = filtered.mkString(", ")

System.out.println(results)

sc.parallelize(List(results)).saveAsTextFile(args(1))

Here is the output, which is the same as the output from our Hadoop version:

