TOBB University of Economics and Technology
Department of Computer Engineering

BİL401/501 Distributed Data Processing and Analysis
Spring 2016

Homework 3

Due by Feb 22 (Monday) 24:00

Subject: Counting bigrams in Spark

(Adapted from Jimmy Lin's class at Waterloo)

In this assignment you will "port" the MapReduce implementations of the bigram frequency count program from Bespin over to Spark (in Scala). Your starting points are ComputeBigramRelativeFrequencyPairs and ComputeBigramRelativeFrequencyStripes in package io.bespin.java.mapreduce.bigram (in Java). You are welcome to build on the BigramCount (Scala) implementation here for tokenization and "boilerplate" code like command-line argument parsing. To be consistent in tokenization, you should copy over the Tokenizer trait here. You'll also need to grab missing Maven dependencies from here (update for your repo accordingly so that Scala/Spark code will compile with the same Maven build command:

$ mvn clean package

Following the Java implementations, you will write both a "pairs" and a "stripes" implementation in Spark. Not that although Spark has a different API than MapReduce, the algorithmic concepts are still very much applicable. Your pairs and stripes implementation should follow the same logic as in the MapReduce implementations. In particular, your program should only take one pass through the input data.

Make sure your implementation runs on the Shakespeare collection you used in the previous assignment.

You can verify the correctness of your algoritm by comparing the output of the MapReduce implementation with your Spark implementation. The output should be the same.
Clarification on terminology: informally, we often refer to "mappers" and "reducers" in the context of Spark. That's a shorthand way of saying map-like transformations (map, flatMap, filter, mapPartitions, etc.) and reduce-like transformations (e.g., reduceByKey, groupByKey, aggregateByKey, etc.). Hopefully it's clear from lecture that while Spark represents a generalization of MapReduce, the notions of per-record processing (i.e., map-like transformation) and grouping/shuffling (i.e., reduce-like transformations) are shared across both frameworks.

Brief explanation about the relationship between --num-executors and --reducers. The --num-executors flag specifies the number of Spark workers that you allocate for this particular job. The --reducers flag is the amount of parallelism that you set in your program in the reduce stage. If --num-executors is larger than --reducers, some of the workers will be sitting idle, since you've allocated more workers for the job than the parallelism you've specified in your program. If --reducers is larger than --num-executors, then your reduce tasks will queue up at the workers, i.e., a worker will be assigned more than one reduce task. In the above example we set the two equal.
Note that the setting of these two parameters should not affect the correctness of your program. The setting of ten above is a reasonable middle ground between having your jobs finish in a reasonable amount of time and not monopolizing cluster resources.

A related but still orthogonal concept is partitions. Partitions describes the physical division of records

across workers during execution. When reading from HDFS, the number of HDFS blocks determines the number of partitions in your RDD. When you apply a reduce-like transformation, you can optionally specify the number of partitions (or Spark applies a default) — in this case, the number of partitions is equal to the number of reducers.

You can use Hortonworks sandbox, or some other built toolbox, or a Spark installation on your computer. Here is another one you can use from databricks (code+data link under intro workshop).

When you've done everything, commit to your repo and remember to push back to origin. You should be able to see your edits in the web interface. Before you consider the assignment "complete", I would recommend that you verify everything above works by performing a clean clone of your repo and going through the steps above.
That's it!

Put all your assignments (previous) under your git repo / bigdatacourse / (asg1, asg2, …) and share the top repo (**bigdatacourse**) with the instructor (edogdu@github, erdogandogdu@bitbucket).

Upload date/time in your repo should be before the submission date/time to get full credit. 10 points are deducted for each late day, and your submission is not accepted after 2 late-days unless permitted by the instructor for legitimate excuses.