Lab 11: First Steps with Apache Spark

Instructor: Mackale Joyner, Co-Instructor: Zoran Budimlić

Course Wiki: `http://comp322.rice.edu`

Staff Email: comp322-staff@mailman.rice.edu

1 Acknowledgements

This provided code in the lab presents the Spark implementation of word count and PI computation described at `http://spark.apache.org`.

2 Overview

The purpose of this lab is to write two example applications using Apache Spark. The focus will be on functionality, not performance. Also, we will not use HJlib in today's lab, so you do not need to add a javaagent configuration today.

Although Apache Spark is best applied to distributed computation, it can be be run in "local mode," where it will simply make use of the available cores on your computer. Using local mode, we can view Spark as another viable model for multicore parallel computing.

In today's lab, you **do not** need to use NOTS or the autograder to run performance tests. For the purposes of this lab, we will run in local mode. However, the commands you will execute are identical to what you would use to run a distributed computation on a cluster with Spark installed.

This lab can be downloaded from the following svn repository:

- `https://svn.rice.edu/r/comp322/turnin/S19/`*`NETID`*`/lab_11`

Use the subversion command-line client or IntelliJ to checkout the project into appropriate directories locally.

3 Selective Wordcount

As shown in class, we can use the map/reduce pattern to count the number of occurrences of each word in the RDD:

```
final JavaPairRDD<String, Integer> counter = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")))
    .mapToPair(s -> new Tuple2<>(s, 1))
    .reduceByKey((a, b) -> a + b);
```

We can then view the result of running our word count via the `collect` operation:

```
counter.collect();
```

Your first task is to alter our map/reduce operation on `textFile` *so that only words of length 5 are counted, and then display the counts of all (and only) words of length 5.*

Hints:

- An `if` expression can be used in any context that an expression can be used, and returns the value returned by whichever branch of the `if` expression is executed.
- The length of a string `s` can be found by using the method `s.length()`
- The elements of a pair can be retrieved using the accessors `_1` and `_2`.
- RDDs have a method `filter()` that takes a boolean test and returns a new RDD that contains only the elements for which the testing function passed. For example, the following application of filter to a list of ints returns a new list containing only the even elements:

  `List(1, 2, 3, 4, 5).filter(n -> n % 2 == 0) will produce List(2, 4)`

4 Estimating π

We can now walk through a Spark program to estimate π in parallel from random trials. We can estimate π in Spark with the following code snippet:

```
final int reducedValue = context.parallelize(terms)
    .map(i -> {
        final double x = Math.random();
        final double y = Math.random();
        return (x * x + y * y < 1) ? 1 : 0;
    })
    .reduce((a, b) -> a + b);

return String.valueOf(4.0 * reducedValue / terms.size());
```

where terms is a list of terms from 0 to N.

As we saw in Lab 9 on Actors, the following formula can also be used to compute π:

$$\pi = \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \left(\frac{1}{16} \right)^n$$

Your next task in the lab is to use the map-reduce technique in Spark to compute the sum of the first N terms of this series and use the `BigDecimal` *class to achieve better accuracy in the computation of pi.*

5 Turning in your lab work

For this lab, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab.
2. Check that all the work for today's lab is in the `lab_11` turnin directory. It's fine if you include more rather than fewer files — don't worry about cleaning up intermediate/temporary files.