**CME 323: Distributed Algorithms and Optimization**

**Instructor: Reza Zadeh (rezab@stanford.edu)**

**HW#3 – Due Thursday May 23 11:59pm (on Gradescope)**

1. **Intro to Spark** Download the following materials.

   - Slides
   - In this question, you will need to use two files, `README.md` and `contributing-to-spark.md`. You can get these files from either build usb or Google Colab.
   - For Spark, you can either install Spark locally (refer to the instructions on the Spark Website), or use Google Colab colab link. We have provided the configuration in the Colab file. However, Colab currently does not support Scala, the language used in the slides, so you will need to write Python code to achieve the same tasks.

   Now, answer the following questions.

   (a) Checkpoint on slide 11.

   (b) Checkpoint on slide 55.

   (c) Checkpoint on slide 60. Note: slide 59 references the file `contributing-to-spark.md`.

   Submit your code and answers.

2. Write a Spark program to find the *least squares fit* on the following 10 data points. The variable `y` is the response variable, and `X1`, `X2` are the independent variables.

| | X1 | X2 | y |
|---|---|---|---|
| [1,] | -0.5529181 | -0.5465480 | 0.009519836 |
| [2,] | -0.5428579 | -1.5623879 | 0.982464609 |
| [3,] | -1.3038629 | 0.5715549 | 0.499441144 |
| [4,] | 0.6564096 | 1.1806877 | 0.495705999 |
| [5,] | -1.2061171 | 1.3430651 | 0.153477135 |
| [6,] | 0.2938439 | -1.7966043 | 0.914381381 |
| [7,] | -0.2578953 | 0.2596407 | 0.815623895 |
| [8,] | 0.9659582 | 2.3697927 | 0.320880634 |
| [9,] | -0.4038109 | 0.9846071 | 0.488856619 |
| [10,] | 0.6029003 | -0.3202214 | 0.380347546 |

   More precisely, find $w_1, w_2$, such that $\sum_{i=1}^{10}(w_1 X1_i + w_2 X2_i - y_i)^2$ is minimized. Report $w_1$, $w_2$, and the Root Mean Square Error and submit code in Spark. Analyze the resulting algorithm in terms of all-to-all, one-to-all, and all-to-one communication patterns.

3. **Intro to Map Reduce** Assume you are given a typical MapReduce implementation where you only have to write the Map and Reduce functions. The Map function you will write takes as input a (key, value) record and returns either a (key, value) record

or nothing. The Reduce function you will write takes as input (key, list of all values for that key) and returns either a record or nothing. The framework already takes care of iterating the Map function over all the records in the input file, key-based intermediate data transfer between Map and Reduce, and storing the returned value of Reduce. For all the following questions, provide algorithms at the level of pseudocode.

(a) Given as set of records (for example, movie names and ranking), provide a MapReduce algorithm to output the top K movies of the set.

(b) Suppose you are given an input file which contains comprehensive information about a social network that has asymmetrical (directed) links, i.e., a network where users follow other users but not necessarily vice-versa (e.g., Twitter). Each record in this input file is (userid-a, userid-b), where userid-a follows userid-b (i.e., points to it). Note that this record tells you nothing about whether or not userid-b follows userid-a. Write a MapReduce program (i.e., Map function and Reduce function) that outputs all pairs of userids who follow each other.

4. **Connected Components with MapReduce** Finding out the number of connected components in a graph is a key subroutine in many graph algorithms. Provide and prove the correctness of a MapReduce algorithm to count the number of connected components in a graph (represented as an edge list).

5. **Sampling from multiple streams** Suppose we have numerous sub-streams of data (say $S_1, \ldots, S_n$), provide and prove the correctness of an algorithm to generate k random samples from the aggregate stream.

6. **Word Count Shuffle** Consider counting the number of occurrences of words in a collection of documents, where there are only k possible words. Write a MapReduce to achieve this, and analyze the shuffle size with and without combiners being used (assuming B mappers are used).

7. **Prefix Sum** The *prefix-sum* operator takes an array $a_1, \ldots, a_n$ and returns an array $s_1, \ldots, s_n$ where $s_i = \sum_{j \leq i} a_j$. For example, starting with an array `[17 0 5 32]` it returns `[17 17 22 54]`. Describe (in detail) how to implement *prefix-sum* in MapReduce, where the input is stored as $\langle i, a_i \rangle$. That is, the key is the position in the array, and the value is the value at that position. Analyze the shuffle size and the reduce-key space and time complexity.