# DSC 204A: Scalable Data Systems

Programming Assignment 1

January 2024

## 1 Introduction

The goal of this programming assignment is to get you comfortable with big data processing using Modin and Ray. You will learn to setup Modin on AWS and compute descriptive statistics. You will also peek into the Ray backend used by Modin with a simple task on using the Ray Core API. First, a word from our sponsors:

**IMPORTANT: Please download your progress to your local machine at regular intervals and terminate your instance when you decide to pause working. You only have $50 for this course, so DO NOT leave instances running. If you terminate without downloading, you WILL LOSE all your work. Every time you start a new instance, you must download the dataset from S3 to your instance. Also, start only AWS Spot Instances and NOT On-Demand instances.**

## 2 Task 1: Setting up the Environment

## 3 Task 2: Data Manipulation with Modin (45)

In this question, you will be required to perform some basic data manipulation with Modin (Pandas on Ray). Modin is a library that allows users to perform Pandas workloads at scale. In this assignment, we will focus on parallelizing dataframe operations with Modin.

You are given the Amazon Reviews dataset with the schema shown in Table 3

| Column name | Column description |
|---|---|
| reviewerID | ID of the reviewer |
| vote | Helpful votes of the review |
| overall | Rating of the product |
| unixReviewTime | Time of the review (unix time) |
| reviewTime | Time of the review (raw) |

Table 1: The Amazon Reviews dataset

### 3.1 Task 2.1

Perform a few data manipulation operations on this dataset and generate a new table with the schema shown in Table 2. We have provided expected output statistics for you so that you can verify your work. Please

use the output function we provided to save the table you generated.

| Column name | Column description |
|---|---|
| reviewerID | ID of the reviewer |
| num_product_reviewed | Total number of products reviewed by this reviewer |
| mean_rating | The average rating this reviewer has given across all reviewed products |
| latest_review_year | The latest year this reviewer has given a review |
| num_helpful_votes | The total number of helpful votes this reviewer has gotten |

Table 2: Schema for the processed dataframe

### 3.2 Task 2.2

Change the number of cpus used by Modin or the Ray backend (documentation here) on your local machine and run your data manipulation code. Document the execution times you see with 2, 3, and 4 CPUs. Can you observe any speed up? If you see any speed up, please report how much speed up you see and give a brief explanation. If there is no significant speed up, please also explain the possible reason briefly.

### 3.3 Grading Scheme

For Task 2.1, there are 4 columns (apart from the ID) in the output table. If all descriptive stats (mean, std dev, min, and max) with 1% error margin match the ground truth, we award 10 points per column. Task 2.2 is worth 5 points.

## 4 Task 3: The Ray Core API (55)

This question will focus on becoming familiar with the Ray Core API. Recall the three key abstractions: Ray tasks, Ray actors and Ray objects. Don't worry if you're not very familiar with the third one. Strictly speaking, you only need to know about parallelizing computations with Ray tasks to solve this question.

### 4.1 Task 3.1

We will implement a distributed merge sort algorithm in Python. The standard merge sort algorithm uses a divide and conquer strategy to partition a given sequence into two halves, and then recursively sorts these two halves. The crucial phase is the merge step, where two sorted halves are merged to get a final sorted list. We will focus on a slightly modified version of the algorithm:

- In the first stage, the input sequence is partitioned into 4 subsequences.
- Next, each subsequence is sorted through a call to the standard merge sort algorithm.
- Finally, the 4 sorted subparts are merged using a heap-based merge algorithm.

A plain implementation of this algorithm is provided to you. Your task is to use Ray to parallelize computations to utilize all 4 CPUs in your AWS instance. The file to modify is `merge_sort_ray.py`. Parts of the code that can and cannot be modified are also highlighted in the comments. Your goal should be to get a runtime of about 84s.

P.S: You're not supposed to make algorithmic changes here (i.e change the heap-based algorithm to something else, etc). Focus on what tasks can and can't be parallelized and use Ray.

### 4.2 Task 3.2

The ideal speedup you can get for a task with 4 workers is 75%. Can you estimate the theoretical maximum speedup you can get for the above merge sort algorithm? How do you account for the difference between theoretical result and the observed speedup with Ray?

### 4.3 Grading Rubric

Task 3.2 is worth 5 points. For Task 3.1, your merge sort code will first be checked for accuracy: We expect the code to accurately sort the given list. Incorrect code will not receive any points. For runtime, we will run your scripts thrice and get the average runtime. We will use the following rubric for runtime:

| **Speedup** | **Score** |
|---|---|
| Less than 90s | 50 |
| Between 90s and 110s | 40 |
| Between 110 and 170s | 20 |
| 170s or more | 0 |

Table 3: Rubric for Task 3.1

## 5 Deliverables

For Task 2.1 and 3.1, you will need to fill in the code in the respective .py files. For Task 2.2 and 3.2, please write down your answers in a document titled "Report" (can be a word document or latex formatted) and submit it along with your code.