DSC 204A: Scalable Data Systems

Programming Assignment 2

February 2024

1 Introduction

The goal of this programming assignment is to go further with Ray - focusing on multi-node data processing and collective communication. We'll first have an easy introduction to the Ray Data API. We'll then revisit the Ray Core API and introduce Ray Actors. Finally, we will build some of the communication algorithms discussed in class through the Ray Collective Communications library. Prior to attempting any of the tasks, please take the time to review Section 5 for detailed instructions on the setup process.

2 Task 1: Ray Data (20 points)

In this question, we will introduce Ray Data, specifically how to work with Ray Datasets, and apply a few basic methods that help us transform and process them. Ray Data is a part of Ray AI Runtime, and provides a flexible API for distributed data processing. Ray Data is built on top of Ray Core, hence we get the same scaling and parallel processing benefits.

2.1 Problem

You are given a subset of the Amazon Reviews Dataset on DataHub[1]. This dataset contains more than 6 million reviews for various electronic products. You have been given a Jupyter notebook (`task1.ipynb`). Follow the instructions in the given notebook, and complete it. You should go through all documentation links provided in the notebook to be able to solve this problem and develop understanding of Ray Data. Ideally, Ray data should maximally utilize CPUs on all workers, which you can check on your Ray Dashboard's 'Cluster' section. Instructions to access the dashboard are given in Section 5.

2.2 Grading Rubric

For this problem, you need to submit a completed version of `task1.ipynb`. The section '`Cleaning Up reviewText`' is worth 10 points, and all sections prior to it are worth a combined 10 points. You will only be graded on the correctness of your code for this problem. We have also provided a couple of helpful asserts to verify that your processing is correct.

[1] ~/public/pa2_dev

3 Task 2: MapReduce with Ray Actors (40 points)

This question focuses on understanding how to process data in a distributed manner using MapReduce. We ask you to implement this algorithm for a simple, parallelizable task using Ray Actors.

3.1 MapReduce

MapReduce is a programming model designed to process large datasets in a distributed computing environment. It breaks down large, complex tasks into smaller units and these units can be distributed across multiple nodes in your cluster. A basic guide to the algorithm is here. For reference, you can refer to this notebook which has an implementation of MapReduce with Ray tasks (Make sure to scroll to the end of the notebook). MapReduce will also be covered in week 7 of the quarter for a detailed overview.

3.2 Problem

The task is to use MapReduce to count the number of occurrences for each word in a block of text. To do so, you will define your Mappers and Reducer both as Ray Actors.

For this problem, you need to implement the MapReduce algorithm in the following way -

- You will employ Mappers equivalent to the count of workers in your Ray cluster, and segment the text into partitions to evenly distribute the workload among the workers.
- You will then assign these partitions to each Mapper, and using the `map` method, compute the word counts on the partitions assigned to them.
- Once the counts are computed by the Mappers, you will use the `reduce` method from your Reducer to combine the results from every partition.

You need to modify `task2.ipynb` to complete this task. For testing, we've given a text file `essay.txt`, which contains an essay written by ChatGPT regarding large language models.

3.3 Grading Rubric

For this question, we will only be grading you based on correctness rather than specific runtime requirements. Firstly, you have to use the Ray actors API and parallelize across `NUM_CPUS` workers. If you do not parallelize or end up not using Ray actors then your submission is invalid. The public test case is the `essay.txt` file provided to you, and producing the correct counts for this file is awarded 20 points. We also have hidden test cases that make up the remaining 20 points.

4 Task 3: Collective Communication (40 points)

In this task, you will be using the point-to-point communication APIs in `ray.util.collective` to implement the AllReduce collective communication operation. To be specific, we have provided you a template of a `Worker` class. The Ray Actor processes will be running the functions of this class. You need to complete the functions of this class so that the Actor processes can perform AllReduce communication among them. You need to implement AllReduce in three different ways. We have provided you with profiling functions so that you can see the difference between these implementations.

4.1 Problem

4.1.1 P2P Communication

This is going to be the building block of the allreduce implementations. We expect you to implement the P2P functions in the `Worker` class using Ray's built-in P2P API.

4.1.2 AllReduce

You will implement 3 variants of AllReduce:

- `ray_all_reduce`: A simple AllReduce implementation using Ray's built-in.
- `bde_all_reduce`: An implementation of the Bidirectional exchange (BDE) AllReduce algorithm.
- `mst_all_reduce`: An implementation of the Minimum Spanning Tree (MST) AllReduce algorithm. As building blocks, you should first implement the MST version of Reduce and Broadcast communication collectives discussed in class. Then construct the implementation of the MST AllReduce algorithm.

For BDE and MST AllReduce, you can use the pseudocode in the required reading. The pseudocode is detailed and thus understanding it thoroughly should get you to the respective implementations. For BDE and MST AllReduce, you can only rely on functions that you define (i.e the P2P communication operations you wrote previously and any helper functions that you implement).

4.2 Grading Rubric

We will be grading this task based on correctness. You would need to ensure that the right P2P communication operations happen for each algorithm (we've provided logging so that you can inspect the same).

- The P2P communication methods : 5 points
- `ray_all_reduce` : 3 points
- BDE AllReduce: 10 points
- MST AllReduce: 15 points
- Workers instantiation : 5 points
- Profiling results: 2 points

You will find instructions in different parts of the `task3.ipynb` notebook for each delivrable.

5 Development Instructions

- We will be using DataHub (datahub.ucsd.edu) for this assignment. You should use the `ray-notebook` server. This is a multi-node cluster with 1 head node and 2 worker nodes. The head node has 2 CPUs and 12GB RAM, which is what you will see in the DataHub UI as in Figure 1. Each of the worker nodes has 3 CPUs, so your Ray cluster has a total of 8 CPUs.

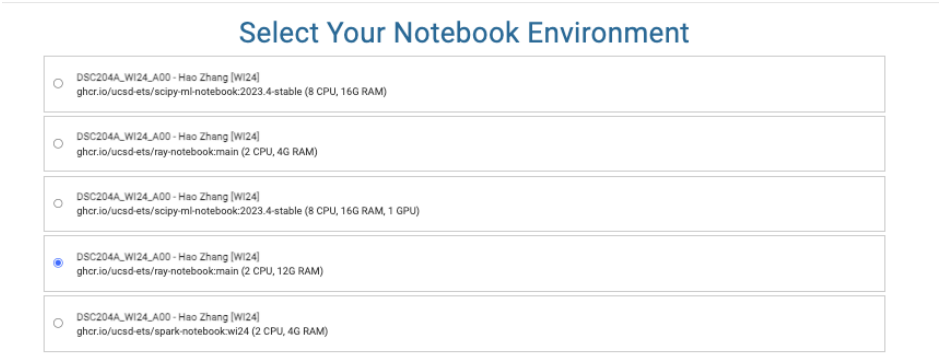

Figure 1: Selecting the correct server on DataHub. Make sure to select the `ray-notebook` server with 12GB RAM.

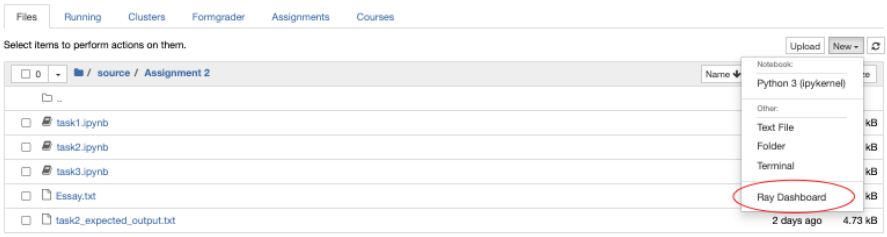

Figure 2: Ray Dashboard

- The Ray Dashboard can be viewed in the "New" dropdown as shown in Figure 2. At startup, the cluster will take some time to be fully operational since the worker nodes need to be provisioned. You can monitor the status of the cluster in the "Cluster" tab of the Ray Dashboard. In a few minutes, you should be able to see 3 "alive" nodes as in Figure 3.
- We will make use of `nbgrader` for this assignment. The assignment will be listed in the "Assignments" tab on JupyterHub, as in Figure 4.
- Once you fetch the assignment you should see the assignment files in "Downloaded assignments". Use the dropdown to navigate to different notebooks (`task1`, `task2`, `task3`), similar to Figure 5.
- In each notebook, you should only complete code blocks with a `raise NotImplementedError()` statement. One example is shown in Figure 6. Cell blocks without this statement are read-only, and any modifications here will not show up in the final version you submit.
- After you've completed a problem, you can validate whether your answers are correct with the "Validate" button (Figure 5). This will run the public test cases for that assignment and raise errors if any. You can now submit the assignment with the "Submit" button. You are free to submit as many times as you wish before the due date.

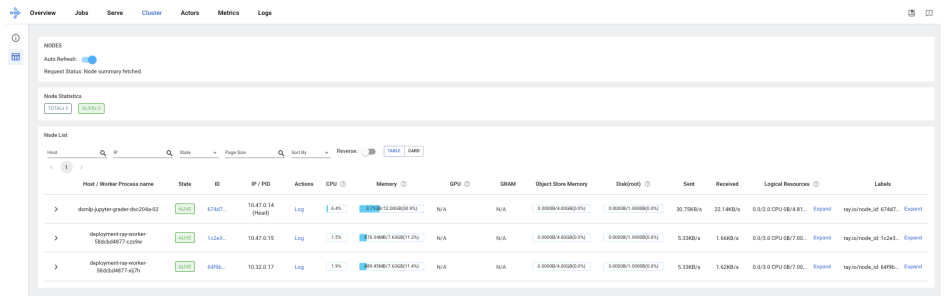

Figure 3: Ray Cluster table once all the nodes have been provisioned.

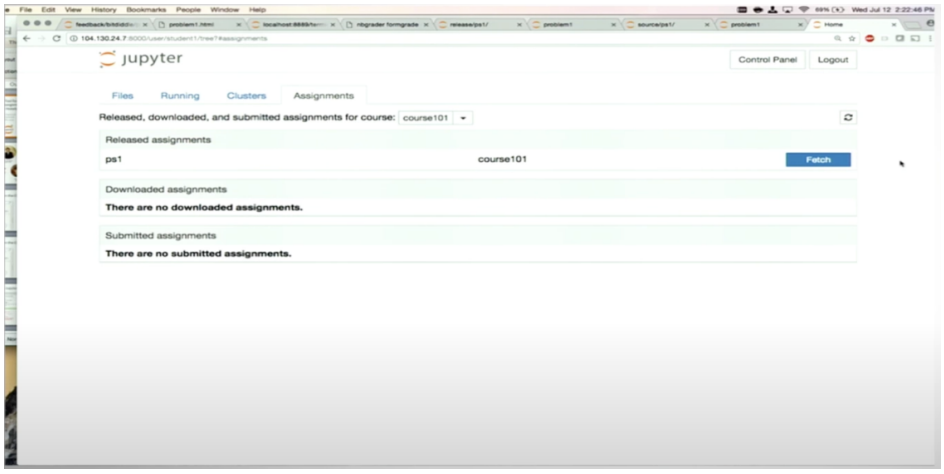

Figure 4: Fetching the assignment in JupyterHub (source)

- Note: If you're running "Validate" for Task 1, it will take a considerable amount of time because we're dealing with multi-node data processsing. We will only be checking for correctness for this question. Thus, if you've verified that the code is correct, you need not trigger `nbgrader`'s validation and directly submit.

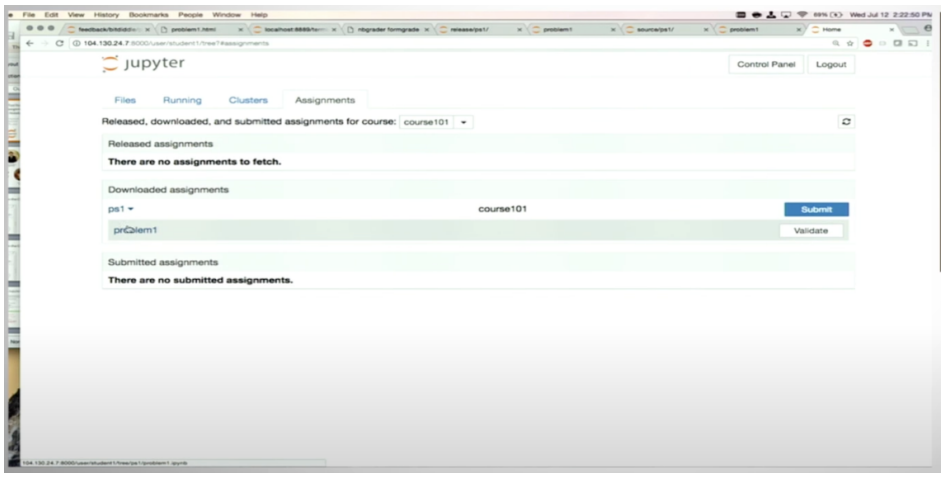

Figure 5: Selecting a notebook from the downloaded assignment(source)

In [ ]:

```
@ray.remote
class Mapper:
    def __init__(self):
        self.word_counts = {}

    def map(self, lines):
    # YOUR CODE HERE
    raise NotImplementedError()

    def get_counts(self):
        return self.word_counts
```

Figure 6: A code block to complete from Task 2. Only the `map` method is meant to be modified.