

DSC 204A: Scalable Data Systems

Programming Assignment 2

October 2025

1 Introduction

The goal of this programming assignment is to go further with **Ray**—focusing on multi-node data processing and collective communication. We will first have an easy introduction to the Ray Data API. We will then revisit the Ray Core API and introduce Ray Actors. Finally, we will build several communication algorithms through the Ray Collective Communications library.

2 Task 1: Ray Data (30 points)

In this question, you will work with **Ray Datasets** and apply basic transformation methods for distributed data processing. Ray Data is part of Ray AI Runtime and is built on top of Ray Core, inheriting the same scaling and parallelism benefits. First upgrade ray in Python 3 (ipykernel) in Jupyter, as instructed in `task1.ipynb`.

```
pip install "ray[default]" --upgrade
```

Note. Before working on the tasks, check Section 5 for Ray Dashboard access.

2.1 Problem

You are given a 100K subset of the **Amazon Reviews** dataset on DataHub¹. A starter notebook `task1.ipynb` is provided. Follow the instructions and linked documentation to complete the notebook.

Ray Data should *maximally utilize* CPUs on all workers, which you can validate via the *Ray Dashboard* (*Cluster* section). See Section 5 for instructions.

2.2 Grading Rubric

Submit a completed `task1.ipynb`. Correctness only.

Section	Points
All sections prior to “Cleaning Up reviewText”	20 (Auto graded)
“Cleaning Up reviewText”	10 (Auto graded)

Helpful `asserts` are included in the notebook to verify correctness.

3 Task 2: MapReduce with Ray Actors (30 points)

This task focuses on implementing a distributed **MapReduce** word count using **Ray Actors**.

¹Dataset path: `~/public/pa2`.

3.1 MapReduce

MapReduce breaks down large tasks into smaller units that can be distributed across multiple nodes. For reference, see the provided guide and example notebook (which includes a Ray *tasks*-based implementation at the end). A detailed overview of MapReduce will also appear in the future lectures.

3.2 Problem

Implement word-count via MapReduce using *Actors* in `task2.ipynb`. We provide a test file `essay.txt` (an LLM-related essay).

- **Parallelism:** Launch **Mappers** equal to the *number of workers/CPU*s in your Ray cluster and partition the text evenly.
- **Map stage:** Each Mapper processes its partition to produce local word counts.
- **Reduce stage:** A **Reducer** Actor aggregates partial counts from all Mappers.

Modify `task2.ipynb` to:

1. Define Mapper and Reducer as *Ray Actors*.
2. Partition input text across Mappers.
3. Compute local counts (map) and combine them (reduce).

3.3 Grading Rubric

Correctness only. **You must use Ray Actors and parallelize across `NUM_CPUS`**; otherwise, the submission is invalid.

Criterion	Points
Correct counts for public case <code>essay.txt</code>	20 (Auto graded)
Hidden test cases	10 (Auto graded)

4 Task 3: Collective Communication (40 points)

In this task, you will use point-to-point (P2P) APIs from `ray.util.collective` to implement the **AllReduce** collective in three ways. A `Worker` class template is provided; complete its functions so Actors can perform AllReduce among themselves. Profiling helpers are provided to compare implementations.

4.1 Problem

P2P Communication (Building Block)

Implement the **P2P** send/receive functions in the `Worker` class using Ray's built-in P2P API.

AllReduce Variants

Implement three AllReduce algorithms:

1. **ray_all_reduce:** A simple implementation leveraging Ray's built-ins.
2. **bde_all_reduce:** *Bidirectional Exchange (BDE)* AllReduce.

3. **mst_all_reduce:** *Minimum Spanning Tree (MST) AllReduce*. As building blocks, first implement **MST Reduce** and **Broadcast** collectives covered in class, then compose **MST AllReduce**.

For BDE and MST, follow the required reading and pseudocode. You may only rely on your implemented P2P operations and any helper functions you define.

4.2 Grading Rubric

Component	Points
Workers instantiation	5 (Manually graded)
P2P communication methods	5 (Auto graded)
RAY AllReduce	3 (Auto graded)
BDE AllReduce	10 (Auto graded)
MST AllReduce	15 (Auto graded)
Profiling results	2 (Manually graded)

Follow the prompts in `task3.ipynb` for each deliverable.

5 Dashboard Access

Instructions for Ray Dashboard are provided in the first code cell of `task2.ipynb`. Use the dashboard to confirm CPU utilization across workers for Task 1 and to monitor Actor placement and activity for Tasks 2–3.

Since Datahub doesn't have browser, you can't view the dashboard using the URL provided in the screenshot. If you want to check the dashboard when working on PA2, you can download the notebooks and the data in `~/public/pa2` to your local machine.

```


: import ray
import re
import time
import string
import json
import os

NUM_CPUS=8

ray.shutdown()
ray.init(include_dashboard=True)

```

2025-10-18 23:26:06,703 INFO worker.py:2004 -- Start /home/grader-dsc204a-02/.local/lib/python3.11/site-packages env var if num_gpus=0 or num_gpus=None (default) warnings.warn(



Python version: 3.11.9

Ray version: 2.50.1

Dashboard: <http://127.0.0.1:8266>

Disconnect

Submission

Submit the completed `task1.ipynb`, `task2.ipynb`, and `task3.ipynb`. Ensure notebooks run end-to-end on the target environment and that all required outputs and logs are present.