Carnegie Mellon University in Qatar

Distributed Systems

15-440 - Fall 2023

Project 4

Out: November 16, 2023

Due: November 30, 2023

Contents

1 Summary & Intended Learning Outcomes **3**

2 Project Objectives **3**

3 Implementation Guidelines **3**

4 Experimentation and Analysis **4**

5 Final Deliverables **5**

6 Rubric **6**

7 Late Policy **6**

1 Summary & Intended Learning Outcomes

Ray is an active open-source project developed at the University of California, Berkeley. Ray provides a general-purpose distributed computing framework for scaling Machine Learning models or workloads and writing parallel and distributed Python applications. It provides fundamental primitive abstractions that allow taking the existing program and turn it into a distributed one. It also allows developers to train and deploy models faster and more efficiently. Ray is currently enjoying wide popularity.

In this project, you will implement the *K-Means* clustering algorithm using Ray. You will compare and contrast the performance of your MPI *K-Means* implementation (from **P3**) against your Ray *K-Means* implementation from this project.

In short, the learning outcomes of this project are as follows:

1. Apply Ray to a popular real problem, namely cluster analysis using *K-Means* algorithm.
2. Compare and contrast your MPI and Ray implementations of *K-Means* in terms of performance and development effort.

2 Project Objectives

The overall goal of this project is to get a clear understanding on Ray parallelism and how different parallel implementations of the same algorithm compare against each other. You will conduct and analyze some scalability studies on various degrees of parallelism for *K-Means*. The project will provide students with: **(1)** deep insights into how parallelism affects performance in large-scale settings, and **(2)** a practical experience augmented with a methodology for solving clustering problems (and alike) on a distributed system using Ray.

3 Implementation Guidelines

In this project, you will provide a Ray implementation for *K-Means* with two types of data sets; a data set of data points and a data set of DNA strands (as was done in **P3**). Please use the datasets you generated in **P3** to run and test your Ray *K-means* implementation. For a complete explanation of the *K-Means* algorithm, please refer to the write-up of **P3**.

For this project, use the 4-VM virtual cluster (which can be deemed as a private cloud) provided to you in **P3**. We have already installed and tested Ray 2.8.0 for you. Hence, the clusters are ready to run your Ray code.

4 Experimentation and Analysis

After implementing *K-Means* using Ray, please conduct experiments and report on the following:

- A comparison between your 3 different *K-Means* implementations (the sequential and the MPI ones from **P3**, and the Ray one from this project) in terms of performance and development effort.
- Two scalability studies; similar to what you did in **P3**. In particular:
  - A scalability study on the number of VMs with a fixed data set size (use only the data set of the 2D data points for this study). Specifically, use 1, 2, 3 and 4 VMs.
  - A scalability study on the number of data points in your data set of the 2D data points with a fixed number of VMs (e.g., 4). Specifically, use 20 million, 30 million, and 40 million data points.
- A discussion on:
  - Your experience in applying Ray to the *K-Means* clustering algorithms.
  - Your insights concerning the performance trade-offs of MPI and Ray with *K-Means*.
  - Your thoughts on the applicability of *K-Means* to Ray.
  - Your recommendations regarding the usage of Ray for algorithms similar to *K-Means*.

5 Final Deliverables

As final deliverables, you should submit:

1. An archive containing a fully tested and debugged code for your Ray *K-Means* implementation. Specifically, you must submit **two programs for the *K-Means* implementation**:

   (1) `points_ray.py`

   (2) `dna_ray.py`

   They must follow the specifications below:

   - Input: (***in this order***)
     - The **Input file** containing the data
     - The **Output file** containing the data
     - Total number of **points/strands** provided
     - Number of **clusters**
     - Number of **iterations** (*if it's given as 0, your program should stop as necessary*)
     - **l**: length of DNA strand (*only applies for the DNA strands K-Means implementations*)

       * Sample program execution:

       ```
       // Runs K-Means on a dataset of 100 points, 3 clusters,
       // and 5 iterations
       $ python3 points_ray.py inputFile outputFile 100 3 5
       ```

   - Output: ***a file with the following information for each cluster*** c_i ***(all separated by new-lines)***:
     - The *final centroid value* for c_i
     - The *number of points/strands assigned* to the cluster c_i

       * Sample output for 2D points (the above, separated by commas on each line):

       ```
       // x-coordinate, y-coordinate, number of points
       5.212, 9.880, 400
       1.511, 3.201, 320
       ...
       ```

2. An **article** with a maximum of 5 pages (*similar to research articles*) that presents your solution, findings, observations and analysis.

6 Rubric

Ray K-Means (50 Points)

$\boxed{\text{25pts}}$
- K-Means using Ray for 2D data points
- K-Means using Ray for DNA strands.

  Details (*these are same for DNA and 2D data points versions*):

  - $\boxed{\text{1pt}}$ The code compiles and runs correctly
  - $\boxed{\text{6pts}}$ Create Ray Parallel Tasks Properly
  - $\boxed{\text{6pts}}$ The workers and the head nodes apply K-means to their portions of data (adopting a good mechanism to compute new centroids)
  - $\boxed{\text{6pts}}$ The head node receives intermediate data from workers and aggregates the results of the parallel tasks to calculate the new means
  - $\boxed{\text{3pts}}$ The head loops over for a new round
  - $\boxed{\text{3pts}}$ Outputting final results and aborting cleanly

Write-up (47.5 Points)

- $\boxed{\text{10pts}}$ Performance comparison between sequential, MPI, and Ray with a fixed dataset size
- $\boxed{\text{1.5pts}}$ Development effort comparison between sequential, MPI and Ray
- $\boxed{\text{15pts}}$ Ray scalability w.r.t 20, 30, 40 million datapoints
- $\boxed{\text{15pts}}$ Ray scalability w.r.t 1, 2, 3, 4 VMs
- $\boxed{\text{1.5pts}}$ Performance trade-offs between MPI and Ray
- $\boxed{\text{1.5pts}}$ Thoughts on the applicability of K-Means to Ray
- $\boxed{\text{1.5pts}}$ Recommendations regarding the usage of Ray for algorithms similar to K-Means
- $\boxed{\text{1.5pts}}$ Paper structure, level of writing, and language

Code Style (2.5 Points)

- Method Comments, Block comments, Readability, Dead code, Code Design

7 Late Policy

- If you hand in on time, there is no penalty.
- 0-24 hours late = 25% penalty.
- 24-48 hours late = 50% penalty.
- More than 48 hours late = you lose all the points for this project.

NOTE: You **CANNOT** use your grace-days quota. For details about the quota, please refer to the syllabus.