15-440
Distributed Systems

Recitation 11:
Project 4 & Ray Demo

Hend Gedawy
P4 Out. Due Nov. 30 (No Grace Days can be used)
P3 Due Today

Announcements
Project 4 Overview/Objective

• Implement the K-Means clustering algorithm using Ray. Two new code files:
  • points_ray.py
  • dna_ray.py

• You will compare and contrast the performance of your MPI K-Means implementation (from P3) against your Ray K-Means implementation from this project
  • Varying the number of data points
  • Varying the cluster size (number of workers/VMs)
Ray Custer

Head node
- Driver
- Worker
- Raylet
- Scheduler
- Object Store
- Global Control Store (GCS)

Worker Node (1)
- Worker
- Worker
- Raylet
- Scheduler
- Object Store

Worker Node (n)
- Worker
- Worker
- Raylet
- Scheduler
- Object Store
Running A Program on Ray Cluster

- Make sure ray is stopped in all nodes (sudo ray stop --force)
- Start Ray @ Head Node:
  - sudo ray start --head --port=6379 --redis-password=my_password --include-dashboard 1 --dashboard-host headNodeIP
- To include more worker machines:
  - Ssh to the worker node and start ray using the following command:
    - sudo ray start --address='headNodeIP:headPortNum'
- Run the program @Head Node:
  - sudo python3 points_ray.py <Program Parameters>
- To view the dashboard of your cluster, go to your web browser and put headNodeIP:dashboardPortNumber
- When Done, run (sudo ray stop --force) on all nodes

Default is 6379
Optional argument, you can give it if you’d like to select the port that you want

Given when head started
Ray Dashboard

It shows:
• all the machines that you have connected to this cluster (3 in this example)
• Information about them (uptime, CPU, RAM, )
Parallel Sum Using Ray
import time

if (__name__ == '__main__'):

    startTime = time.time()
    N = 1000
    total_sum=0
    for i in range(0, N):
        total_sum += i

    print("The sum is {0}\n".format(total_sum))
    print("Time ", time.time()-startTime)

How to turn this sequential Sum program into a parallel/distributed one using Ray?
K-Means Clustering With Ray – General Guidelines

• Identify the parts of the algorithm that you need to run in parallel or distribute (As we did in MPI)
• Put these parts in separate functions
• Turn these functions into Ray tasks using the `@ray.remote` decorator
• Every invocation to the function, creates a Ray task that can run in parallel and returns the result objectID as a future
• Wait for the set of futures in your spawned parallel tasks
• Aggregate the returned partial results