

15-440  
Distributed Systems  
**Recitation 11:**  
**Project 4 & Ray Demo**

Hend Gedawy



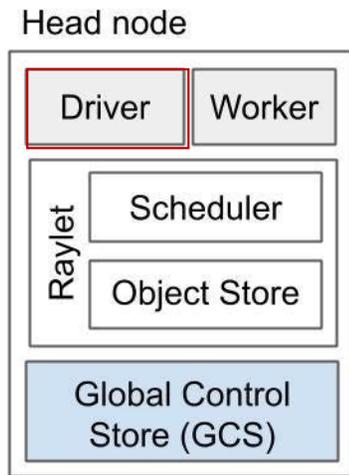
# Announcements

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

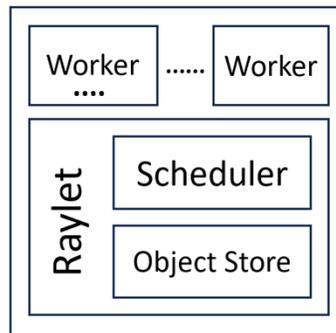
# Project 4 Overview/Objective

- 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)

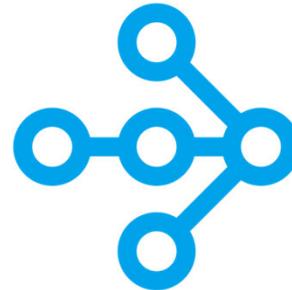
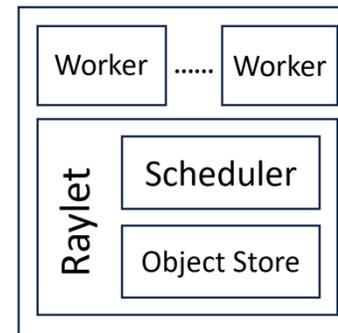
# Ray Cluster



Worker Node (1)



Worker Node (n)



# RAY

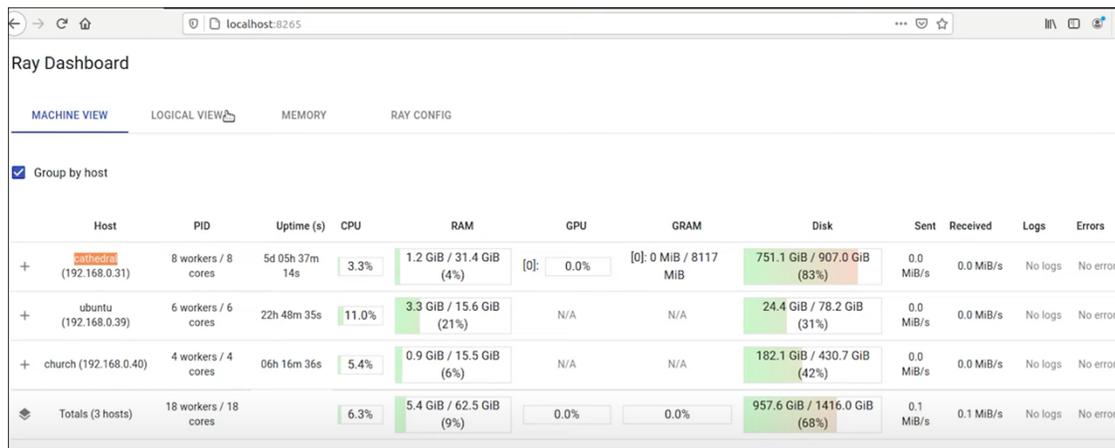
# Running A Program on Ray Cluster

Default is 6379

Optional argument, you can give it if you'd like to select the port that you want

- 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 ← Given when head started
- When Done, run (sudo ray stop --force) on all nodes

# Ray Dashboard



The screenshot shows the Ray Dashboard interface. At the top, there are tabs for 'MACHINE VIEW', 'LOGICAL VIEW', 'MEMORY', and 'RAY CONFIG'. Below the tabs, there is a checkbox labeled 'Group by host' which is checked. The main content is a table with the following columns: Host, PID, Uptime (s), CPU, RAM, GPU, GRAM, Disk, Sent, Received, Logs, and Errors. The table lists three worker nodes and a summary row for all three hosts.

Host	PID	Uptime (s)	CPU	RAM	GPU	GRAM	Disk	Sent	Received	Logs	Errors
+ ubuntu (192.168.0.31)	8 workers / 8 cores	5d 05h 37m 14s	3.3%	1.2 GiB / 31.4 GiB (4%)	[0]: 0.0%	[0]: 0 MiB / 8117 MiB	751.1 GiB / 907.0 GiB (83%)	0.0 MiB/s	0.0 MiB/s	No logs	No errors
+ ubuntu (192.168.0.39)	6 workers / 6 cores	22h 48m 35s	11.0%	3.3 GiB / 15.6 GiB (21%)	N/A	N/A	24.4 GiB / 78.2 GiB (31%)	0.0 MiB/s	0.0 MiB/s	No logs	No errors
+ church (192.168.0.40)	4 workers / 4 cores	06h 16m 36s	5.4%	0.9 GiB / 15.5 GiB (6%)	N/A	N/A	182.1 GiB / 430.7 GiB (42%)	0.0 MiB/s	0.0 MiB/s	No logs	No errors
⚙ Totals (3 hosts)	18 workers / 18 cores		6.3%	5.4 GiB / 62.5 GiB (9%)	0.0%	0.0%	957.6 GiB / 1416.0 GiB (68%)	0.1 MiB/s	0.1 MiB/s	No logs	No errors

## 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

# Sequential to 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



جامعة كارنيغي ميلون في قطر  
Carnegie Mellon University Qatar