UC San Diego

# DSC 102
# Systems for Scalable Analytics

Sai Sree Harsha

PA0 Discussion Session: Setting up AWS and Dask

## Miscellaneous

- Office hours
  - Regular OHs: Wed 10am to 11am
  - Extra in PA0 interval: 4pm to 6pm on Apr 20, Apr 21, Apr 24, Apr 25, Apr 27
  - Location: Open seating area near CSE 3230
- AWS link
  - https://ets-apps.ucsd.edu/individual/DSC102_SP23_A00/
- AWS credentials
  - https://ets-apps.ucsd.edu/individual/DSC102_SP23_A00?mode=env

## Agenda

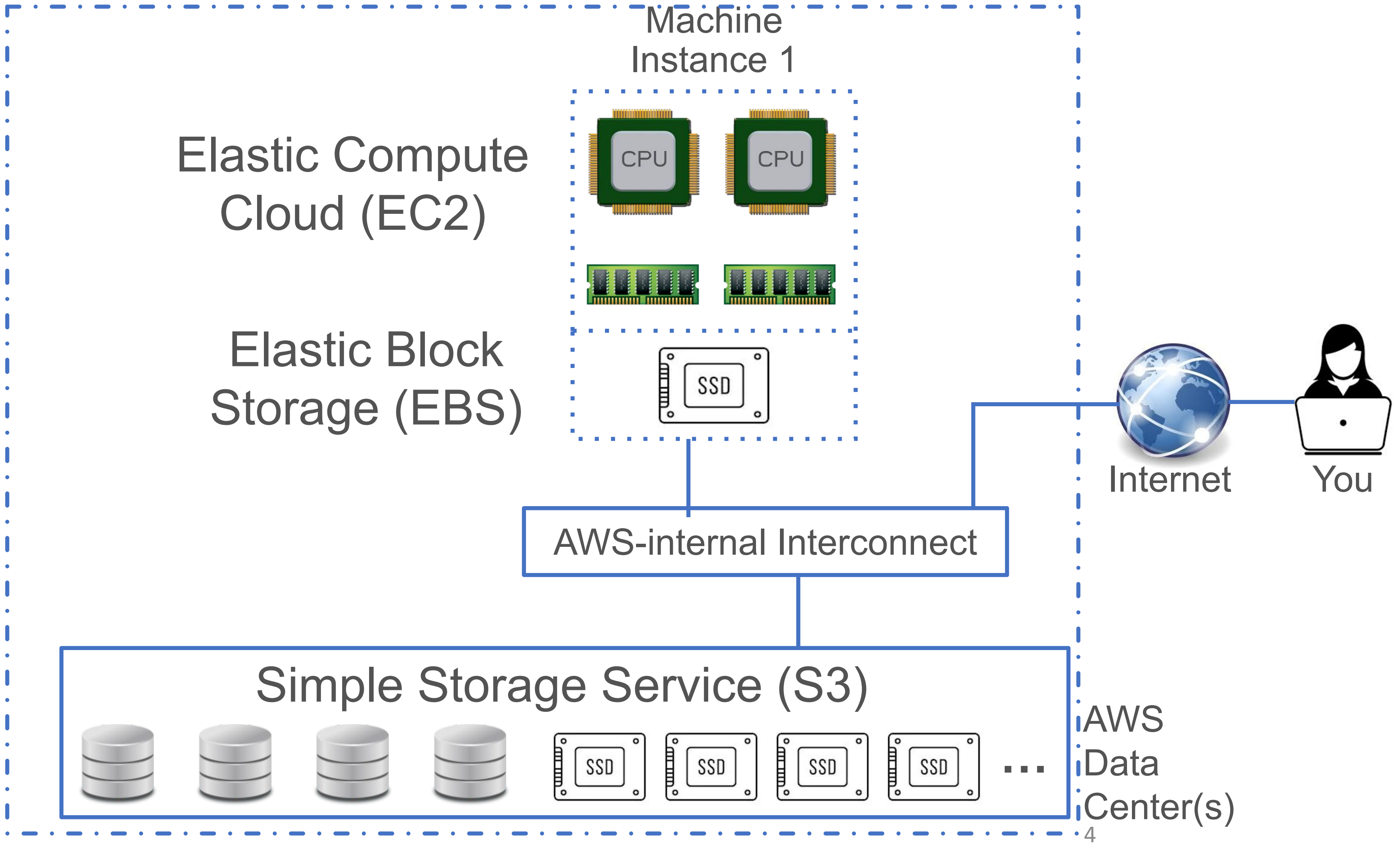
1. Fundamentals of Dask

2. Demo
   a. Setting up AWS
   b. Commonly used Dask functions

3. Assignment task and Grading scheme

4. Best Practices and tips for PA0

# AWS Services



Machine Instance 1
Elastic Compute Cloud (EC2)
CPU
CPU
Elastic Block Storage (EBS)
SSD
AWS-internal Interconnect
Internet
You
Simple Storage Service (S3)
SSD
SSD
SSD
SSD
...
AWS Data Center(s)

# Dask: Overview

- Parallel computing framework that scales existing Python ecosystems

```
np.zeros((10000,10000,10000)) -> OOM!
dask.array.zeros((10000,10000,10000))) -> SUCCESS!
```

- Breaks up work into tasks and executes them in task parallel manner
- Dask provides APIs (called collections) to create a task graph
- Dask also provides a scheduler that runs the task graph by assigning tasks to workers

# Dask: APIs

High-level APIs:

- Dask Array (Parallel NumPy)
- $\boxed{\text{Dask DataFrame (Parallel Pandas)}}$
- Dask Bag (Parallel Dictionary)
- Dask ML (Parallel Scikit-Learn)

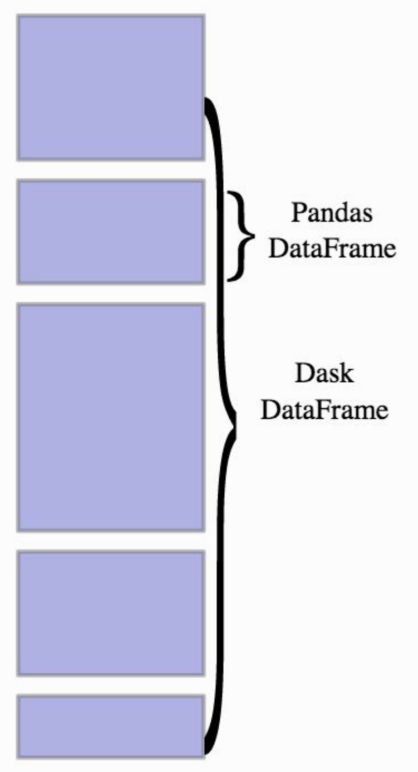
DataFrame APIs enough for this assignment, feel free to check out other APIs if needed

Low-level APIs:

- Dask Delayed (Parallel lazy objects)
- Dask Futures (Parallel eager objects)

# Dask: DataFrame API

- A Pandas DataFrame needs data to fit entirely in DRAM.
- A Dask DataFrame consists of multiple smaller Pandas DFs called “partitions”. These partitions reside on the disk.
- Operations on a Dask DF trigger operations on each partition (smaller Pandas DF) in a way that is mindful of potential parallelism and memory constraints.
- Dask handles staging of partitions between disk and DRAM.
- The number of partitions is often automatically determined based on available memory and the number of cores, but can also be manually specified.
- Each partition should fit comfortably in memory (DRAM).



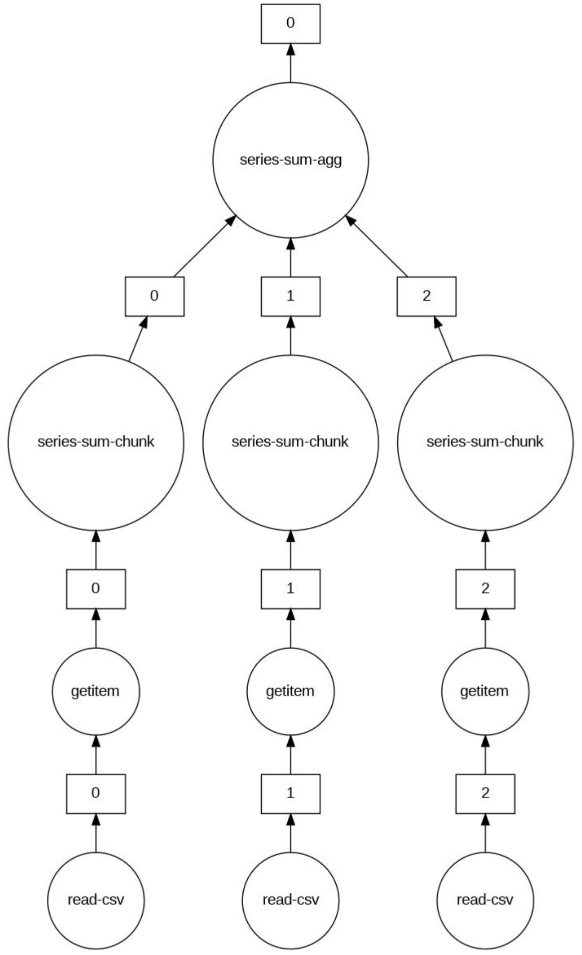
# Dask: DataFrame API

- Dask operations are evaluated ***lazily:*** Dask constructs the logic (called task graph) of the computation immediately but "evaluates" them only when necessary.
- Use *.compute()* method to trigger computation

```
import dask.dataframe as dd
df = dd.read_csv("my_huge_file.csv")
s = df.column.sum()

# visualize task graph
s.visualize()

# trigger computation to
# calculate sum of column
s.compute()
```



## AWS and Dask Demo

# Assignment: Dataset Description

*Amazon Reviews* table

| **Column name** | **Column description** | **Example** |
|---|---|---|
| reviewerID | ID of the reviewer | A32DT10X9WS4D0 |
| asin | ID of the product | B003VX9DJM |
| reviewerName | name of the reviewer | Slade |
| helpful | helpfulness rating of the review | [0, 0] |
| reviewText | text of the review | this was a gift for my friend who loves touch lamps. |
| overall | rating of the product | 1 |
| summary | summary of the review | broken piece |
| unixReviewTime | summary of the review | 1397174400 |
| reviewTime | time of the review (raw) | 04 11, 2014 |

# Assignment Task:

We will be using Dask library to explore secondary storage aware data access on a single machine.

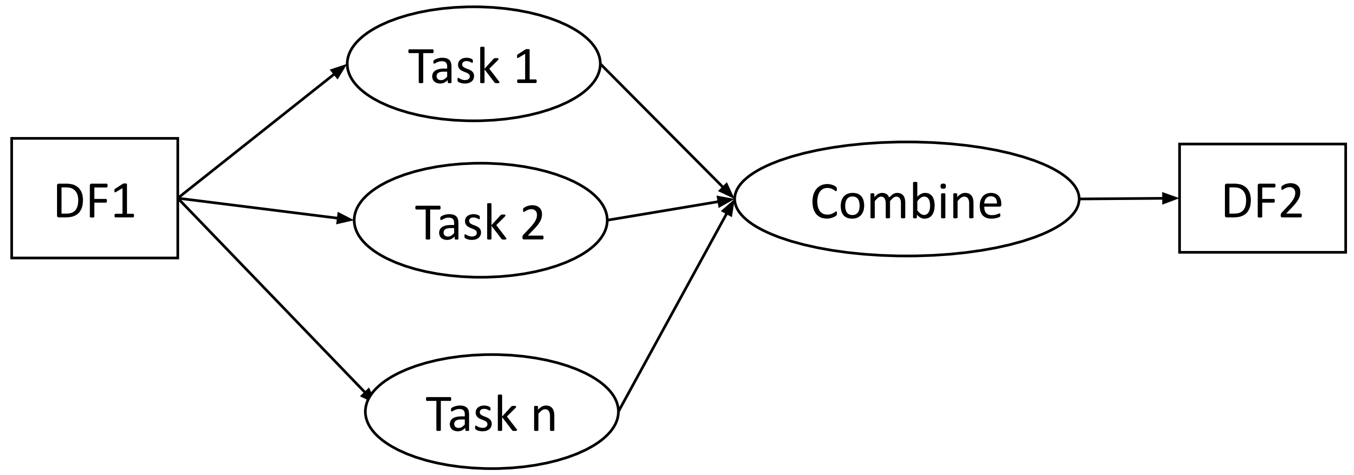
Create a new *users* table using only the *reviews* table with the following schema:

| **Column name** | **Column description** |
|---|---|
| reviewerID (PRIMARY KEY) | ID of the reviewer |
| number_products_rated | Total number of products rated by the reviewer |
| avg_ratings | Average rating given by the reviewer across all the reviewed products |
| reviewing_since | The year in which the user gave their first review |
| helpful_votes | Total number of helpful votes received for the users' reviews |
| total_votes | Total number of votes received for the users' reviews |

# Assignment Approach

Break up the “task” (creating a new *users* dataframe) into multiple sub-tasks (creating columns of the *users* dataframe)

Specify operations using Dask DataFrame APIs, which generates the task graph



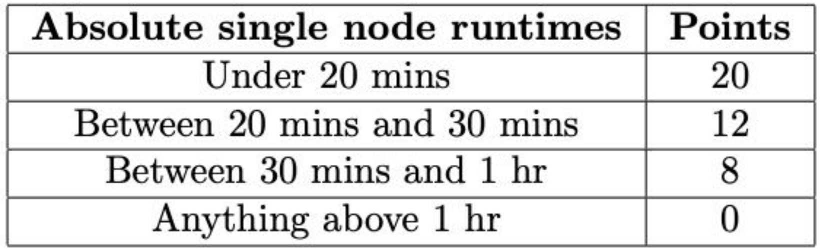
Given the task graph, Dask scheduler will take care of computing them in a task-parallel manner

# Grading Scheme

Accuracy(80)

- 5 columns
- If all the descriptive stats (mean, std dev, min, and max) rounded to 2 decimal points match the ground truth with a **1% error margin**, then **16 points awarded per column**

Runtime(20)



| **Absolute single node runtimes** | **Points** |
|---|---|
| Under 20 mins | 20 |
| Between 20 mins and 30 mins | 12 |
| Between 30 mins and 1 hr | 8 |
| Anything above 1 hr | 0 |

- Run function thrice and take average for getting the runtime measurement
- If accuracy points ≥ 40, runtime evaluation is automated (grading is based on above table)
- If accuracy points < 40, then partial credit based on manual inspection by TAs

## Files and Submission

All files necessary for the assignment are provided in the s3://dsc102-public bucket

- **user_reviews.csv** – Amazon reviews dataset
- **PA0.py** – function signature
- **results_PA0.json** – expected result on the user_reviews.csv dataset

Files used in this discussion session are provided in the s3://dsc102-discussion-demo bucket

- **demo_data.csv** – small subset of user_reviews.csv used for the demo in this discussion session
- **dask_demo_notebook.ipynb** – Jupyter notebook used for the demo in this discussion session

Submit your source code as <**YOUR-TEAM-NAME>.py** on Canvas.

Your source code must confirm to the function signatures provided to you.

Make sure that your code is writing results to **results_PA0.json**.

# Best Practices for PA0

- Use private GitHub repo if possible for handling code and logs.
- Terminate the AWS instance every time after usage; launch again & read from S3 again next time to save budget. (**Backup your code** at regular intervals/before terminating).
- Since the development data set is large, **work on a smaller subset first** (you can use the demo_data.csv (3.5 GB) and move to the full dataset user_reviews.csv (28.5 GB) later).
- Some helpful Dask APIs: groupby(), map_partitions(), str.split()
- While performing **groupby()** aggregations on large no. of groups (millions or more), use **split_out** to split output into multiple partitions to avoid memory error. (see this and this for tuning **split_out**)
- Call **.compute()** only once in your code (avoid computing intermediate dataframes).

## Other Helpful Links

- https://tutorial.dask.org/01_dataframe.html
- https://docs.dask.org/en/latest/dataframe-best-practices.html
- https://docs.dask.org/en/latest/dataframe-design.html
- https://examples.dask.org/dataframes/03-from-pandas-to-dask.html
- https://distributed.dask.org/en/latest/memory.html
- https://distributed.dask.org/en/latest/manage-computation.html
- https://docs.dask.org/en/latest/dataframe-indexing.html
- https://docs.dask.org/en/stable/generated/dask.dataframe.DataFrame.reset_index.html

Questions