Scaling up Interactive Data Analysis in Jupyter Lab: From Laptop to HPC

Peter Rose (pwrose@ucsd.edu)
Director, Structural Bioinformatics Lab

COMPUTING WITHOUT BOUNDARIES

EXPANSE

SAN DIEGO SUPERCOMPUTER CENTER

UNIVERSITY OF CALIFORNIA SAN DIEGO

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

Outline

- When to run on Expanse
- Setup a portable and reproducible software environment
- Run Jupyter Lab on Expanse
- Scale up calculations on CPU/GPU - Dataframes
- Measure parallel efficiency
- Get ready to use Expanse: accounts, allocations

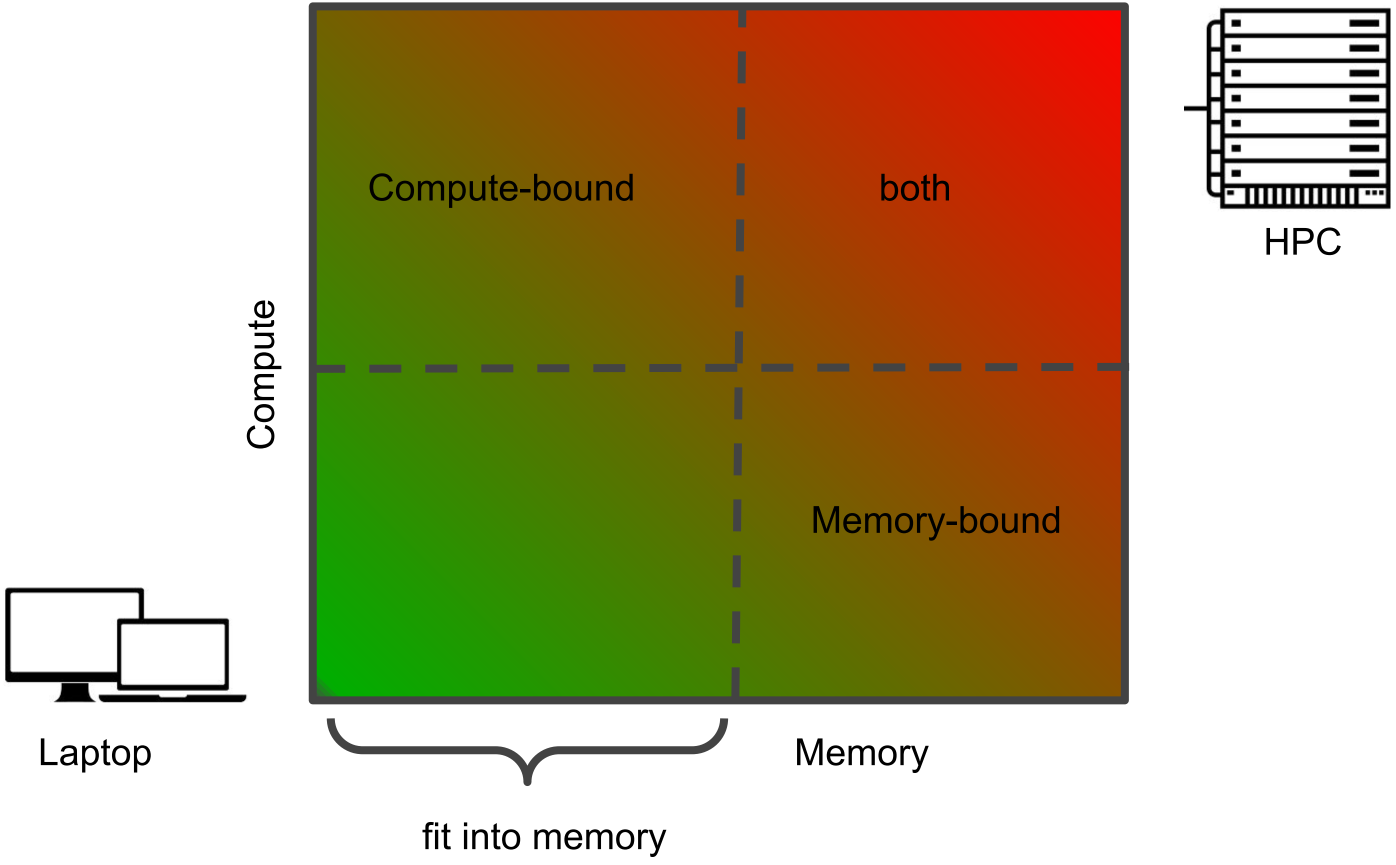
When to run on Expanse

Laptop/Desktop

- Coding
- Exploratory phase
- Small datasets
- Run on single or few cores

Expanse

- Scaling up to
  - large datasets
  - many datasets
  - long runtimes
- Run on many cores
- Run on GPU


Compute-bound
both
Memory-bound
Compute
Memory
fit into memory
Laptop
HPC

Setup a portable and reproducible software environment

From laptop to HPC

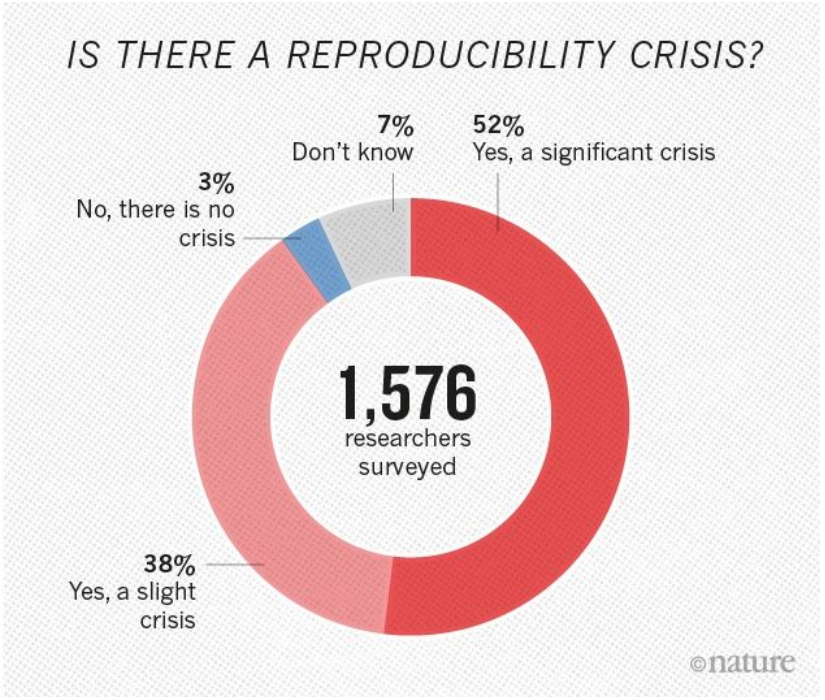
Reproducibility Crisis?

"More than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments."

Nature, 2016, M. Baker, 1,500 scientists lift the lid on reproducibility

"Nature journal editors … will, on a case-by-case basis, ask reviewers to check how well the code works."

Nature, 2018, Does your code stand up to scrutiny?



Reproducibility*	**Reusability**	**Scalability**
obtaining **consistent** results using	obtaining **new** results using	obtaining **new** results using
same input data or parameters	**different** input data or parameters	**large** input data or parameter sets
same computational steps, methods, and code	**same** computational steps, methods, and code	**same** computational steps, methods, and code
same analysis conditions	**same** analysis conditions	**same** analysis conditions

* L. Barba, https://figshare.com/articles/Next_in_Reproducibility_standards_policies_infrastructure_and_human_factors/8194328/1

Tools and Infrastructure

Jupyter — Computational notebooks: combine documentation, code, and results

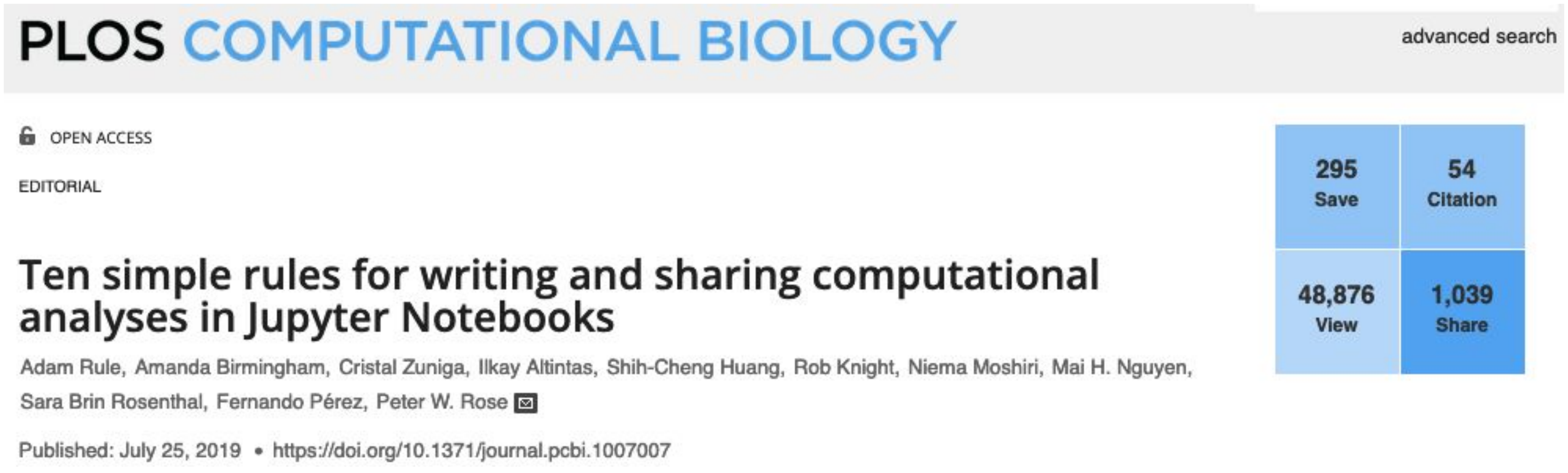
git — Version-control system for tracking changes in source code

GitHub — Source code repository

CONDA — Open-source package and environment management system

SINGULARITY — Container that packages software and OS in a portable way

EXPANSE — Scalable compute infrastructure



PLOS COMPUTATIONAL BIOLOGY

advanced search

OPEN ACCESS

EDITORIAL

Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

Adam Rule, Amanda Birmingham, Cristal Zuniga, Ilkay Altintas, Shih-Cheng Huang, Rob Knight, Niema Moshiri, Mai H. Nguyen, Sara Brin Rosenthal, Fernando Pérez, Peter W. Rose

Published: July 25, 2019 • https://doi.org/10.1371/journal.pcbi.1007007

295 Save	54 Citation
48,876 View	1,039 Share

Paper: https://doi.org/10.1371/journal.pcbi.1007007

Git repo: https://github.com/jupyter-guide/ten-rules-jupyter

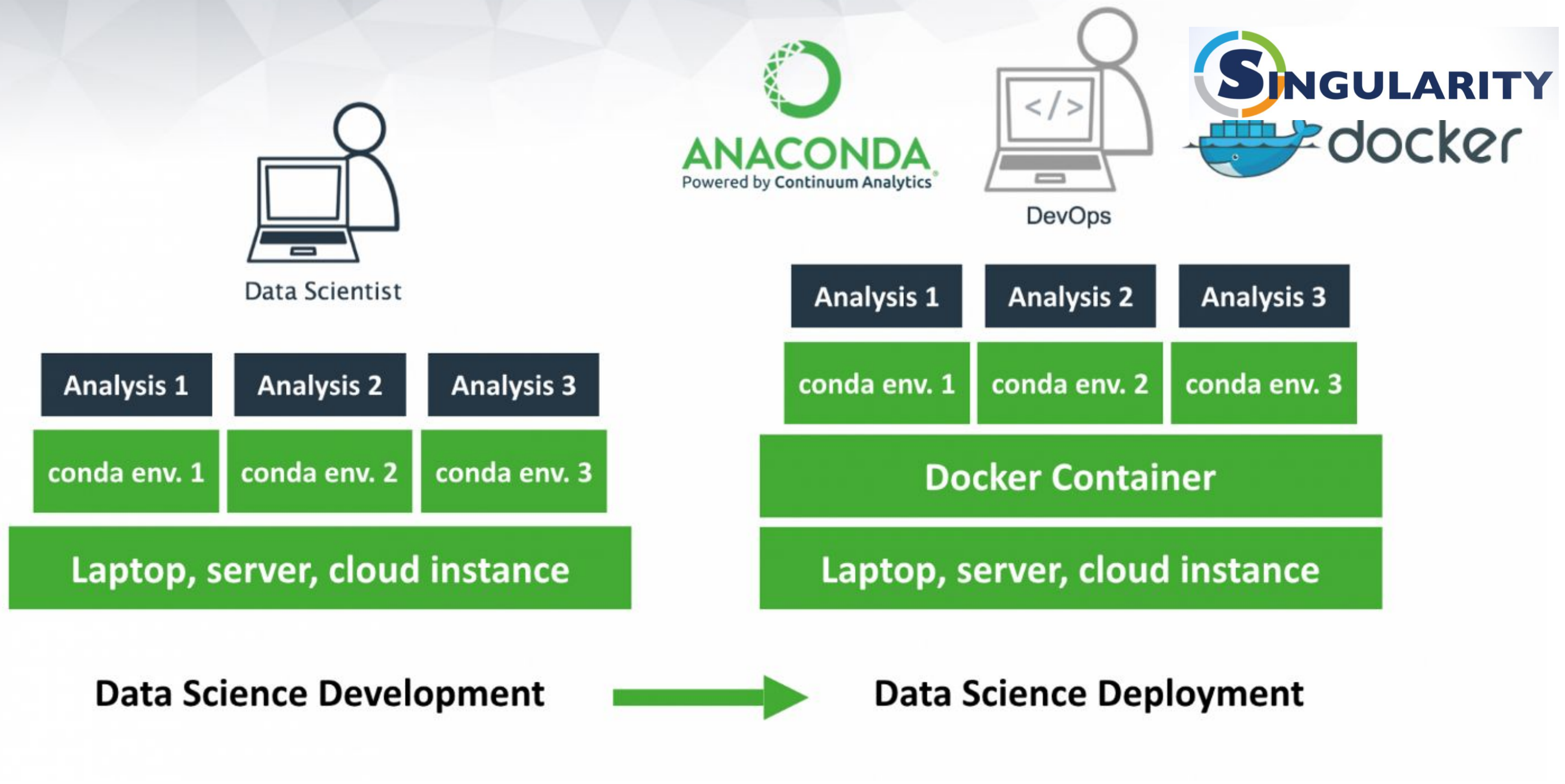
Reproducible Environments

CONDA

- Beginner
- Experience with Conda
- Exploratory development
- Frequently changing dependencies
- Easy to compose an environment with many dependencies
- Run on single node on Expanse
- Supported on Linux, Mac, Windows
- Run on native OS

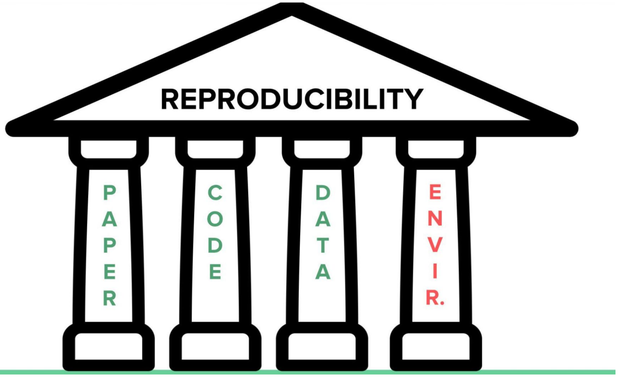
SINGULARITY

- Advanced user
- Experience with containers
- Production environment
- Often setup for a single tool
- Optimized containers for Expanse
  - pytorch, tensorflow, …
  - support for multi-node
- Supported on Linux
- Mac, Windows requires a VM
- Run on packaged OS, e.g. Ubuntu



Source: https://medium.com/@patrickmichelberger/getting-started-with-anaconda-docker-b50a2c482139

CONDA®



- **Package management system**
  - Conda installs, runs, and updates open source packages (e.g., NumPy, Pandas) and their dependencies, while checking compatibility with all preexisting packages.
- **Environment management system**
  - Conda allow you to create, save, load and switch between multiple environments on your local computer, as well as share instructions for how to recreate that environment on a different computer.
- **Multi-platform (Windows, MacOS, and Linux)**
- **Multi-language (Python, R, Ruby, Scala, Java, JavaScript, C/ C++, etc.)**

Why Conda Environments?

```
pip install pandas
pip install scikit-learn

Or

conda install pandas
conda install scikit-learn
```

Directly installing packages into your base environment will lead to version conflicts, errors, and non-reproducible results.

```
environment_1

python=3.7
pandas=0.25.0
scikit-learn=0.20.0
```

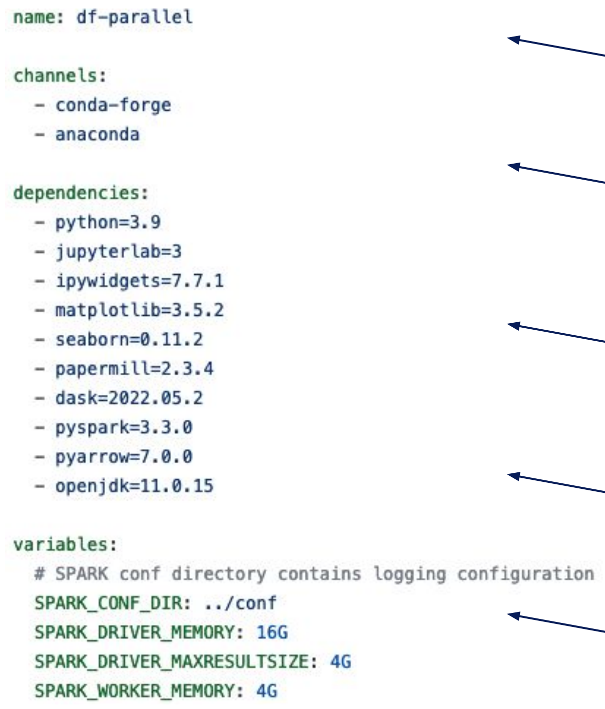
```
environment_2

python=3.9
pandas=1.2.4
scikit-learn=0.24.2
```

By creating conda environments, multiple versions of software packages can co-exists without interference.

Conda environment are portable and can be installed on multiple platforms.

Define a Conda Environments

Create an **environment.yml** file in the top level of a Git Repository (https://github.com/sbl-sdsc/df-parallel)



```
name: df-parallel

channels:
  - conda-forge
  - anaconda

dependencies:
  - python=3.9
  - jupyterlab=3
  - ipywidgets=7.7.1
  - matplotlib=3.5.2
  - seaborn=0.11.2
  - papermill=2.3.4
  - dask=2022.05.2
  - pyspark=3.3.0
  - pyarrow=7.0.0
  - openjdk=11.0.15

variables:
  # SPARK conf directory contains logging configuration
  SPARK_CONF_DIR: ../conf
  SPARK_DRIVER_MEMORY: 16G
  SPARK_DRIVER_MAXRESULTSIZE: 4G
  SPARK_WORKER_MEMORY: 4G
```

Use the same name as your Git repository

Specify the channels where to look for packages. Order matters! The conda-forge channel has newer versions than anaconda.

Specify (**"pin"**) version number to ensure reproducibility and compatibility.

Specify non-Python packages (e.g., Java).

Set environment variables (e.g., configuration options).

Create a Conda Environment

Prerequisite: Miniconda3 (light-weight, preferred) or Anaconda3 installed

https://docs.conda.io/en/latest/miniconda.html

Create a Conda environment

```
conda env create -f environment.yml
    or
mamba env create -f environment.yml (faster)
```

Activate a Conda environment

```
conda activate <environment_name>
```

Run Jupyter Lab

```
jupyter lab
```

Deactivate conda environment

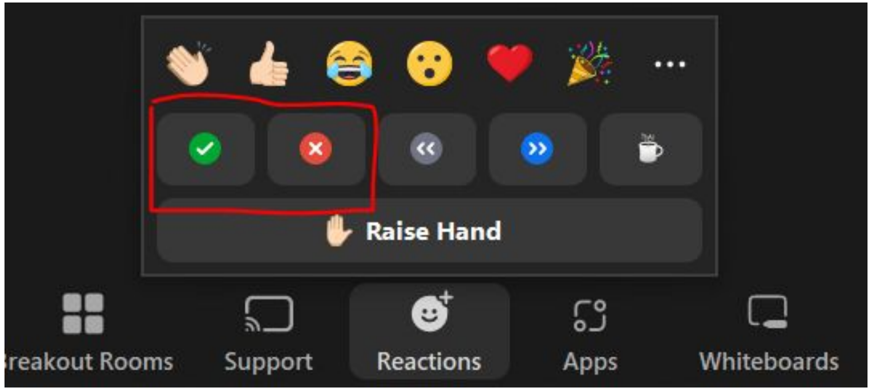
```
conda deactivate
```

✓ Mac, Windows, Linux

✗ Expanse: **Do not** create a Conda environment in your home directory (network file system)
-> Use the **galyleo** script!

Run Jupyter Lab on Expanse

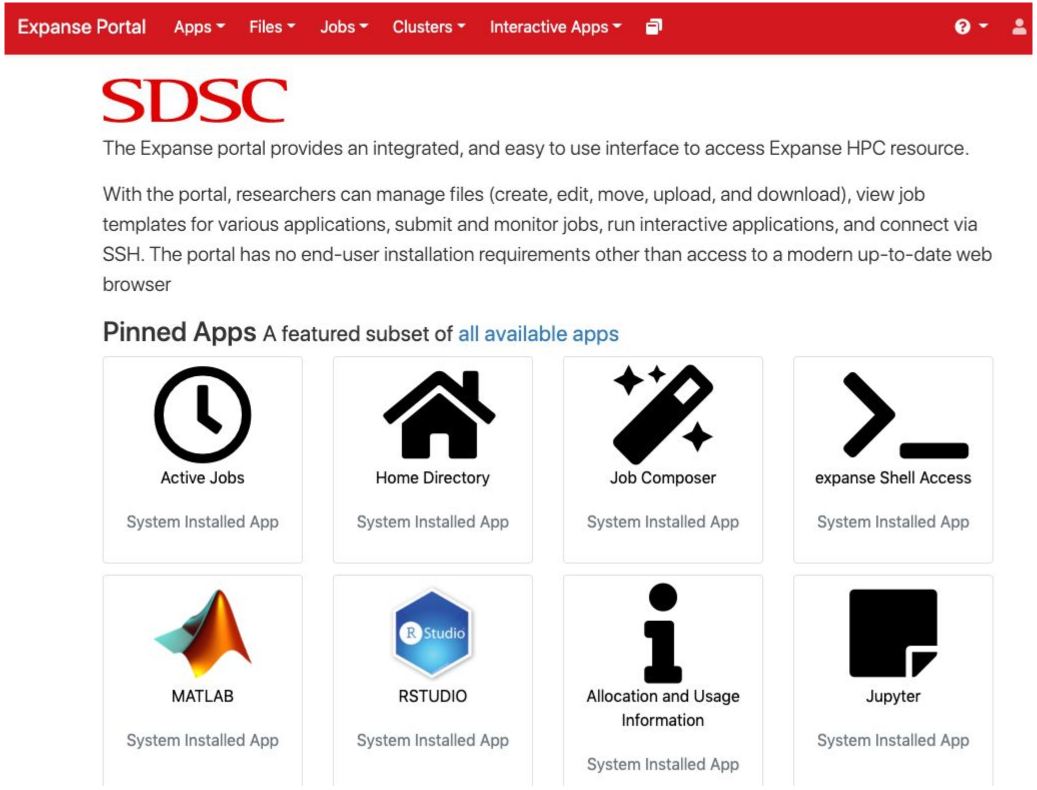
Feedback using Zoom Reactions



We will use Reactions to get feedback during the hands-on exercises

- Yes ✓ I've successfully completed the task
- No ✗ I have a problem (go to Slack and describe your problem or raise your hand)

Expanse Portal



SDSC

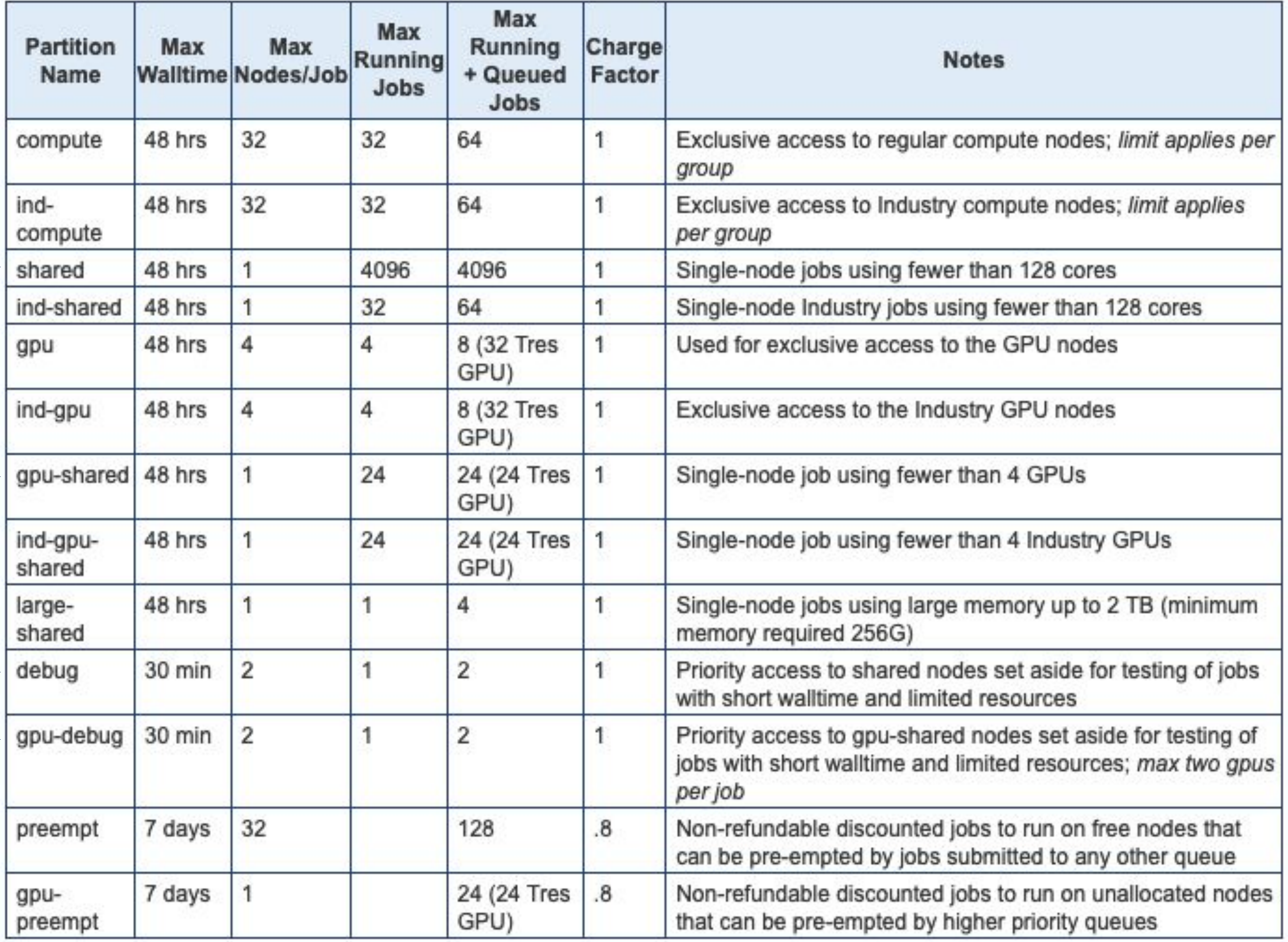
The Expanse portal provides an integrated, and easy to use interface to access Expanse HPC resource.

With the portal, researchers can manage files (create, edit, move, upload, and download), view job templates for various applications, submit and monitor jobs, run interactive applications, and connect via SSH. The portal has no end-user installation requirements other than access to a modern up-to-date web browser

Pinned Apps A featured subset of all available apps

Active Jobs — System Installed App
Home Directory — System Installed App
Job Composer — System Installed App
expanse Shell Access — System Installed App
MATLAB — System Installed App
RSTUDIO — System Installed App
Allocation and Usage Information — System Installed App
Jupyter — System Installed App

- Login with your XSEDE credentials (trainxx): https://portal.expanse.sdsc.edu/
- Open a terminal window



	Partition Name	Max Walltime	Max Nodes/Job	Max Running Jobs	Max Running + Queued Jobs	Charge Factor	Notes
	compute	48 hrs	32	32	64	1	Exclusive access to regular compute nodes; *limit applies per group*
	ind-compute	48 hrs	32	32	64	1	Exclusive access to Industry compute nodes; *limit applies per group*
Jupyter Notebook (CPU) →	shared	48 hrs	1	4096	4096	1	Single-node jobs using fewer than 128 cores
	ind-shared	48 hrs	1	32	64	1	Single-node Industry jobs using fewer than 128 cores
	gpu	48 hrs	4	4	8 (32 Tres GPU)	1	Used for exclusive access to the GPU nodes
	ind-gpu	48 hrs	4	4	8 (32 Tres GPU)	1	Exclusive access to the Industry GPU nodes
Jupyter Notebook (GPU) →	gpu-shared	48 hrs	1	24	24 (24 Tres GPU)	1	Single-node job using fewer than 4 GPUs
	ind-gpu-shared	48 hrs	1	24	24 (24 Tres GPU)	1	Single-node job using fewer than 4 Industry GPUs
	large-shared	48 hrs	1	1	4	1	Single-node jobs using large memory up to 2 TB (minimum memory required 256G)
Testing (CPU) →	debug	30 min	2	1	2	1	Priority access to shared nodes set aside for testing of jobs with short walltime and limited resources
Testing (GPU) →	gpu-debug	30 min	2	1	2	1	Priority access to gpu-shared nodes set aside for testing of jobs with short walltime and limited resources; *max two gpus per job*
	preempt	7 days	32		128	.8	Non-refundable discounted jobs to run on free nodes that can be pre-empted by jobs submitted to any other queue
	gpu-preempt	7 days	1		24 (24 Tres GPU)	.8	Non-refundable discounted jobs to run on unallocated nodes that can be pre-empted by higher priority queues

Example using Galyleo

Prepend path to galyleo to your path (e.g., append to ~/`.bash_profile` file)

```
export PATH="/cm/shared/apps/sdsc/galyleo:${PATH}"
```

1. Clone the Git repository

```
https://github.com/sbl-sdsc/df-parallel.git
```

2. Launch your Jupyter Notebook session using a Conda environment.yml file

```
galyleo launch --account <account_number> --partition shared --cpus 10
--memory 20 --time-limit 00:30:00 --conda-env df-parallel --conda-yml
"${HOME}/df-parallel/environment.yml"  --mamba
```

3. Copy and paste generated URL into your web browser

```
https://anchovy-passion-placidly.expanse-user-content.sdsc.edu?token
=48ee984b9ea07a96c17aaec000bc5fcf
```

How to Create and Use a Packed Conda Environment

Galyleo creates a Conda environment on the fly. If you use an environment often, you can save a packed Conda environment.

Create packed Conda Environment (**`df-parallel.tar.gz`**)

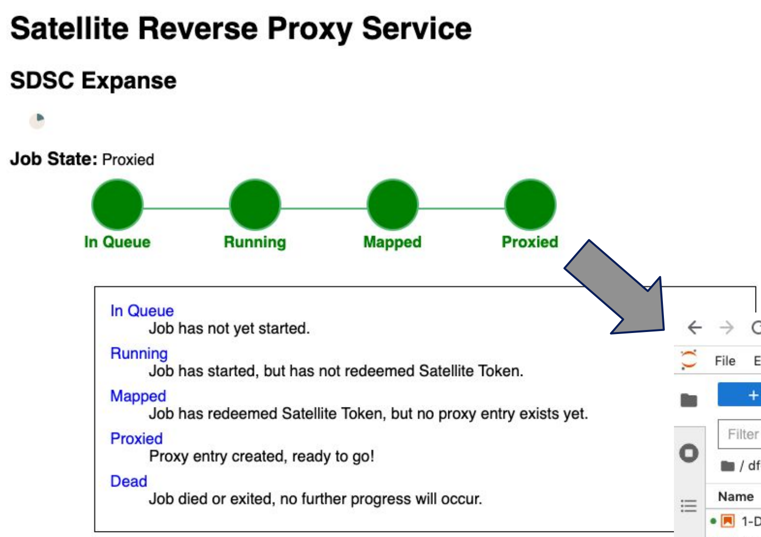
```
./df-parallel/pack.sh --account <account_number> --conda-env df-parallel
--conda-yml "${HOME}/df-parallel/environment.yml"
```

Use packed Conda Environment

```
galyleo launch --account <account_number> --partition shared --cpus 8 --memory 16
--time-limit 00:30:00 --conda-env df-parallel
--conda-pack "${HOME}/df-parallel.tar.gz"
```

More details about Conda environment: https://github.com/mkandes/galyleo#conda-environments

Progress Bar and Jupyter Launch

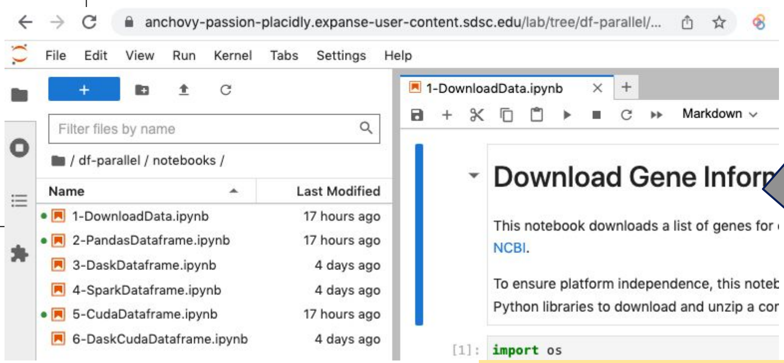


Satellite Reverse Proxy Service

SDSC Expanse

Job State: Proxied

In Queue
Job has not yet started.
Running
Job has started, but has not redeemed Satellite Token.
Mapped
Job has redeemed Satellite Token, but no proxy entry exists yet.
Proxied
Proxy entry created, ready to go!
Dead
Job died or exited, no further progress will occur.



anchovy-passion-placidly.expanse-user-content.sdsc.edu/lab/tree/df-parallel/...

File Edit View Run Kernel Tabs Settings Help
Filter files by name
/ df-parallel / notebooks /
1-DownloadData.ipynb
Markdown

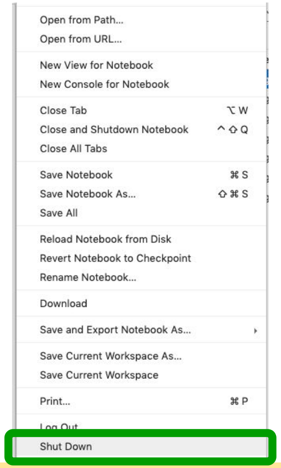
Name	Last Modified
1-DownloadData.ipynb	17 hours ago
2-PandasDataframe.ipynb	17 hours ago
3-DaskDataframe.ipynb	4 days ago
4-SparkDataframe.ipynb	4 days ago
5-CudaDataframe.ipynb	17 hours ago
6-DaskCudaDataframe.ipynb	4 days ago

Download Gene Infor

This notebook downloads a list of genes for
NCBI.

To ensure platform independence, this noteb
Python libraries to download and unzip a cor

```
[1]: import os
```



File-> Shut Down to terminate process, or job continues to use up your allocation!

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

Running the Dataframe Examples

Clone the Git repo

```
git clone https://github.com/sbl-sdsc/df-parallel.git
```

Run on CPU (Pandas, Dask, Spark dataframes)

```
galyleo launch --account <account_number> --partition shared --cpus 10
--memory 20 --time-limit 01:00:00 --conda-env df-parallel
--conda-yml "${HOME}/df-parallel/environment.yml" --mamba
```

Run on GPU (Pandas, Dask, Spark, cuDF, Dask-cuDF dataframes)

```
galyleo launch --account <account_number> --partition gpu-shared --cpus 10
--memory 92 --gpus 1 --time-limit 01:00:00 --conda-env df-parallel-gpu
--conda-yml "${HOME}/df-parallel/environment-gpu.yml" --mamba
```

Task 1

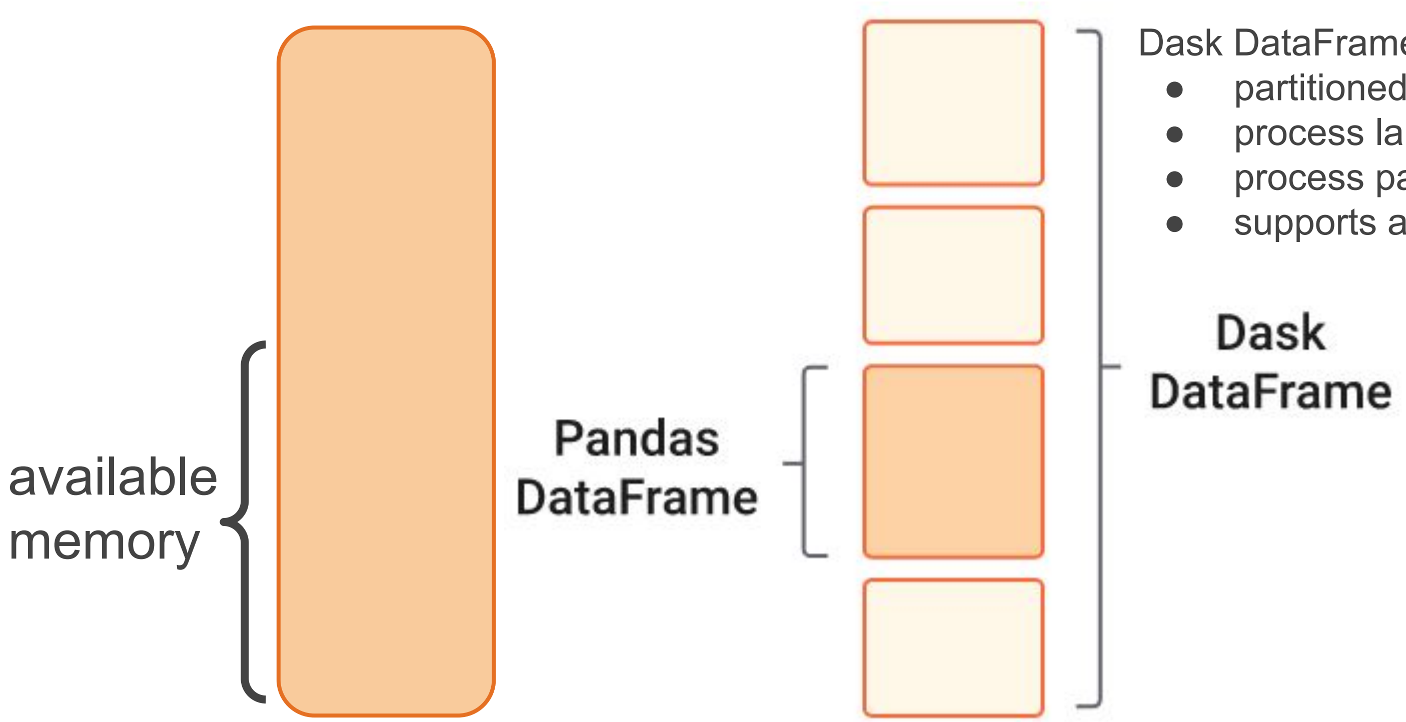
- Launch Jupyter Lab using Galyleo script

- Follow the instructions in section 6.2: Task 1
  https://github.com/sdsc/sdsc-summer-institute-2022/tree/main/6.2_scaling_up_interactive_data_analysis_jupyter_lab#task-1-launch-jupyter-lab-on-expanse-using-a-conda-environment

Scale up calculations on CPU/GPU

Example: Data Analysis with Dataframes

Processing large Datasets on CPU



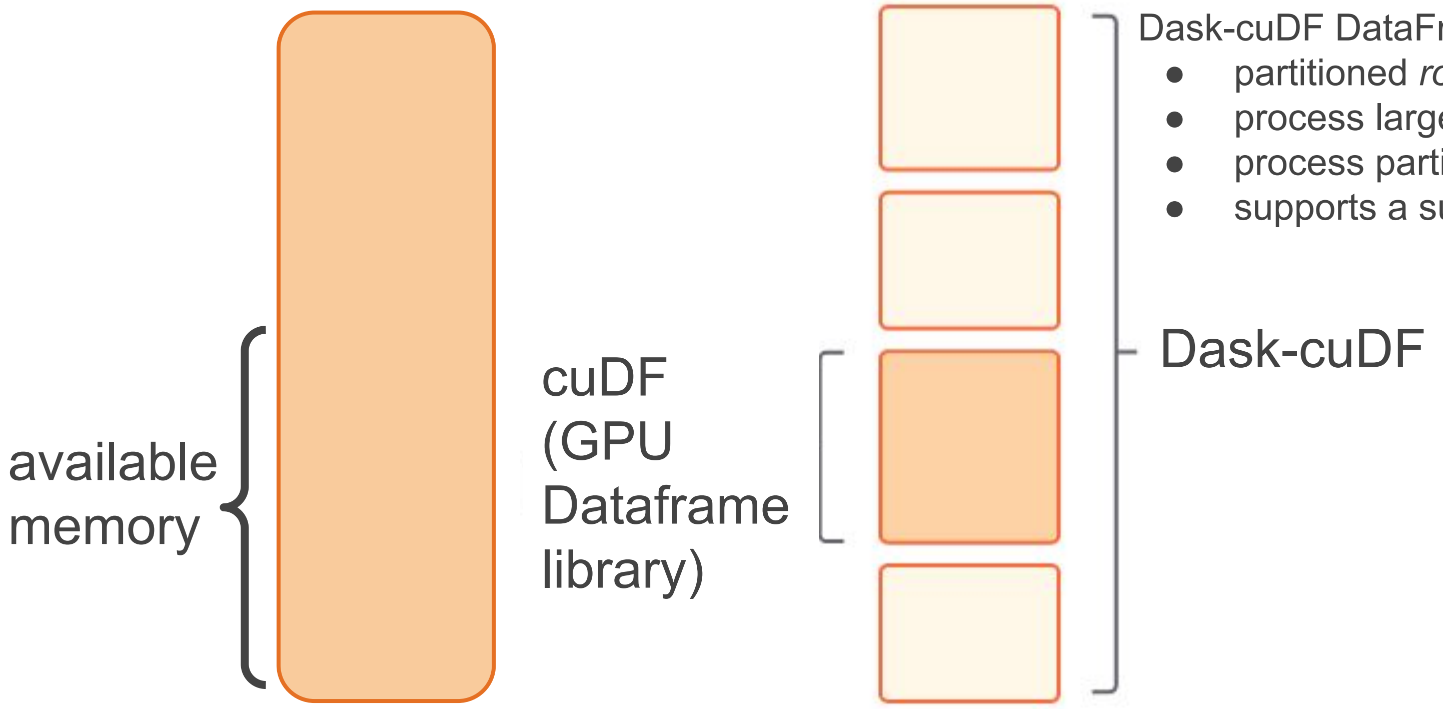
Dask DataFrame

- partitioned *row-wise*
- process large (out of core) datasets
- process partitions in parallel on CPU
- supports a subset of the Pandas API

https://pandas.pydata.org/docs/index.html

https://docs.dask.org/en/stable/dataframe.html

Processing large Datasets on GPU



Dask-cuDF DataFrame

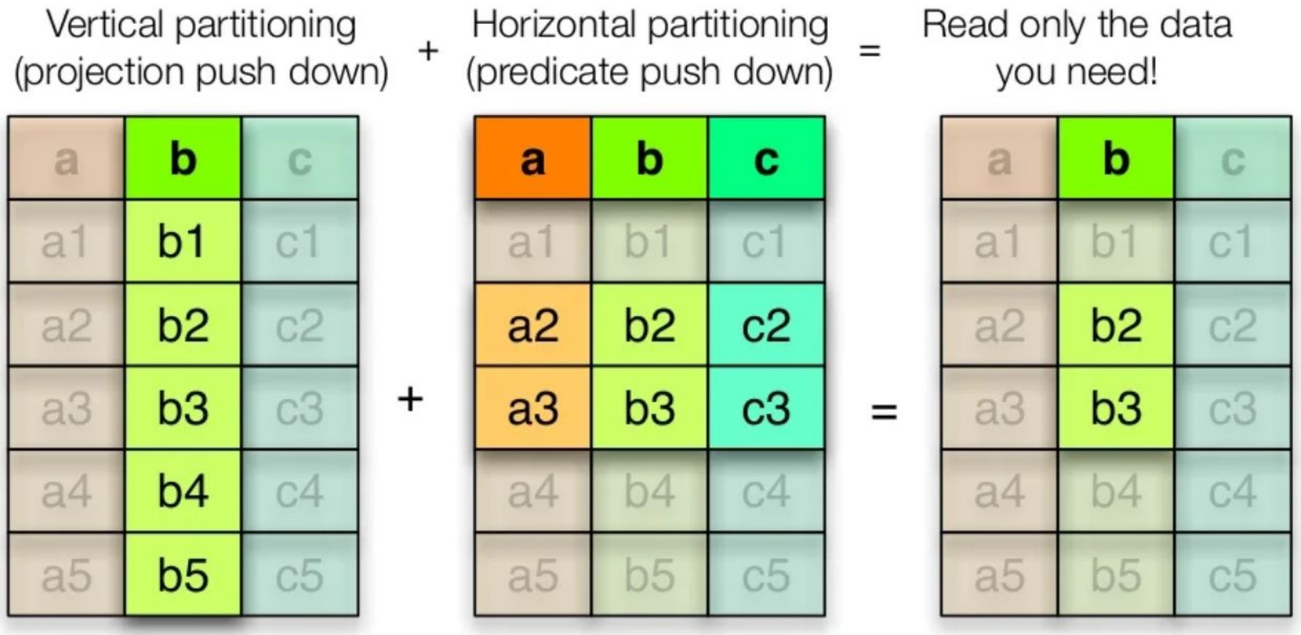
- partitioned *row-wise*
- process large (out of core) datasets
- process partitions on GPU
- supports a subset of the Pandas API

https://docs.rapids.ai/api

Columnar Storage Format - Parquet

Query and space efficient file format (default: Snappy compression)

@EmrgencyKittens



Vertical partitioning (projection push down)
+
Horizontal partitioning (predicate push down)
=
Read only the data you need!
a b c
a1 b1 c1
a2 b2 c2
a3 b3 c3
a4 b4 c4
a5 b5 c5
+
=

Horizontal partitioning uses column min/max statistics from Parquet metadata.

https://www.slideshare.net/julienledem/if-you-have-your-own-columnar-format-stop-now-and-use-parquet

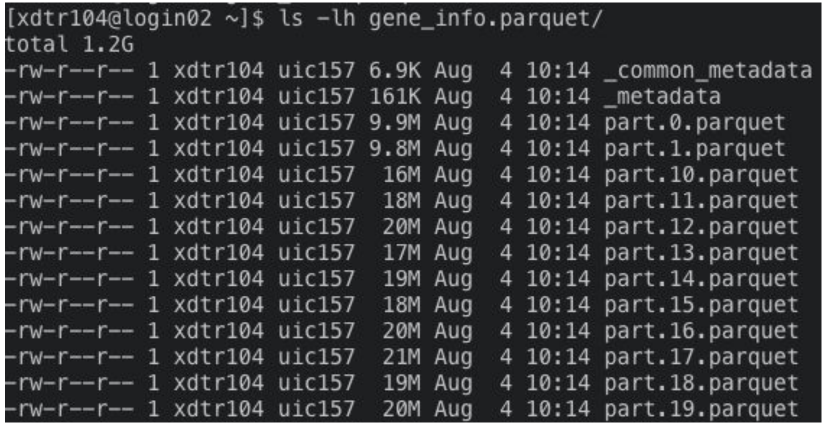
Vertical & Horizontal Partitioning

```
column_names = ["GeneID", "Symbol", "Synonyms", "description", "type_of_gene", "#tax_id", "chromosome"]
filters=[[("type_of_gene", "==", "protein-coding")]]

# Parquet
genes = pd.read_parquet(filename, columns=column_names, filters=filters)
# Dask
genes = dd.read_parquet(filename, columns=column_names, filters=filters)
# cuDF
genes = cudf.read_parquet(filename, columns=column_names, filters=filters)

# Dask-cuDF (filter does not work or is not fully implemented)
genes = dask_cudf.read_parquet(filename, columns=column_names)
genes = genes[genes["type_of_gene"] == "protein-coding"]

# Spark
genes = spark.read.parquet(filename)
genes = genes.select(column_names)
genes = genes.filter("type_of_gene == 'protein-coding'")
```

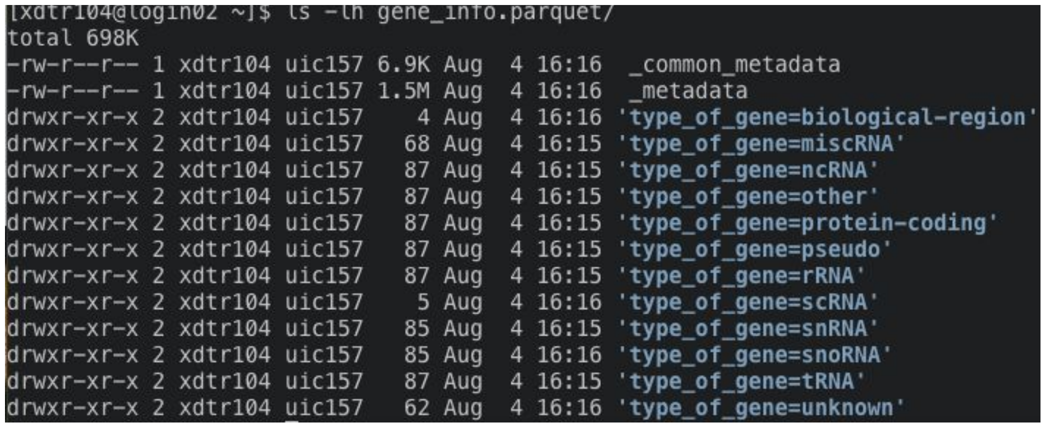
Creating Parquet "Files" with Dask

```
genes = dd.read_csv(input, dtype=str, sep="\t")

genes.to_parquet(output, write_index=False,
write_metadata_file=True, engine="pyarrow")
```

```
genes = dd.read_csv(input, dtype=str, sep="\t")

genes.to_parquet(output, write_index=False,
write_metadata_file=True, engine="pyarrow",
partition_on=["type_of_gene"])
```



```
[xdtr104@login02 ~]$ ls -lh gene_info.parquet/
total 1.2G
-rw-r--r-- 1 xdtr104 uic157 6.9K Aug  4 10:14 _common_metadata
-rw-r--r-- 1 xdtr104 uic157 161K Aug  4 10:14 _metadata
-rw-r--r-- 1 xdtr104 uic157 9.9M Aug  4 10:14 part.0.parquet
-rw-r--r-- 1 xdtr104 uic157 9.8M Aug  4 10:14 part.1.parquet
-rw-r--r-- 1 xdtr104 uic157  16M Aug  4 10:14 part.10.parquet
-rw-r--r-- 1 xdtr104 uic157  18M Aug  4 10:14 part.11.parquet
-rw-r--r-- 1 xdtr104 uic157  20M Aug  4 10:14 part.12.parquet
-rw-r--r-- 1 xdtr104 uic157  17M Aug  4 10:14 part.13.parquet
-rw-r--r-- 1 xdtr104 uic157  19M Aug  4 10:14 part.14.parquet
-rw-r--r-- 1 xdtr104 uic157  18M Aug  4 10:14 part.15.parquet
-rw-r--r-- 1 xdtr104 uic157  20M Aug  4 10:14 part.16.parquet
-rw-r--r-- 1 xdtr104 uic157  21M Aug  4 10:14 part.17.parquet
-rw-r--r-- 1 xdtr104 uic157  19M Aug  4 10:14 part.18.parquet
-rw-r--r-- 1 xdtr104 uic157  20M Aug  4 10:14 part.19.parquet
```



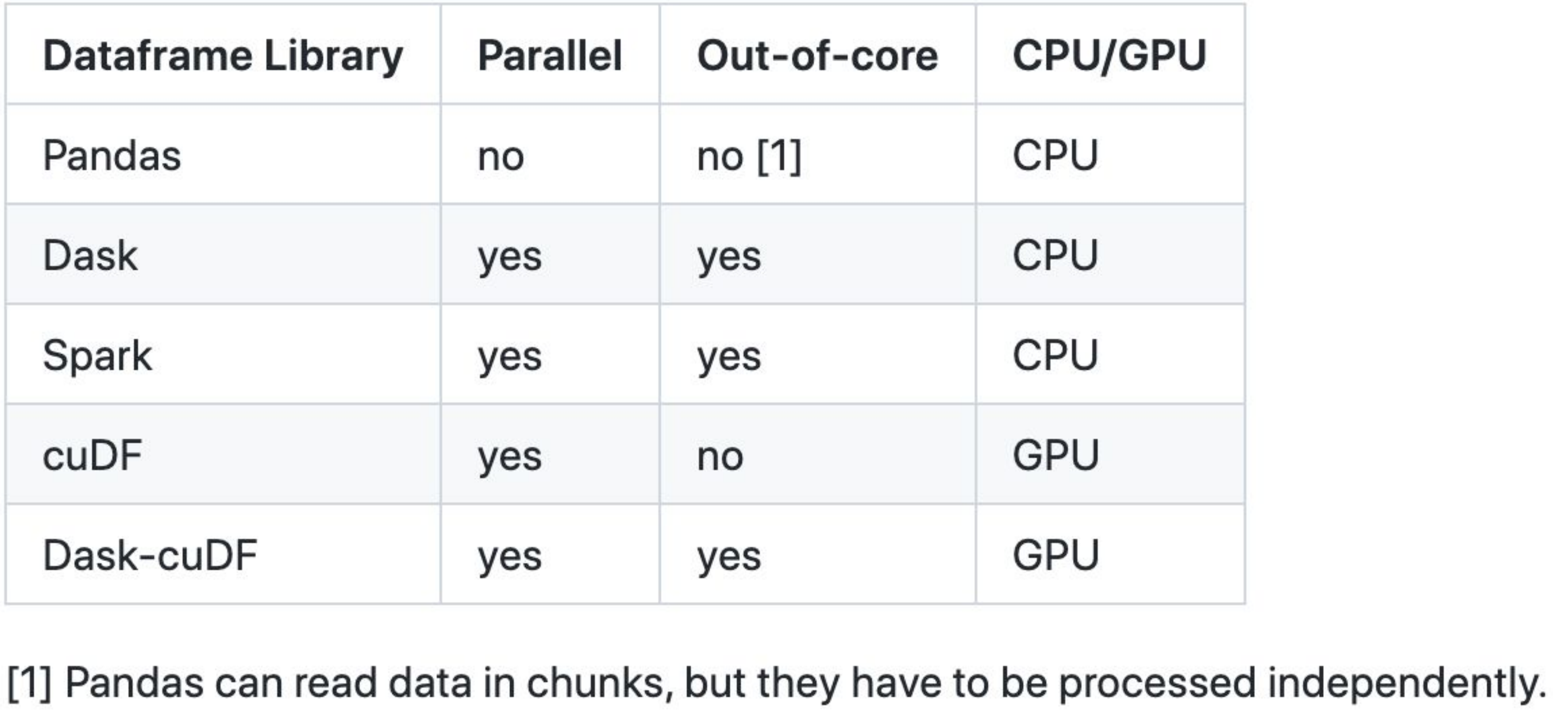
```
[xdtr104@login02 ~]$ ls -lh gene_info.parquet/
total 698K
-rw-r--r-- 1 xdtr104 uic157 6.9K Aug  4 16:16  _common_metadata
-rw-r--r-- 1 xdtr104 uic157 1.5M Aug  4 16:16  _metadata
drwxr-xr-x 2 xdtr104 uic157    4 Aug  4 16:16 'type_of_gene=biological-region'
drwxr-xr-x 2 xdtr104 uic157   68 Aug  4 16:15 'type_of_gene=miscRNA'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=ncRNA'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=other'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=protein-coding'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=pseudo'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=rRNA'
drwxr-xr-x 2 xdtr104 uic157    5 Aug  4 16:16 'type_of_gene=scRNA'
drwxr-xr-x 2 xdtr104 uic157   85 Aug  4 16:15 'type_of_gene=snRNA'
drwxr-xr-x 2 xdtr104 uic157   85 Aug  4 16:16 'type_of_gene=snoRNA'
drwxr-xr-x 2 xdtr104 uic157   87 Aug  4 16:15 'type_of_gene=tRNA'
drwxr-xr-x 2 xdtr104 uic157   62 Aug  4 16:16 'type_of_gene=unknown'
```

- Parquet files are typically directories of files.

- They can be partitioned for query efficiency

Example Notebooks

https://github.com/sbl-sdsc/df-parallel



Dataframe Library	Parallel	Out-of-core	CPU/GPU
Pandas	no	no [1]	CPU
Dask	yes	yes	CPU
Spark	yes	yes	CPU
cuDF	yes	no	GPU
Dask-cuDF	yes	yes	GPU

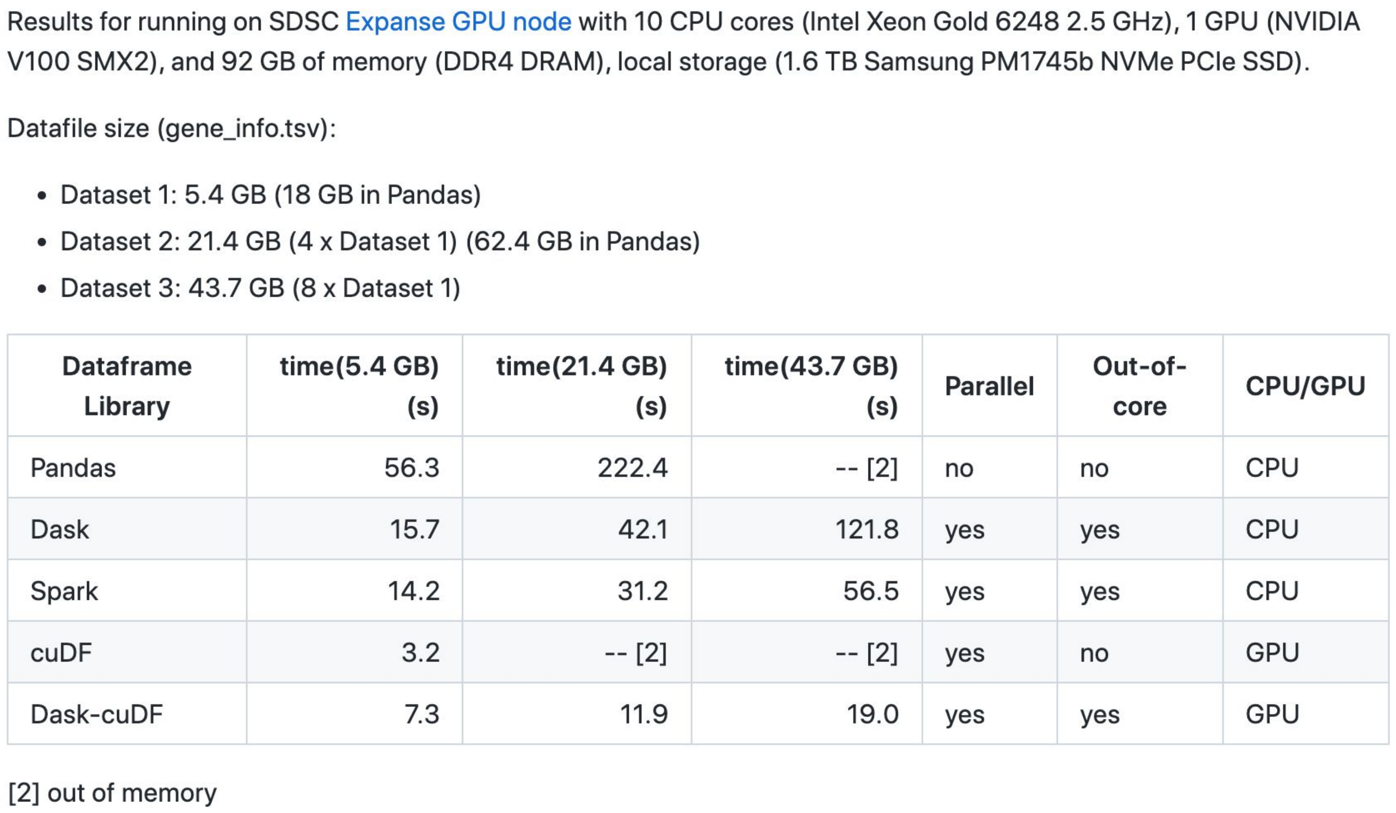
[1] Pandas can read data in chunks, but they have to be processed independently.

Task 2

- Compare the runtime of 5 dataframe libraries
- Compare csv vs. parquet files for Cuda dataframe

- Follow the instructions in section 6.2: Task 2
  https://github.com/sdsc/sdsc-summer-institute-2022/tree/main/6.2_scaling_up_interactive_data_analysis_jupyter_lab#task-2-run-notebooks-in-jupyter-lab

Dataframe Comparison



Results for running on SDSC Expanse GPU node with 10 CPU cores (Intel Xeon Gold 6248 2.5 GHz), 1 GPU (NVIDIA V100 SMX2), and 92 GB of memory (DDR4 DRAM), local storage (1.6 TB Samsung PM1745b NVMe PCIe SSD).

Datafile size (gene_info.tsv):

- Dataset 1: 5.4 GB (18 GB in Pandas)
- Dataset 2: 21.4 GB (4 x Dataset 1) (62.4 GB in Pandas)
- Dataset 3: 43.7 GB (8 x Dataset 1)

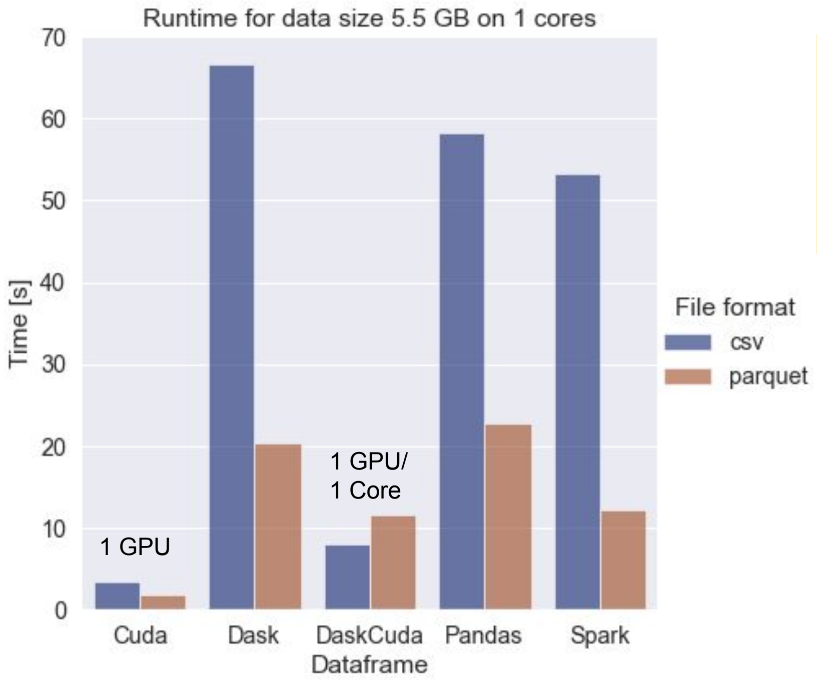
Dataframe Library	time(5.4 GB) (s)	time(21.4 GB) (s)	time(43.7 GB) (s)	Parallel	Out-of-core	CPU/GPU
Pandas	56.3	222.4	-- [2]	no	no	CPU
Dask	15.7	42.1	121.8	yes	yes	CPU
Spark	14.2	31.2	56.5	yes	yes	CPU
cuDF	3.2	-- [2]	-- [2]	yes	no	GPU
Dask-cuDF	7.3	11.9	19.0	yes	yes	GPU

[2] out of memory

Measure Parallel Efficiency

Overall Performance - 1 Core or 1GPU

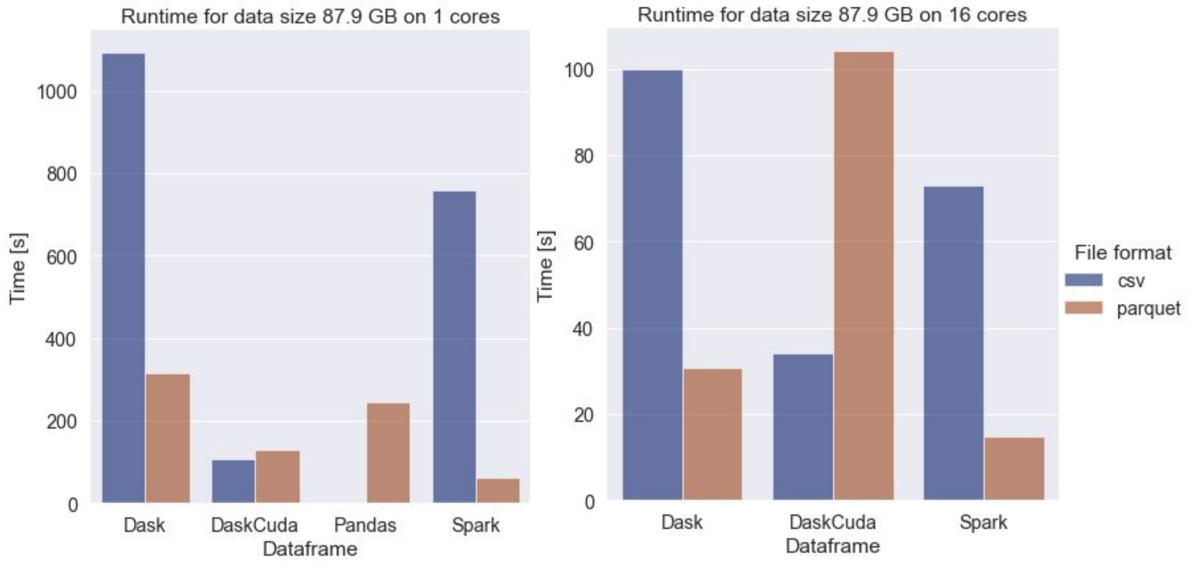
Small Dataset (csv: 5.5 GB, in-memory: 15.2 GB)



- GPU outperforms CPU
- Parquet outperforms CSV file format (except for DaskCuda)
- DaskCuda adds overhead to Cuda

Overall Performance - Large Dataset

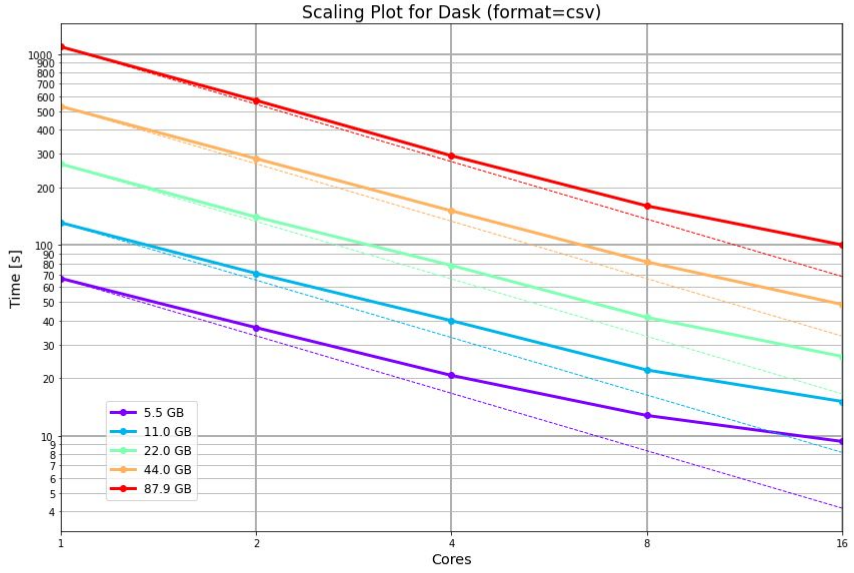
Large Dataset (csv: 87.9 GB) = 16x Small Dataset

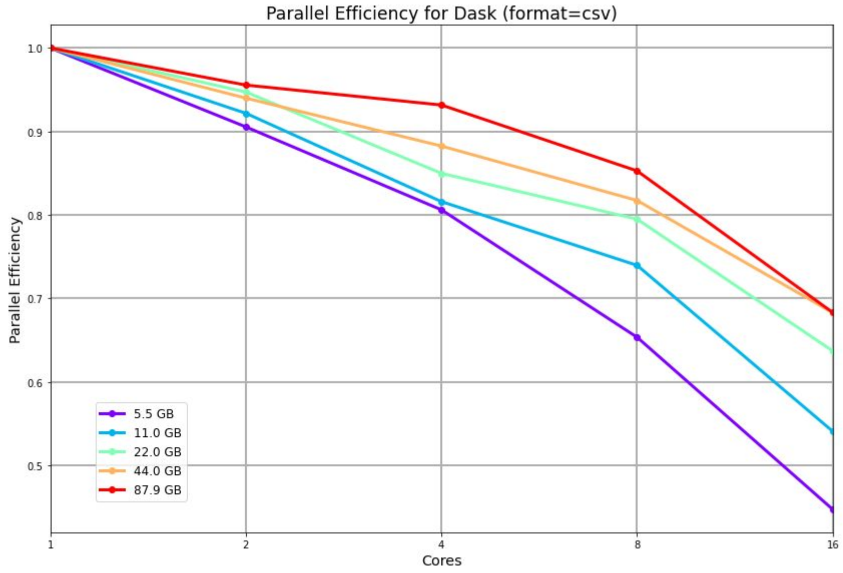


- Cuda out of memory error
- Pandas can handle large Parquet files (not CSV)
- Parquet outperforms CSV file format (except for DaskCuda)
- Spark w/ Parquet file has overall best performance

Scaling for Dask Dataframe

Runtime as function of dataset size and number of cores





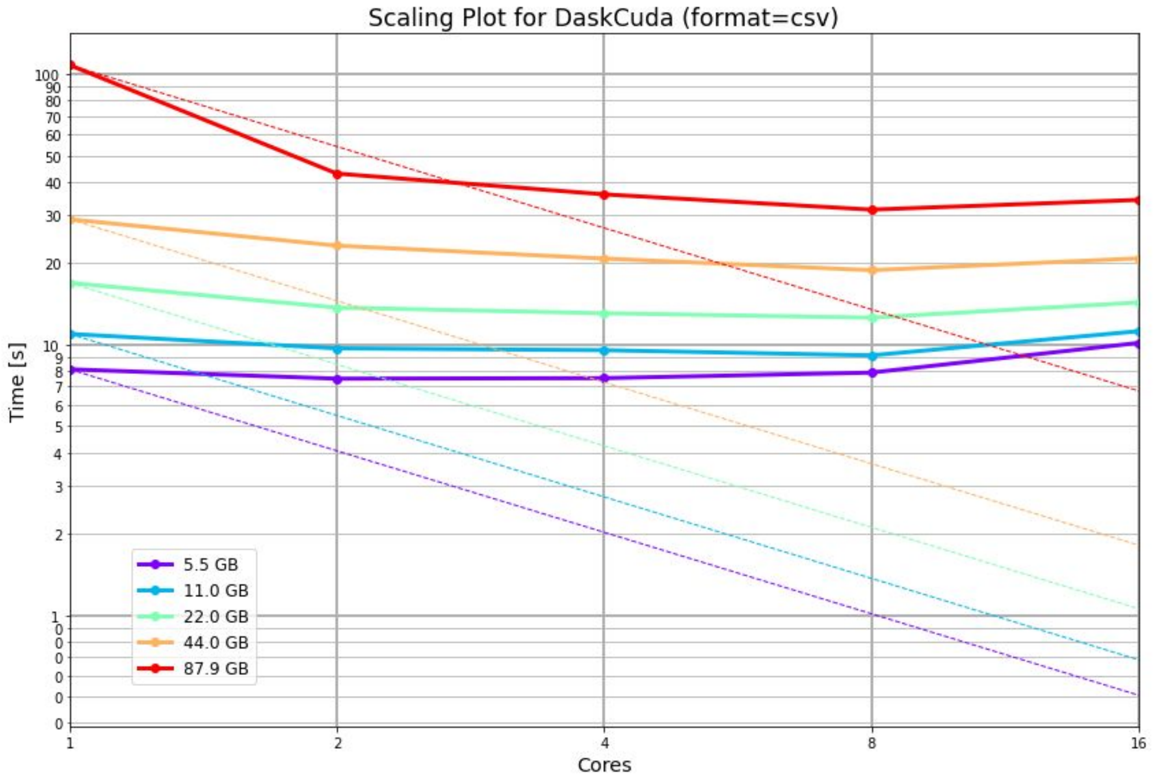
Interactive benchmark: 7-ParallelEfficiency.ipynb

Batch benchmark: slurm script: benchmark.sb notebook: 8-BenchmarkSummary.ipynb

SDSC SAN DIEGO SUPERCOMPUTER CENTER

UC San Diego

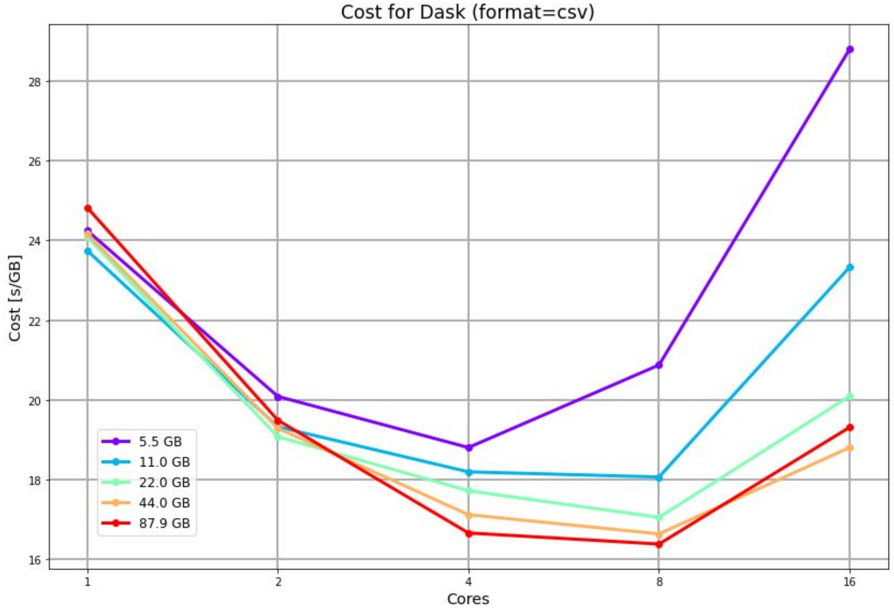
GPU Dataframe

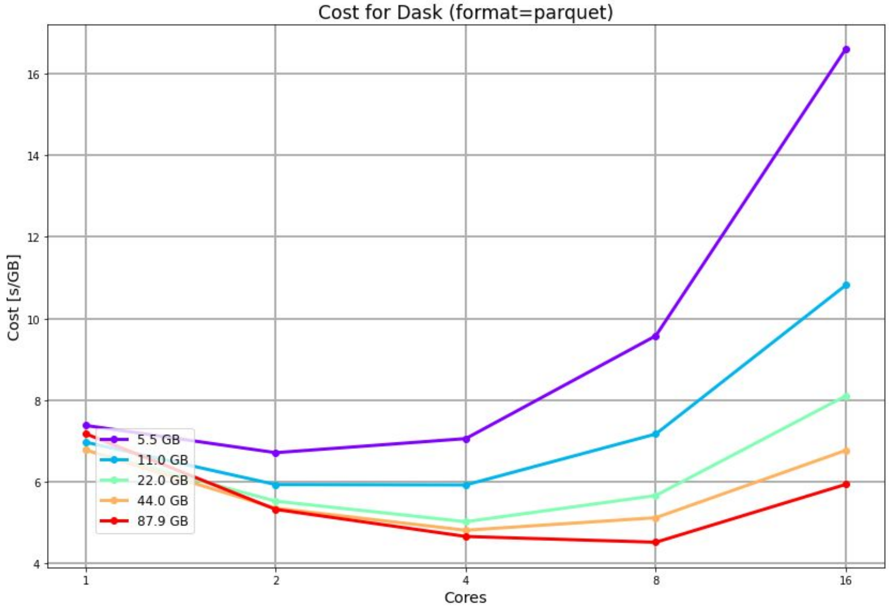


- DaskCuda can handle large datasets
- Number of CPU cores has little effect, except for the large datasets

How many Cores should I use?

```
cost = core * time/size                        # this favors a single core

opprotunity_cost = 1                           # adjustable parameter
cost += opportunity_cost * time/size           # give priority to finish task faster
```





How much does it cost to run the jobs?

Run on CPU (Pandas, Dask, Spark dataframes)

```
galyleo launch --account <account_number> --partition shared --cpus 10
--memory 20 --time-limit 00:30:00 --conda-env df-parallel
--conda-yml "${HOME}/df-parallel/environment.yml" --mamba
```

1 CPU or 2GB of memory are charged 1 CPU Service Unit (SU)/hour.
This job will be charged 10 SU/hour or 5 SUs for 30 minutes.

Run on GPU (Pandas, Dask, Spark, cuDF, Dask-cuDF dataframes)

```
galyleo launch --account <account_number> --partition gpu-shared --cpus 10
--memory 92 --gpus 1 --time-limit 00:30:00 --conda-env df-parallel-gpu
--conda-yml "${HOME}/df-parallel/environment_gpu.yml" --mamba
```

1 GPU or 10 CPUs or 92 GB of memory are charged 1 GPU Service Unit (SU)/hour.
This job will be charged 1 GPU SU/hour. **The minimum charge for any job is 1 SU.** So this job will use 1 SU even though it's just run for 30 minutes.

Get ready to use Expanse

Accounts and allocations

Expanse Allocation

- Apply for an Expanse trial allocation or submit a proposal for a larger allocation.
- Using the trial account, run some performance evaluations
- Sept. 1, 2022: The ACCESS Resource Allocations Marketplace and Platform Services (RAMPS) will replace the XSEDE
- Check for updates here:

  https://portal.xsede.org/allocations/announcements

Summary

- When to run on Expanse
- Setup a portable and reproducible software environment
- Run Jupyter Lab on Expanse
- Scale up calculations on CPU/GPU - Dataframes
- Measure parallel efficiency
- Get ready to use Expanse: accounts, allocations

Homework

- Install df-parallel on your laptop/desktop
- Run the CPU dataframe examples
- Compare the performance with Expanse
- Apply what you’ve learned to you own work
- Commit yourself to write portable and reproducible code

Acknowledgements

Marty Kandes

Mary Thomas

Scott Sakai

Robert Sinkovitz

Funding: NSF Award Number 2017767

CyberTraining: Implementation: Small: Developing a Best Practices Training Program in Cyberinfrastructure-Enabled Machine Learning Research