NYU Center for Data Science

Week 07.2: Dask / HPC

DS-GA 1004: Big Data

slido

We're half way through the semester. How are you doing?

ⓘ Start presenting to display the poll results on this slide.

Announcements

- Quiz 3 this week
  - Spark + Column-oriented storage
- Lab 3 due next week 04/01
- Lab 4 coming soon
  - Dask on the Greene cluster

This week

- Dask
- High-performance computing (HPC)
- Open Q&A

Roadmap for the semester

1.	01/24	Introduction	**9.**	**03/21**	**HPC and Dask**
2.	01/31	Relational databases	10.	03/28	Text and similarity search
3.	02/07	Map-reduce	11.	04/04	Reproducibility
4.	02/14	Hadoop distributed file system	12.	04/11	Recommender systems
5.	*02/21*	*President's day, no meeting*	13.	04/18	Graph algorithms
6.	02/28	Spark	14.	04/25	Differential privacy
7.	03/07	Column-oriented storage	15.	05/02	Graphical processing units
8.	*03/14*	*Spring break, no meeting*	16.	05/09	TBA

Dask

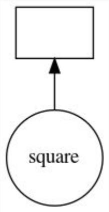
[Rocklin, 2015]

- Python-based distributed computation
- Many common design principles with Spark
  - Delayed computation
  - Computation graphs
  - Collections-based interfaces (e.g. DataFrames)
- Some key differences:
  - Prioritizes array-based computation
  - Designed to support single-machine, out-of-core use

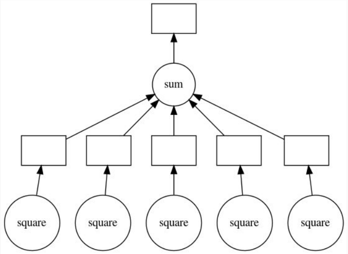
Delayed computation and task graphs

- Dask builds complex computations by composing deferred computations into a **task graph**

```
import dask

def square(x):
        return x**2

f = dask.delayed(square)
y = f(5)
y.visualize()  # draw computation graph
```




```
g = dask.delayed(sum)

z = g([f(x) for x in range(5)])

z.visualize()
```



Collections interfaces: **Bags**

- Dask **bags** are loosely analogous to Spark RDDs
- Unordered collection of generic Python objects
  - Partitions into subsets (sub-bags)
- Implements some basic operations
  - map, filter, join, sum, etc.
- A good choice for initial processing and structured objects
  - If your data is tabular or array-based, probably not the best choice

```
import dask.bag as db

b = db.from_sequence(range(5))

c = b.map(square)

c.compute()   # [0, 1, 4, 9, 16]

c.sum().compute()  # 30
```

Dask Bags vs. Spark RDDs

- Both partition a collection across multiple machines
- Both are immutable
- Both can be **transformed** Bag→Bag, RDD→RDD
- RDDs have **types** (e.g. RDD[Integer])
- Bags are, more or less, **untyped**. You need to be more careful using them!

Bag folding vs grouping

```python
import dask.bag as db

b = db.from_sequence(range(10))
iseven = lambda x: x % 2 == 0
add = lambda x, y: x + y

dict(b.foldby(iseven, add))
```

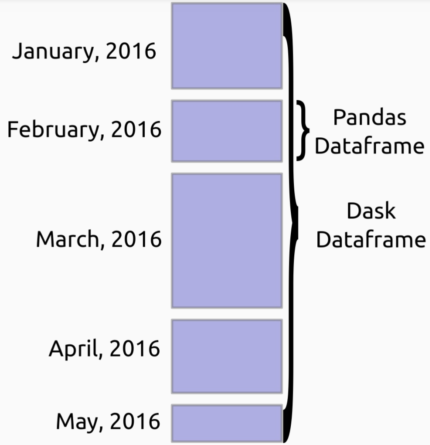
- Try to **avoid using groupBy** on bags
  - This requires much inter-worker communication, and is slow!
- Use **foldBy** if possible
  - Similar benefits to a combiner in map-reduce
  - Perform local aggregation first to reduce the amount of shuffling
  - Like combiners, not always applicable, may require some cleverness

- You supply a key function and a binary operation

Or better yet, move into **DataFrames** before any heavy processing…

Collections interfaces: **DataFrames**

- Just like you'd expect, similar to Spark DataFrames
  - Uses Pandas internally, interface is basically the same
- Parallelism (partitioning) is over subsets of **rows**
- Good choice for data that can naturally split into multiple CSV files (or Parquet partitions)

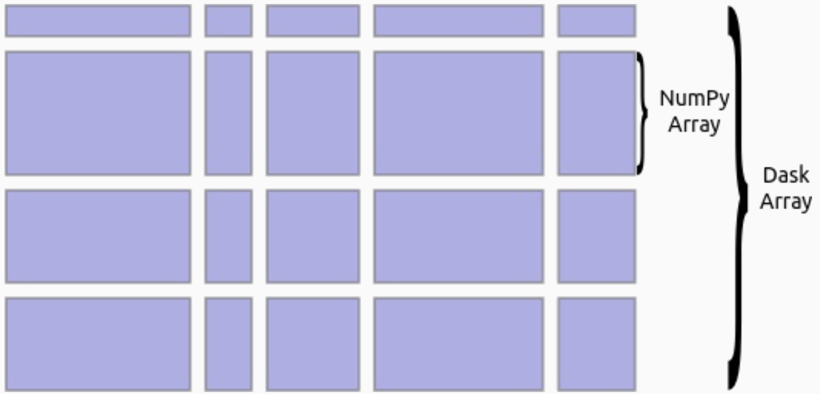



```
import dask.dataframe as dd

df = dd.read_csv('*.csv')
df.mean().compute()
```

Example image from https://docs.dask.org/en/latest/dataframe.html

Collections interfaces: **Arrays**

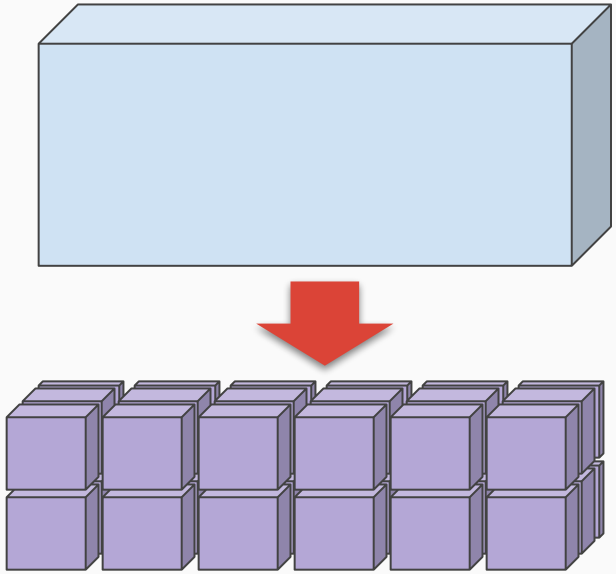
- Dask Arrays work like NumPy arrays
- **Parallelism is not limited to rows**
  - You can define **chunks** along each dimension
- Large arrays are assembled implicitly from many small arrays
- Most* numpy operations work automatically



Example image from https://docs.dask.org/en/latest/array.html

Array chunking example

```
import numpy as np
import dask.array as da

# Make some noise: 2000 x 6000 x 5
x = np.random.randn(2000, 6000, 5)

# Slice it up into chunks
x_parallel = da.from_array(x, chunks=(1000, 1000, 2))
```



Things that don’t work well with chunks

- Sorting. But you can often get away with **topk** instead!
- Operations where output size depends on values in data (e.g., masking: `x[x>0]`)
- Some linear algebra operations (**np.linalg**)

Handling large numerical data

- CSV and Parquet are not great options here!
- Collections of npy / npz files can be okay
  - x.to_npy_stack('/output/folder/') *# Size and chunking info stored as metadata 'info' file*
  - y = **da**.from_npy_stack('/folder/containing/files/')
- Hierarchical Data Format (**HDF5**) is a pretty good solution
  - Data can be accessed without fully loading into memory
- **ZARR** is a newer project with better distributed storage support

HDF5 (not the same as HDFS!)

- Basically a file-system within a file
  - (Hierarchical) Directory structures
- In python, use the **h5py** package
- Data is memory-mapped, not loaded
- Parallel reads are okay!

```
import h5py

data = h5py.File('myfile.h5', mode='r')

x = data['/x']
y = data['/y']
z = data['/path/to/z']
```

```
import dask.array as da

x_parallel = da.from_array(x, chunks=(1000, 1000))
```

Does Dask replace Spark?

- Eh… it depends 🤷
  - https://docs.dask.org/en/latest/spark.html summarizes use-cases and differences
- Pros for Dask:
  - Do you need to integrate with the SciPy stack? (Matplotlib, sklearn, etc)
  - Do you need to work with dense / multi-dimensional data?
  - Custom algorithms / advanced machine learning? GPUs?
- Pros for Spark:
  - More mature, possibly more stable / safe
  - More "high-level" -- you don't need to think as much about the compute graph
  - Probably faster / better optimized for DataFrame crunching
  - Better support for large graph data

HPC: Peel and Greene

- So far, we've been using a Hadoop cluster (**Peel**)
  - HDFS storage
  - MapReduce + Spark jobs (YARN)
- We also have the **Greene** cluster for less restrictive computation
  - Network-accessible file storage (IBM GPFS, **not** HDFS)
  - SLURM job system: job scheduling is unaware of storage layout!
  - Traditionally preferred if you have "embarrassing parallelism"
    (i.e., independent computation / almost no communication)

Next week

- Approximate nearest neighbors
- Spatial data structures

slido

Audience Q&A Session

ⓘ Start presenting to display the audience questions on this slide.