High Performance Computing For Data Science

Gabriele Masina
220496

Giovanni Zotta
223898

UNIVERSITY OF TRENTO

June 2021

Overview

Overview

What is Dask?

Dask is a flexible library for **parallel computing** in Python.[1]

It features many useful interfaces that mimic popular Data Science tools like **Pandas, NumPy, Scikit-Learn** and it allows them to scale to larger memory requirements and **distributed computing environments**.

Advantage

Dask allows data scientists to feel comfortable with their familiar tools, while allowing them to scale to **High-Performance-Computing** environments.

[1] Dask documentation

Why Dask?

Python offers many advanced tools for Data Science. However, without the ability to scale them to large scale computing infrastructures, often the data analysis computations can be unbearable to run on a single machine.

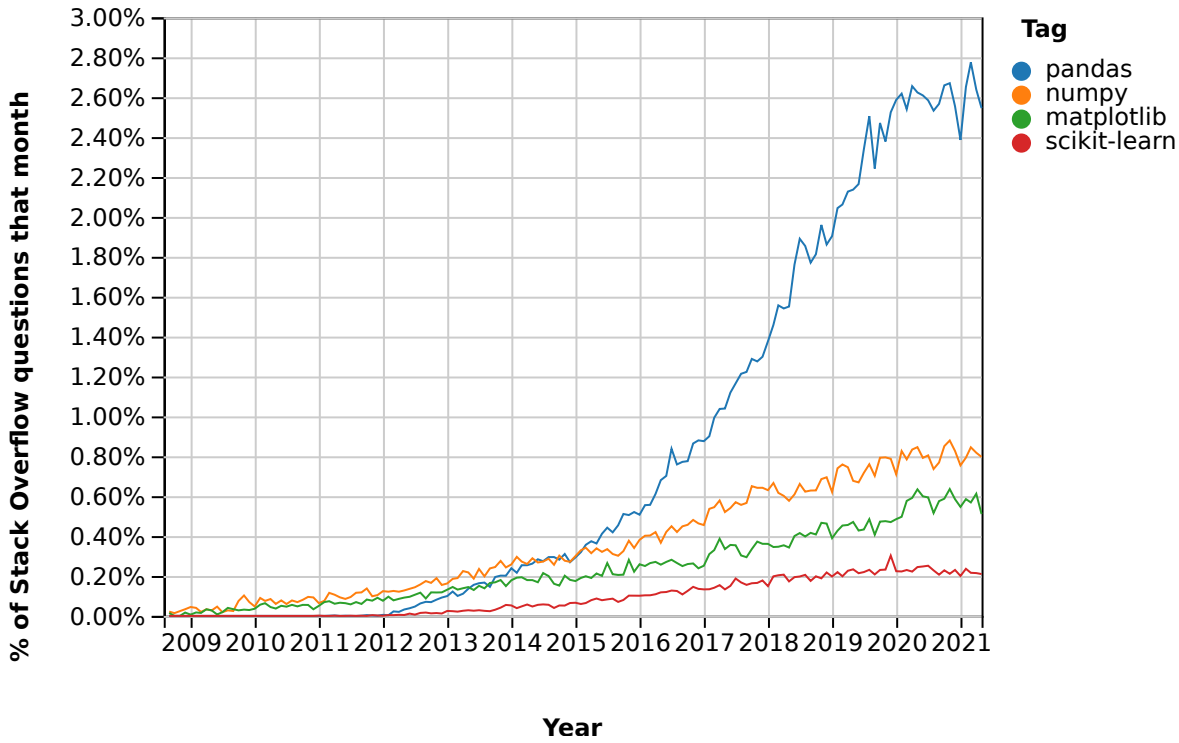

Figure: In the last few years, Python has grown a lot in the Data Science community. [Source]

Who uses Dask?

Dask is used in many fields, both in research and in the industry. Here are a few examples:

- Life sciences;
- Finance;
- Geophysical and Climate sciences;
- Retail Logistics and Demand Forecasting;
- ...

An example: Meteorology

Many national weather services use Dask[2] to run simulations and analyse results, such as:

- UK: Met office and Hydrographic office:
- USA: NCAR, NASA, USGS;
- Australia: CSIRO;
- Europe: European Centre for Medium-Range Weather Forecasting;
- ...

Scientific data formats

Dask provides native support for popular scientific data formats, like **NetCDF**, **HDF5** and others.

[2] https://youtu.be/t_GRK4L-bnw?t=999

Overview

Dask user interfaces

Dask supports several user interfaces:

low-level flexible annotations that allow to parallelize **custom snippets of code**:

- **Delayed** Lazy parallel function evaluation;
- **Futures** Real-time parallel function evaluation.

high-level complex **data structures** that resemble interfaces of **popular data science libraries**:

- **Arrays** Parallel NumPy;
- **Bags** Flexible data structure that allows several operations like map, reduce, filter, ...;
- **DataFrames** Parallel Pandas;
- **Machine Learning** Parallel Scikit-Learn;
- **Many more** ...

A serial example

```
from time import sleep

def inc(x):
    sleep(1)
    return x + 1

def add(x, y):
    sleep(1)
    return x + y

x = inc(1)
y = inc(2)
z = add(x, y)
```

Execution time: $\sim 3s$

Distributed scheduler

In order to parallelize the workload among different processes and CPU cores, we need to define our working environment.

```
from dask.distributed import Client
client = Client(n_workers=4)
```

The above code snippet initializes a local cluster on our laptop composed of 4 parallel processes.

Parallelizing with Delayed

```
from dask import delayed

x = delayed(inc)(1)
y = delayed(inc)(2)
z = delayed(add)(x, y)
z.visualize()
```

Execution time: $< 2ms$

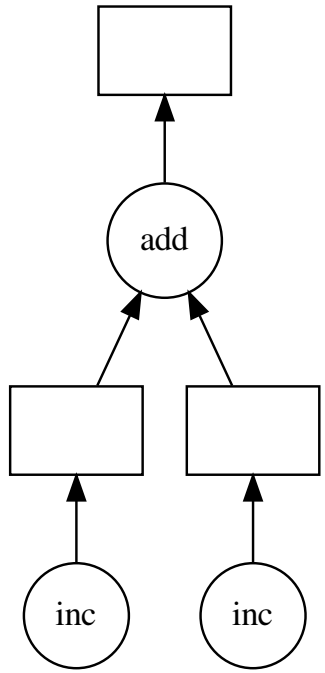

Figure: Resulting task graph

Parallelizing with Delayed

```
from dask import delayed

x = delayed(inc)(1)
y = delayed(inc)(2)
z = delayed(add)(x, y)
z.visualize()
```

Execution time: $< 2ms$

```
z.compute()
```

Execution time: $\sim 2s$

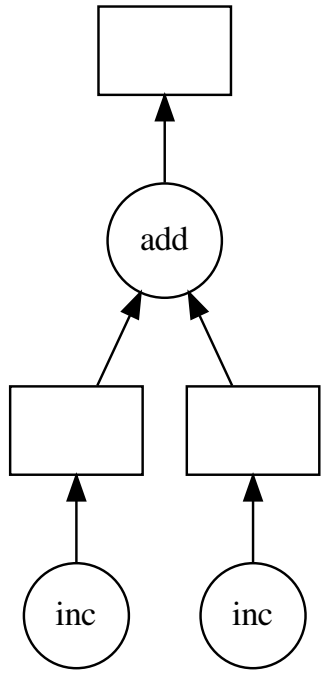

Figure: Resulting task graph

A more involved example

```
data = [3, 5, 4]

def double(x):
    sleep(1)
    return 2 * x

results = []
for x in data:
    a = double(x)
    b = inc(x)
    c = add(a, b)
    results.append(c)

total = sum(results)
```

A more involved example - First parallelization

```
data = [3, 5, 4]

results = []
for x in data:
    a = delayed(double)(x)
    b = delayed(inc)(x)
    c = delayed(add)(a, b)
    results.append(c)

total = sum(results)
total.visualize()
```

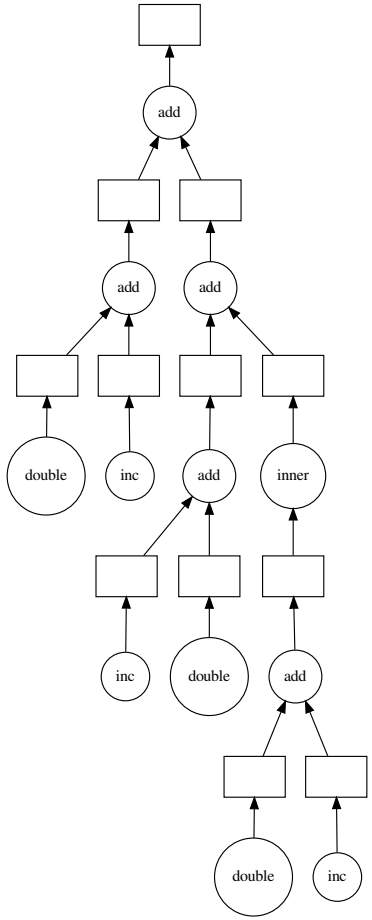

Figure: Partially delayed sum

A more involved example - Second parallelization

```
data = [3, 5, 4]

results = []
for x in data:
    a = delayed(double)(x)
    b = delayed(inc)(x)
    c = delayed(add)(a, b)
    results.append(c)

total = delayed(sum)(results)
total.visualize()
```

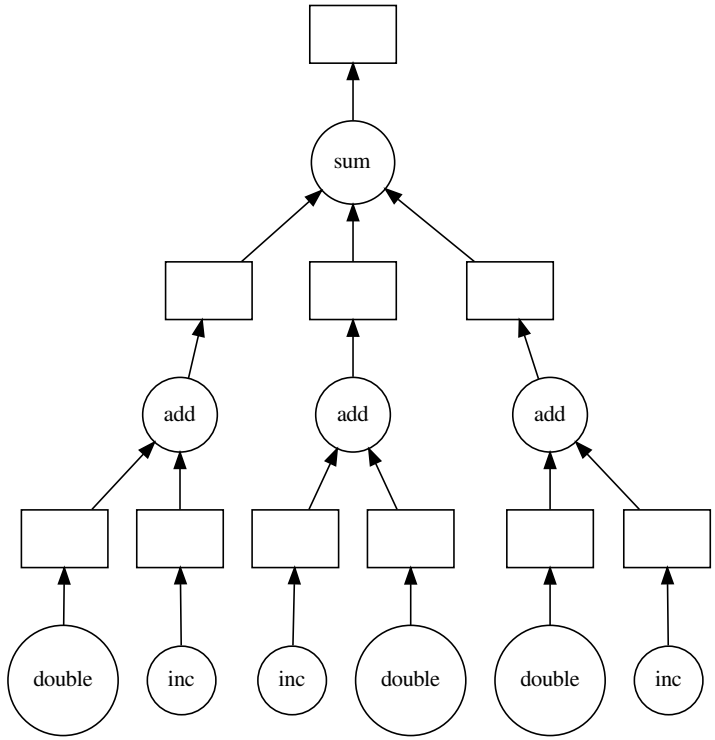

Figure: Completely delayed sum

Dask arrays

Dask arrays implement many NumPy arrays features. Unlike NumPy arrays, Dask arrays can be **split in** multiple small arrays in order to be able to be computed on **many machines** and **many cores**. They support several NumPy interfaces, such as:

- Arithmetic and scalar mathematics: `+, *, exp, log, ...`;
- Reductions: `sum, mean, sum(axis=0), ...`;
- Slicing: `x[100:500]`;
- Many more ...

Handling large datasets

Dask can handle dataset that could not be usually loaded in the RAM of a machine by applying smart decisions on the allocation and deletion of data structures.

Dask arrays in action

```
import dask.array as da

x = da.random.random((600,
    200), chunks=(200,100))
```

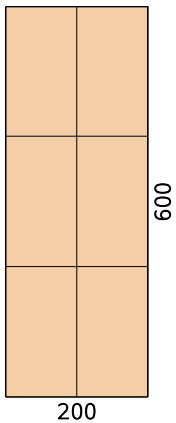

Figure: Shape of x divided in chunks

Dask arrays in action

```
import dask.array as da

x = da.random.random((600,
    200), chunks=(200,100))

y = x.mean(axis=0)
y.compute()
```

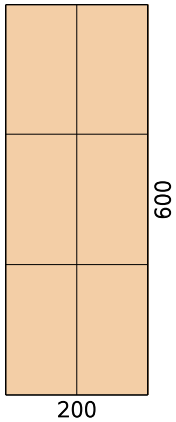

Figure: Shape of x divided in chunks

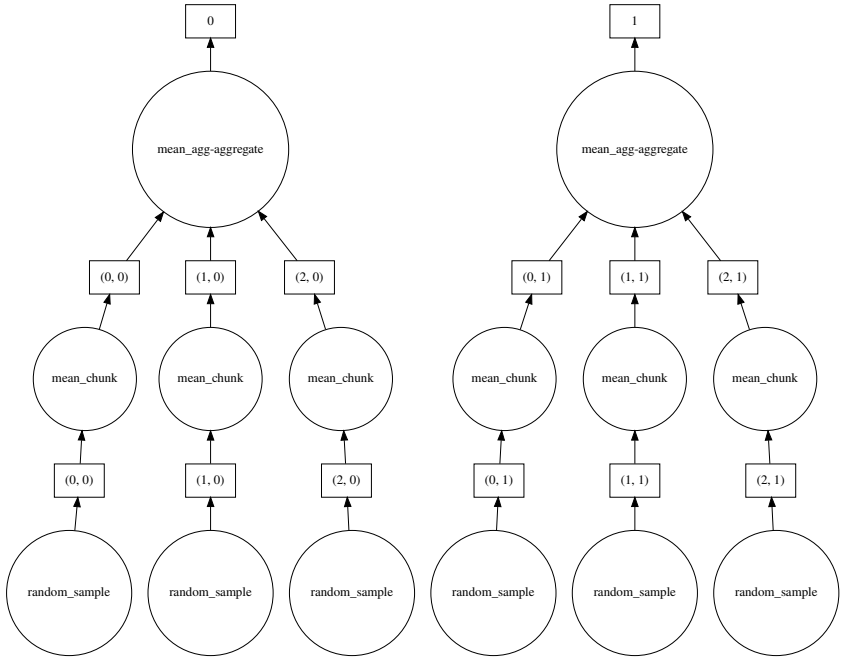

Figure: Resulting task graph

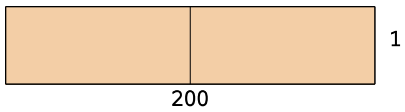

Figure: Shape of output y

The Dask dashboard

The Dask dashboard

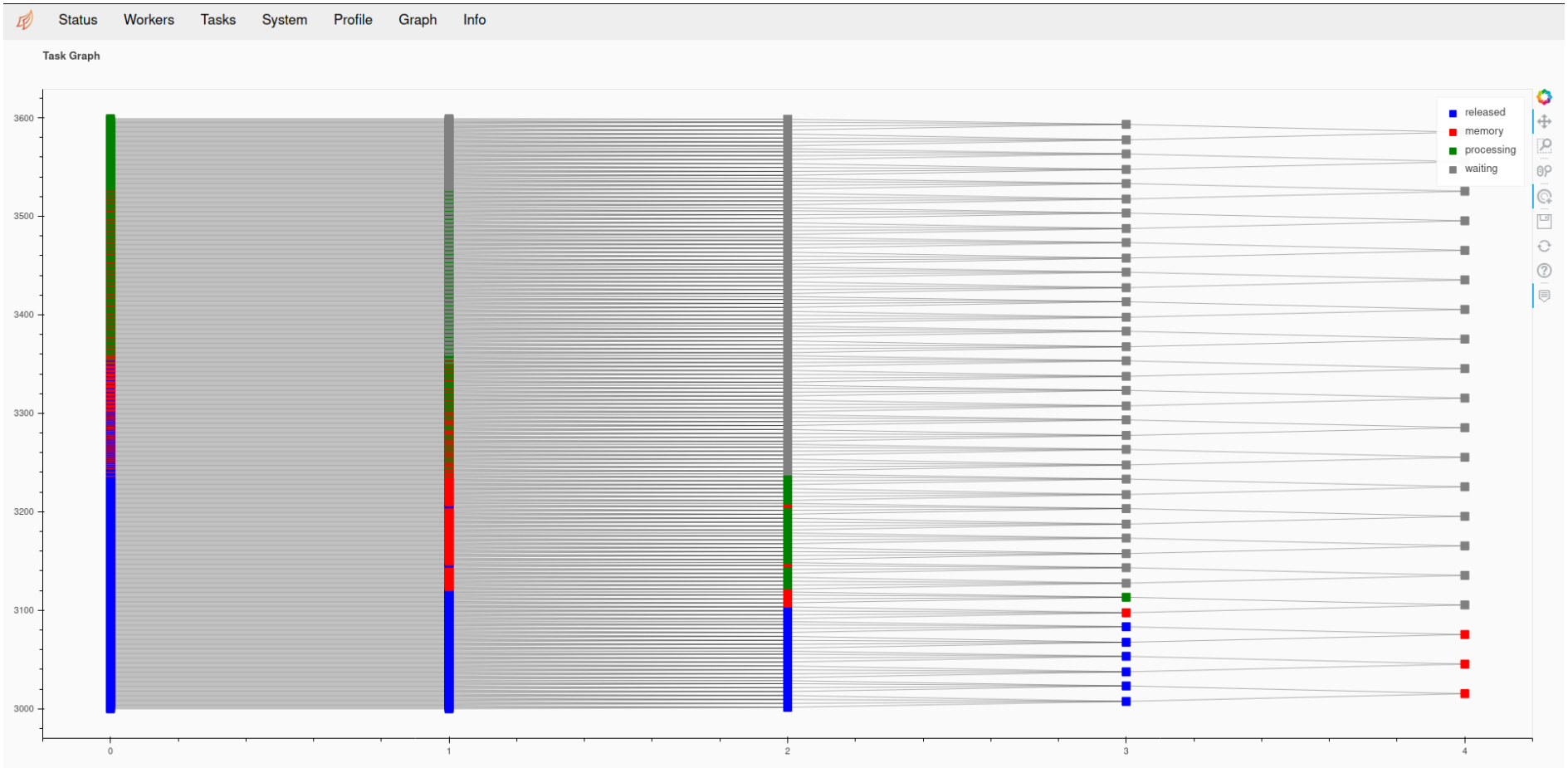

Figure: Interactive task graph

Overview

Dask and MPI

Dask can be easily deployed to an HPC cluster using MPI and PBS.

Requirements

The only requirement is that we have to make sure we install the Python packages `dask`, `dask[distributed]`, `dask_mpi` and `mpi4py`.

Deploy with MPI

```
#!/bin/bash
#PBS -l select=4:ncpus=4:mem=8gb
#PBS -l walltime=0:10:00
#PBS -q short_cpuQ
module load python-3.7.2
module load mpich-3.2
# load a virtual env where Dask for Python is installed
source dask_env/bin/activate

mpirun.actual -n 16 python dask_test.py
```

Deploy with MPI

```
from dask_mpi import initialize
from dask.distributed import Client
import  dask.array  as da

initialize() # initialize MPI

client = Client()  # Connect process to other workers

x = da.random.random((6000, 2000), chunks = (200 ,100))

y = x.mean(axis =0)
z = y.compute()
print(z)

client.close() # terminate the client
```

Listing 1: dask_test.py

Overview

Take-home message

Finding out that there is a tool for the job at hand is often relieving, since most of the heavylifting has already been done by someone else.

However...

In HPC, the learning curve of such tools may sometimes look very steep.

That's only true to an extent!

For example, Dask is very **user-friendly**, and once one has learned the basics it can give the user very **powerful** tools while staying familiar to the well-known Data Science libraries.