Introduction to Dask: Scalable Analytics in Python

DoD High Performance Computing and Modernization Program (HPCMP)
Productivity Enhancement and Training (PET)

USER PRODUCTIVITY ENHANCEMENT AND TRAINING

Zachary Lamb, PET Computational Scientist
November 12 2020

GENERAL DYNAMICS
Information Technology

Overview

- What is Dask?
- Installation and setup
- The task graph
- Delayed and futures
- Bag, Array, Dataframe
- Distributed
- Machine Learning
- Dask Dashboard
- Jupyter Notebook demo

What is Dask?

- Dask is defined as a parallel computing library that scales the existing Python ecosystem
- Operates by constructing and executing a task graph
- Very good at scaling to different hardware
- It provides multi-core and distributed parallel execution on larger-than-memory datasets

Installation

- **Dask is easily installed via conda or pip and can be built from source**
- **Included in the default Anaconda distribution**
- **More details: https://docs.dask.org/en/latest/install.html**
- **Single machine and distributed schedulers**

Dependency	Version	Description
bokeh	>=1.0.0	Visualizing dask diagnostics
cloudpickle	>=0.2.2	Pickling support for Python objects
cityhash		Faster hashing of arrays
distributed	>=2.0	Distributed computing in Python
fastparquet		Storing and reading data from parquet files
fsspec	>=0.6.0	Used for local, cluster and remote data IO
gcsfs	>=0.4.0	File-system interface to Google Cloud Storage
murmurhash		Faster hashing of arrays
numpy	>=1.13.0	Required for dask.array
pandas	>=0.23.0	Required for dask.dataframe
partd	>=0.3.10	Concurrent appendable key-value storage
psutil		Enables a more accurate CPU count
pyarrow	>=0.14.0	Python library for Apache Arrow
s3fs	>=0.4.0	Reading from Amazon S3
sqlalchemy		Writing and reading from SQL databases
cytoolz/toolz	>=0.8.2	Utility functions for iterators, functions, and dictionaries
xxhash		Faster hashing of arrays

Task Graph

- Dask encodes algorithms in a simple format involving Python dicts, tuples, and functions

```
{'x': 1,
 'y': 2,
 'z': (add, 'x', 'y'),
 'w': (sum, ['x', 'y', 'z']),
 'v': [(sum, ['w', 'z']), 2]}
```

- Task scheduling is often used as a means to parallelize a program by breaking the work into smaller tasks
- Dask represents these tasks as nodes in a graph with edges between nodes if one task depends on data produced by another
- A task scheduler executes this graph in a way that respects data dependencies and leverages parallelism where possible

Delayed

- **Delayed allows developers to parallelize their own algorithms**
- **Decorating functions with delayed will allow Dask to parallelize any portion of your problem that can be run in parallel**
- **Decorated (wrapped) functions will return a Delayed object and perform no computations**

```python
from dask import delayed

@delayed
def inc(x):
    return x + 1
```

```python
import dask

def inc(x):
    return x + 1

inc_delayed = dask.delayed(inc)
```

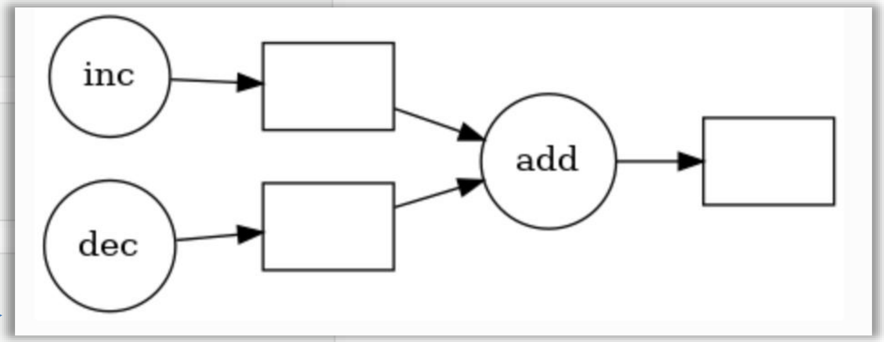
Delayed

```python
import time
import random

def inc(x):
    time.sleep(random.random())
    return x + 1

def dec(x):
    time.sleep(random.random())
    return x - 1

def add(x, y):
    time.sleep(random.random())
    return x + y
```

```python
import dask
inc = dask.delayed(inc)
dec = dask.delayed(dec)
add = dask.delayed(add)
```

```python
%%time
x = inc(1)
y = dec(2)
z = add(x, y)
z
```

- **Here z is a Delayed object**
- **The only work that has been done is the construction of the task graph**



Questions?

Dask Futures

- **Futures extends the concurrent.futures interface of Python and is very similar to the ThreadPoolExecutor (or ProcessPoolExecutor) that is in the standard library**
- **The major difference from Delayed is that futures starts the computation immediately**

```python
from dask.distributed import Client

client = Client()  # by default workers are processes - pass processes=False for threads
```

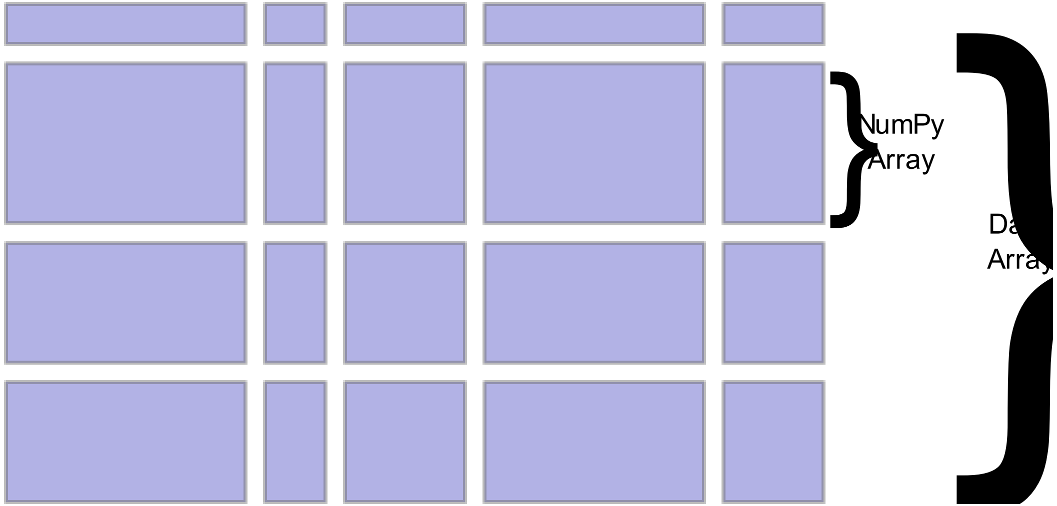
```python
def inc(x):
    return x + 1

futures = client.map(inc, range(1000))

results = [future.result() for future in futures]
results = client.gather(futures)  # may be faster
```

Dask Array

- **Dask arrays use Numpy under the hood and many of the numpy functionalities are supported**
- **Dask arrays are simply chunked numpy arrays**

In [4]:
```python
import numpy as np

arr = np.arange(50)
arr.shape
```
Out[4]: (50,)

In [5]:
```python
import dask.array as da

arr = da.arange(50)
arr.shape
```
Out[5]: (50,)

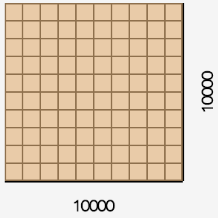


Dask Array

- **Dask provides some nice visualization of arrays in Jupyter Notebooks**

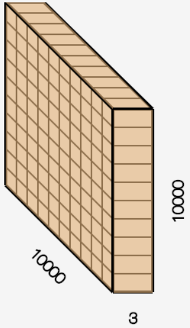
```python
import dask.array as da

arr = da.random.random((10000, 10000), chunks=(1000, 1000))
arr
```

	Array	Chunk
Bytes	800.00 MB	8.00 MB
Shape	(10000, 10000)	(1000, 1000)
Count	100 Tasks	100 Chunks
Type	float64	numpy.ndarray




```python
import dask.array as da

arr = da.random.random((10000, 10000, 3), chunks=(1000, 1000, 3))
arr
```

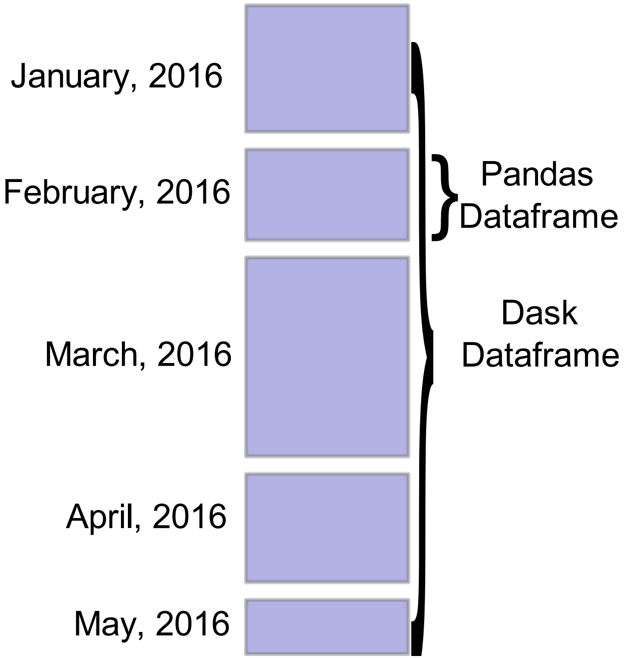
	Array	Chunk
Bytes	2.40 GB	24.00 MB
Shape	(10000, 10000, 3)	(1000, 1000, 3)
Count	100 Tasks	100 Chunks
Type	float64	numpy.ndarray



Dask Dataframe

- **Dask DataFrames coordinate many Pandas DataFrames/Series arranged along the index**
- **A Dask DataFrame is partitioned row-wise, grouping rows by index value for efficiency**

```python
import pandas as pd
df = pd.read_csv('some data')

import dask.dataframe as dd
df = dd.read_csv('some data')
```



Dask Bag

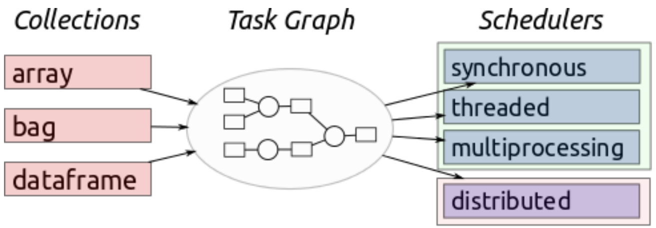
- Dask Bag implements operations like map, filter, fold, and groupby on collections of generic Python objects
- Typically used to parallelize simple computations on unstructured or semi-structured data like text data, log files, JSON records, or user defined Python objects

```
In [7]: import dask.bag as db

        bag = db.from_sequence([0,1,2,3,4,5,6,7,8,9], npartitions=4)
        bag.map(lambda x: x**2)

Out[7]: dask.bag<lambda, npartitions=4>

In [8]: bag.compute()

Out[8]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Questions?

Dask Distributed

- **Up to this point we have seen that Dask handles the computations for us**
- Dask has two families of task schedulers:
  - Single machine scheduler: This scheduler provides basic features on a local process or thread pool. This is the default scheduler
  - Distributed scheduler: This scheduler is more sophisticated, offers more features, but also requires a bit more effort to set up
- **Dask comes with four available schedulers:**
  - Threaded: a scheduler backed by a thread pool
  - Processes: a scheduler backed by a process pool
  - Single-threaded (aka "sync"): a synchronous scheduler, good for debugging
  - Distributed: a distributed scheduler for executing graphs on multiple machines



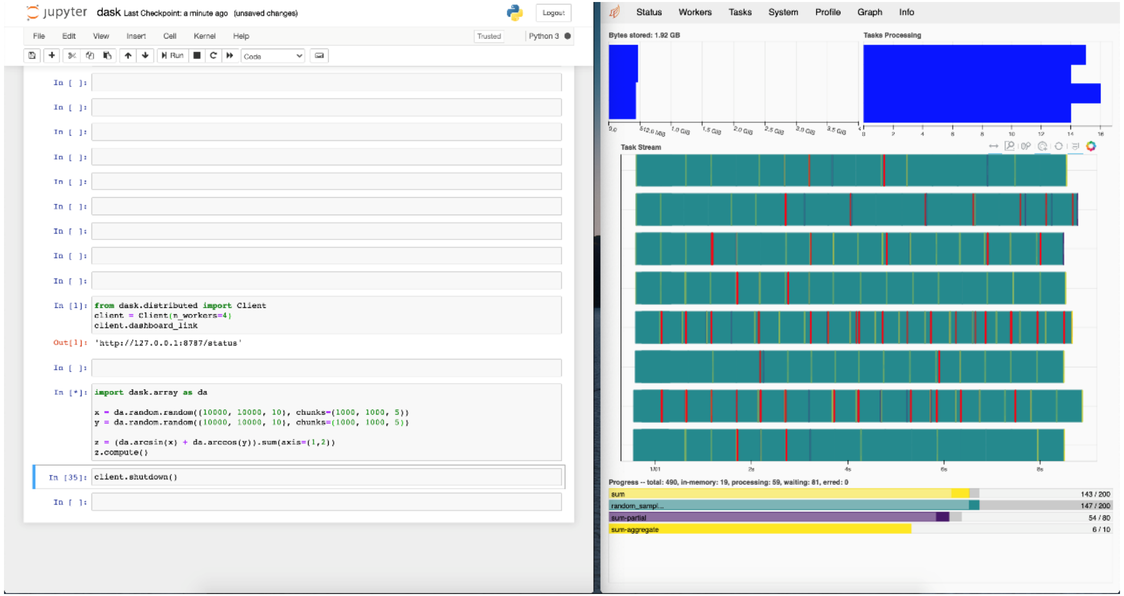
Dask for Machine Learning

- Dask-ML provides scalable machine learning in Python using Dask alongside popular machine learning libraries like scikit-learn
- Dask extends the parallelism in scikit-learn from a single machine to distributed resources
- Allows you to use some scikit-learn algorithms with larger-than-memory datasets
- Tips from Dask documentation:
  - For in-memory problems, use scikit-learn (or your favorite ML library).
  - For large models, use dask_ml.joblib and your favorite scikit-learn estimator
  - For large datasets, use dask_ml estimators

```python
from dask.distributed import Client
import joblib

client = Client()  # Connect to a Dask Cluster

with joblib.parallel_backend('dask'):
    # Your normal scikit-learn code here
```

Dask Dashboard

- **The Dashboard provides live information in the form of plots and tables**
- **Information on memory and CPU usage, task execution (progress, distribution), and detailed profiling information**



PET
USER PRODUCTIVITY ENHANCEMENT AND TRAINING

GENERAL DYNAMICS
Information Technology

Questions?

Contact

Questions and Information

https://training.hpc.mil/

pet@hpc.mil

References/Sources

1. Figures (Array, DataFrame, distributed, logo): https://docs.dask.org/en/latest/

USER PRODUCTIVITY
ENHANCEMENT AND TRAINING

GENERAL DYNAMICS
Information Technology