Aneesha Bakharia Follow
Data Science, Topic Modelling, Deep Learning, Algorithm Usability and Interpretation, Learning Analytics, Electronics — Brisbane, Australia
Jun 6 · 4 min read

# Why every Data Scientist should use Dask?

Dask is simply the most revolutionary tool for data processing that I have encountered. If you love Pandas and Numpy but were sometimes struggling with data that would not fit into RAM then Dask is definitely what you need. Dask supports the Pandas dataframe and Numpy array data structures and is able to either be run on your local computer or be scaled up to run on a cluster. Essentially you write code once and then choose to either run it locally or deploy to a multi-node cluster using a just normal Pythonic syntax. This is a great feature in itself, but it is not why I am writing this blog post and saying that every Data Scientist (at least the one's using Python) should use Dask. The magic Dask feature for me, has been that with minimal code changes, I can run code code in parallel taking advantage of the processing power already on my laptop. Processing data in parallel, means less time to execute, less time to wait and more time to analyse! This blog post will talk about dask.delayed and how it fits into the data science workflow.

Photograph from https://pixabay.com/en/skiing-departure-wag-trace-curves-16263/

> *"Dask provides advanced parallelism for analytics, enabling performance at scale for the tools you love"— https://dask.pydata.org/en/latest/*

## Getting Familiar with Dask

As an introduction to Dask, I'll start with a few examples just to give you an indication of its completely unobtrusive and natural syntax. The

main take away here is that you can use what you already know without needing to learn a new big data tool like Hadoop or Spark.

Dask introduces 3 parallel collections that are able to store data that is larger than RAM, namely Dataframes, Bags and Arrays. Each of these collection types are able to use data partitioned between RAM and a hard disk as well distributed across multiple nodes in a cluster.

Dask DataFrame is made up of smaller split up Pandas dataframes and therefore allows a subset of Pandas query syntax. Here is example code that loads all csv files in 2018, parses the timestamp field and then runs a Pandas query:

```
import dask.dataframe as dd

df = dd.read_csv('logs/2018-*.*.csv', parse_dates=['timesta
df.groupby(df.timestamp.dt.hour).value.mean().compute()
```

Dask Dataframe example

A Dask Bag is able to store and process collections of Pythonic objects that are unable to fit into memory. Dask Bags are great for processing logs and collections of json documents. In this code example, all json files from 2018 are loaded into a Dask Bag data structure, each json record is parsed and users are filtered using a lambda function:

```
import dask.bag as db
import json

records = db.read_text('data/2018-*-*.json').map(json.loads
```

Dask Bag example

Dask Arrays support Numpy like slicing. In the following code example, an HDF5 dataset in chunked into (5000, 5000) dimension blocks:

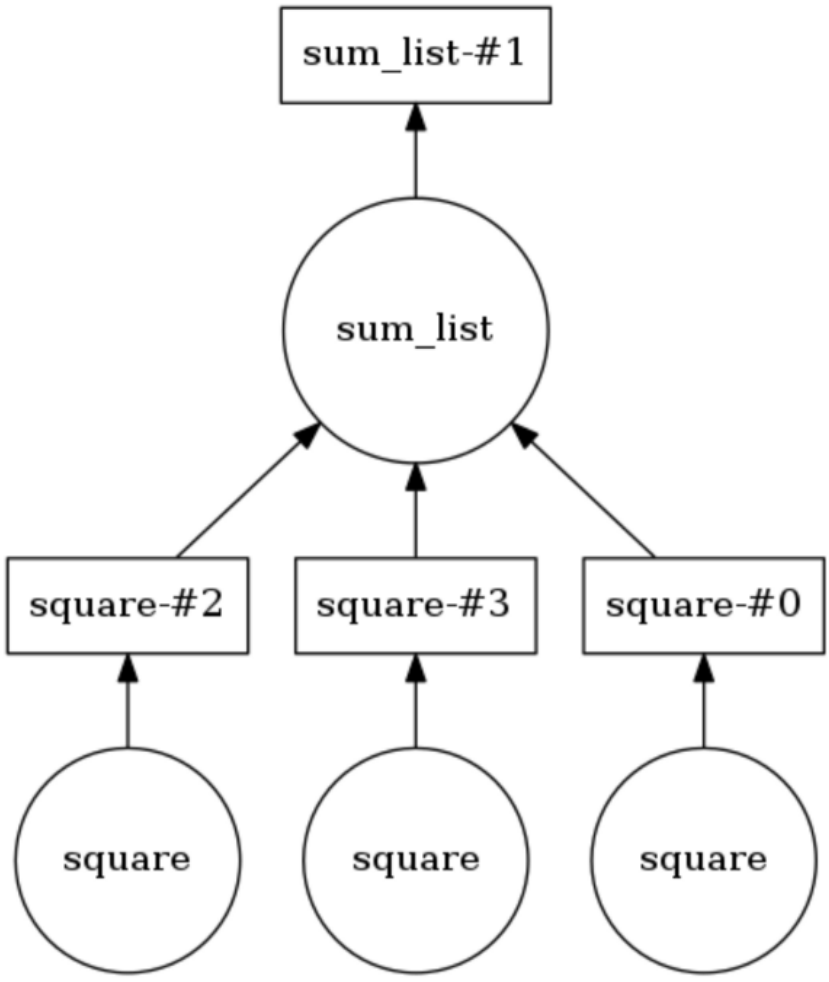
```
import h5py
f = h5py.File('myhdf5file.hdf5')
dset = f['/data/path']

import dask.array as da
```

Dask Array example

## Parallel Processing with Dask

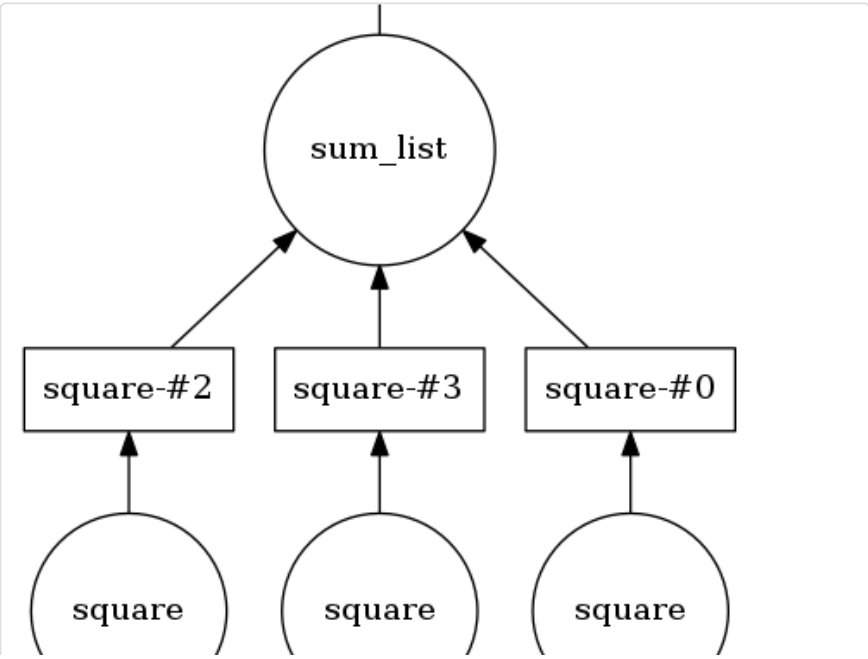
An alternate accurate name for this section would be “Death of the sequential loop”. A common pattern I encounter regularly involves looping over a list of items and executing a python method for each item with different input arguments. Common data processing scenarios include, calculating feature aggregates for each customer or performing log event aggregation for each student. Instead of executing a function for each item in the loop in a sequential manner, Dask Delayed allows multiple items to be processed in parallel. With Dask Delayed each function call is queued, added to an execution graph and scheduled.

Writing custom thread handling or using asyncio has always looked a bit tedious to me, so I’m not even going to bother with showing you comparative examples. With Dask, you don’t need to change your programming style or syntax! You just need to annotate or wrap the method that will be executed in parallel with @dask.delayed and call the compute method after the loop code.




Example Dask computation graph

In the example below, two methods have been annotated with @dask.delayed. Three numbers are stored in a list which must be squared and then collectively summed. Dask constructs a computation graph which ensures that the “square” method is run in parallel and that the output is collated as a list and then passed to the sum_list method. The computation graph can be printed out by calling .visualize(). Calling .compute() executes the computation graph. As you can see in the output, the the list items are not processing in order and are run in parallel.




Dask Delayed demonstration

The number of threads can be set (i.e., dask.set_options( pool=ThreadPool(10) ) and its also easy to swap to use processes on your laptop or personal desktop (i.e., dask.config.set( scheduler=’processes’ ).

I have illustrated that adding parallel processing to your data science workflow is trivial with Dask. I’ve used Dask recently to split user clickstream data into 40 minute sessions and build user aggregate features for clustering. Please share a description of how you are using Dask as a comment to this blog post. Happy Dasking…..

## Additional Resources

- Dask Cheatsheet https://dask.pydata.org/en/latest/_downloads/daskcheatsheet.pdf

- Great Dask overview video in 16 minutes
  https://www.youtube.com/watch?v=ods97a5Pzw0
- Detailed Dask overview video (40 minutes)
  https://www.youtube.com/watch?v=mjQ7tCQxYFQ