NYU | Center for Data Science

Week 07: Dask

DS-GA 1004: Big Data

Where are we now?

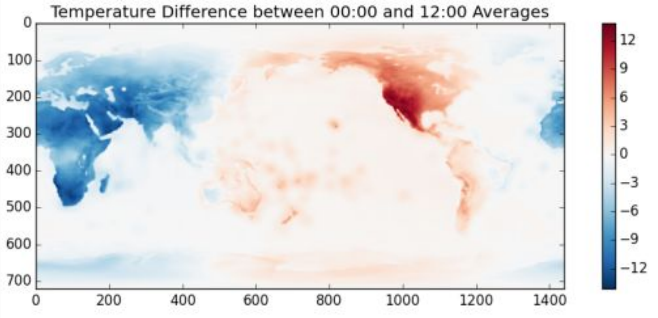
RDBMS and SQL	Map-Reduce and HDFS	Spark
efficiently store and organize data for common analyses	distributed computation and storage for simple algorithms	improved distributed computation for complex algorithms

Spark is cool!

- Spark integrates nicely with Java-based tools (Hadoop framework)
  - HDFS, Parquet, YARN scheduler, etc…
- Spark is great for DataFrames and SQL-like processing
  - *Graphs also, though we haven’t gotten to that (yet)*
- >10 years old now, implementation is mature and stable

Some things are still difficult…

- Data that doesn't naturally fit RDD/DataFrame model
- Python (SciPy stack) integration
- (Modern) Machine learning:
  - sklearn, pytorch, etc…
- Scaling down vs. out:
  - do you really need a cluster?



Dask

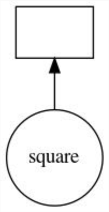
[Rocklin, 2015]

- 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**

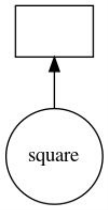
```
import dask

def square(x):
        return x**2

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



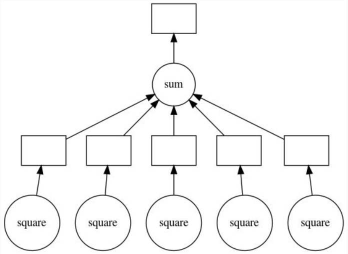
Delayed computation and task graphs

- 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(z) for z in range(5)])

z.visualize()
```

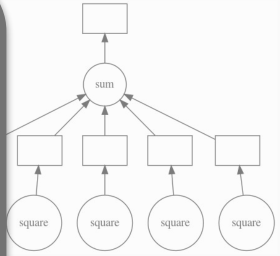


Delayed computation and task graphs

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

```
import dask

def square(x):
        return x*

f = dask.delaye
y = f(5)
y.visualize()  # dr
```

```
g = dask.delayed(sum)
```



Delayed functions are similar to **transformations** in Spark

No **action** is taken until we call **y.compute()** or **z.compute()**

Key difference from Spark: no strong type requirements!
(inputs and outputs need not be RDDs)

Collections interfaces: **Bags**

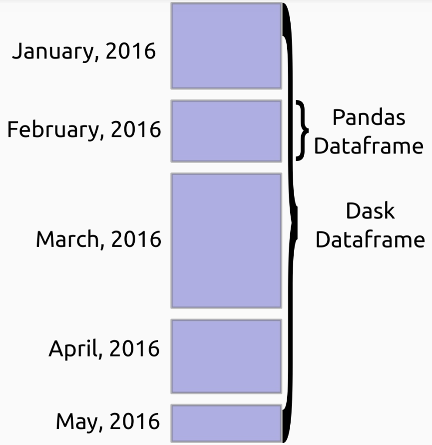
- Dask **bags** are loosely analogous to Spark RDDs
- Unordered collection of generic Python objects
    - Partitions into subsets (sub-bags)
- Implements some basic operations
    - map, filter, join, sum, etc.
- A good choice for initial processing and structured objects
    - If your data is tabular 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
```

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)

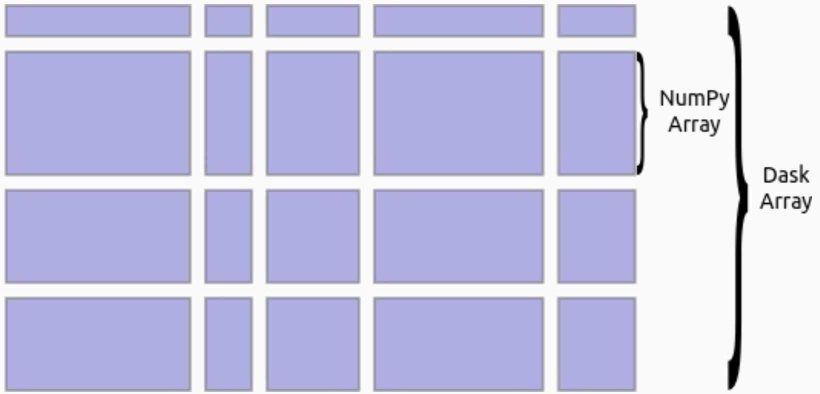
```
import dask.dataframe as dd

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

Schedulers: how does code run?

- On a single machine, you have three options:
  - threads
  - processes
  - synchronous (debugging mode)
- Or you can run on a cluster
  - Execution can look very similar to Spark (eg YARN jobs)
  - Data is transferred automatically / as needed

Schedulers: how does code run?

- On a single machine, you have three optio
  - threads
  - processes
  - synchronous (debugging mode)

- Or you can run on a cluster
  - Execution can look very similar to Spark (eg YAR
  - Data is transferred automatically / as needed

Threads

+ Single python process
+ Shared memory between threads
- Certain python operations can block computation for all threads

Processes

+ Shared data must be serialized and sent between processes
+ Processes do not block each other

Scaling down: why a single machine?

- **Many "big data" jobs do not really need a cluster!**
  - Data fits on a hard drive, but not in RAM ("core memory")
  - NumPy (& friends) generally assume fully observed, in-memory data
- In this case, working on small chunks at a time is sufficient
  - Coding this by hand can be tedious / error-prone
- Dask simplifies this, and makes it easy to migrate to a cluster if necessary

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?

- Pros for Spark:
  - More mature, possibly more stable / safe
  - Probably faster / better optimized for DataFrame crunching
  - Better support for large graph data