



DASK FOR PARALLEL COMPUTING CHEAT SHEET

See full Dask documentation at: <http://dask.pydata.org/>

These instructions use the conda environment manager. Get yours at <http://bit.ly/getconda>

DASK QUICK INSTALL

Install Dask with conda `conda install dask`

Install Dask with pip `pip install dask[complete]`

DASK COLLECTIONS

EASY TO USE BIG DATA COLLECTIONS

DASK DATAFRAMES

PARALLEL PANDAS DATAFRAMES FOR LARGE DATA

Import `import dask.dataframe as dd`

Read CSV data `df = dd.read_csv('my-data-*.csv')`

Read Parquet data `df = dd.read_parquet('my-data.parquet')`

Filter and manipulate data with Pandas syntax `df['z'] = df.x + df.y`

Standard groupby aggregations, joins, etc. `result = df.groupby(df.z).y.mean()`

Compute result as a Pandas dataframe `out = result.compute()`

Or store to CSV, Parquet, or other formats `result.to_parquet('my-output.parquet')`

EXAMPLE

```
df = dd.read_csv('filenames.*.csv')
df.groupby(df.timestamp.day)\
    .value.mean().compute()
```

DASK ARRAYS

PARALLEL NUMPY ARRAYS FOR LARGE DATA

Import `import dask.array as da`

Create from any array-like object `import h5py
dataset = h5py.File('my-data.hdf5')['/group/dataset']`

Including HFD5, NetCDF, or other on-disk formats. `x = da.from_array(dataset, chunks=(1000, 1000))`

Alternatively generate an array from a random distribution. `da.random.uniform(shape=(1e4, 1e4), chunks=(100, 100))`

Perform operations with NumPy syntax `y = x.dot(x.T - 1) - x.mean(axis=0)`

Compute result as a NumPy array `result = y.compute()`

Or store to HDF5, NetCDF or other on-disk format `out = f.create_dataset(...)
x.store(out)`

EXAMPLE

```
with h5py.File('my-data.hdf5') as f:
    x = da.from_array(f['/path'], chunks=(1000, 1000))
    x -= x.mean(axis=0)
    out = f.create_dataset(...)
    x.store(out)
```

DASK BAGS

PARALLEL LISTS FOR UNSTRUCTURED DATA

Import `import dask.bag as db`

Create Dask Bag from a sequence `b = db.from_sequence(seq, npartitions)`

Or read from text formats `b = db.read_text('my-data-*.json')`

Map and filter results `import json
records = b.map(json.loads)
 .filter(lambda d: d["name"] == "Alice")`

Compute aggregations like mean, count, sum `records.pluck('key-name').mean().compute()`

Or store results back to text formats `records.to_textfiles('output-*.json')`

EXAMPLE

```
db.read_text('s3://bucket/my-data-*.json')
    .map(json.loads)
    .filter(lambda d: d["name"] == "Alice")
    .to_textfiles('s3://bucket/output-*.json')
```

DASK COLLECTIONS (CONTINUED)

ADVANCED

Read from distributed file systems or cloud storage	<code>df = dd.read_parquet('s3://bucket/myfile.parquet')</code>
Prepend prefixes like <code>hdfs://</code> , <code>s3://</code> , or <code>gcs://</code> to paths	<code>b = db.read_text('hdfs:///path/to/my-data.*.json')</code>
Persist lazy computations in memory	<code>df = df.persist()</code>
Compute multiple outputs at once	<code>dask.compute(x.min(), x.max())</code>

CUSTOM COMPUTATIONS

FOR CUSTOM CODE AND COMPLEX ALGORITHMS

DASK DELAYED

LAZY PARALLELISM FOR CUSTOM CODE

Import	<code>import dask</code>
Wrap custom functions with the <code>@dask.delayed</code> annotation	<code>@dask.delayed</code> <code>def load(filename):</code> <code> ...</code>
Delayed functions operate lazily, producing a task graph rather than executing immediately	<code>@dask.delayed</code> <code>def process(data):</code> <code> ...</code>
Passing delayed results to other delayed functions creates dependencies between tasks	<code>load = dask.delayed(load)</code> <code>process = dask.delayed(process)</code> <code>...</code>
Call functions in normal code	<code>data = [load(fn) for fn in filenames]</code> <code>results = [process(d) for d in data]</code>
Compute results to execute in parallel	<code>dask.compute(results)</code>
CONCURRENT.FUTURES	ASYNCHRONOUS REAL-TIME PARALLELISM
Import	<code>from dask.distributed import Client</code>
Start local Dask Client	<code>client = Client()</code>
Submit individual task asynchronously	<code>future = client.submit(func, *args, **kwargs)</code>
Block and gather individual result	<code>result = future.result()</code>
Process results as they arrive	<code>for future in as_completed(futures):</code> <code> ...</code>
EXAMPLE	<code>L = [client.submit(read, fn) for fn in filenames]</code> <code>L = [client.submit(process, future) for future in L]</code> <code>future = client.submit(sum, L)</code> <code>result = future.result()</code>

SET UP CLUSTER

HOW TO LAUNCH ON A CLUSTER

MANUALLY

Start scheduler on one machine	<code>\$ dask-scheduler</code> Scheduler started at <code>SCHEDULER_ADDRESS:8786</code>
Start workers on other machines Provide address of the running scheduler	<code>host1\$ dask-worker SCHEDULER_ADDRESS:8786</code> <code>host2\$ dask-worker SCHEDULER_ADDRESS:8786</code>
Start Client from Python process	<code>from dask.distributed import Client</code> <code>client = Client('SCHEDULER_ADDRESS:8786')</code>

ON A SINGLE MACHINE

Call <code>Client()</code> with no arguments for easy setup on a single host	<code>client = Client()</code>
--	--------------------------------

CLOUD DEPLOYMENT

See <code>dask-kubernetes</code> project for Google Cloud	<code>pip install dask-kubernetes</code>
See <code>dask-ec2</code> project for Amazon EC2	<code>pip install dask-ec2</code>

MORE RESOURCES

User Documentation	dask.pydata.org
Technical documentation for distributed scheduler	distributed.readthedocs.org
Report a bug	github.com/dask/dask/issues