# 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')`<br>`df.groupby(df.timestamp.day)\`<br>`.value.mean().compute()` |
| **Dask Arrays** | *Parallel NumPy arrays for large data* |
| Import | `import dask.array as da` |
| Create from any array-like object.<br>This includes HFD5, NetCDF, or other on-disk formats.<br>Alternatively generate an array from a random distribution. | `import h5py`<br>`dataset = h5py.File('my-data.hdf5')['/group/dataset']`<br>`x = da.from_array(dataset, chunks=(1000, 1000))`<br>`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 non-disk format | `out = f.create_dataset(...)`<br>`x.store(out)` |
| EXAMPLE | `with h5py.File('my-data.hdf5') as f:`<br>`x = da.from_array(f['/path'], chunks=(1000, 1000))`<br>`x -= x.mean(axis=0)`<br>`out = f.create_dataset(...)`<br>`x.store(out)` |

| DASK COLLECTIONS (CONT.) | |
|---|---|
| **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`<br>`records = b.map(json.loads)`<br>`  .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')`<br>`  .map(json.loads)`<br>`  .filter(lambda d: d["name"] == "Alice")`<br>`  .to_textfiles('s3://bucket/output.*.json')` |
| **Advanced** | |
| Read from distributed file systems or cloud storage<br>Prepend prefixes like hdfs://, s3://, or gcs:// to paths | `df = dd.read_parquet('s3://bucket/myfile.parquet')`<br>`b = db.read_text('hdfs:///path/to/my-data.*.json')` |
| Persist lazy computations in memory | `df = df.persist()` |
| Compute multiple outputs at once | `dask.compute(x.min(), x.max())` |
| **CUSTOM COMPUTATIONS** | *For custom code and complex algorithms* |
| **Dask Delayed** | *Lazy parallelism for custom code* |
| Import | `import dask` |
| Wrap custom functions with the @dask.delayed annotation<br>Delayed functions operate lazily, producing a task graph rather than executing immediately<br>Passing delayed results to other delayed functions creates dependencies between tasks | `@dask.delayed`<br>`def load(filename):`<br>`  ...`<br>`@dask.delayed`<br>`def process(data):`<br>`  ...` |
| Call functions in normal code | `data = [load(fn) for fn in filenames]`<br>`results = [process(d) for d in data]` |
| Compute results to execute in parallel | `dask.compute(results)` |
| **Concurrent.futures** | *Asynchronous real-time parallelism* |
| Import | `from dask.distributed import Client` |
| Start local Dask Client | `client = Client()` |
| Submit individual task asynchronously | `future = client.submit(func, *args, **kwargs)` |
| Block and gather individual result | `result = future.result()` |
| Process results as they arrive | `for future in as_completed(futures):`<br>`  ...` |
| EXAMPLE | `L = [client.submit(read, fn) for fn in filenames]`<br>`L = [client.submit(process, future) for future in L]`<br>`future = client.submit(sum, L)`<br>`result = future.result()` |

| SET UP CLUSTER | https://docs.dask.org/en/latest/setup.html |
|---|---|
| **Manually** | |
| Start scheduler on one machine | `$ dask-scheduler`<br>`Scheduler started at SCHEDULER_ADDRESS:8786` |
| Start workers on other machines<br>Provide address of the running scheduler | `host1$ dask-worker SCHEDULER_ADDRESS:8786`<br>`host2$ dask-worker SCHEDULER_ADDRESS:8786` |
| Start Client from Python process | `from dask.distributed import Client`<br>`client = Client('SCHEDULER_ADDRESS:8786')` |
| **On a single machine** | |
| Call Client() with no arguments for easy setup on a single host | `client = Client()` |
| **Cloud Deployment** | |
| Install Dask on Kubernetes | `pip install dask-kubernetes` |
| Install Dask on AWS / GCP / AzureML | `pip install dask-cloudprovider` |
| Multi-user server for managing Dask Clusters | `pip install dask-gateway` |

## More resources

| | |
|---|---|
| User Documentation | https://docs.dask.org |
| Technical Documentation for distributed scheduler | https://distributed.dask.org |
| Ask for help on StackOverflow | https://stackoverflow.com/questions/tagged/dask |
| Report a bug | https://github.com/dask/dask/issues |