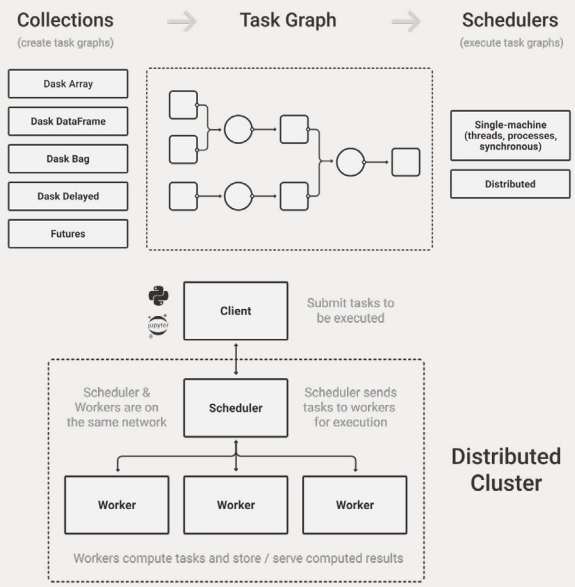
DASK Cheat Sheet

Dask

Summary
Library for parallel and distributed computing in Python.



Collections

Summary
- Provide the APIs used to write Dask code.
- Create task graphs (stepwise instructions) for executing the computation in parallel.
- Uses all of the cores on your computer
- Work on datasets larger than available memory by effectively streaming data from disk

High-Level collections:

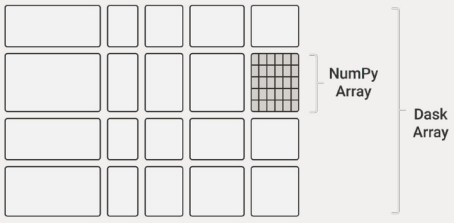
Arrays	Parallel NumPy	Bags	Parallel lists
DataFrames	Parallel Pandas	Machine Learning	Parallel Scikit-Learn, XGBoost, and others

Low-Level APIs:

Delayed	Lazy parallel function evaluation	Futures	Real-time parallel function evaluation

Dask Array

Summary
For parallel NumPy
- Composed of many NumPy Arrays




```
import dask.array as da
x = da.random.random((10000, 10000), chunks=(1000, 1000))

import h5py
dataset = h5py.File('data.hdf5')['/data/path']
x = da.from_array(dataset, chunks=(1000, 1000))
result = x.sum()
result.compute()
```

Dask Array (continued)

Dask Array supports most of the NumPy interface:

Arithmetic and scalar mathematics	`+, *, exp, log, etc.`
Reductions along axes	`sum(), mean(), std(), sum(axis=0), etc.`
Tensor contractions, dot products, matrix multiply	`tensordot`
Axis reordering/transpose	`transpose`
Slicing and Indexing	`x[:50, 500:100:-2]`
Linear algebra	`svd, qr, solve, solve_triangular, lstsq`

Limitations:
- Arrays with unknown shapes do not support all operations.
- `tolist` and `np.linalg` not implemented
- Call `topk` instead of `sort`

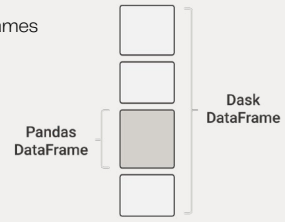
Dask DataFrame

Summary
For parallel pandas
- Composed of multiple small pandas DataFrames

```
import dask.dataframe as dd

df = dd.read_csv('data.csv')
df.head()

df_new = df[df.y == 'a'].x + 1
df_new.compute()
```



Implements pandas interface:

Element-wise operations	`df.x + df.y`
Row-wise selections	`df[df.x > 100]`
Common aggregations	`df.x.max(), df.max()`
Date time/string accessors	`df.timestamp.month`
groupby-aggregate (with common aggregations)	`df.groupby(df.x).y.max(), df.groupby('x').max()`
Join on index	`dd.merge(df1, df2, left_index=True, right_index=True)` or `dd.merge(df1, df2, on=['idx', 'x'])` where idx is the index name for both df1 and df2
Join with Pandas DataFrames	`dd.merge(df1, df2, on='id')`

Limitations
- Expensive to set new index from unsorted column
- Operations like `groupby-apply` and `join` on unsorted columns are also expensive

Delayed

Summary
Parallelize custom algorithms
- Evaluates computations lazily

```
x = dask.delayed(inc)(1)
y = dask.delayed(inc)(2)
z = dask.delayed(add)(x, y)

@dask.delayed
def inc(x):
    return x + 1

z.compute()
z.visualize()
```

Futures

Summary
Extends Python's `concurrent.futures` interface
- Good for arbitrary task scheduling (like Delayed)

Futures (continued)

- Immediate computation (not lazy)

```
x = client.submit(inc, 10) # Submit function application to scheduler, returns Future
z = client.submit(add, x, x) # Pass Futures as input

futures = client.map(inc, range(1000)) # Map function on sequence, returns Future

c.result() # Wait until computation completes, gather result to local process

results = client.gather(futures) # Gather futures from distributed memory
df = pd.read_csv('data.csv')
remote_df = client.scatter(df) # Scatter data into distributed memory

future.cancel() # delete data even if other futures point to it

wait(futures) # Wait until all futures are complete
```

Dask Bag

Summary

Implements `map`, `filter`, `fold`, `groupby`, etc. on collections of generic Python objects.

- Parallel computation
- Lazy evaluation

```
import dask.bag as db

b = db.read_text('data/*.json').map(json.loads)
b.take(2)

b.filter(lambda record: record['count'] > 10).take(2)

b.map(lambda record: record['text']).take(2)

b.count().compute()
```

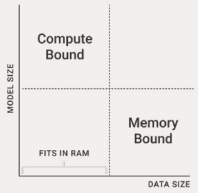
Limitations:

- Relies on the multiprocessing scheduler
- Computations slower than Dask Array/DataFrame
- `groupby` is slow, use `foldby` instead

Dask-ML

Summary

For distributed machine learning



- **Compute bound**: Use Dask cluster to parallelize regular collections (NumPy, ndarray, pandas, DataFrame, etc.) on many machines to scale model size.

```
from dask.distributed import Client
import joblib

client = Client(processes=False)

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

- **Memory bound**: Use high-level collections (Dask Array, Dask DataFrame, etc.) with Dask-ML estimators designed for it.

```
from dask_ml.preprocessing import Categorizer

from dask_ml.linear_model import LogisticRegression

from dask_ml.cluster import KMeans

from dask_ml.naive_bayes import GaussianNB

from dask_ml.model_selection import GridSearchCV
```

Diagnostic Dashboards

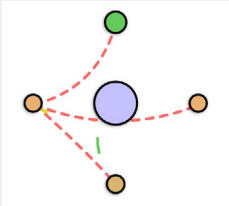
Summary

Real-time visualization of cluster state

- Access it on localhost:8787, or
- Using the JupyterLab extension for Dask

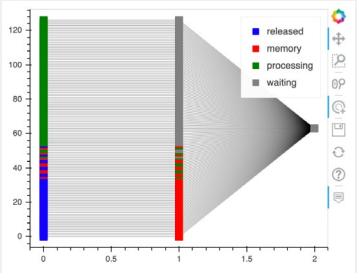
Visualizations:

- **Cluster Map**: Interactions between workers and the scheduler

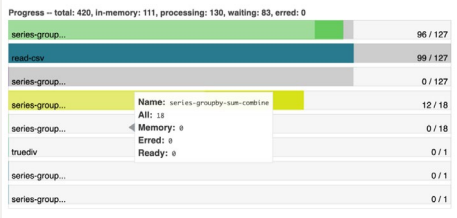


- **Task Stream**: Tasks being performed at a given time

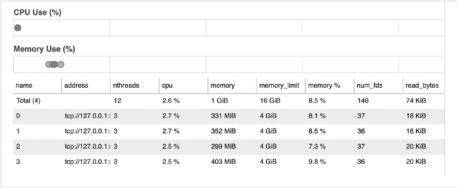
- **Task Graph**: Stepwise process and current status of parallel computation



- **Progress Bar**: Progress being made on each task during execution



- **Workers**: CPU and memory usage of workers



References

- Dask Documentation
- Dask Examples
- Dask Tutorial
- Best Practices
- Community