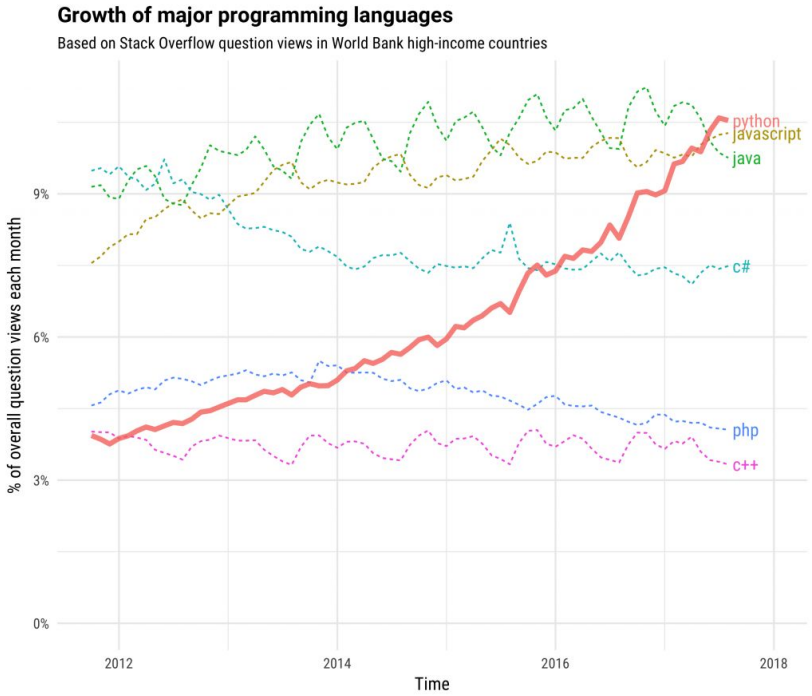
Scaling your workflow with Dask

Tom Augpsurger / Anaconda

Problem: Python is powerful, but doesn't scale well

Python is great for medium data




```
>>> import pandas as pd
>>> df = pd.read_parquet("accounts")
MemoryError
```

Powerful: Leading platform today for analytics
Limited: Fails for big data or scalable computing
Frustration: Alternatives fracture development

https://stackoverflow.blog/2017/09/06/incredible-growth-python/

DASK

General purpose Python library for parallelism

Scales existing libraries, like Numpy, Pandas, and Scikit-Learn

Flexible enough to build complex and custom systems

Accessible for beginners, secure and trusted for institutions

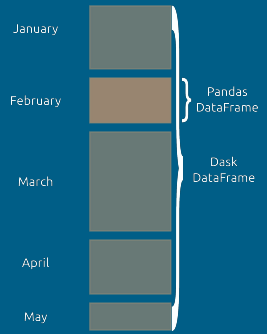
Dask accelerates the existing Python ecosystem

Built alongside the current community

NumPy

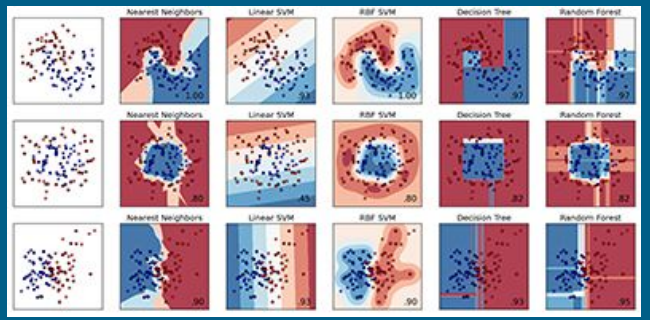

```
import numpy as np

x = np.ones((1000, 1000))
x + x.T - x.mean(axis=0
```

Pandas




```
import pandas as pd

df = pd.read_csv(“file.csv”)
df.groupby(“x”).y.mean()
```

Scikit-Learn



```
from scikit_learn.linear_model \
    import LogisticRegression

lr = LogisticRegression()
lr.fit(data, labels)
```

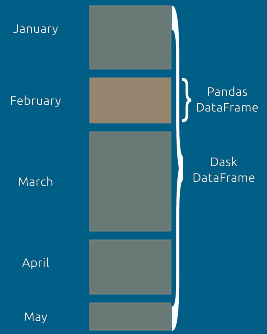
Dask accelerates the existing Python ecosystem

Built alongside the current community

NumPy

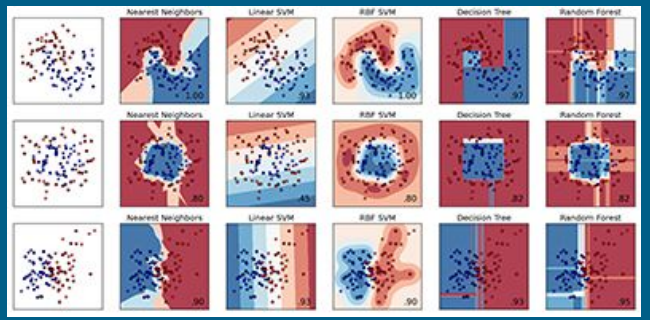

```
import dask.array as da

x = da.ones((10000, 10000))
x + x.T - x.mean(axis=0
```

Pandas




```
import dask.dataframe as dd

df = dd.read_csv("s3://*.csv")
df.groupby("x").y.mean()
```

Scikit-Learn



```
from dask_ml.linear_model \
    import LogisticRegression

lr = LogisticRegression()
lr.fit(data, labels)
```

Use Case: Banking with Capital One

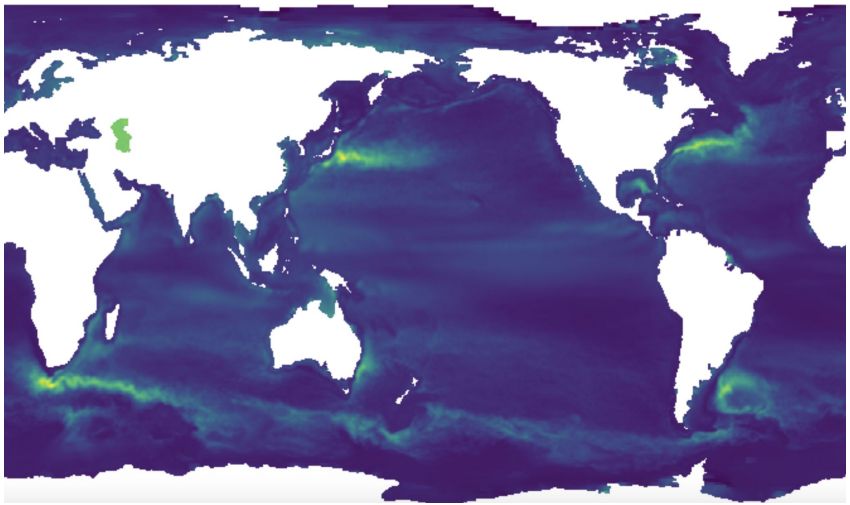
Data cleaning, Feature engineering, and Machine learning

Capital One engineers analyze financial and credit data to build cloud-based machine learning models

- Datasets range in scale from **1 GB - 100 TB**
- CSV and Parquet datasets **100s - 1000s of columns**
- Data Cleaning, Feature Selection, Engineering, Training, Validation, and Governance
- Dask DataFrames used to train with **dask-ml** and **XGBoost**
- **10x** speed up in computational performance with Dask

Use Case: Analyze Sea Levels

Columbia/NCAR leverage Dask Array to understand our planet



Sea level altitude variability over 30 years

Learn more about Pangeo in this talk

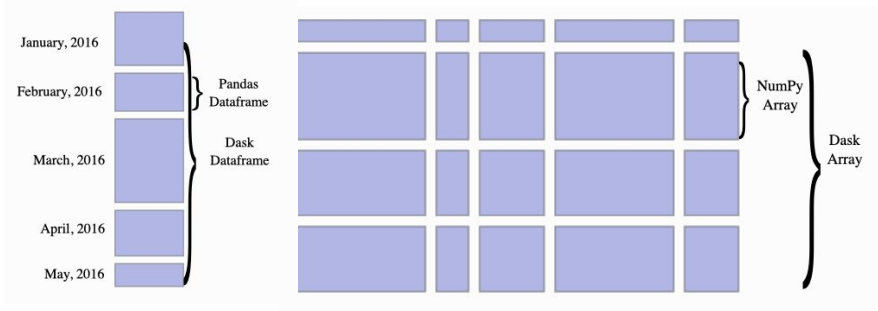
Pangeo researchers analyze simulated and observed climate and imaging data

- 1 GB - 100 TB
- HPC and Cloud
- HDF5/NetCDF/Zarr storage
- Interactive computing

Includes collaborators NASA, NOAA, USGS, UK-Met, CSIRO, and various industries

Concepts

High level parallelism



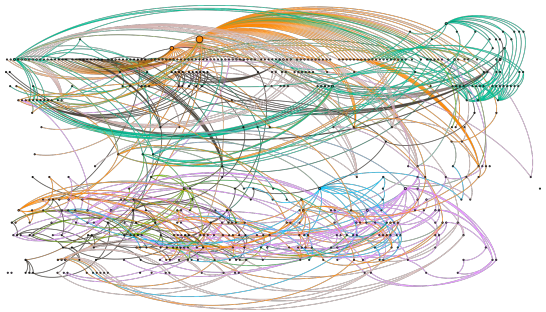
Scaling familiar interfaces

High Level: Parallel versions of popular libraries

Scales NumPy, Pandas, Scikit-Learn, XGBoost, RAPIDS, …

Similar to Spark, MapReduce, Databases

Low level parallelism



Enabling advanced workloads

Low Level: Distributed, real-time task scheduling.

Scales custom systems, enables sophisticated workflows

Similar to Make, Airflow, Flink, …

High level parallelism

Handles important common cases

Common entry point and familiar

Low level parallelism

Enables advanced situations

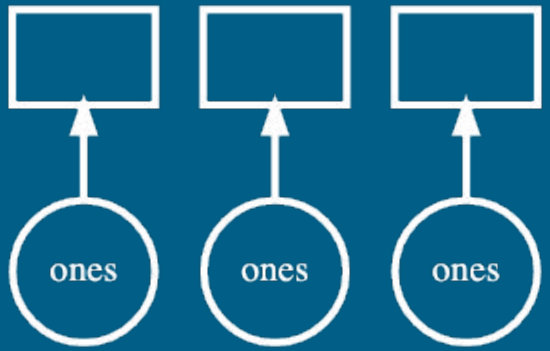
Critical for outside-of-the-box problems

Parallel Algorithms

Dask parallelizes common Python operations

Dask APIs produce task graphs

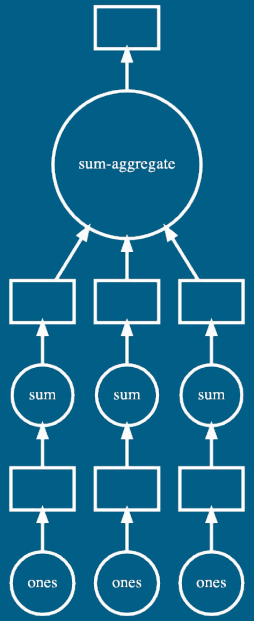
Create a 1D array of ones




```
import dask.array as da

x = da.ones(15, chunks=(5,))
```

Dask APIs produce task graphs

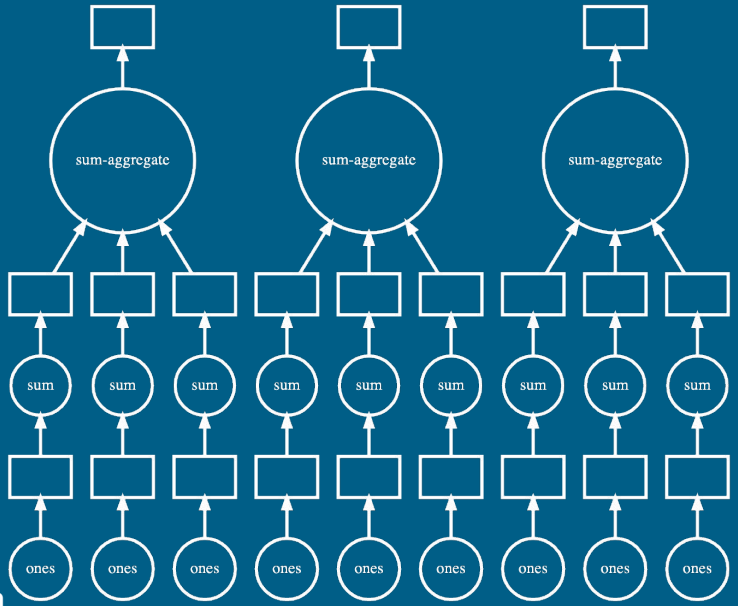
Sum that array




```
import dask.array as da

x = da.ones(15, chunks=(5,))
x.sum()
```

Dask APIs produce task graphs

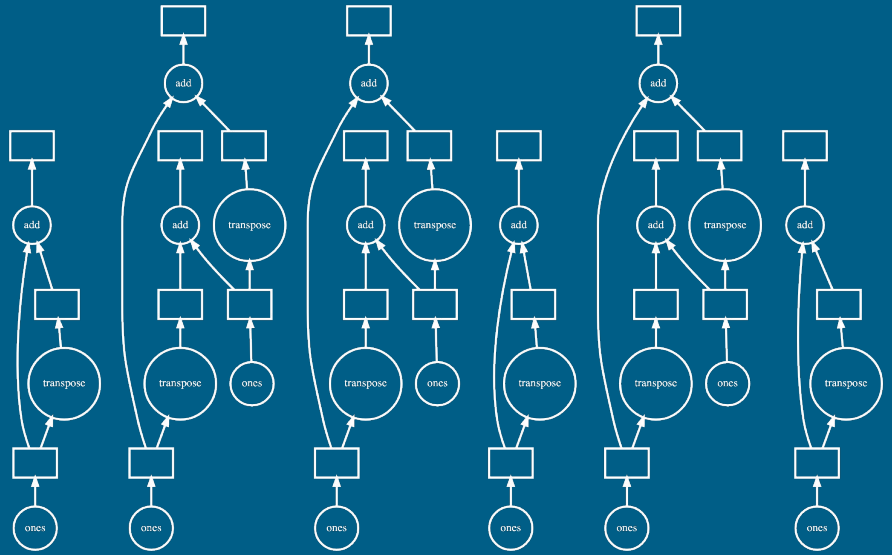
Sum a 2D array of ones




```
import dask.array as da

x = da.ones((15, 15), chunks=(5, 5))
x.sum(axis=0)
```

Dask APIs produce task graphs

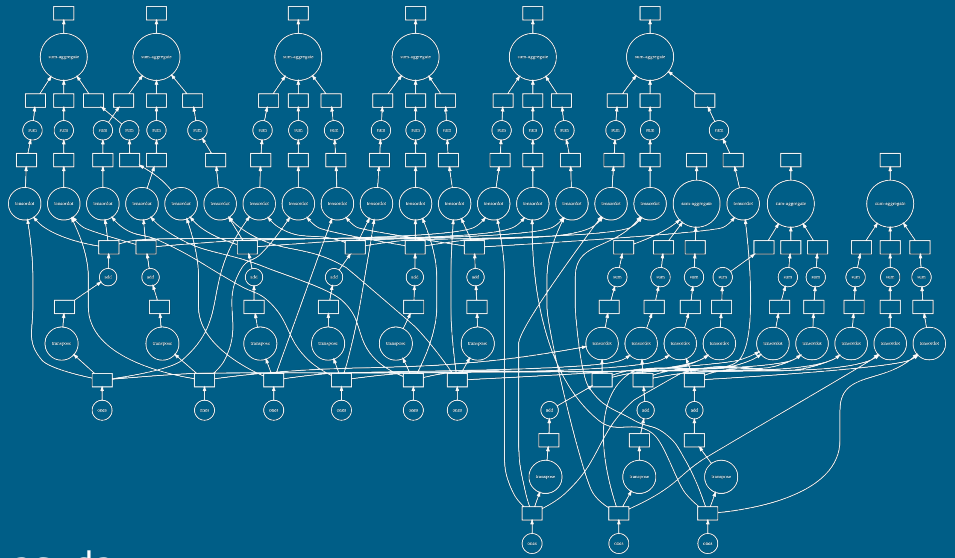
Add array to its transpose




```
import dask.array as da

x = da.ones((15, 15), chunks=(5, 5))
x + x.T
```

Dask APIs produce task graphs

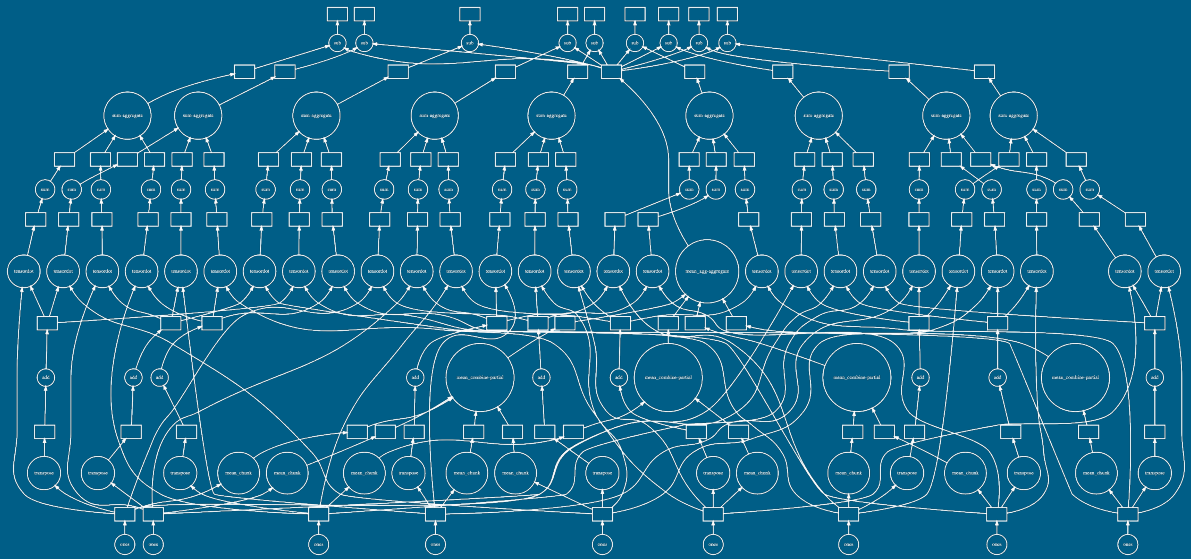
Add in a matrix multiply



```
import dask.array as da

x = da.ones((15, 15), chunks=(5, 5))
x.dot(x.T + 1)
```

Dask APIs produce task graphs

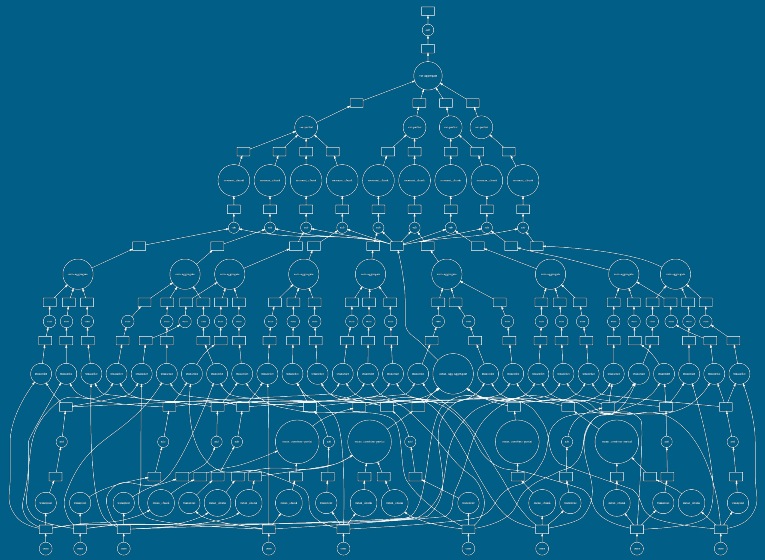
Add in a matrix multiply and a mean



```
import dask.array as da

x = da.ones((15, 15), chunks=(5, 5))
x.dot(x.T + 1) - x.mean(axis=0)
```

Dask APIs produce task graphs

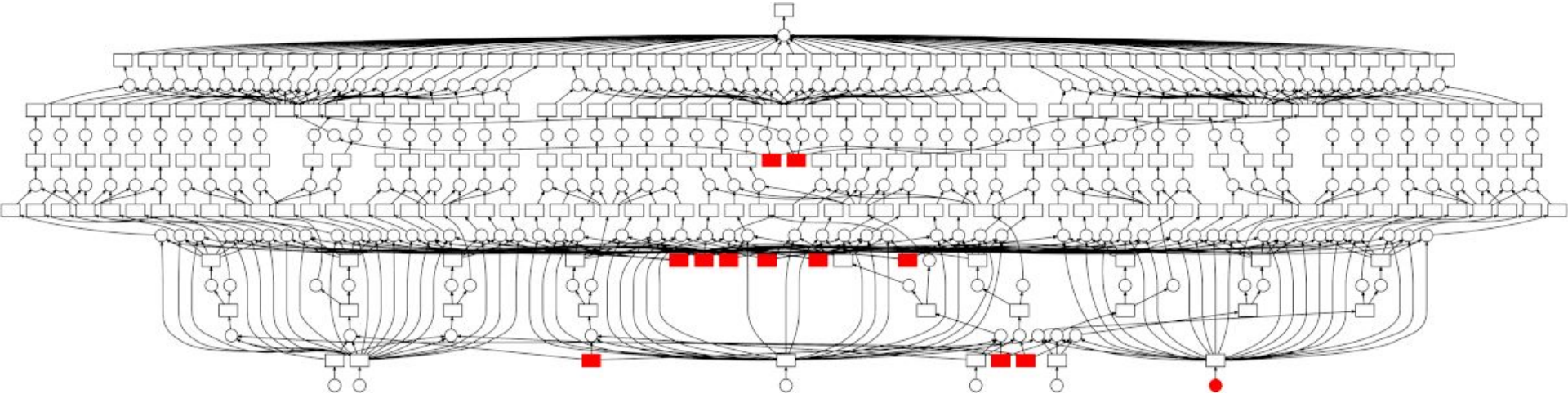
Add in a matrix multiply and a mean and a standard deviation



```
import dask.array as da

x = da.ones((15, 15), chunks=(5, 5))
(x.dot(x.T + 1) - x.mean(axis=0)).std()
```

Dask schedulers then compute them

We execute task graphs efficiently over parallel hardware



Scikit-Learn cross-validated grid search over a pipeline

Dask APIs build graphs, Dask schedulers execute them

Collections
(create task graphs)

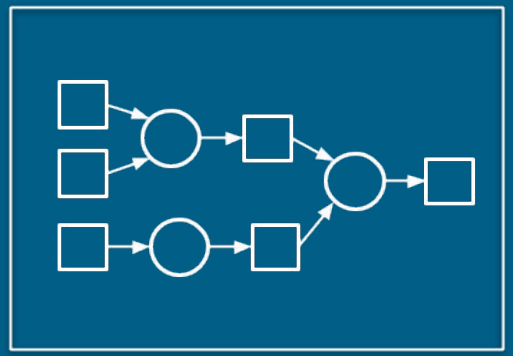
→

Task Graph

→

Schedulers
(execute task graphs)

- Dask Array
- Dask DataFrame
- Dask Bag
- Dask Delayed
- Futures



- Single-machine (threads, processes, synchronous)
- Distributed

A variety of frontend APIs

A variety of backend Execution systems

Demo

Deployment

Dask deploys on all major resource managers

We execute task graphs efficiently over parallel hardware

Cloud

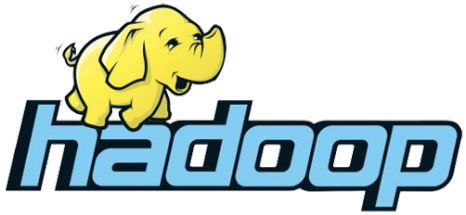
Google Cloud Platform


```
cluster = KubeCluster()
cluster = ECSCluster()
cluster = FargateCluster()

df = dd.read_parquet(...)
```

HPC


```
cluster = PBSCluster()
cluster = LSFCluster()
cluster = SLURMCluster()
...
df = dd.read_parquet(...)
```

Hadoop/Spark




```
cluster = YarnCluster()


df = dd.read_parquet(...)
```

Dask deploys on all major resource managers

We execute task graphs efficiently over parallel hardware

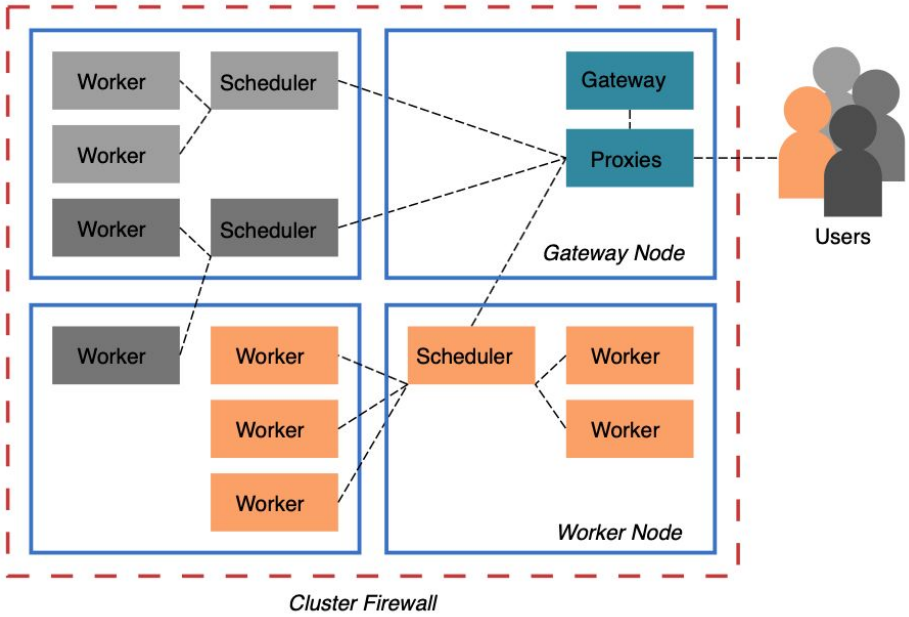
```python
from dask_kubernetes import KubeCluster

cluster = KubeCluster.from_yaml('worker-spec.yml')
cluster.scale_up(10)  # specify number of nodes explicitly

cluster.adapt(minimum=1, maximum=100)  # or dynamically scale based on current workload
```

```yaml
# worker-spec.yml

kind: Pod
metadata:
  labels:
    foo: bar
spec:
  restartPolicy: Never
  containers:
  - image: daskdev/dask:latest
    imagePullPolicy: IfNotPresent
    args: [dask-worker, --nthreads, '2', --no-bokeh, --memory-limit, 6GB, --death-timeout, '60'
    name: dask
    env:
      - name: EXTRA_PIP_PACKAGES
        value: fastparquet git+https://github.com/dask/distributed
    resources:
      limits:
        cpu: "2"
        memory: 6G
      requests:
        cpu: "2"
        memory: 6G
```

```yaml
# /path/to/spec.yaml
name: dask
queue: myqueue

services:
  dask.worker:
    # Restrict workers to GPU nodes only
    node_label: GPU
    # Don't start any workers initially
    instances: 0
    # Workers can infinite number of times
    max_restarts: -1
    # Restrict workers to 4 GiB and 2 cores each
    resources:
      memory: 4 GiB
      vcores: 2
    # Distribute this python environment to every worker node
    files:
      environment: /path/to/my/environment.tar.gz
    # The bash script to start the worker
    # Here we activate the environment, then start the worker
    script: |
      source environment/bin/activate
      dask-yarn services worker
```

Centralized deployment with Dask Gateway

Let IT handle deployments, users handle data science



- Centrally managed
- Secure by default
- Flexible cluster backends
- Robust to failure

Centralized deployment with Dask Gateway

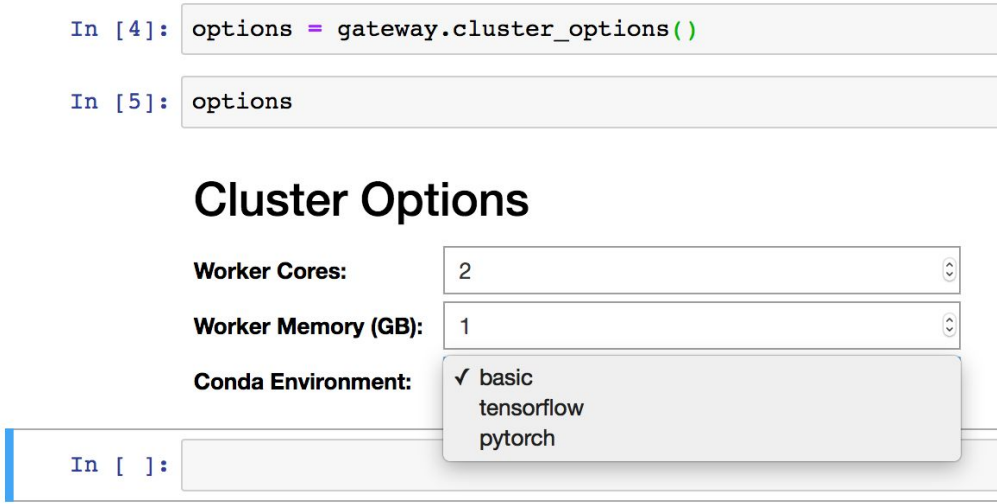
Let IT handle deployments, users handle data science

```
>>> from dask_gateway import Gateway

>>> gateway = Gateway()

>>> gateway.list_clusters()

[]

>>> cluster = gateway.new_cluster()

>>> cluster.scale(40)
```

Centralized deployment with Dask Gateway

Let IT handle deployments, users handle data science

- Cluster admins provide configuration options
- Set user resource limits



Wrap-up

Pleasant to use and adopt

Dask is designed for experts and novices alike

Beautiful interactive dashboard

Builds intuition on parallel performance

Familiar APIs and data models

Dask looks and feels like well known libraries

Co-developed with the ecosystem

Built by NumPy, Pandas, and Scikit-Learn devs

Dask complements the existing ecosystem

Software Community

Run by people you know, built by people you trust

- **Developed**: **300** contributors, **20** active maintainers
  From: Numpy, Pandas, Scikit-Learn, Jupyter, and more
- **Used**: **10k** weekly visitors to documentation
- **Discussed**: Dask is the most common parallel framework at PyData/SciPy/PyCon conferences today.
- **Safe:** BSD-3 Licensed, fiscally sponsored by NumFOCUS, community governed

Product portfolio

ANACONDA.

ANACONDA.
INDIVIDUAL EDITION

Open-source tools for individual data scientists

ANACONDA.
TEAM EDITION

Data science packages in an enterprise-grade repository

ANACONDA.
ENTERPRISE EDITION

End-to-end machine learning ops platform

ANACONDA.
CLOUD

On-demand access to collaboration and compute resources in a subscription offering; coming later in 2020

Documentation: dask.org

Examples: examples.dask.org

Community: github.com/dask