MAS DSE 230
Scalable Analytics
Dask

Mai H. Nguyen

TODAY'S TOPICS

- Model Evaluation
- Dask
- Cloud Computing
- AWS

SPARK REVIEW

- Spark
  - History
  - RDDs
  - DataFrames
  - Spark Architecture
  - Spark API
  - Spark Core & Libraries

SPARK

- Computing platform for distributed computing
  - Built-in parallelism & fault-tolerance on commodity cluster
  - Extends MapReduce operations
  - Provides interactive querying, iterative analytics, streaming processing
  - Adopts Python Pandas style of function calls
  - Goals: speed, ease of use, generality, unified platform
- History
  - Research project began in 2009 at UC Berkeley's AMPlab
  - Paper published in 2010
  - Contributed to Apache Software Foundation in 2013
  - Commercial version by Databricks
- Compatible with Hadoop

RDDs

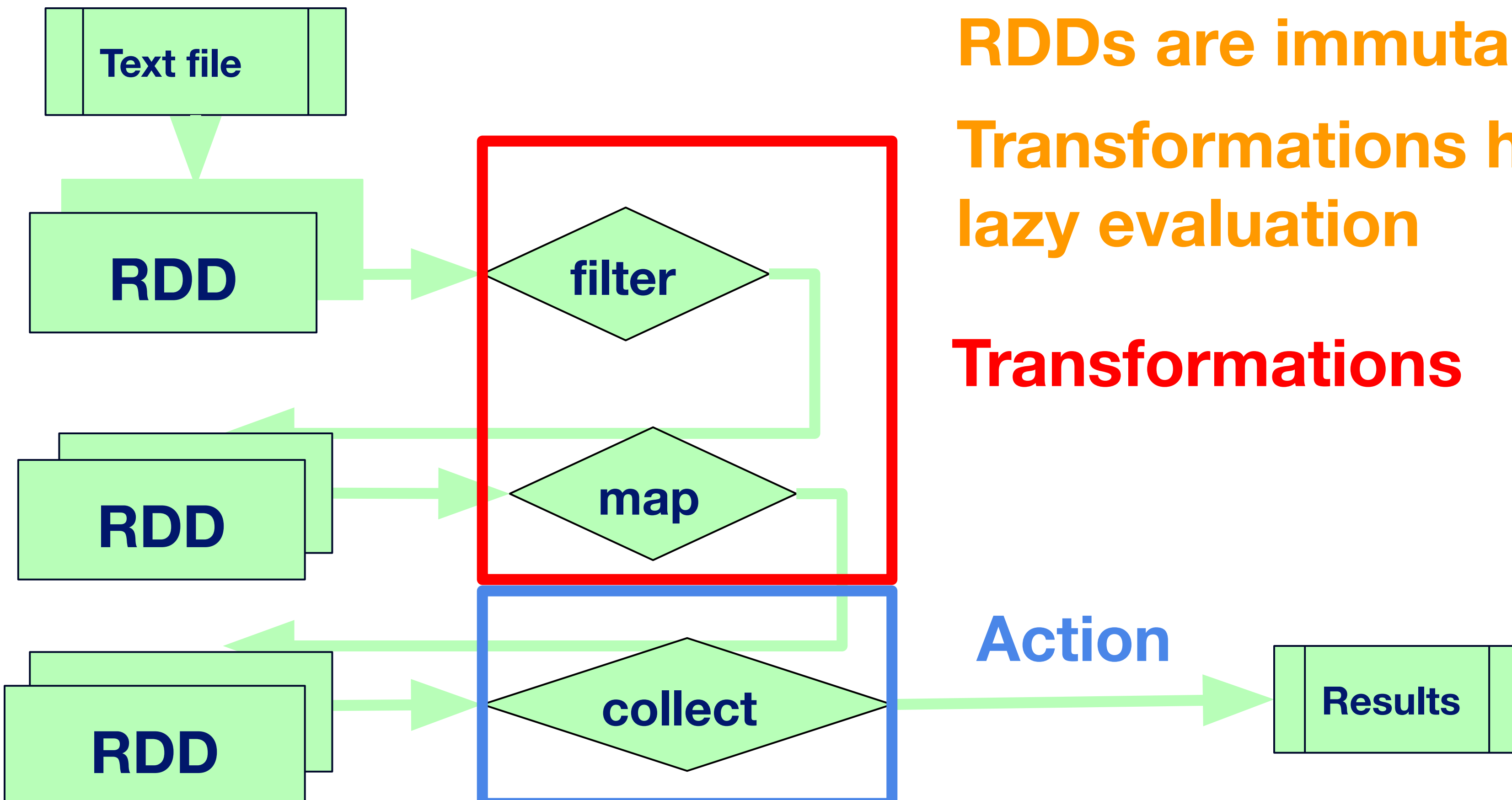
- RDDs
  - Abstraction of data as distributed collection of objects
- Resilient Distributed **Dataset**
  - Collection of data
    - From files in local filesystem (text, JSON, etc.)
    - From data store (HDFS, RDBMS, NoSQL, etc.)
    - Created from another RDD
- Resilient **Distributed** Dataset
  - Data is divided into partitions
  - Partitions are distributed across nodes in cluster
- **Resilient** Distributed Dataset
  - Provides resilience (e.g., fault tolerance) to failures
  - History of operations performed on each partition is tracked to provide lineage-based fault tolerance
- All provided automatically by Spark engine

PROCESSING RDDs

- RDDs can be processed using 2 types of operations
  - **Transformation**: Creates new RDD from existing RDD
  - **Action**: Runs computation(s) on RDD and returns value

RDDs are immutable

Transformations have lazy evaluation



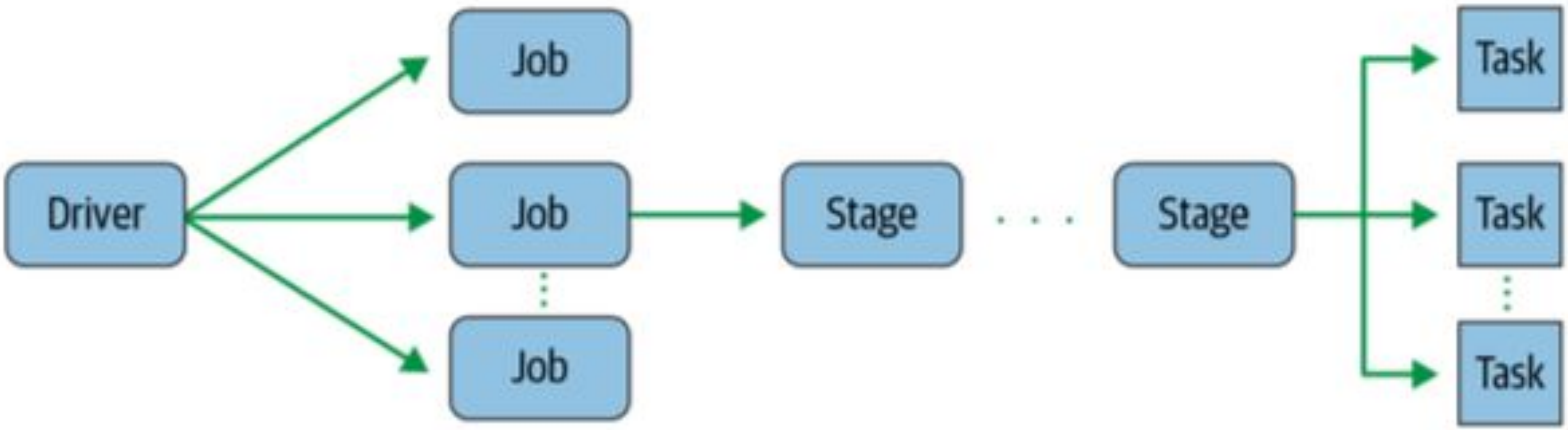
LAZY EVALUATION

- Transformations on RDDs have **lazy evaluation**
  - Transformation are not immediately processed
  - Plan of operations is built
- Operations executed when **action** is performed
  - i.e., actions force computation
- Allows for optimizations in generating physical plan
- Example:
  - filtered = strings.filter(strings.value.contains(“Spark”))
    - Nothing is returned
  - filtered.count()
    - ‘filter’ is performed, and count is returned

DATAFRAMES & DATASETS

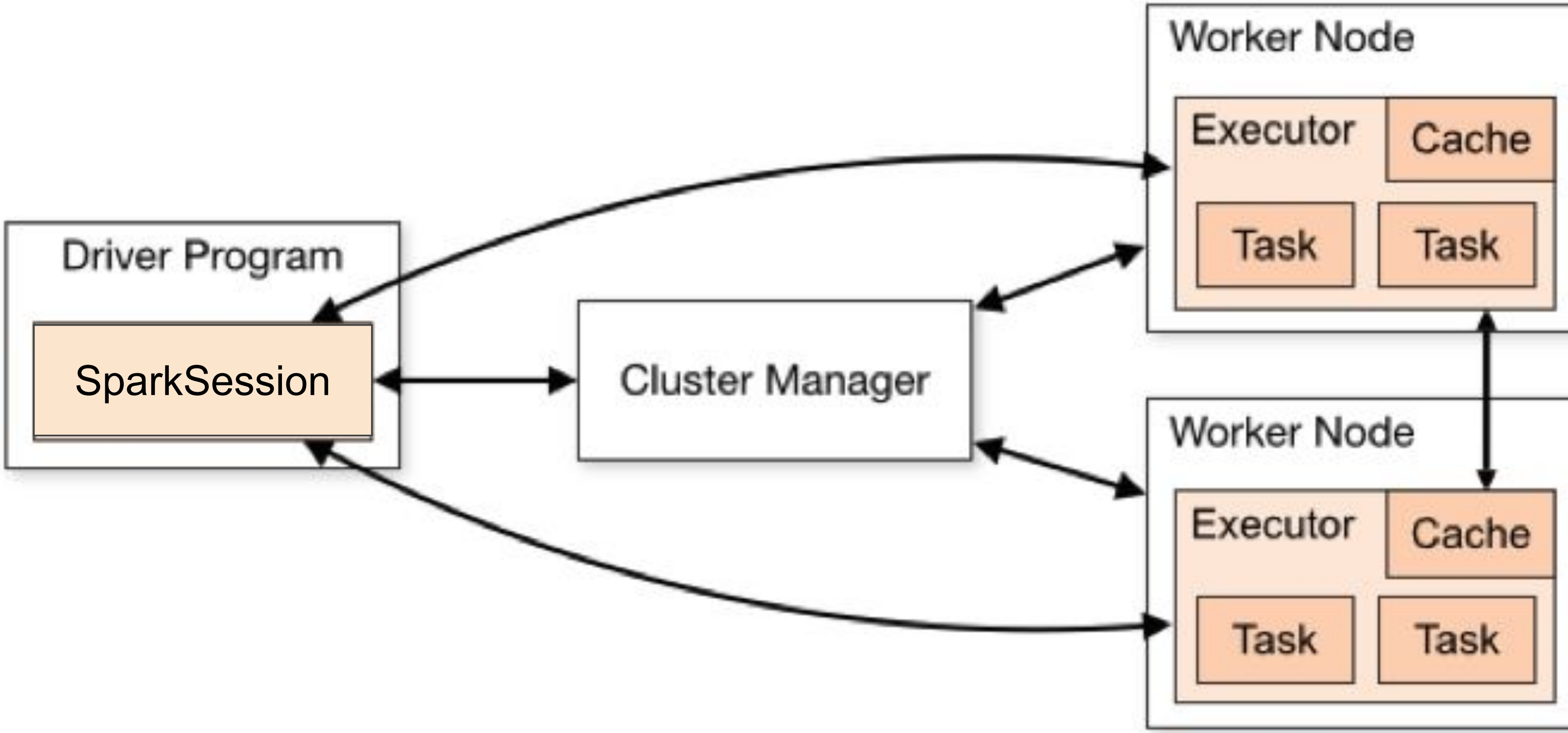
- Extensions to RDDs
  - Higher-level abstractions
  - Improved performance
  - Better scalability
- DataFrame
  - No static type checking
  - APIs in Java, Scala, Python, R
- DataSet
  - Static type checking
  - APIs in Java and Scala
- Can convert to/from RDDs and use with RDDs

SPARK APPLICATION



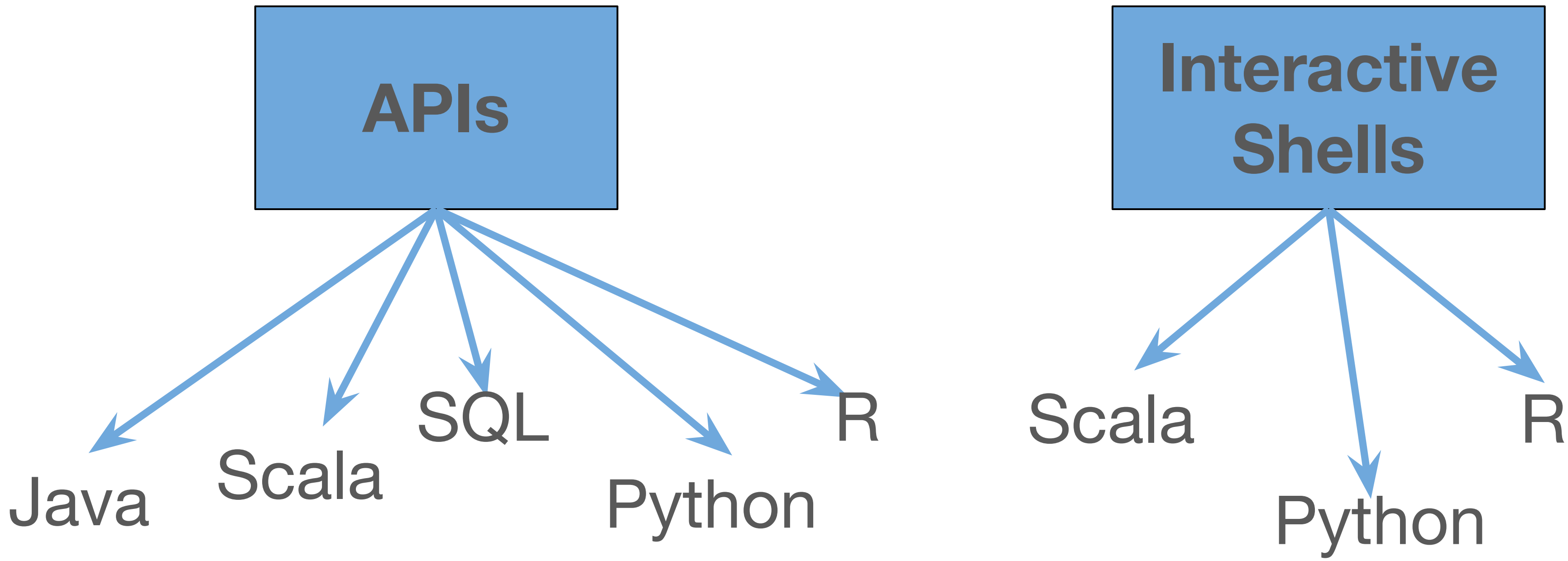
- Driver converts Spark application into one or more jobs
- An action creates a job
- DAG of instructions built for each job
- Each node in DAG is single or multiple Spark stages
- Each stage is broken down into tasks
- Tasks are distributed to executors

SPARK ARCHITECTURE



SPARK INTERFACE

Goals: speed, **ease of use**, generality, unified platform

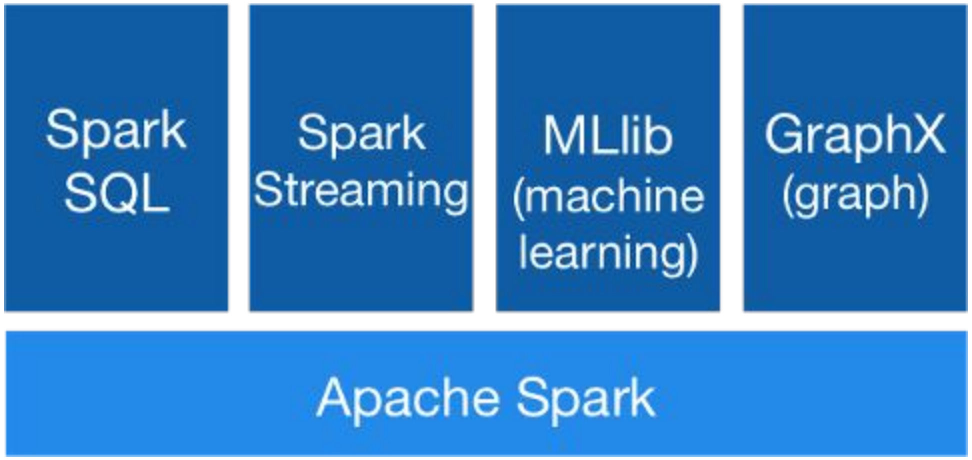


SPARK AS UNIFIED PLATFORM

- Goals: speed, ease of use, **generality,** unified platform
- Support for several data sources
  - Local file systems, HDFS, RDBMSs, MongoDB, Kafka, AWS S3, etc.
- Can run on various platforms
  - Hadoop, Kubernetes, cloud, standalone
- Support for multiple workloads
  - batch, streaming
  - machine learning, SQL, graph processing

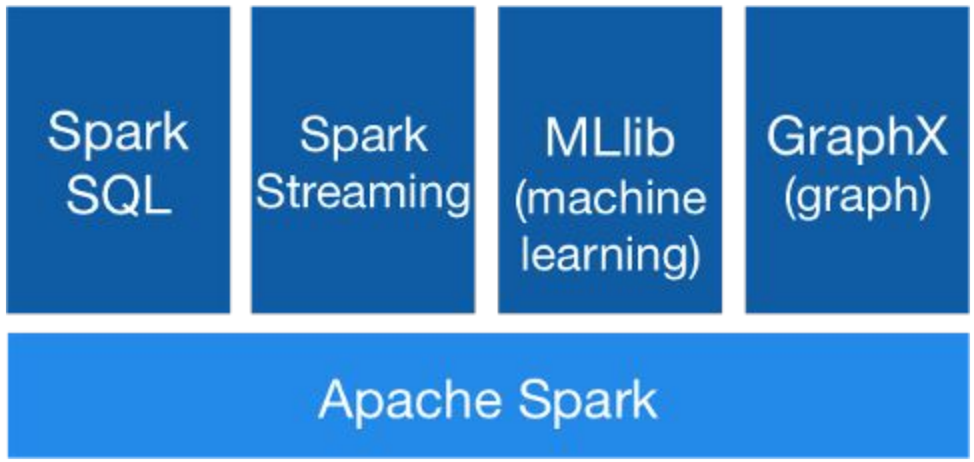
SPARK AS UNIFIED PLATFORM

- Goals: speed, ease of use, generality, **unified platform**



- Provides unified platform for various analytics processing
- **Spark engine** provides core capabilities for distributed processing
- **Spark libraries** provide additional higher-level functionality for diverse workloads

SPARK LIBRARIES



- Spark SQL: structured data processing
- Spark Streaming: real-time analytics
- MLlib: machine learning
- GraphX: graph processing

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

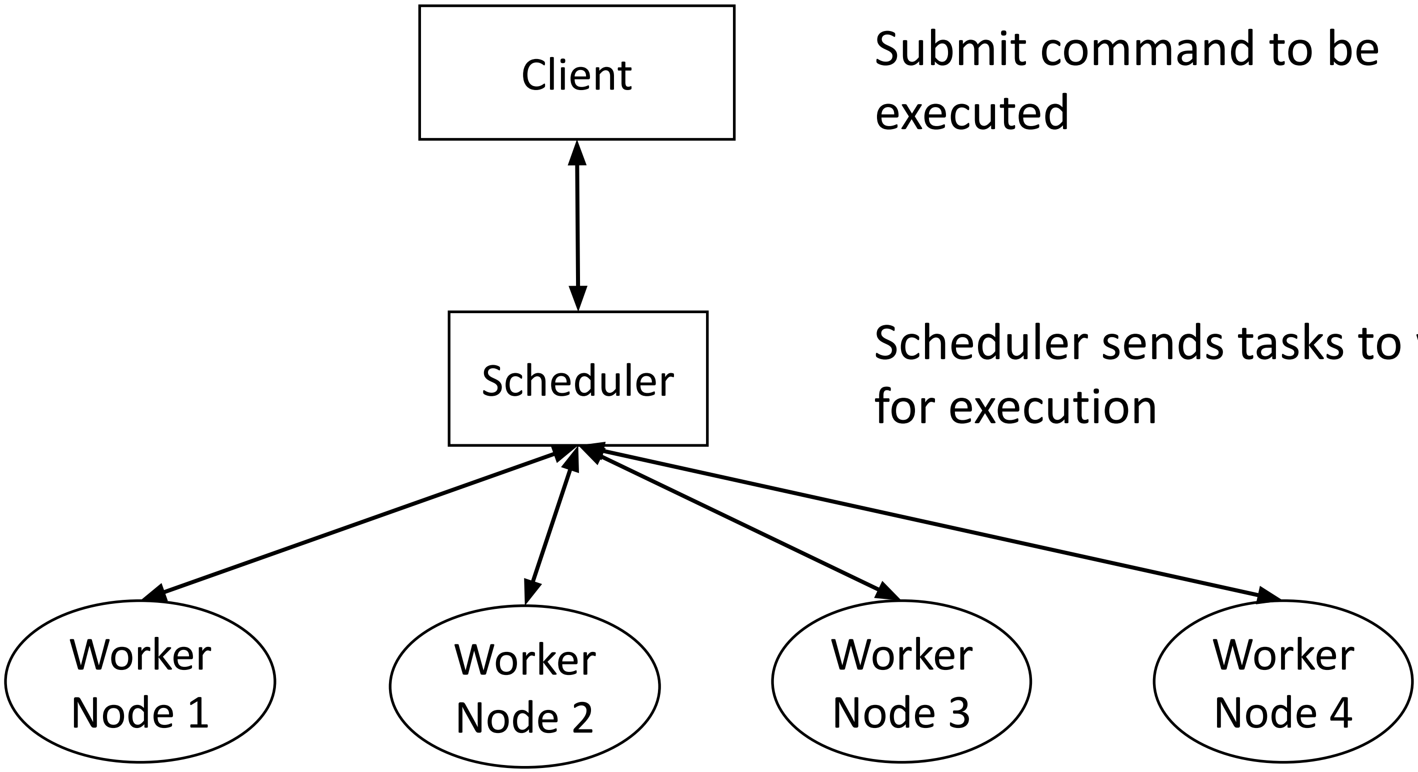
DASK

- Parallel computing library for analytics
- Python library
  - Scales existing Python ecosystem
- Can be used for
  - parallel processing on single machine
  - distributed processing on cluster of machines

DASK DESIGNING PRINCIPLES

- Python-Based
  - API is similar to numpy, pandas, scikit-learn
  - Integrates natively with Python code
- Scalable
  - Can run on clusters with 1000s of cores
  - Can also run on single system
- Flexible
  - Can run on on-premise, cloud, or HPC systems.
  - Can be used for custom workloads
- Customizable
  - Provides low-level APIs to parallelize custom computations
- Responsive
  - Provides real-time dashboard

DASK DISTRIBUTED EXECUTION



Submit command to be executed

Scheduler sends tasks to workers for execution

Workers execute tasks and return computed results

DASK SCHEDULING

Dask API creates task graphs

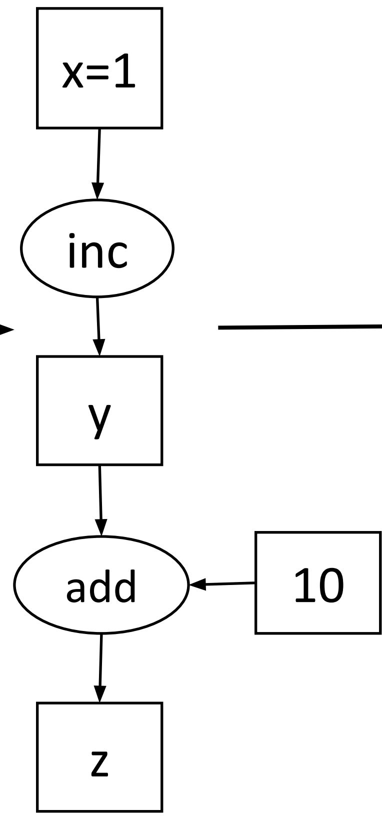
Dask schedulers execute them

```python
def inc(i):
    return i + 1

def add(a, b):
    return a + b

x = 1
y = inc(x)
z = add(y, 10)
```

Dask API



Task Graph

Single-machine scheduler

Distributed scheduler

Schedulers

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK APIs

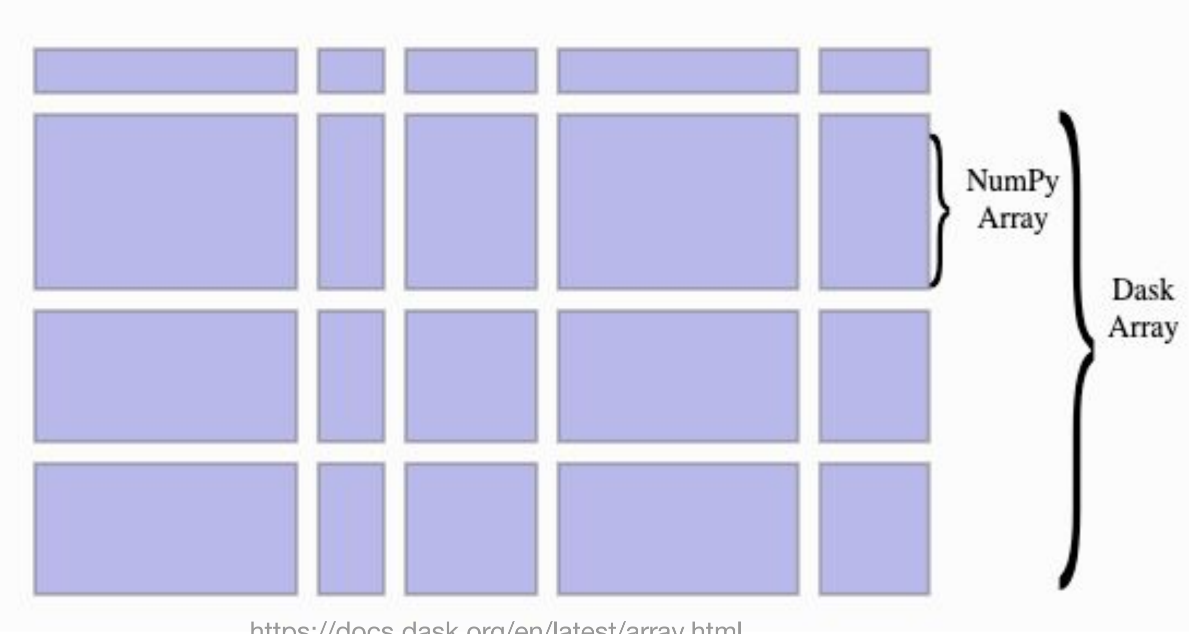
- High-Level
  - Provides scalable versions of numpy, pandas, scikit-learn
    - Arrays: parallel numpy
    - Bags: parallel multisets
    - DataFrames: parallel pandas
    - Others from external libraries
- Low-level
  - Provides operations to parallelize custom, user-defined workloads
    - Delayed: parallel function evaluation with lazy evaluation
    - Futures: real-time parallel function evaluation

DASK APIs

- High-Level
  - Provides scalable versions of numpy, pandas, scikit-learn
    - Arrays: parallel numpy
    - Bags: parallel multisets
    - DataFrames: parallel pandas
    - Others from external libraries
- Low-level
  - Provides operations to parallelize custom, user-defined workloads
    - Delayed: parallel function evaluation with lazy evaluation
    - Futures: real-time parallel function evaluation

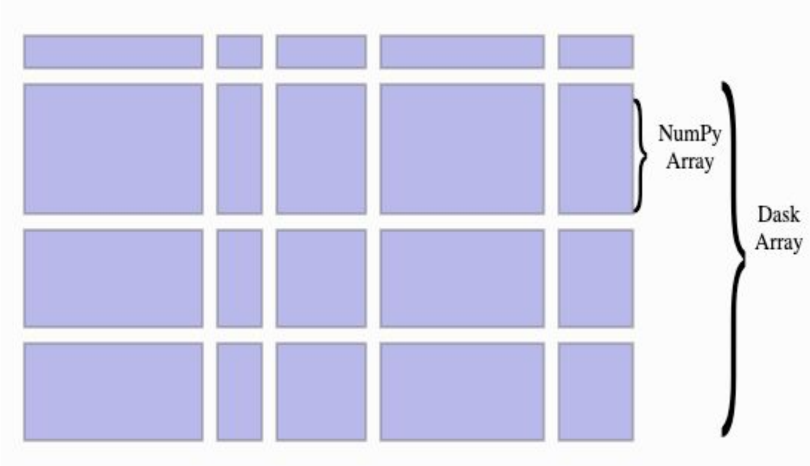
DASK ARRAY

- Parallel NumPy arrays
  - Extends numpy array for processing large arrays
  - Splits large array into small arrays that can reside on disk or distributed in cluster
  - Implements subset of numpy functionality



https://docs.dask.org/en/latest/array.html

DASK ARRAY



- Parallel NumPy arrays
  - Consists of many numpy arrays

- Create random array

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

- Persist in memory

```
x = x.persist()
```

- Use numpy syntax

```
y = x + x.T
y.sum().compute()
```

To execute operations and get results

DASK BAG

- Parallel bag or multiset
  - Collection of generic Python objects
  - Multiset: Set that allows for multiple instances of elements
    - **List**: ordered collection with repeats: [1,2,2,3]
    - **Set**: unordered collection without repeats: {1,2,3}
    - **Bag** = **Multiset**: unordered collection with repeats: {1,2,3,2}

  - Implements operations on objects *in parallel*
    - e.g., map, filter, groupy, aggregation

  - Common uses
    - Used to parallelize computations on unstructured or semi-structured data or user-defined Python objects
    - e.g., text data, log files

DASK BAG

- Read in data from JSON files

```
import dask.bag as db
import json
b = db.read_text('data/*.json').map(json.loads)
```

- Bag operations
  - Get first 2 rows

    ```
    b.take(2)
    ```

  - Select people over 30 years old and return first 2 results

    ```
    b.filter(lambda sample: sample['age']>30).take(2)
    ```

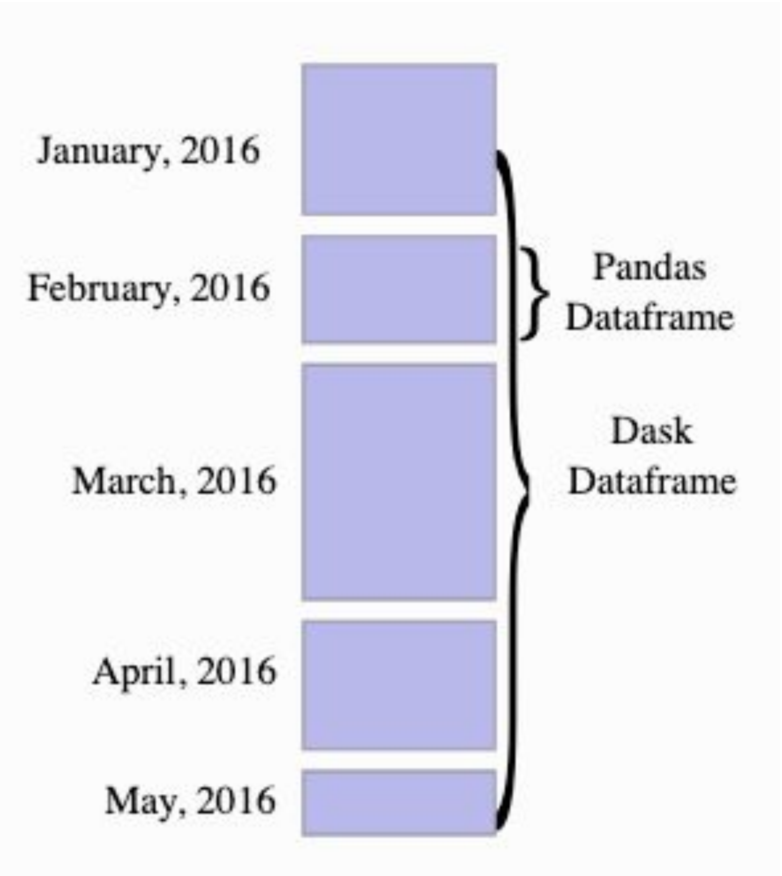
  - Count number of samples

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

DASK DATAFRAME

Parallel Pandas DataFrames

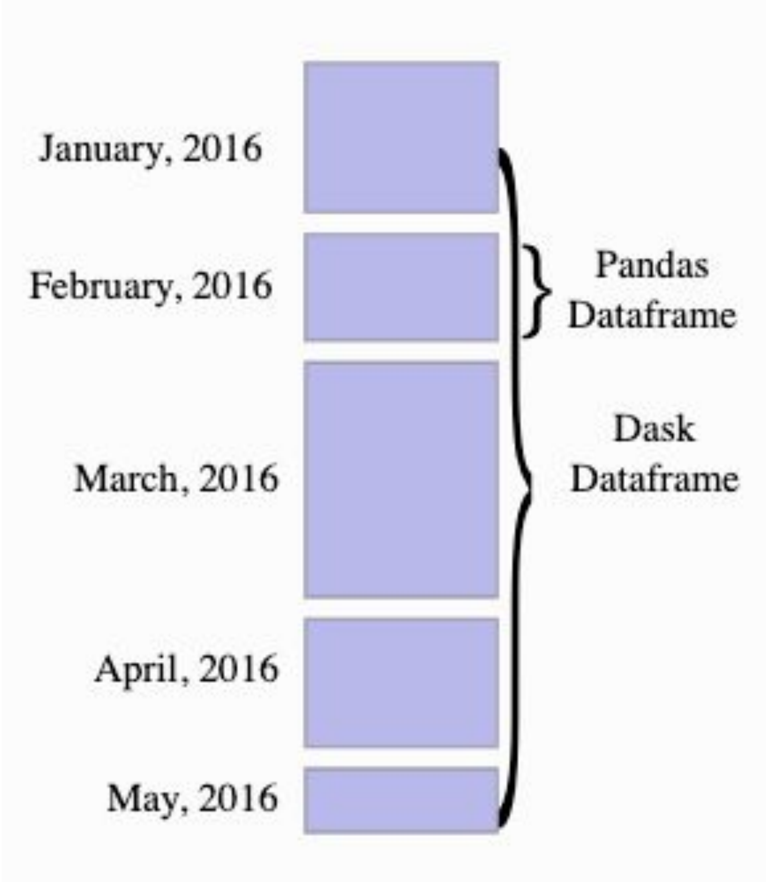
- Large parallel DataFrame made up of many smaller Pandas DataFrames
- Pandas DataFrames can reside on disk or distributed across many machines in cluster
- Operation on Dask DataFrame triggers operations on smaller Pandas DataFrames
- Implements subset of Pandas functionality



https://docs.dask.org/en/latest/dataframe.html

DASK DATAFRAME

- Parallel Pandas DataFrames
  - Consists of many Pandas DataFrames



- Load data

```
import dask.dataframe as dd
df = dd.read_csv('*.csv')
```

- Use pandas syntax

```
df.head()
len(df)
df.groupby('occupation').age.mean().compute()
df.age.max().visualize()
```

To see underlying task graph

DASK APIs

- High-Level
  - Provides scalable versions of numpy, pandas, scikit-learn
    - Arrays: parallel numpy
    - Bags: parallel multisets
    - DataFrames: parallel pandas
    - Others from external libraries
- Low-level
  - Provides operations to parallelize custom, user-defined workloads
    - Delayed: parallel function evaluation with lazy evaluation
    - Futures: real-time parallel function evaluation

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK APIs

- High-Level
  - Provides scalable versions of numpy, pandas, scikit-learn
    - Arrays: parallel numpy
    - Bags: parallel multisets
    - DataFrames: parallel pandas
    - Others from external libraries
- Low-level
  - Provides operations to parallelize custom, user-defined workloads
    - Delayed: parallel function evaluation with lazy evaluation
    - Futures: real-time parallel function evaluation

DASK DELAYED

- Usage
  - To parallelize custom code
  - To build complex algorithms
- How it works
  - Used for parallelizing functions
  - Usually involves loops in code
- Lazy evaluation
  - Tasks to carry out computation is lazily evaluated
  - To optimize execution efficiency

DASK DELAYED

- Some problems may not be well represented using provided collections (e.g., Dask DataFrame)
- Can use Delayed API to customize parallel processing
- How can this code be parallelized?

```python
def inc(x):
    return x + 1

def double(x):
    return x * 2

def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = sum(output)
```

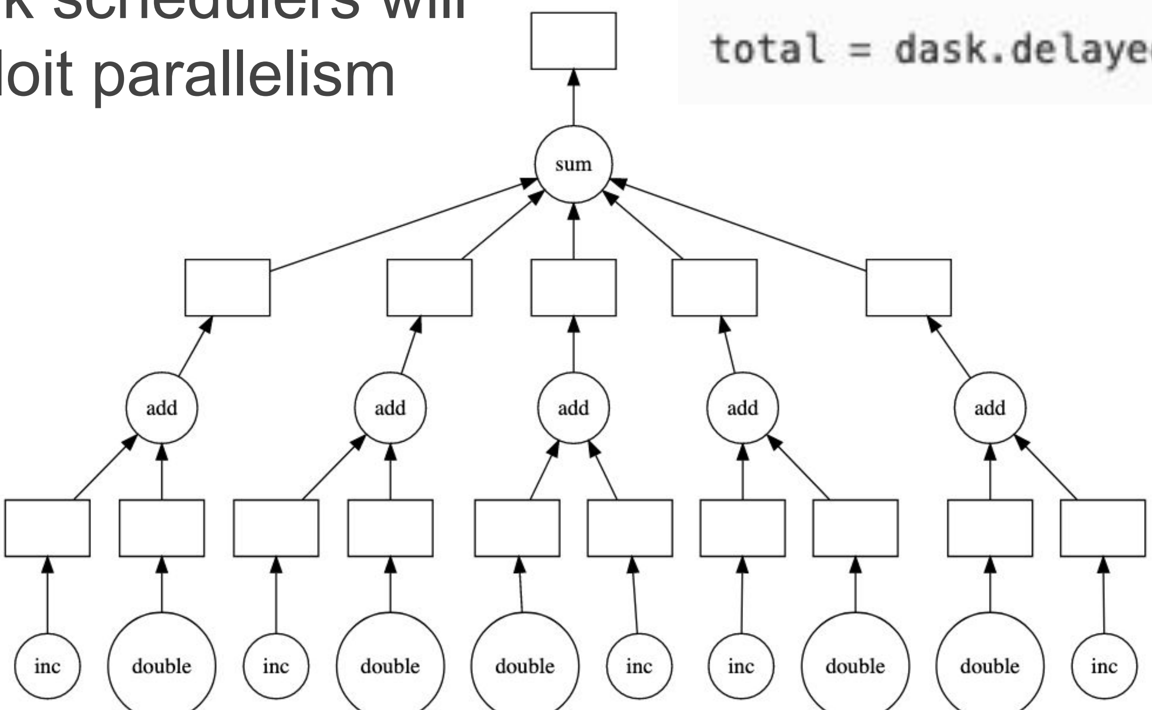
DASK DELAYED

- Wrap functions in 'delayed'
- Execution is delayed
- Task graph is generated instead
- Dask schedulers will exploit parallelism

```
import dask

output = []
for x in data:
    a = dask.delayed(inc)(x)
    b = dask.delayed(double)(x)
    c = dask.delayed(add)(a, b)
    output.append(c)

total = dask.delayed(sum)(output)
```



data = [1,2,3,4,5]

total.visualize()

total.compute()

=> 50

DASK DELAYED

Can also used ‘delayed’ as function decorator:

```python
def inc(x):
    return x + 1

def double(x):
    return x * 2

def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = sum(output)
```

```python
import dask

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

@dask.delayed
def double(x):
    return x * 2

@dask.delayed
def add(x, y):
    return x + y

data = [1, 2, 3, 4, 5]

output = []
for x in data:
    a = inc(x)
    b = double(x)
    c = add(a, b)
    output.append(c)

total = dask.delayed(sum)(output)
```

DASK FUTURES

- Usage
  - For computations that may change over time
- How it works
  - Allows for parallelizing custom code
  - But tasks are executed immediately instead of lazily
- Steps
  - Start client
  - Define functions
  - Submit tasks to remote thread/process/worker
  - Gather result from thread/process/worker

DASK FUTURES

Start client

```
client = Client()
```

Define functions

```
def inc(x):
    return (x+1)
def add(x,y):
    return (x+y)
```

Submit tasks to worker

```
a = client.submit (inc,10)
a
```

<Future: status: pending, type: int, key: inc-b8aaf26b99466...>

<Future: status: finished, type: int, key: inc-b8aaf26b99466...>

Gather result

```
a.result()
```

11

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK ML

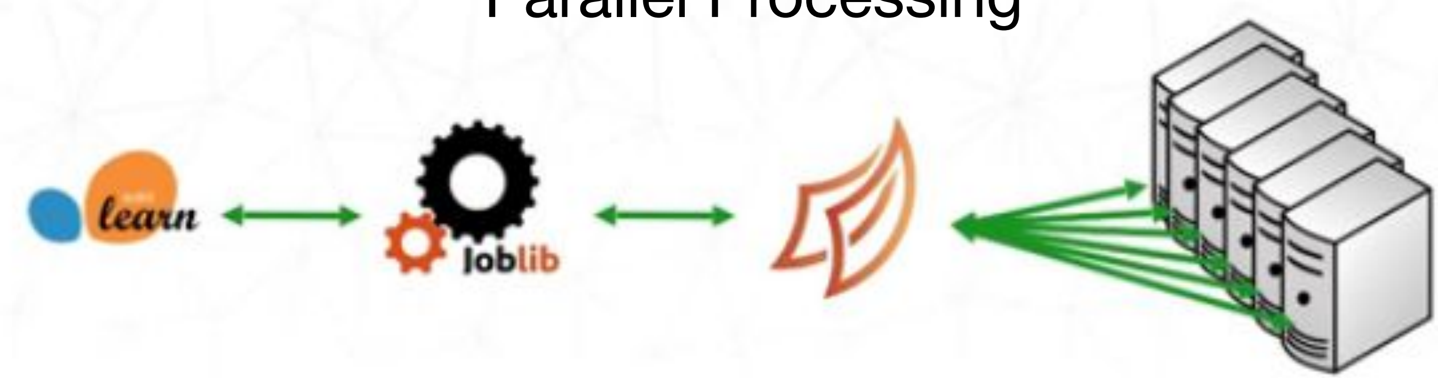
- Scalable machine learning in Python
  - Using Dask with scikit-learn, XGBoost, etc.
- Approaches
  - (1) Parallelize scikit-learn
    - For execution on many machines in a cluster
  - (2) Reimplement algorithms to be scalable
    - Replace numpy arrays with Dask arrays to achieve scalability
    - Done for linear models, pre-processing, and clustering
  - (3) Partner with other distributed libraries
    - XGBoost
- Scikit-learn API
  - Similar syntax as scikit-learn

(1) PARALLELIZING SCIKIT-LEARN

- (1a) Single-machine parallelism with scikit-learn
  - Provided in scikit-learn for some estimators
- (1b) Multi-machine parallelism with Dask
  - Provides distributed scikit-learn by using Dask backend
  - Data has to fit in memory
- (1c) Parallel meta-estimators
  - Train with scikit-learn
  - Predict on large dataset in parallel
- (1d) Incremental Learning
  - Training is done by fitting estimator in batches

SCIKIT-LEARN VS. DASK

Scikit-Learn: Single-Machine Parallelism
Parallel Processing



Scikit-Learn & Dask: Multiple-Machine Parallelism
Distributed Processing

(1a) PARALLELISM IN SCIKIT-LEARN

- ***Single***-machine parallelism
- Some estimators and utilities provide parallelism
  - Via n_jobs parameter (using *joblib* library underneath)
  - n_jobs = <integer> to specify number of cores to use
  - n_jobs = None means unset (n_jobs=1)
  - n_jobs = -1 to use all available cores
  - n_jobs = -N to use all but 1 available cores (N>1)

3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

(1b) DISTRIBUTED SCIKIT-LEARN WITH DASK

- ***Multi***-machine parallelism with Dask
  - Makes use of all cores of *cluster*
  - Dask provides distributed processing for scikit-learn algorithms written with parallel execution (i.e., with n_jobs)
  - Using Dask backend with joblib library
  - Wrap code to be executed in parallel with
    - joblib.parallel_backend('dask')
- Data still needs to fit in RAM
  - Operations on data done in parallel
- Useful for
  - Training ensemble model
  - Hyperparameter tuning

(1c) DASK PARALLEL META-ESTIMATORS

- Distributed processing on larger-than-memory datasets
- Provides parallel prediction and transformation
  - Training is done using scikit-learn, so is *not* parallelized
  - Post-fit tasks are parallelized
    - Use trained scikit-learn estimator with Dask backend
    - Speeds up prediction and transformation
    - Allows for prediction on large, larger-than-memory datasets

(1d) INCREMENTAL LEARNING

- Training is performed incrementally in batches
- Idea
  - Some scikit-learn estimators have partial_fit()
  - Estimator is wrapped in Dask’s Incremental()
  - Dask passes each batch of data to estimator’s partial_fit()

(2) SCALABLE DASK-ML ALGORITHMS

- Dask re-implementations of algorithms
- Data preparation
  - MinMaxScaler, StandardScaler, OneHotEncoder, etc.
  - train_test_split()
- Hyperparameter tuning
  - GridSearchCV, RandomizedSearchCV, etc.
- Models
  - LogisticRegression
  - LinearRegression, ElasticNet, etc.
  - k-Means, etc.

(3) DASK AND XGBOOST

- XGBoost
  - Scalable implementation of gradient boosted trees
- Dask & XGBoost
  - Dask setups up XGBoost
  - Dask prepares data
  - Dask hands off data to XGBoost
  - Dask gets results from XGBoost
  - Integrated in same code

DASK ML

- Scalable machine learning in Python
  - Using Dask with scikit-learn, XGBoost, etc.
- Approaches
  - (1) Parallelize scikit-learn
    - For execution on many machines in a cluster
  - (2) Reimplement algorithms to be scalable
    - Replace numpy arrays with Dask arrays to achieve scalability
    - Done for linear models, pre-processing, and clustering
  - (3) Partner with other distributed libraries
    - XGBoost
- Scikit-learn API
  - Similar syntax as scikit-learn

SCIKIT-LEARN

- Random forest in scikit-learn
  - No parallelism

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier()
rf_model.fit (X_train, y_train)
rf_model.predict (X_test)
```

SINGLE-MACHINE PARALLELISM

- Available in scikit-learn
  - For estimators with n_jobs parameter
  - Distribute tasks over all cores in a *single* machine
- Example
  - Random forest
  - fit() and predict() will be executed in parallel

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_jobs=-1)
rf_model.fit (X_train, y_train)
rf_model.predict (X_test)
```

Use all cores

MULTI-MACHINE PARALLELISM

- Distributed machine learning with Dask
  - Distribute tasks over all cores in *cluster* of machines
  - Using Dask backend
  - Works with scikit-learn algorithms with n_jobs parameter
- Steps
  - Create and connect to Dask cluster
  - Wrap code to be executed using Dask backend with
    - with joblib.parallel_backend('dask')

MULTI-MACHINE PARALLELISM

- Using joblib with Dask backend
- Create and connect to Dask cluster

```
from dask.distributed import Client
client = Client()
# client = Client(<IP_address>)
```

If using existing client

- Specify code to be executed in parallel

```
import joblib
from scipy.stats import randint as sp_randint
param_dist = {'max_depth':sp_randint(1,10)}
rf_model = RandomForestClassifier()
search   = RandomizedSearchCV(rf_model,param_dist,
                n_iter=50, cv=10, random_state=123)
```

MULTI-MACHINE PARALLELISM

- Using joblib with Dask backend
- Create and connect to Dask cluster

```
from dask.distributed import Client
client = Client()
# client = Client(<IP_address>)
```

If using existing client

- Specify code to be executed in parallel

```
import joblib
from scipy.stats import randint as sp_randint
param_dist = {'max_depth':sp_randint(1,10)}
rf_model = RandomForestClassifier()
search   = RandomizedSearchCV(rf_model,param_dist,
                n_iter=50, cv=10, random_state=123)
with joblib.parallel_backend('dask'):
    search.fit(<data>,<target>)
search.best_params_
client.close()
```

Specify code to be run in parallel

DASK PARALLEL META-ESTIMATOR

- Provides parallel prediction and transformation
  - Wrap scikit-learn estimator to provide parallel execution of prediction and transformation operations

```
from sklearn.tree import DecisionTreeClassifier
from dask_ml.wrappers import ParallelPostFit

model= ParallelPostFit(estimator=DecisionTreeClassifier())

model.fit(<data>,<target>)

model.predict(<big_test_data>)
```

Uses scikit-learn to fit model to data & target

Uses Dask for distributed prediction

DASK INCREMENTAL META-ESTIMATOR

- Provides incremental learning

```
from sklearn.tree import SGDClassifier
from dask_ml.wrappers import Incremental

estimator = SGDClassifier(random_state=10, max_iter=100)

model = Incremental(estimator)

model.fit(<input>,<target>,classes=[0,1])
```

Estimator will be trained in batches

Underlying scikit-learn model's fit() is used on batch

DASK SCALABLE ALGORITHMS

- Algorithms implemented in Dask
  - For distributed processing on cluster of machines
- Example
  - K-means cluster analysis

```
from dask_ml.cluster import KMeans

km = KMeans(n_cluster=<K>)

km.fit(<data>)

predictions = km.predict(<data>)
```

Create k-means model with K clusters

Fit model to data

Find cluster assignments for data

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK BEST PRACTICES

- Optimize code for single machine
  - Explore better algorithms and/or data structures
  - Compile code (Use Numba or Cython)
  - Sample data
  - Profile code to find problem spots
  - May not need distributed processing
- Cache data when possible
  - ddf = ddf.persist()
- Use dashboard
  - Info about task runtimes, memory use, etc.
  - http://<scheduler-url>:8787/status
  - More info at https://docs.dask.org/en/latest/diagnostics-distributed.html

DASK BEST PRACTICES

- Avoid very large or very small partitions
  - Number of chunks should be >= number of cores
  - Too few large partitions will reduce parallelism
  - Too many small partitions will incur too much overhead
  - Can set partition size when reading in data
  - Repartition if needed
- Use compute() carefully
  - Instead of calling compute() several times:
    - ddf = dd.read_csv(“file”)
    - xmin = ddf.x.min().compute()
    - xmax = ddf.x.max().compute()
  - Call it just once
    - ddf = dd.read_csv(“file”)
    - xmin,xmax = dask.compute(df.x.min(),df.x.max())

DASK BEST PRACTICES

- Load data using Dask
  - Instead of reading in data with Pandas then handing to Dask:
    - ❖ ddf = <...>
    - ❖ for file in filenames:
      - ✔ df = pandas.read_csv(file)
      - ✔ ddf = ddf.append(df)
  - Let Dask read in data directly into Dask collection
    - ❖ ddf = dd.read_csv(filenames)
- Use Pandas if data fits into RAM
  - ddf = dd.read_csv("file")
  - ddf = ddf.groupby('name').mean()
  - df = ddf.compute()

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK VS. SPARK

Dask

- Component in Python ecosystem
- Used in conjunction with numpy, pandas, scikit-learn
- Python-based
- Lacks high level optimization, but has flexibility to adapt to custom workloads
- Single node or cluster

Spark

- Big Data platform with its own ecosystem
- All-in-one project. Integrates with other Apache projects
- Has APIs in Scala, Java, Python, R, SQL
- Provides high level optimizations, but lacks flexibility for more complex or adhoc algorithms
- Single node or cluster

https://docs.dask.org/en/latest/spark.html

QUIZ

QUIZ

Dask provides APIs in …

A. Java
B. C
C. Python
D. All of the above
E. None of the above

QUIZ

Which of the following is true about using joblib?

A. On a single node, setting n_jobs=-1 specifies that all available cores will be used except for 1
B. The value of n_jobs specifies the degree of parallelism for all scikit-learn estimators
C. Using Dask as the backend with joblib provides distributed processing on all nodes in the cluster
D. None of the above
E. A & C

QUIZ

In Dask ...

A. A Dask Bag consists of many smaller numpy arrays
B. An operation on a Dask DataFrame triggers operations on smaller Pandas DataFrames
C. Dask Arrays have all the same functionality as numpy arrays, implemented in parallel
D. The low-level APIs provide scalable versions of numpy, pandas, and scikit-learn
E. All of the above

QUIZ

Dask parallel meta-estimators…

A. Parallelize all operations of scikit-learn estimators
B. Speeds up training
C. Allow for training to be performed on larger-than-memory datasets
D. None of the above
E. B & C

QUIZ

Dask Delayed and Futures …

A. Are used to parallelize custom code
B. Provide low-level access to the Dask scheduler
C. Provide scalable versions of Pandas
D. All of the above
E. A & B only

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise

DASK RESOURCES

- Website
  - https://dask.org/
- Documentation
  - https://docs.dask.org/en/latest/
- Tutorials
  - https://tutorial.dask.org/00_overview.html
- Examples
  - https://examples.dask.org/index.html
- API
  - https://ml.dask.org/modules/api.html

DASK

- Dask Overview
- High-Level APIs
- Low-Level APIs
- Dask ML
- Dask Best Practices
- Dask vs. Spark
- Dask Exercise