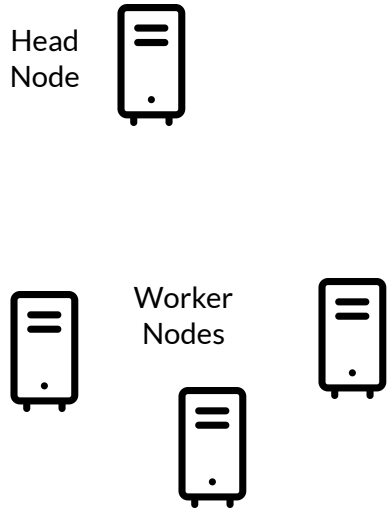
STATEFUL DISTRIBUTED COMPUTING WITH RAY

Tom Baldwin
Data Scientist
PyData PDX
October 7, 2020

DISTRIBUTED COMPUTING FRAMEWORKS

Tools for running your code on multiple computers.



Enterprisey! APACHE Spark™

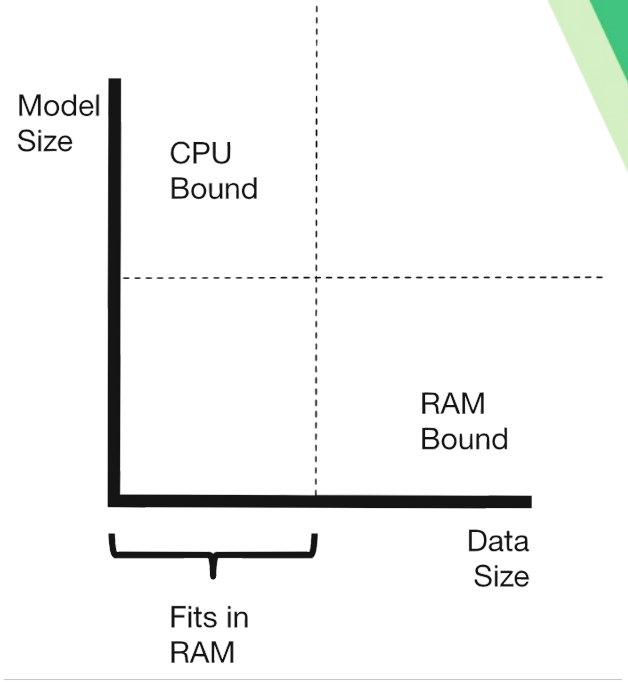
The Python native: DASK

This talk: RAY

WHEN TO USE DISTRIBUTED COMPUTING

- Memory bound
  - Dataset is too big
  - Each computer gets a portion

- CPU bound
  - “Embarrassingly parallel” operations can be divided into independent parts
  - Each computer does some tasks



See: Tom Augspurger’s talk at PyData NYC 2019 (video) (slides)


```python
@F.udf(T.LongType())
def my_func(x):
    time.sleep(1)
    return x * 2
```


```python
@dask.delayed
def my_func(x):
    time.sleep(1)
    return x * 2
```


```python
@ray.remote
def my_func(x):
    time.sleep(1)
    return x * 2
```

```python
# spark
[row[0] for row in df.select(my_func(F.col("col"))).collect()]

# dask
dask.compute([my_func(i) for i in range(4)])

# ray
ray.get([my_func.remote(i) for i in range(4)])
```

```
@decorator
def my_function(x):
    # ... very serious business ...
    return result


@decorator
def MyClass(x):
    def my_method(self, x):
        # very serious business
        # using internal state
        return result
```

The **decorator** takes your custom function and replaces it with one that:

- **Pickles** your code
- Pickles any **arguments** passed to it
- Ships code + args off to an idle computer somewhere (a **worker node**), which runs it
- Returns a reference to a **task** in progress
- That can be **traded for the real result** when you call `compute()` (or `get()` or...)

Ray can also do this with **classes**.

THE STATEFUL WORKER ABSTRACTION

In Ray, you can

- Decorate & call a **function** to create a **task**
- Decorate & instantiate a **class** to create an **actor**

Actors are like tasks (your code on a remote node) that hang around until you explicitly stop them.

While an actor is active, you can call its methods to trigger computations and manipulate its internal state.

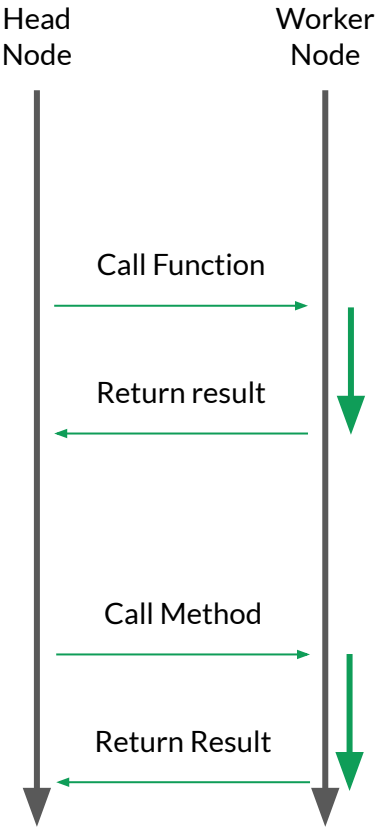
Dask can also do this (via experimental API)

```python
@ray.remote
class MyActor(object):
    def __init__(self, coef=2):
        self.coef = coef

    def set_coef(self, coef):
        self.coef = coef

    def my_method(self, x):
        time.sleep(1)
        return x * self.coef
```

```python
actor_a = MyActor.remote(coef=2)
actor_b = MyActor.remote(coef=5)

ray.get(
    [
        actor_a.my_method.remote(x=2),
        actor_b.my_method.remote(x=2),
    ]
)  # [4, 10]
```

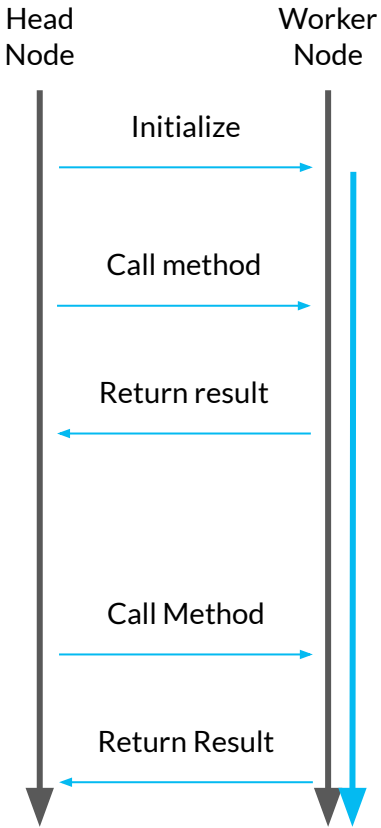
```python
ray.get(actor_b.set_coef.remote(coef=3))

ray.get(
    [
        actor_a.my_method.remote(x=2),
        actor_b.my_method.remote(x=2),
    ]
)  # [4, 6]
```

Task Abstraction



Local variables disappear on exit

Actor Abstraction



Store state in instance attributes

REINFORCEMENT LEARNING

ML paradigm in which an agent interacts with an environment, seeking to maximize some reward

Agents contain a model which they can update/retrain in response to feedback from actions taken

RL algorithms succeed or fail based on

- How well the model fits
- How well they balance explore/exploit

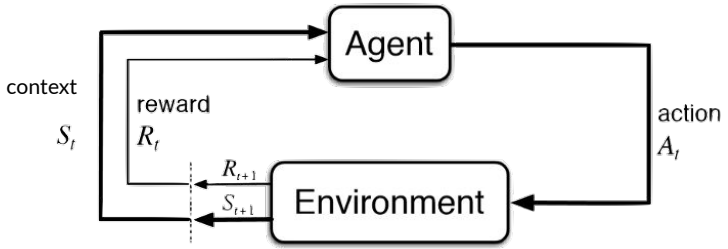
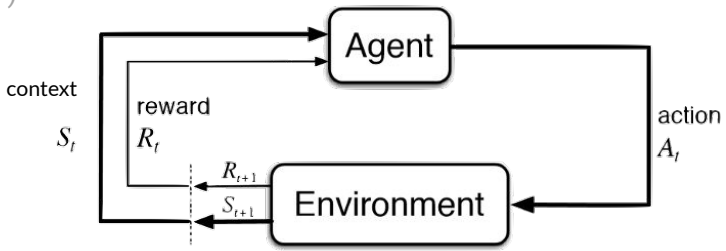


Diagram from [Towards Data Science]

```python
@ray.remote
class LearningAgent(object):
    def __init__(self, training_data):
        self.training_data = training_data
        self.model = MyFancyModel()
        self.model.fit(training_data)

    def make_prediction(self, context):
        action = self.model.predict(context)
        return action

    def receive_feedback(self, context, action, reward):
        self.training_data.append([context, action, reward])
        self.model.retrain(self.training_data)
```



Using actors for RL avoids pickling/unpickling the model on each cycle.

PROJECTS RELATED TO RAY

Bundled with Ray

- Tune (for hyperparameter tuning)
- RLlib (for reinforcement learning)

External projects

- RayOnSpark (Analytics Zoo)

THINGS TO LIKE ABOUT RAY

- First-class support for the stateful worker (actor) abstraction
  - Experimental support in Dask
- Pretty easy to set up
  - Can run on local computer or on a cluster
  - I even made a toy cluster out of two laptops!
  - Scales up to running on cloud clusters
- Built-in toolkits for RL and HPT

You might not need it if:

- Your problem is memory-bound (use a Spark or Dask dataframe)
- Your distributed CPU-bound problem doesn't require keeping state
- You might not need distributed computing at all!

https://ray.io

CASCADE DATA LABS

cascadedatalabs.com