Course Outline

Scaling Python with Dask

Course Summary

Description

If you've taken your data skills from zero to one with PyData (Pandas, Scikit-Learn, and friends) then this class will help you use larger data sets that won't fit in memory and will help you distribute your workloads to accelerate your code with Dask.

During the first two sessions, you'll learn to use Python skills you already have to query and transform data, build models, and scale your custom code. During the last two sessions, we'll peek under the hood to learn how Dask works and "look inside" with real-time animated dashboards. We'll cover options for deploying clusters, troubleshooting, sample use cases, and best practices.

This entire class is delivered through interactive, web-based JupyterLab notebooks that you can keep and refer to whenever you need. You'll also receive 10,000 Coiled Cloud credits per month for 3 months so you can continue your learning journey without limitations.

Topics

- Introduction
- Parellelize Python Code
- Dask Dataframe
- Dask Array
- Scaling Your Own Code
- Graphical User Interfaces
- Machine Learning
- Thinking About Distributed Deployment
- Distributed Dask: Cast of Characters
- Basic Operation of Dask Clusters
- Tasks
- Distributed Data
- Resource usage and Resilience
- Debugging
- Use Case Example: Orchestrating Batch ML Scoring [optional, per timing]
- Best Practices

Audience

If you've taken your data skills from zero to one with PyData (Pandas, Scikit-Learn, and friends) then this class will help you use larger data sets that won't fit in memory and will help you distribute your workloads to accelerate your code with Dask.

Prerequisites

- Python, basic level
- PyData stack (Pandas, NumPy, scikit-learn), basic level

Duration

Four half days

Course Outline

Scaling Python with Dask

Course Outline

I. Introduction
- A. About Dask – what it is, where it came from, what problems it solves
- B. Examples: one-line AutoML, Dask Dataframe, and custom parallelization

II. Parellelize Python Code
- A. Fundamentals of parallelism in Python
- B. concurrent.futures, Dask Delayed, Futures
- C. Example: building a parallel Dataframe

III. Dask Dataframe
- A. How Dask Dataframe works
- B. Pandas-style analytics with Dask Dataframe
- C. Applying custom computations to Dataframe

IV. Dask Array
- A. How Dask Array is related to NumPy and NDArray
- B. Operations on Dask Array

V. Scaling Your Own Code
- A. Basic scheduling
- B. Example: Monte Carlo simulation
- C. Visualizing task graphs
- D. Dynamic scheduling

VI. Graphical User Interfaces
- A. Monitoring workers, tasks, and memory
- B. Principal performance and troubleshooting challenges with big data
- C. Using Dask's dashboards to understand performance

VII. Machine Learning
- A. Scikit-Learn-style featurization with Dask
- B. Algorithm support and integration
- C. Modeling

VIII. Thinking About Distributed Deployment
- A. About Dask and Coiled Computing: Making scale-out computing easy
- B. Simplest distributed cluster: manual setup
- C. Changes in transitioning to distributed environment
- D. Implications for users (devs) and admin (IT)

IX. Distributed Dask: Cast of Characters
- A. Client, Scheduler, Nanny, Worker
- B. Where these services are located, their relationships and roles
- C. Supporting Players: cluster resource manager (e.g., k8s, Coiled Cloud, YARN, etc.)

X. Basic Operation of Dask Clusters
- A. Creating clusters with helper tools: Cloud Provider, Coiled Cloud, etc.
- B. Cluster API
- C. Sizing and scaling your cluster
- D. Admin perspective

XI. Tasks
- A. Submitting tasks and directing output
- B. Scheduling policy
- C. Finding your tasks and data (programmatically)
- D. Seeing your tasks and data: the Dask Dashboard

XII. Distributed Data
- A. Source data via tasks
- B. Source data scatter
- C. Storing data worker-local
- D. Handling output (result) data, direct parallel write vs. gather/result

XIII. Resource usage and Resilience
- A. Output spill location and resource management
- B. Work stealing
- C. Loss of processes
- D. Loss of storage on workers

XIV. Debugging
- A. Additional GUIs (e.g., profiler)
- B. Remote debugging
- C. client.run

XV. Use Case Example: Orchestrating Batch ML Scoring [optional, per timing]
- A. Source data on disk
- B. ML model
- C. Options for interference, pros/cons
- D. Supplying dependencies via code or container image
- E. Basic workflow
- F. Improvements and optimizations (e.g., batch size)

XVI. Best Practices
- A. Managing partitions and tasks
- B. File formats
- C. Caching
- D. Integrating with more Python (and non-Python) tools like xgboost, plotting libraries, and GPUs
- E. Q&A