Table of Contents

Chapter 1. Learning Dask

Preface

This book focuses on Dask because of it's comparatively deep integration into the Python ecosystem. Alternatives such as Ray, Spark, and buying really expensive computers exist, and we'll talk a bit more about those in the next chapter.

Audience

We wrote this book for Python data engineers and scientists looking to scale or speed-up their Python powered data powered applications. This includes everything from the traditional exchange-transform-load (ETL) to the new fancy machine-learning (aka three statisicians in a trench-coat) tools.

This book is not for people who love setting up machines, fine-tuning memory allocations, micro-optimizations, or people looking to build distributed systems from scratch. We'll the basics of these topics that a practioner need to understand, but not at the level of systems implementor.

This book assumes you know enough Python, and the libraries you are using, to complete your work on a smaller scale, but not necessarily how to handle larger data sizes or bigger expirements. If you don't

Supporting Books

If you're coming from a language other than Python, or otherwise new to Python Learning Python and Head First Python from O'Reilly are really solid introductions.

If your Python solid, but your data analysis experience could use some reinforcement (see joke) Python for Data Analysis, 2nd Edition, by Wes McKinney (O'Reilly), Data Science from Scratch, 2nd Edition, by Joel Grus (O'Reilly), and Machine Learning with Python Cookbook by Chris Albon (O'Reilly) are all excellent resources to dig deeper.

Dask is not the only option for you. If as your going through the first chapter you find one of the alternative frameworks mentioned to be a better fit we have some suggestions for books on those topics. Suggestions which we might financially benefit from, but I assure you book royalties are pretty low. If you decide PySpark is better suited to what your working on "Learning Spark" (the latest edition) is a great jumping off point for the Spark ecosystem. If you decide that Ray is better suited to your needs, we'd recommend the soon-to-be-started "Learning Ray" book from or "Serverless Python with Ray" (both from O'Reilly).

Also if you're interested in teaching your kid about the area that you're working on, I am working on a book targetted to children introducing distributed programming with Spark called "Distributed Computing 4 Kids."

Layout

This introductory book is meant to get up you up to speed with the basics of dask in the first three chapters. After you've completed "Understanding the basics of Dask" the rest of the book is a choose-your-own-adventure type novel where the if you go down the wrong path you get a stack-trace you don't undeerstand.

1.1. What is Dask, why do we need it, and where does it fit?

Dask is a framework for parallelized computing with Python supporting scaling from multiple-cores all the way to multiple data centers. In addition to managing the work, Dask has APIs at different levels of abstraction ranging from scheduling invidual tasks to operations on Dataframes. Dask's lineage traces back to Continum Analytics, now known as Anaconda Inc, and grew out of the DARPA funded BLAZE project [https://blaze.readthedocs.io/en/latest/index.html], which aimed to provide a numpy/pandas like interface to big-ish data. Continum has participated in the development of many key libraries, and even conferences, in the Python data analytics space.

1.1.1. Why do we need dask?

Dask simplifies our problems of growing our Python code to handle larger and more complex data and problems. While the term "big data" is perhaps less in vogue now than it was a few years ago, the data sizes of the problems were handling and the complexity of the computation & models we are building have not gotten smaller or simpler. Dask allows us to largely use the existing interfaces that we are used to (such as pandas or Python's concurrency libraries) while providing the ability to go beyond what is possible with a single machine.

1.1.2. Where does Dask fit in the ecosystem

Dask sits at the intersection of multiple, traditionally distinct, ecosystems. The first is the "big data ecosystem" of things like Hadoop and friends. Dask is also a part of the Python data science ecosystem, including things like Pandas. The 3rd ecosystem which Dask is a part of is that of accelerated Python, including things like RAPIDS, numba, and multi-processing. Few tools sit at the same intersection, with the closest similar tool being Ray.

Dask and Ray are both written in Python, with underlying extensions when needed. There is a GitHub issue [https://github.com/ray-project/ray/issues/642] where the creators of both system compare their similarities and differences. From a systems point of view, the biggest differences between Ray and Dask come down to handling state + recovery as well as centralized v.s. de-centralized scheduling.

Big Data

Dask occupies a slightly different position than then the traditional big data Python tools such as PySpark (from Apache Spark), Streaming Python MR (from Apache Hadoop), PyBeam (from Apahce Beam), and PyFlink (from Apache Flink). From an implementation point of view, the biggest difference between Dask and many of these systems is that of Java.

Many of the Python system in the big data space fundamentally delegate

much of the work to Java. This can add substantail overhead both in terms of per-record (with copying data) and per-task (with slower startup times). On the other hand, delegating to existing Java solutions allows these tools to integrate with the output of other big data tools and take advantage of the optimizations at the Java level (such as cost based query optimization).

An important question for people working in a Hadoop ecosystem is that of formats. The ability to easily load the output of different toals and interact with shared libraries can be very useful. While Dask can read some of the formats, often the data in these environments is structured with the assumption that filters or columnar reads will be automatically pushed down. While it is of course possible to write out the data in a way that is suited for Dask to query from such systems, the overhead and dealy of additional ETL jobs and copies of data may not be worth it.

Many of the "big data" ecosystem projects package everything together (e.g. Spark SQL, Spark Kubernetes, etc. all got released together), Dask takes a more modular approach with it's components following their own development and release cadence.

Data Science

In the data science ecosystem many Python user's first library is that of `pandas`. While Spark has started to integrate the Koalas project [https://koalas.readthedocs.io/en/latest/], Dask's support of data science library APIs is one of the best. In addition to the higher-level pandas' APIs, Dask implements support for scaling `numpy`, `sci-kit-learn`, and other data science tools.

Accelerated Python

The accelerated Python ecosystem takes a few different looks, ranging from special code generation (like `numba`) or hardware specific accelerators and libraries like NVidia's `CUDA` (and wrappers like `cuDF`) and Intel's `MKL` libraries. While Dask can take advantage of these libraries, and there are some integrations like `dask-cuDF`, Dask is not primarily focused on optimizing your code. The other side of accelerated Python is that of `threading` or

`multiprocessing` where the focus is on allowing parallel compution. Dask implements these APIs and supports extending them beyond the concept of just a single computer.

Dask, with a little help from friends

While Dask core is relatively light weight and leaves out many "traditional" components found in other systems, these tend to be built on top as libraries.

SQL Engine

Dask it's self does not have a SQL engine, however BlazingSQL [https://github.com/BlazingDB/blazingsql] uses dask to provide a GPU accelerated SQL engine.

Workflow Scheduling

The household name of workflow scheduling in the big data ecosystem is Apache Airflow. While Apache Airflow has a wonderful collection of operators, making it easy to express complex task types conscisely, it is notoriously difficult to scale [1: With one thousand tasks per-hour taking substantail tuning and manual consideration https://medium.com/@keozchan/scaling-airflow-to-1000-tasks-hour-aac3207b26ec]. Dask can be used to run Airflow tasks [https://airflow.apache.org/docs/apache-airflow/1.10.1/howto/executor/use-dask.html], or as a backend for other task scheduling systems like the very humbly named perfect [https://github.com/prefecthq/prefect]

What Dask is not

While Dask is many things it is not a magic wand you wave over your code and have it go faster. There are places where Dask has largely compatible drop-in APIs, but if used incorrectly may result in slower execution. Dask is, for the most part, not a code re-writing or JIT tool, although those tools can be used in conjuction with Dask [2: Even for languages other than Python, like with BlazingSQL].

1.1.3. Conclusion