Dask Stories Documentation

Dask Community

May 08, 2019

Contents

This page shares stories from other Dask users.

CHAPTER 1

Sidewalk Labs: Civic Modeling

1.1 Who am I?

I'm Brett Naul. I work at Sidewalk Labs.

1.2 What problem am I trying to solve?

My team @ Sidewalk ("Model Lab") uses machine learning models to study human travel behavior in cities and produce high-fidelity simulations of the travel patterns/volumes in a metro area. Our process has three main steps:

- Construct a "synthetic population" from census data and other sources of demographic information; this population is statistically representative of the true population but contains no actual identifiable individuals.
- Train machine learning models on anonymous mobile location data to understand behavioral patterns in the region (what times do people go to lunch, what factors affect an individual's likelihood to use public transit, etc.).
- For each person in the synthetic population, generate predictions from these models and combine the resulting into activities into a single model of all the activity in a region.

For more information see our blogpost Introducing Replica: A Next-Generation Urban Planning Tool.

1.3 How Dask Helps

Generating activities for millions of synthetic individuals is extremely computationally intensive; even with for example, a 96 core instance, simulating a single day in a large region initially took days. It was important to us to be able to run a new simulation from scratch overnight, and scaling to hundreds/thousands of cores across many workers with Dask let us accomplish our goal.

1.4 Why we chose Dask originally, and how these reasons have changed over time

Our code consists of a mixture of legacy research-quality code and newer production-quality code (mostly Python). Before I started we were using Google Cloud Dataflow (Python 2 only, massively scalable but generally an astronomical pain to work with / debug) and multiprocessing (something like ~96 cores max).

Dask let us scale beyond a single machine with only minimal changes to our data pipeline. If we had been starting from scratch I think it's likely we would have gone in a different direction (something like C++ or Go microservices, especially since we have strong Google ties), but from my perspective as a hybrid infrastructure engineer/data scientist, having all of our models in Python makes it easy to experiment and debug statistical issues.

1.5 Some of the pain points of using Dask for our problem

There is lots of special dask knowledge that only I possess, for example:

- In which formats we can serialize data that will allow for it to be reloaded efficiently? Sometimes we can use parquet, other times it should be CSVs so we can easily chunk them dynamically at runtime
- Sometimes we load data on the client and scatter to the workers, and other times we load chunks directly on the workers
- The debugging process is sufficiently more complicated compared to local code that it's harder for other people to help resolve issues that occur on workers
- The scheduler has been the source of most of our scaling issues: when the number of tasks/chunks of data gets too large, the scheduler tends to fall over silently in some way.

  Some of these failures might be to Kubernetes (if we run out of RAM, we don't see an OOM error; the pod just disappears and the job will restart). We had to do some hand-tuning of things like timeouts to make things more stable, and there was quite a bit of trial and error to get to a relatively reliable state
- This has more to do with our deploy process but we would sometimes end up in situations where the scheduler and worker were running different dask/distributed versions and things will crash when tasks are submitted but not when the connection is made, which makes it take a while to diagnose (plus the error tends to be something inscrutable like `KeyError: ...` that others besides me would have no idea how to interpret)

1.6 Some of the technology that we use around Dask

- Google Kubernetes Engine: lots of worker instances (usually 16 cores each), 1 scheduler, 1 job runner client (plus some other microservices)
- Make + Helm
- For debugging/monitoring I usually kubectl port-forward to 8786 and 8787 and watch the dashboard/submit tasks manually. The dashboard is not very reliable over port-forward when there are lots of workers (for some reason the websocket connection dies repeatedly) but just reconnecting to the pod and refreshing always does the trick

CHAPTER 2

Genome Sequencing for Mosquitos

2.1 Who am I?

I'm Alistair Miles and I work for Oxford University Big Data Institute but am also affiliated with the Wellcome Sanger Institute. I lead the malaria vector (mosquito) genomics programme within the malaria genomic epidemiology network, an international network of researchers and malaria control professionals developing new technologies based on genome sequencing to aid in the effort towards malaria elimination. I also have a technical role as Head of Epidemiological Informatics for the Centre for Genomics and Global Health, which means I have some oversight and responsibility for computing and software architecture and direction within our teams at Oxford and Sanger.

2.2 What problem am I trying to solve?

Malaria is still a major cause of mortality, particularly in sub-Saharan Africa. Research has shown that the best way to reduce malaria is to control the mosquitoes that transmit malaria between people. Unfortunately mosquito populations are becoming resistant to the insecticides used to control them. New mosquito control tools are needed. New systems for mosquito population surveillance/monitoring are also needed to help inform and adapt control strategies to respond to mosquito evolution. We have established a project to perform an initial survey of mosquito genetic diversity, by sequencing whole genomes of approximately 3,000 mosquitoes collected from field sites across 18 African countries, The Anopheles gambiae 1000 Genomes Project. We are currently working to scale up our sequencing operations to be able to sequence ~10,000 mosquitoes per year, and to integrate genome sequencing into regular mosquito monitoring programmes across Africa and Southeast Asia.

2.3 How does Dask help?

Whole genome sequence data is a relatively large scale data resource, requiring specialised processing and analysis to extract key information, e.g., identifying genes involved in the evolution of insecticide resistance. We use conventional bioinformatic approaches for the initial phases of data processing (alignment, variant calling, phasing), however beyond that point we switch to interactive and exploratory analysis using Jupyter notebooks.

Making interactive analysis of large-scale data is obviously a challenge, because inefficient code and/or use of computational resources vastly increases the time taken for any computation, destroying the ability of an analyst to explore many different possibilities within a dataset. Dask helps by providing an easy-to-use framework for parallelising computations, either across multiple cores on a single workstation, or across multiple nodes in a cluster. We have built a software package called scikit-allel to help with our genetic analyses, and use Dask within that package to parallelise a number of commonly used computations.

2.4 Why did I choose Dask?

Normally the transition from a serial (i.e., single-core) implementation of any given computation to a parallel (multi-core) implementation requires the code to be completely rewritten, because parallel frameworks usually offer a completely different API, and managing complex parallel workflows is a significant challenge.

Originally Dask was appealing because it provided a familiar API, with the dask.array package following the numpy API (which we were already using) relatively closely. Dask also handled all the complexity of constructing and running complex, multi-step computational workflows.

Today, we're also interested in Dask's offered flexibility to initially parallelise over multiple cores in a single computer via multi-threading, and then switch to running on a multi-node cluster with relatively little change in our code. Thus computations can be scaled up or down with great convenience. When we first started using Dask we were focused on making effective use of multiple threads for working on a single computer, now as data is growing we are moving data and computation into a cloud setting and looking to make use of Dask via Kubernetes.

2.5 Pain points?

Initially when we started using Dask in 2015 we hit a few bugs and some of the error messages generated by Dask were very cryptic, so debugging some problems was hard. However the stability of the code base, the user documentation, and the error messages have improved a lot recently, and the sustained investment in Dask is clearly adding a lot of value for users.

It is still difficult to think about how to code up parallel operations over multidimensional arrays where one or more dimensions are dropped by the function being mapped over the data, but there is some inherent complexity there so probably not much Dask can do to help.

The Dask code base itself is tidy and consistent but quite hard to get into to understand and debug issues. Again Dask is handling a lot of inherent complexity so maybe not much can be done.

2.6 Technology I use around around Dask

We are currently working on deploying both JupyterHub and Dask on top of Kubernetes in the cloud, following the approach taken in the Pangeo project. We use Dask primarily through the scikit-allel package. We also use Dask primarily with the Zarr array storage library (in fact the original motivation for writing Zarr was to provide a storage library that enabled Dask to efficiently parallelise I/O bound computations).

2.7 Anything else to know?

Our analysis code is still quite heterogeneous, with some code making use of a bespoke approach to out-of-core computing which we developed prior to being aware of Dask, and the remainder using Dask. This is just a legacy of

timing, with some work having started prior to knowing about Dask. With the stability and maturity of Dask now I am very happy to push towards full adoption.

One cognitive shift that this requires is for users to get used to lazy (deferred) computation. This can be a stumbling block to start with, but is worth the effort of learning because it gives the user the ability to run larger computations. So I have been thinking about writing a blog post to communicate the message that we are moving towards adopting Dask wherever possible, and to give an introduction to the lazy coding style, with examples from our domain (population genomics). There are also still quite a few functions in scikit-allel that could be parallelised via Dask but haven't yet been, so I still have an aspiration to work on that. Not sure when I'll get to these, but hopefully conveys the intention to adopt Dask more widely and also help train people in our immediate community to use it.

CHAPTER 3

Full Spectrum: Credit and Banking

3.1 Who am I?

My name is Hussain Sultan. I am a partner at Full Spectrum Analytics. I create personalized analytics software within banks for the sake of equitable and profitable decision making.

3.2 What problem am I trying to solve?

Lending businesses create and manage valuations and cashflow models that output the profitability expectations for customer segments. These models are complex because they form a network of equations that need to be scored efficiently and keep track of inputs/outputs at scale.

3.3 How Dask helps

Dask is instrumental in my work for creating efficient cashflow model management systems and general data science enablement on data lakes.

Dask provides a way to construct the dependencies of cashflow equations as a DAG (using the dask.delayed interface) and provides a good developer experience for building scoring/gamification/model tracking applications.

3.4 Why I chose Dask originally

I chose dask for three reasons:

1. It was lightweight
2. The granular task scheduling approach to scaling both dataframes and arbitrary computations fit my use case well

3. It is easy to scale my team with Python programmers

3.5 Some of the pain points of using Dask in our problem

It's hard to get organization buy-in to adopt an open-source technology without vendored support and enterprise SLAs.

In a recent project, we had to integrate with the Orc data format that turned out to be more expensive than I originally anticipated (compounded by enterprise hadoop set-up and encryption requirements). These changes have since been upstreamed though, and so things are easier now.

3.6 Some of the technology that we use around Dask

We deployed on generic internal server with Jenkins scheduling a Jupyter notebook to execute. We built everything out using our internal analytics platform. We didn't have to worry about security because everything was behind a corporate firewall.

CHAPTER 4

IceCube: Detecting Cosmic Rays

4.1 Who am I?

I'm James Bourbeau, I'm a graduate student in the Physics department at the University of Wisconsin at Madison. I work at the IceCube South Pole Neutrino Observatory studying the cosmic-ray energy spectrum.

4.2 What problem am I trying to solve?

Cosmic rays are energetic particles that originate from outer space. While they have been studied since the early 1900s, the sources of high-energy cosmic rays are still not well known. I analyze data collected by IceCube to study how the cosmic-ray spectrum changes with energy and particle mass; this can help provide valuable insight into our understanding of the origin of cosmic rays.

This involves developing algorithms to perform energy reconstruction as well as particle mass group classification for events detected by IceCube. In addition, we use detector simulation and an iterative unfolding algorithm to correct for inherit detector biases and the finite resolution of our reconstructions.

4.3 How Dask Helps us

I originally chose to use Dask because of the Dask Array and Dask Dataframe data structures. I use Dask Dataframe to load thousands of HDF files and then apply further feature engineering and filtering data preprocessing steps. The final dataset can be up to 100GB in size, which is too large to load into our available RAM. So being able to easily distribute this load while still using the familiar pandas API has become invaluable in my research.

Later I discovered the Dask delayed iterface and now use it to parallelize code that doesn't easily conform to the Dask Array or Dask Dataframe use cases. For example, I often need to perform thousands of independent calculations for the pixels in a HEALPix sky map. I've found Dask delayed to be really useful for parallelizing these types of embarrassingly parallel calculations with minimal hassle.

I also use several of the diagnostic tools Dask offers such as the progress bar and resource profiler. Working in a large collaboration with shared computing resources, it's great to be able to monitor how many resources I'm using and scale back or scale up accordingly.

4.4 Pain points of using Dask

There were two main pain points I encountered when first using Dask:

- Getting used to the idea of lazy computation. While this isn't an issue that is specific to Dask, it was something that took time to get used to.
- Dask is a fairly large project with many components and it took some time to figure out how all the various pieces fit together. Luckily, the user documentation for Dask is quite good and I was able to get over this initial learning curve.

4.5 Technology that we use around Dask

We store our data in HDF files, which Dask has nice read and write support for. We also use several other Python data stack tools like Jupyter, scikit-learn, matplotlib, seaborn, etc. Recently, we've started experimenting with using HTCondor and the Dask distributed scheduler to scale up to using hundreds of workers on a cluster.

CHAPTER 5

Pangeo: Earth Science

5.1 Who Am I?

I am Ryan Abernathey, a physical oceanographer and professor at Columbia University / Lamont Doherty Earth Observatory.

I am a founding member of the Pangeo Project, an initiative aimed at coordinating and supporting the development of open source software for the analysis of very large geoscientific datasets such as satellite observations or climate simulation outputs. Pangeo is funded by National Science Foundation Grant 1740648, of which I am the principal investigator.

5.2 What Problem are We Trying to Solve?

Many oceanographic and atmospheric science datasets consist of multi-dimensional arrays of numerical data, such as temperature sampled on a regular latitude, longitude, depth, time grid. These can be real data, observed by instruments like weather balloons, satellites, or other sensors; or they can be "virtual" data, produced by simulations. Scientists in these fields perform an extremely wide range of different analyses on these datasets. For example:

- simple statistics like mean and standard deviation
- principal component analysis of spatio-temporal variability
- intercomparison of datasets with different spatio-temporal sampling
- spectral analysis (Fourier transforms) over various space and time dimensions
- budget diagnostics (e.g. calculating terms in the equation for heat conservation)
- machine learning for pattern recognition and prediction

Scientists like to work interactively and iteratively, trying out calculations, visualizing the results, and tweaking their code until they eventually settle on a result that is worthy of publication.

The traditional workflow is to download datasets to a personal laptop or workstation and peform all analysis there. As sensor technology and computer power continue to develop, the volume of our datasets is growing exponentially. This

workflow is not feasible or efficient with multi-terabyte datasets, and it is impossible with petabyte-scale datasets. The fundamental problem we are trying to solve in Pangeo is **how do we maintain the ability to perform rapid, interactive analysis in the face of extremely large datasets?** Dask is an essential part of our solution.

5.3 How Dask Helps

Our large multi-dimensional arrays map very well to Dask's `array` model. Our users tend to interact with Dask via Xarray, which adds additional label-aware operations and group-by / resample capabilities. The Xarray data model is explicitly inspired by the Common Data Model format widely used in geosciences. Xarray has incorporated dask from very early in its development, leading to close integration between these packages.

Pangeo provides configurations for deploying Jupyter, Xarray and Dask on high-performance computing clusters and cloud platforms. On these platforms, our users load data lazily using xarray from a variety of different storage formats and perform analysis inside Jupyter notebooks. Working closely with the Dask development team, we have tried to simplify the process of launching Dask clusters interactively by using packages such as dask-kubernetes and dask-jobqueue. Users employ those packages to interactively launch their own Dask clusters across many nodes of the compute system. Dask then automatically parallelizes the xarray-based computations without users having to write much specialized parallel code. Users appreciate the Dask dashboard, which provides a visual indication of the progress and efficiency of their ongoing analysis. When everything is working well, Dask is largely transparent to the user.

5.4 Why We Chose Dask Originally

Pangeo emerged from the Xarray development group, so Dask was a natural choice. Beyond this, Dask's flexibility is a good fit for our applications; as described above, scientists in this domain perform a huge range of different types of analysis. We need a parallel computing engine which does not strongly constrain the type of computations that can be performed nor require the user to engage with the details of parallelization.

5.5 Pain Points

Dask's flexibility comes with some overhead. I have the impression that the size of the graphs our users generate, which can easily exceed a million tasks, is pushing the limits of the dask scheduler. It is not uncommon for the scheduler to crash, or to take an uncomfortably long time to process, when these tasks are submitted. Our workaround is mostly to fall back on the sort of loop-based iteration over large datasets that we had to do pre-Dask. All of this undermines the interactive experience we are trying to achieve.

However, the first year of this project has made me optimistic about the future. I think the interaction between Pangeo users and Dask developers has been pretty successful. Our use cases have helped identify several performance bottlenecks that have been fixed at the Dask level. If this trend can continue, I'm confident we will be able to reach our desired scale (petabytes) and speed.

A broader issue relates to onboarding of new users. While I said above that Dask operates transparently to the users, this is not always the case. Users used to writing loop-based code to process datasets have to be retrained around the delayed-evaluation paradigm. It can be a challenge to translate legacy code into a Dask-friendly format. Some sort of "cheat sheet" might be able to help with this.

5.6 Technology around Dask

Xarray is the main way we interact with Dask. We use the `dask-jobqueue` and `dask-kubernetes` projects

heavily.

We also use Zarr extensively for storage, especially on the cloud, where we also employ `gcsfs` and `s3fs` to interface with cloud storage.

CHAPTER 6

NCAR: Hydrological Modeling

6.1 Who am I?

I am Joe Hamman and I am a Project Scientist in the Computational Hydrology Group at the National Center for Atmospheric Research. I am a core developer of the Xarray project and a contributing member of the Pangeo project. I study subjects in the areas of climate change, hydrology, and water resource engineering.

6.2 What problem am I trying to solve?

Climate change will bring widespread impacts to the hydrologic cycle. We know this because many research studies, conducted over the past two decades, have shown what the first order effects of climate change will look like in managed and natural hydrologic systems in terms of things like water availability, drought, wildfire, extreme precipitation, and floods. However, we don't have a very good understanding of the characteristic uncertainties that come from our choice of tools that we use to estimate these changes.

In the field of hydroclimatology, the tools we use are numerical models of the climate and hydrologic systems. These models can be constructed in many ways and it is often difficult understand how specific choices we make when building a model impact the inferences we can draw from them (e.g. the impact of climate change on flood frequency). We are working on methods to expose and constrain methodological uncertainties in the climate impacts modeling paradigm for water resource applications. This includes developing and analyzing large ensembles of climate projections and interrogating these ensembles to understand specific sources of uncertainty.

6.3 How does Dask help?

The climate and hydrologic sciences rely heavily on data stored in formats like HDF5 and NetCDF. We often use Xarray as an interface and friendly data model for these formats. Under the hood, Xarray uses either NumPy or Dask arrays. This allows us to scale the same Xarray computations we would typically do in-memory using NumPy, to larger tasks using Dask.

In my own research, I use Dask and Xarray as the core computational libraries for working with large datasets (10s-100s of terabytes). Often the operations we do with these datasets are fairly simple reduction operations where we may compare the average climate from two periods.

6.4 Why we chose Dask

When working on scientific analysis tasks, I don't want to think about parallelizing my code. We chose to work with Dask because `Dask.array` was nearly drop-in-compatible with NumPy. This meant that adopting Dask, inside or outside of Xarray, was much easier than adopting another parallel framework. Along those same lines, Dask is well integrated with other key parts of the scientific Python stack, including Pandas, Scikit-Learn, etc.

6.5 Pain points

Originally deploying Dask on HPC systems was a bit of a pain. But this has gotten much easier.

Additionally while Dask is easy to use, it's also easy to break. The freedom it provides also means that you have the freedom to shoot yourself in the foot.

Also diagnosing performance issues can be more complex than when just using Numpy. It's still a bit of an art rather than a science.

6.6 Technology we use around Dask

- We use Xarray to provide a higher level (and familiar) interface around Numpy arrays and Dask arrays
- We use NetCDF and HDF files for data storage
- I mostly work on HPC systems and have been helping develop the dask-jobqueue package for deploying Dask on job queueing systems
- In the Pangeo project, we're exploring Dask applications using Kubernetes and Jupyter notebooks

6.7 Other thoughts

I'm quite interested in enabling more intuitive scientific analysis workflows, particularly when parallelization is required. Dask has been a big part of our efforts to facilitate a "beginning-to-end" workflow pattern on large datasets.

CHAPTER 7

Mobile Network Modeling

7.1 Who am I?

I am Sameer Lalwani, and I specialize in modeling wireless networks.

We use these models to help operators with their technology decisions by building digital twins of their wireless networks. These models are used to quantify the impact of technology on user experience, network KPI & economics.

7.2 The problem I'm trying to solve

Mobile network operators face uncertainty whenever a new technology emerges. Their existing networks are complex with several bands and pre-existing sites. They need to understand the implications of bringing a new technology into this mix. By modeling their networks we can help reduce the uncertainty that management faces, while giving them actionable insights about their network.

For example currently we are in the middle of a 4G to 5G transition, presenting operators with decisions like the following:

- Which spectrum band should they bid on?
- What type of user experience and network performance can they expect?
- How much capital & operating expense will be required?
- What site locations should they target to meet coverage and capacity requirements?
- Should they consider refarming or acquire new bands?

These models are used to create digital twins at the scale of a city or a country. Based on known subscriber behavior and data from census, traffic and other sources "synthetic subscribers" are created. These subscribers could easily run into few millions with every individual having its own physical location on a map. We also bring in other GIS layers like building layouts, roads etc. to model indoor and outdoor subscribers.

The network can also have a range of cell sites going from few hundreds to tens of thousands.

Our models calculate data rates at an individual subscriber level by computing RF pathloss loss & SINR from all nearby sites. This takes our datasets into tens of millions of rows on which numerical computation needs to be done.

We also run lots of scenarios on this network to understand its capabilities and limitations.

7.3 How Dask helps

7.3.1 Distributed computing and Dask delayed

Dask Distributed helps run our scenarios across multiple machines while remaining within the memory constraints of each machine. We create approximately 10-20 machines on our VMware infrastructure with one Linux machine running the Dask scheduler, and all other machines running Dask workers with identical Conda environments.

Our team runs Jupyter notebooks on their desktop machines which connect with the Dask scheduler. Their notebooks work on their own datasets but for specialized functions and scenarios their tasks get transparently sent to the Dask scheduler. Each site count scenario takes between 10-15 mins to run, so it's nice to have 40 of them running in parallel.

7.3.2 Dask Dataframe

We use LiDAR data sets to calculate line of sight for mmWave propagation from lamp posts. The LiDAR data sets for the full city are often too large to open on a single machine. Dask Dataframe allows us to pool the resources of multiple machines while keeping our logic similar to Pandas dataframes.

We use Dask delayed to process multiple LiDAR files in parallel and then combine them into a single Dask Dataframe representing the full city. The line of sight calculation is CPU intensive and so it is nice to distribute it across multiple cores.

One of the things we really appreciate about Dask is that it gives us the ability to focus on the model logic without worrying about scalability. Once the code is ready, its seamless to call the functions using delayed and have them run across multiple CPU cores.

7.4 Pain points when using dask

I find Dask Dataframe to be too slow for multi column groupby operations. For such tasks I sort the Dataframe by the columns and partition it by the first column. I then use a delayed operation to use pandas on each partition which end up being much faster.

7.5 Technology I use around Dask

- GIS : Geopandas, Would love to get this natively supported.
- Traffic Flows: networkx
- Analytics: scikit-learn, scipy, pandas
- Visualization: Datashader with Dask distributed for LiDAR data.
- Charts: Holoviews, Seaborn
- Data Storage: HDF5

All our deployments are manually done. We bring up these machines for our analytics with identical Conda environments and tear them down once we are done.

Previously we used IPython parallel but found that Dask was easier to set up, and also allowed us to write more complex logic and also share our computing pool between multiple users.

7.6 Links to this work

Examples of models which use LiDAR to calculate line of sight:

1. 5G mmWave Coverage from Lamp posts
2. mmWave Backhaul & Economics

CHAPTER 8

Satellite Imagery Processing

8.1 Who am I?

I am David Hoese and I work as a software developer at the Space Science and Engineering Center (SSEC) at the University of Wisconsin - Madison. My job is to create software that makes meteorological data more accessible to scientists and researchers.

I am also a member of the open source PyTroll community where I act as a core developer on the SatPy library. I use SatPy in my SSEC projects called Polar2Grid and Geo2Grid which provide a simple command line interface on top of the features provided by SatPy.

8.2 The Problem I'm trying to solve

Satellite imagery data is often hard to read and use because of the many different formats and structures that it can come in. To make satellite imagery more useful the SatPy library wraps common operations performed on satellite data in simple interfaces. Typically meteorological satellite data needs to go through some or all of the following steps:

- **Read:** Read the observed scientific data from one or more data files while keeping track of geolocation and other metadata to make the data the most useful and descriptive.
- **Composite:** Combine one or more different "channels" of data to bring out certain features in the data. This is typically shown as RGB images.
- **Correct:** Sometimes data has artifacts, from the instrument hardware or the atmosphere for example, that can be removed or adjusted.
- **Resample:** Visualization tools often support a small subset of Earth projections and satellite data is usually not in those projections. Resampling can also be useful when wanting to do intercomparisons between different instruments (on the same satellite or not).
- **Enhancement:** Data can be normalized or scaled in certain ways that makes certain atmospheric conditions more apparent. This can also be used to better fit data in to other data types (8-bit integers, etc).

- **Write:** Visualization tools typically only support a few specific file formats. Some of these formats are difficult to write or have small differences depending on what application they are destined for.

As satellite instrument technology advances, scientists have to learn how to handle more channels for each instrument and at spatial and temporal resolutions that were unheard of when they were learning how to use satellite data. If they are lucky, scientists may have access to a high performance computing system, while the rest may have to settle for long execution times on their desktop or laptop machines. By optimizing the parts of the processing that take a lot of time and memory it is our hope that scientists can worry about the science and leave the annoying parts to SatPy.

8.3 How Dask helps

SatPy's use of Dask makes it possible to do calculations on laptops that used to require high performance server machines. SatPy was originally drawn to Xarray's `DataArray` objects for the metadata storing functionality and the support for Dask arrays. We knew that our original usage of Numpy masked arrays was not scalable to the new satellite data being produced. SatPy has now switched to `DataArray` objects backed by Dask and leverages Dask's ability to do the following:

- **Lazy evaluation:** Software development is so much easier when you don't have to remove intermediate results from memory to process the next step.
- **Task caching:** Our processing involves a lot of intermediate results that can be shared between different processes. When things are optimized in the Dask graph it saves us as developers from having to code the "reuse" logic ourselves. It also means that intermediate results that are no longer needed can be disposed of and their memory freed.
- **Parallel workers and array chunking:** Satellite data is usually compared by geographic location. So a pixel at one index is often compared with the pixel of another array at that same index. Splitting arrays in to chunks and processing them separately provides us with a great performance improvement and not having to manage which worker gets what chunk of the array makes development effortless.

Benefiting from all of the above lets us create amazing high resolution RGB images in 6-8 minutes on 3 year old laptops that would have taken 35+ minutes to crash from memory limitations with SatPy's old Numpy implementation.

8.4 Pain points when using Dask

1. Dask arrays are not Numpy arrays. Almost everything is supported or is close enough that you get used to it, but not everything. Most things you can get away with and get perfectly good performance; others you may end up computing your arrays multiple times in just a couple lines of code when you didn't know it. Sometimes I wish that there was a Dask feature to raise an exception of your array is computed without you specifically saying it was ok.
2. Writing to common satellite data formats, like GeoTIFF, can't always be written to by multiple writers (multiple nodes on a cluster) and some aren't even thread-safe. Opening a file object and using it with `dask.array.store` may work with some schedulers and not others.
3. Dimension changes are a pain. Satellite data processing some times involves lookup tables to save on bandwidth limitations when sending data from the satellite to the ground or other similar situations. Having to use lookup tables, including something like a KDTree, can be really difficult and confusing to code with Dask and get it right. It typically involves using `atop`, `map_blocks`, or sometimes suffering the penalty of passing things to a `Delayed` function where the entire data array is passed as one complete memory-hungry array.
4. A lot of satellite processing seems to perform better with the default threaded Dask scheduler over the distributed scheduler due to the nature of the problems being solved. A lot of processing, especially the creation of RGB images, requires comparing multiple arrays in different ways and can suffer from the amount of communication

between distributed workers. There isn't an easy way that I know of to control where things are processed and which scheduler to use without requiring users to know detailed information on the internal of Dask.

8.5 Technology I use around Dask

As mentioned earlier SatPy uses Xarrayto wrap most of our Dask operations when possible. We have other useful tools that we've created in the PyTroll community to help support deploying satellite processing tools on servers, but they are not specific to Dask.

8.6 Links

- PyTroll Community
- SatPy
- SatPy Examples
- PyResample

CHAPTER 9

Prefect: Production Workflows

9.1 Who am I?

I am Chris White; I am the Tech Lead at Prefect, a company building the next generation of workflow automation platforms for data engineers and data scientists. In this role, I am the core developer of our open source engine which allows users to build, schedule and execute robust workflows.

9.2 The Problem I'm trying to solve

Most teams are responsible for maintaining production workflows that are critical to the team's mission. Historically these workflows consisted largely of batch ETL jobs, but more recently include things such as deploying parametrized machine learning models, ad-hoc reporting, and handling event-driven processes.

Typically this means developers need a workflow system which can do things such as:

- retry failed tasks
- schedule jobs to run automatically
- log detailed progress (and history) of the workflow
- provide a dashboard / UI for inspecting system health
- provide notification hooks for when things go wrong

among many other things. We at Prefect like to think of a workflow system as a technical insurance policy - you shouldn't really notice it much when things are going well, but it should be maximally useful when things go wrong.

Prefect's goal is to build the next generation workflow system. Older systems such as Airflow and Luigi are limited by their model of workflows as slow-moving, regularly scheduled, with limited inter-task communication. Prefect, on the other hand, embraces this new reality and makes very few assumptions about the nature and requirements of workflows, thereby supporting more dynamic use cases in both data engineering and data science.

9.3 How Dask helps

Prefect was designed and built with Dask in mind. Historically, workflow systems such as Airflow handled *all* scheduling, of both workflows *and* the individual tasks contained within the workflows. This pattern introduces a number of problems:

- this puts an enormous burden on the central scheduler (it is scheduling *every single action* taken in the system)
- it adds non-trivial latency to task runs
- in practice, this limits the amount of dynamicism workflows can have
- it also tends to limit the amount of data tasks can share, as all information is routed through the central scheduler
- it requires users to have an external scheduler service running to run their workflows at all!

Instead, Prefect handles the scheduling of *workflows*, and lets Dask handle the scheduling and resource management of *tasks* within each workflow. This provides a number of benefits out of the box:

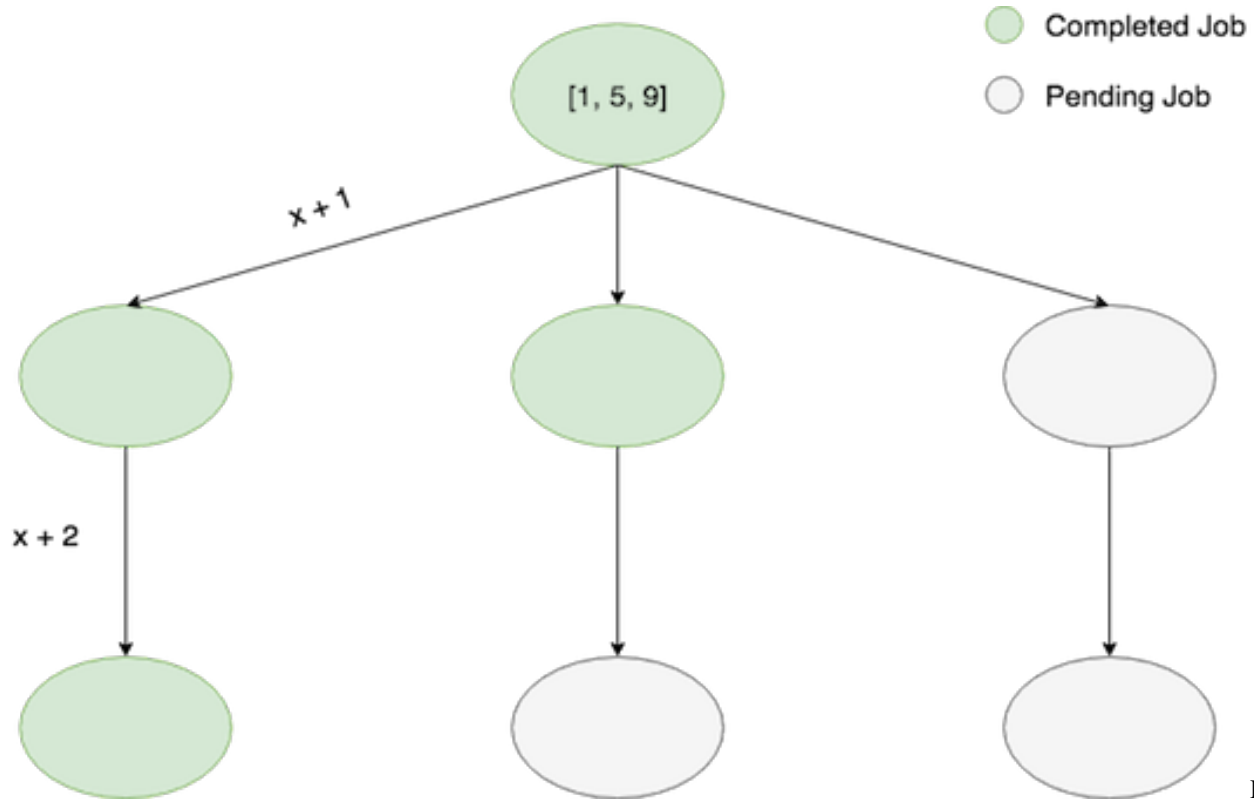
- **Task scheduling:** Dask handles all task scheduling within a workflow, allowing Prefect to incentivize smaller tasks which Dask schedules with millisecond latency
- **"Dataflow":** because Dask handles serializing and communicating the appropriate information between Tasks, Prefect can support "dataflow" as a first-class pattern
- **Distributed computation:** Dask handles allocating Tasks to workers in a cluster, allowing users to immediately realize the benefits of distributed computation with minimal overhead
- **Parallelism:** whether running in a cluster or locally, Dask provides parallel Task execution off the shelf

Additionally, because Dask is written in pure Python and has an active open source community, we can very easily get feedback on possible bugs, and even contribute to improving the software ourselves.

To achieve this ability to run workflows with many tasks, we found that Dask's Futures interface serves us well. In order to support dynamic tasks (i.e., tasks which spawn other tasks), we rely on Dask worker clients. We have also occasionally experimented with Dask Queues to implement more complicated behavior such as future-sharing and resource throttling, but are not currently using them (mainly for design reasons).

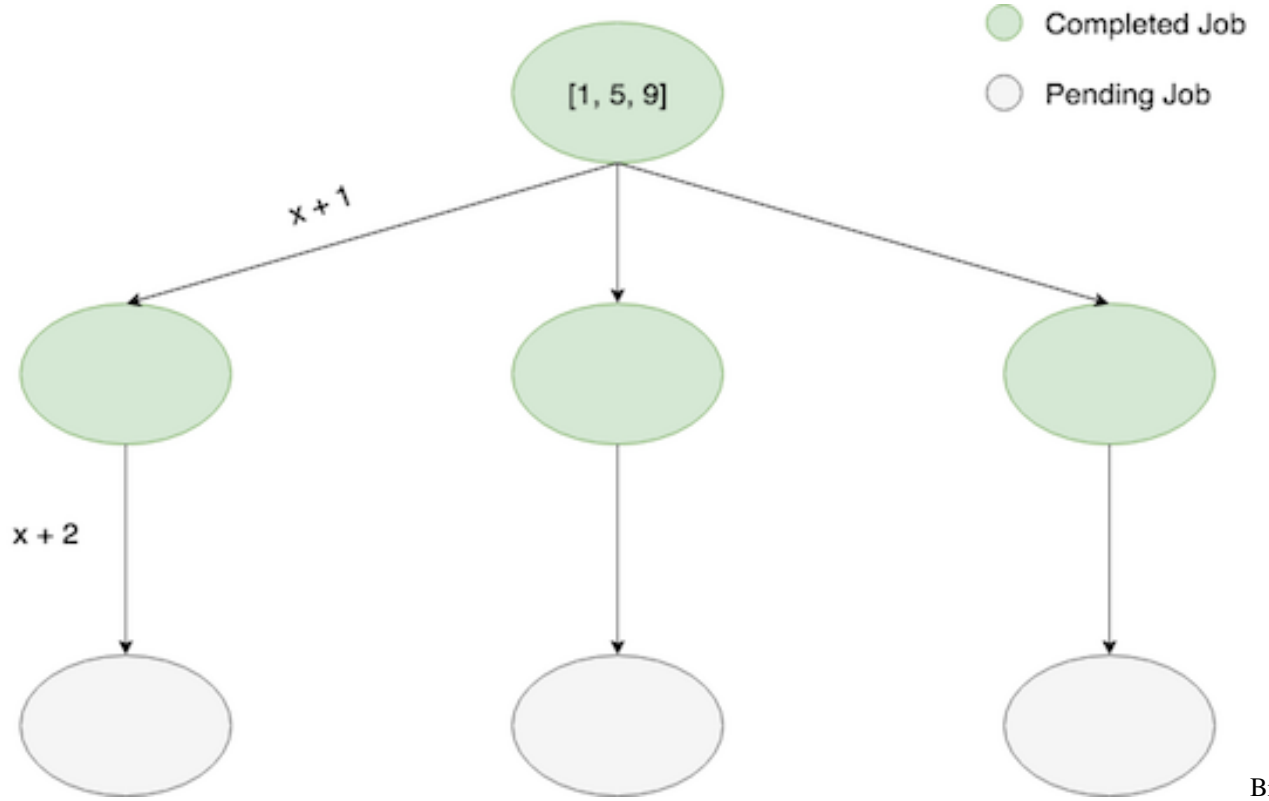
9.4 Pain points when using Dask

Our biggest pain point in using Dask has largely revolved around the ability (or lack thereof) to share futures between clients. To provide a concrete example, suppose we start with a list of numbers and, using `client.map` twice, we proceed to compute `x -> x + 1 -> x + 2` for each element of our list. When using only dask primitives and a single client, these computations proceed asychronously, meaning that the final computation of each branch can begin without waiting on the other middle computations, as in this schematic:



Depth First Execution

However, in Prefect, we aren't simply passing around Dask futures created from a single `Client` - when a `map` operation occurs, the dask futures are actually created by a `worker_client` and attached to a Prefect `State` object. *Ideally*, we would leave these futures unresolved at this stage so that computation can proceed as above. However, because it is non-trivial to share futures between clients we must `gather` the futures with this same client, making our computation proceed in a "breadth-first" manner:



Breadth first execution

This isn't the worst thing, but for longer pipelines it would be very nice to have the faster branches of the pipeline proceed with execution so that final results are produced earlier for inspection.

9.5 Technology we use around Dask

Our preferred deployment of Prefect Flows uses dask-kubernetes to spin up a short-lived Dask Cluster in Kubernetes.

Otherwise, the logic contained within Prefect Tasks can be essentially arbitrary; many tasks in the system interact with databases, GCP resources, AWS, etc.

9.6 Links

- Prefect Repo
- Prefect on Dask Example
- Dask-Kubernetes

CHAPTER 10

Contributing

If you solve interesting problems with Dask then we want you to share your story. Hearing from experienced users like yourself can help newcomers quickly identify the parts of Dask and the surrounding ecosystem that are likely to be valuable to them.

Stories are collected as pull requests to github.com/dask/dask-stories. You may wish to read a few of the stories above to get a sense for the typical level of information. There is a template in the repository with suggestions, but you can also structure your story a different way.

10.1 Template

10.1.1 Who am I?

A brief description of who you are, and the name of the project for which you use Dask.

10.1.2 The Problem I'm trying to solve

Include context and detail here about the problem that you're trying to solve. Details are *very* welcome here. You're probably writing to someone within your own field so feel free to use technical speech.

You shouldn't mention Dask here yet; focus on your problem instead. Why is it important? Why is it hard? Who does this problem affect?

10.1.3 How Dask helps

Describe how Dask helps you to solve this problem. Again, details are welcome. New readers probably won't know about specific API like "we use client.scatter" but probably will be able to follow terms used as headers in documentation like "we used dask dataframe and the futures interface together".

We also encourage you to mention how your use of Dask has *changed* over time. What originally drew you to the project? Is that still why you use it or has your perception or needs changed?

10.1.4 Pain points when using dask

Dask has issues and it's not always the right solution for every problem. What are things that you ran into that you think others in your field should know ahead of time?

10.1.5 Technology I use around Dask

This might be other libraries that you use with Dask for analysis or data storage, cluster technologies that you use to deploy or capture logs, etc.. Anything that you think someone like you might want to use alongside Dask.

10.1.6 Other information

Is there something else that didn't fit into the sections above? Feel free to make your own.

10.1.7 Links

Links and images throughout the document are great. You may want to list links again here. This might be links to your company or project, links to blogposts or notebooks that you've written about the topic, or links to relevant source code. Anything that someone who was interested in your story could use to learn more.

We also strongly encourage you to include images. These might be output results from your analyses, diagrams showing your architecture, or anything that helps to convey who your group is, and the kind of work that you're doing.