EPISODE 1056

[INTRODUCTION]

[00:00:00] JM: Python is the most widely used language for data science, and there are several libraries that are commonly used by Python data scientists including NumPy, Pandas, scikit-learn. These libraries improve the user experience of a Python data scientist by giving them access to high-level APIs. Data science often performed over huge datasets, and the data structures that are instantiated with those datasets are so big that they might need to be spread across multiple machines. To manage large distributed datasets, a library such as a scikit-learn can use a system called Dask. Dask dad ask allows the instantiation of data structures such as a Dask data frame or a Dask array.

Matthew Rocklin is the creator of Dask. He joins the show to talk about distributed computing with Dask, its use cases and the Python ecosystem. Matthew also gives a detailed comparison between Dask and Spark. Spark is also used for distributed data science. It's a show about the future of applied distributed systems for data science. I hope you enjoy it.

I also want to mention, I've started doing some investing. If you are an entrepreneur with a great idea particularly one around engineering, send me an email, jeff@softwareengineeringdaily.com.

[SPONSOR MESSAGE]

[00:01:30] JM: When I'm building a new product, G2i is the company that I call on to help me find a developer who can build the first version of my product. G2i is a hiring platform run by engineers that matches you with React, React Native, GraphQL and mobile engineers who you can trust. Whether you are a new company building your first product, like me, or an established company that wants additional engineering help, G2i has the talent that you need to accomplish your goals.

Go to softwareengineeringdaily.com/g2i to learn more about what G2i has to offer. We've also done several shows with the people who run G2i, Gabe Greenberg, and the rest of his team.

These are engineers who know about the React ecosystem, about the mobile ecosystem, about GraphQL, React Native. They know their stuff and they run a great organization.

In my personal experience, G2i has linked me up with experienced engineers that can fit my budget, and the G2i staff are friendly and easy to work with. They know how product development works. They can help you find the perfect engineer for your stack, and you can go to softwareengineeringdaily.com/g2i to learn more about G2i.

Thank you to G2i for being a great supporter of Software Engineering Daily both as listeners and also as people who have contributed code that have helped me out in my projects. So if you want to get some additional help for your engineering projects, go to softwareengineeringdaily.com/g2i.

[INTERVIEW]

[00:03:19] JM: Matthew Rocklin, welcome to show.

[00:03:21] MR: Thanks for having me. It's great to be here.

[00:03:24] JM: Python is the most widely used language for data science, and there are several libraries that are commonly used for data science within Python. You've got NumPy, Pandas, scikit-learn. What problems do these libraries solve for data scientists?

[00:03:44] MR: Yeah, that's actually a huge question. I'd actually say that, first, there're actually thousands of other libraries. Most people who use Python for data science, they're certainly using those ones as a baseline where there are lots of other libraries they're bringing in. If they're doing natural language processing, they're doing NLP, and they're doing NLTK, and SpaCy, if they're looking at time series analysis, they're bringing in CyPy and other things. There're all of these sort of ecosystem of libraries coming in. They're all being active.

But as you said, NumPy, Pandas, scikit-learn are probably the foundation. These are just people who are good at doing a bit of analysis of the data science. They're probably trained as something not computer science, but they want to do a lot of computation that is fast. Python is

just really a nice language. It is really useful, really usable by humans, but it's also written by computer scientists. It usually touches a lot of C code, a lot of Fortran code, a lot of battle-hardened numeric systems. Python really bridges that gap between usability and hardcore numeric that's really, really useful for computing.

[00:04:49] JM: Okay. I guess a way of looking at this is these are just libraries that are making it easier to perform data science machine learning operations.

[00:05:02] MR: Yes, definitely.

[00:05:03] JM: Okay. For many of these applications that I am building around data science or machine learning, I've got a rather large dataset. Maybe the dataset can fit on a single machine, and my machine has both disk and RAM. Can I use both of these storage mediums for data science? If I've got a really large dataset and I want to process it on a single machine, am I able to utilize both the RAM and the disk on that single machine?

[00:05:36] MR: Yes. If you're clever, using those libraries, you can. You might load up some CSV file into RAM, process it, throw it out of RAM, load up some other CSV file, process it, throw it out of RAM. If you're clever, you can do a lot of this work manually by loading in data processing and then destroying it, of there a variety of libraries that will try to reuse NumPy, Pandas, scikit-learn, but in a more scalable way. We do the manual work for. So you could just lever it like spark. That wouldn't work with those libraries, but it is something. You can use Dask, a library that I work on that will do that for you. I'll just maybe talk about that for a second.

With Dask data frame, you might be given a Pandas-like API, but it will load up data, process it in chunks and give you sort of the illusion that you're operating on terabytes of RAM in-memory, but actually it's using disk in an intelligent way.

[00:06:32] JM: Awesome. Basically, even on just the single machine instance for Dask, it can page the large dataset and use that paging to utilize disk in an efficient fashion.

[00:06:50] MR: Yeah, that's correct. Ideally, we don't have to page. Ideally, we can stream. We can find some very clever way to stream through your dataset in one go without us going back

and forth to disk all the time. But you're right. In the worst case scenario, we go back and forth the disk.

[00:07:05] JM: I see. You're describing paging versus streaming in the sense that if I just want to perform some NLP operation across my entire dataset, that dataset does not fit in RAM. I can just incrementally pull in chunks of the dataset, run the operation on each chunk and have the result. But in some situation where I would need to process the entire dataset multiple times, like if I needed to run one NLP operation and then run another NLP operation and then run another, I'm going to have to be going back-and-forth.

[00:07:42] MR: Yeah, and you can imagine it being more complicated. You're running some NLP operation. You're running some other one. You're then joining those two together. You're subtracting off some aggregate off of the first computation. Actually, it's really complicated workflows. Now the question is can you find some way to walk that entire workflow in a way that minimizes your memory use at any one time? That ends up being actually really interesting like analytic and research problem to do.

[00:08:08] JM: Let's talk about that. When I want to scale up Python data science application beyond a single machine, what are my options?

[00:08:20] MR: You have a ton. You could write it yourself is going to the first answer, right? You could learn how sockets work and you could learn how to NumPy and Pandas works and you could make some for the queuing system. You could use MPI. MPI is like an old scientific computing system that's release used for big iron competition. You could use a sort of Hadoop, Spark, Flink data engineering stack.
You could use Dask. There's a bunch of frameworks that will try to solve this problem for you. It's actually a really good time to be looking for big data solutions in Python because there's a lot of competition right. You could also use a database, something that was really geared for just one kind of system. If all you want to do is SQL queries, you just be using a database.

[00:09:04] JM: You're the creator of Dask, as you've mentioned, and this is a system for scaling Python. Give an overview of what problem Dask solves.

[00:09:15] MR: Yeah. As you mentioned at the top of the show, you have NumPy, Pandas, scikit-learn, they're very popular. Then fortunately don't work well when your data gets too large or when you have lots of computers to use. They were designed fundamentally for single machine.

Dask is a Python library that was designed to scale out those libraries. We made Dask in order to partner with NumPy, partner with Pandas, partner with scikit-learn to make scalable versions of those libraries. What Dask provides is a delivers all of the networking, load-balancing, task scheduling, parallel algorithms to write down one data frame join. There's lots of other smaller data from joins, and integration with things like Kubernetes, the cloud, YARN, general deployment techniques. DASK you think of as being a framework or a development kit that helps developers parallelize existing Python libraries.

[00:10:13] JM: If I'm thinking about the problems in doing distributed processing on a long, a large dataset, I have the dataset, I'm going to load it into some data structure and then I'm going to process that entire data structure. There are two parts of distributed computing to talk about here. One part is the fact that you have this enormous dataset and you've got to figure out how am I going to make a multi-machine data structure that can hold that dataset. The other question is how do I orchestrate my machines to do distributed processing across that distributed dataset? Let's first talk about the collection. If I'm wanting to make a collection like an array or some kind of bag scalable, tell me how to do.

[00:11:10] MR: Yeah, that's a great question. Yeah, I think you separated it really cleanly, and there're actually different kinds of people that talk about both kinds of problems, right? If you want to make a parallel array, you're talking about parallel rhythms. People we think studied math. People studied linear algebra, that sort of thing.

Yeah. I mean, you could think of let's look a couple of different kinds of algorithms. A really simple one, s um. You've got a big array. Let's say you're at looking at the temperature across the globe for the next 50 years. Maybe looking at global warming for the climate. You just want to see does the temperature increase or decrease? Your big array, to cut up the earth into lots of little cubes, maybe every cube for every 1000 km² or 100 km² or something. Maybe those cubes

might be a little NumPy array. We can already use NumPy. NumPy is already really good for handling those sort of multidimensional array data structures.

If you want to compute the sum, you just compute the sum of all those little NumPy arrays and then you would sum up all of their sums. Really simple competition. It's sort of the straight MapReduce kind of thing. A more complicated algorithm, maybe you wanted to do matrix multiply. You could do a big matrix multiply in terms of lots of little small matrix multiplies, but it's actually really complicated the structure of which the little NumPy array has to multiply I guess which other NumPy ends up becoming very complex. Maybe you also want to subtract off some set of deviation. That's also very complex.

You end up building some fairly complicated task graphs, some fairly complicated recipes of how to compute these large array computations in terms of smaller in-memory computations. A product like Dask array will parallelize NumPy by writing all those recipes for you. It knows that if you have a million by million array composed of 1000 by 1000 NumPy, it can figure out which NumPy arrays to talk to each other one in order to do the larger matrix multiply, for example. If it's something simpler, like computing the sum, it will do all the sums.

[00:13:23] JM: Can you tell me more about what you're doing to distribute – In Dask, if I want to instantiate a really, really big distributed array, what kinds of work are you doing in Dask to instantiate that array?

[00:13:42] MR: Yeah. Let's go with the climate science example again for second. Let's say that you are –These are usually run on big iron supercomputers. We're talking maybe around the cloud and Kubernetes. Maybe were on some HPC system, and there's probably some file, maybe sort of like HDF5 or NetCDF. These are file formats commonly used for these large arrays, and there are Python libraries already to read blocks of NumPy arrays out of these larger files. So we just need to call that Python function that already exists thousands of times across your many machines in order to load up all these different chunks of this NumPy array.

Now, we've got these thousand machines each holding maybe 10 NumPy arrays and now we need to sort of map and figure out which for this particular NumPy array, where does it fit in the broader picture? Maybe this is the NumPy array that corresponds to the temperature over

France, for example. On this other computer is a NumPy array corresponding to the block of temperature over Italy. We know that if we want to sort of look at the Italy-France connection, we need to have those two machines.

Dask is a really a system that's watching all those machines and is tracking all those Python objects and is as necessary telling those machines what to do, “Okay. It’s now time for the machine holding France to compute its sum. It’s now time for the machine holding Italy to transfer that array over to the machine holding France so that we can do some interaction.”

There're two problems here. One is figuring out a plan of which arrays need to talk to each other and then executing that plan, which is a lot of talking to all the machines, make sure they’re doing the right thing. If one machine goes down, making sure the work that was on it gets replaced.

[00:15:34] JM: Let's start with the fault tolerance. Let’s say I load this array into memory across all these different machines within my data center, and instantly one of them fails, and I've got to recover the dataset that was on that machine. How does that fault recovery system work?

[00:15:57] MR: Yeah. There is a system within Dask called the scheduler, which tracks what everything it’s doing. They’re sort of the mastermind of the entire system. If it notices that one worker has gone down, it has been storing a record of what that worker was holding and it can replay that work on other workers that are nearby. Similarly, if a new machine comes on, that’s scheduler can say, “Hey, great! I’ve got three workers that are already saturated with work. Let me share the work to this new worker that arrived.”

At its core, the Dask scheduler is just a giant state machine that is watching all the events that all the workers are producing, saying, “Hey, I finished this work. What’s next?” and also washing the various clients who are submitting work and also watching the workers to see if they fail or get recreated.

[SPONSOR MESSAGE]

[00:16:52] JM: DNS allows users to navigate to your web endpoints, and whether they're there hitting your app from their mobile phone or accessing your website from their browser, DNS is critical infrastructure for any piece of software.

F5 Cloud Services builds fast, reliable load-balancing and DNS services. For more than 20 years, F5 has been building load-balancing infrastructure, and today, F5 Cloud Services provides global DNS infrastructure for lightning fast access around the world. If you're looking for a scalable, high-quality DNS provider, visit f5.com/sedaily and get a free trial of F5 Cloud Services. Geolocation-based routing, health checking, DDoS protection and the stability and reliability of F5 Networks. Go to f5.com/sedaily for a free trial of F5 Cloud Services; fast, scalable DNS and load-balancing infrastructure. That's f5.com/sedaily.

[INTERVIEW CONTINUED]

[00:18:01] JM: We'll talk about the contrast with Spark a little bit more in the future, but while we're on the subject of fault tolerance, Spark had this nifty observation that you could re-create an intermediate dataset by replaying the operations that had been applied to the data before the existence of that dataset. I think it's called like lineage-based computing or something. It sounds like you have taken the same approach to failures in the in-memory collection maintenance. Is that correct?

[00:18:43] MR: Yeah, that's fair. Your two options for resilience are either store everything many, many times, which makes sense for databases and makes sense for persistent storage, but it doesn't make sense for computational systems. It's way too slow. The other approach is you just do re-computation when necessary, and that's what most computational systems like Spark or Dask choose to do.

[00:19:05] JM: Just a little bit more on the data structures that people can instantiate on multiple machines, just a couple examples are Dask data frame and Desk array. Can you compare the implementation of these two data structures?

[00:19:23] MR: Sure. Dask data frame is bunch of Pandas data frames. Pandas is often used like where you might use SQL. Think of it maybe as like a like a large Excel table. You have a lot

of columns of homogenous type data to the name column of text and a value column of decibel or float points. Yeah, common for business applications, common for anything that's sort of tabular.

A Dask array is a bunch of NumPy arrays, and NumPy, sort of the climate science example, a bunch of multidimensional gridded structures. Think of genomic data. Think of satellite imagery. Think of medical imagery. Think of simulations of fluid flow over aircraft wings. But there's actually lots of other ways people use Dask. Those maybe the most common two, but they maybe account for 50%.

The Dask maintainers maintain things like Dask data frame or Dask array, but there are other projects like xarray built on top to do a lot of geosciences work, products like Prefect or kind of an Airflow alternative. Airflow itself runs on top of Dask optionally. Dask really gets used in lots of different kinds of systems. It's much more a framework for building parallel frameworks that it is a particular data frame implementation. Again, I'd say half of the Dask usage comes from people building their own stuff on top of Dask, which is where I think it really shines.

[00:20:48] JM: The difference between a data frame and an array, and I think this is beyond the scope of just Dask. Is it the fact that a data frame has named columns? Is that the main difference or are there other differences between a data frame and an array?

[00:21:03] MR: That's really one difference. I would say the biggest difference in my mind is that arrays can be multidimensional. If you look at like the pixels in an image, you can't put the pixel of an image inside of a data frame structure, a linear structure, efficiently. You actually really want all of the neighboring pixels in an image to be close to each other in-memory. Actually, they're just laid out differently. If you go from an image to maybe like an FMRI dataset, if you're looking at a scan of your brain, you really can't put that into like an Excel table or a linear data structure like a data frame.

Certainly, data frames do have names, but like xarray is a project gives names to arrays as well. I think the main difference is are you single dimensional and doing mostly business analytics applications? Group by joins, filtering, or are you looking at multidimensional data and you can

maybe worry about FFT's or overlapping computations or neighboring computations. That tends to be the biggest difference between the two.

[00:22:04] JM: Now, I do want to get to the processing part of things, but I'd like to know what in designing the Dask, designing and implementing the Dask data structures, what are some difficult engineering problems you've had to solve?

[00:22:23] MR: If you're talking about in terms of data structures, think about like Dask arrays, Dask data frames, they weren't actually that difficult to be honest. Certainly, you need to have an understanding of parallel algorithms. How do you do a parallel join in a data frame? How do you do a distributed matrix multiply? Those are hard problems, but they're also solved. There are lots of research papers around how to do those things. There's lots of prior art. You need to be a competence, but not necessarily brilliant engineer to do those things. They exist

I think the real challenge is, is one of community. Desk array follows a NumPy API pretty much exactly, and the Dask data frame follows the Pandas data frame API pretty much exactly. We use those libraries under the hood. We are intimately tied to the existing Python ecosystem. I think the real challenge or the real accomplishment of Dask is that we're able to move an entire ecosystem of software that was not originally designed to be parallel to become parallel, and a common issue of engineering talent or engineering effort certainly, but is a lot more of community understanding and developing rapport and developing protocols and having a lot of groups that historically do not communicate a ton start to communicate about some of these hard problems.

[00:23:43] JM: Okay. Well, let's talk more about the processing side things. If I want to execute an operation across one or more of these large data structures that I've got sitting in-memory across a data center, describe how that works. What's involved in the scheduling and the processing of these tasks?

[00:24:05] MR: Yeah. I feel like most of my answers to you honestly are like, "Oh! Actually it's way more big, way more complex than you're asking about." I'll do that gain. Yeah, there's also a ton of other things. How are we loading data? Is it on S3? Is there an S3 library I Python that's

good enough to load data? How is it compressed? Did the person writing the compression library in Python? Are they still around. There's a fun of ton problems.

[00:24:26] JM: It almost sounds like from the way you're saying it, it almost sounds like none of this stuff is groundbreaking computer science. It's a lot of implementing stuff that is prior art and sanding the edges in order to have a thoroughly designed ecosystem with all the edges sanded and all the imports easy to use.

[00:24:49] MR: Yeah, I really like that analogy actually. We did a lot of standing. Dask really is taking Python's data science stack; NumPy, Pandas, scikit-learn, and Python's web development stack; Tornado, Async IO and smashing them together, right? Desk is a peer-to-peer networking application running with TCP servers as a scheduling bit. It's all written on top of Tornado, a popular Python networking labor. But the message we're sending back-and-forth are these NumPy, Pandas, scikit-learn data structures.

We're taking these really robust pieces that were in this existing ecosystem and we're just smashing them together. Then we're going around and finding all the pieces that we're missing, parquet readers or Amazon S3 readers and making sure that we're partnering with various groups that are building those pieces and make sure that everything works out well.

[00:25:39] JM: Hearing you say that, it's actually really interesting thinking about the history here, because like Java developed these kinds of things much earlier I think just because of happenstance. You had, for example, like trading companies, building Java infrastructure, and they needed the great networking libraries. They needed the great big data libraries. Like same with Google. Google I think built most of their like services infrastructure around Java, for example. I mean, I'm sure Google has plenty of Python also. But for the most part, Java usage – Of course, you just think about the Hadoop ecosystem, like all the distributed systems people were just like Java people.

The byproduct of that is you have this dividend of lots of great libraries and imports and import a parquet file kind of things, import from S3 kind of things, just pre-baked into the ecosystem. From what I'm hearing, it sounds like Desk is sort of a really strong effort to give that same robust filled-out nature to the Python ecosystem.

[00:26:56] MR: Yeah, I think that's fair. I mean, it's not just Dask. Dask is one library and sort of a broader effort that many people are currently working towards. Yeah, I think you've hit on something that's really interesting actually, and it's actually at the core of a lot of tension that we see today in business. The data engineering had the most businesses, as you said, is written in Java or some JVM language, for example, Skala with Spark. But the data science stack is all written in Python. The same time that businesses were building out critical infrastructure in Java, all the scientists and all the math people and all the data science people were building out a lot of really exciting analytics capabilities in Python, C, Kudu, C++. Now we want to do both of those at the same time. Want to do some engineering and then switch it to data science? Do data science at scale. That, no one has a good answer for.

Typically, what happens today is that you have data scientists working in Python on some data MLR machine and they want to read at scale and the rewrite it in Spark. That process is troublesome, right? For switching from one stack to another stock. The Spark folks are trying to make Spark more friendly for data science. That's a huge effort. There are so many algorithms to write. There is so much user research to do. It's incredibly hard, and that's where I think the Java stack really falls down, is understanding how to do complex algorithms in ways that data scientists are happy using.

The Python stack was never good at distributed computing, except with systems like MPI. Both stacks are really trying to fill in the gaps that the other stack does well. Yeah, Dask is definitely a major effort, and all the things we do around Python are trying to fill in all those gaps to make us work at scale more effectively.

[00:28:46] JM: The approach of high PySpark I think is Python APIs into the Java-based Spark infrastructure, which makes for a better experience for a data scientist. It makes more familiar experience for a data scientist. I know there's some issues there, because if you're operating in Python, then you have these Python-based data structures and you have to figure out how to serialize them and deserialize them into Java data structures, but you have Apache Arrow which helps with that. But it just sounds very kludgy altogether. Maybe that's the issue.

[00:29:23] MR: Yeah. Actually, people always point to serialization as being the major problem. I think that's not the major problem with PySpark. I think that causes some performance woes, but it's actually not a huge deal. Lots of times, for example, using Arrow, that's not like – You can work around that. I think what you can't work around though is a problem of complexity. Again, Spark is really designed for more data engineering workloads. I want to read a bunch of JSON data, I want to parse it, I want to filter it, maybe do some database merge, throw it into parquet.

When you start doing more advanced workloads, more data science or machine learning, the Spark execution engine really starts to fall down. It's just not sophisticated enough. Spark us fundamentally at its core a MapReduce system. It runs maps. It runs shuffles, sort of all-to-all communication and it is aggregations. Everything that Spark does fundamentally rolls down to one of those system, one of those operations.

If you start looking at something like a matrix multiply or like SVD or looking at random-access or time series operations, that fundamental system of execution, that MapReduce paradigm, is not sufficiently flexible to handle it. You run to a lot of flexibility problems when you're starting hitting data science machine workloads. There's a reason why Tensorflow was not built on Spark, right? The kinds of executions that Tensorflow does, you just can't sort of fit it into the MapReduce paradigm of Spark.

This is where think – We started off with Spark. Our original goal was to make NumPy parallel and we said, "Great! Let's try to figure out how to do that with PySpark," and that failed very, very fast. Again, when you hit more complicated workloads, the Spark paradigm doesn't work well. But if you're just doing data engineering, if you just want to do data frame stuff, Spark is an excellent choice, and I think a JVM structure should definitely stick with that.

[00:31:19] JM: Can you say more about why the MapReduce-based paradigm falls over for machine learning workloads?

[00:31:29] MR: Sure. Let's look at a couple of examples. I'm trying to find an example that is accessible and also clear. It's like, "Oh, yeah. You can't do parameter servers with MapReduce, but that's probably not going to be super successful."

Let's do something simpler. Let's look at time series for a second. Let's say you have a time series dataset and you want to get a data for a particular day, a particular day of data. March 1st, 2018. You can do that by mapping a function overall of the data. This is like are you of this day? You get back a result. You shouldn't have to do that, right? You shouldn't build it to zero-in on the machine that has that particular day of data and pick out that particular result.

Spark doesn't provide any sort of random-access mechanism. You can't uniquely identify one piece of data and just get that piece of data out. You have to map over that. That's like a very simple case where Spark falls down.

If you look at I think like Two Sigma is a good example there. They actually had to fork Spark in order to provide this kind of functionality. It was like a very deep change to the system to provide that sort of simple change. Another example, let's say you want to do like a rolling average or anything that requires neighboring information, right? I have – Maybe all my data is partitioned by week and I want to partition that by month. Well, the weeks and months don't quite align, and so you need to do a little bit of work to sort of have neighboring tasks talk to each other. The way that Spark is arranged, that's actually not super easy to do. You can sort of identify individual partitions. It doesn't have that same kind of flexibility. You can do those operations, but it requires a more complicated sort of shuffle to all system, which ends up just being a lot slower.

[00:33:17] JM: Are you saying that there might be instances where you have more of a complex flow, a complex sequence of operations where you want subsets of the data to be processed in different parts of the DAG without having these shuffle operations that have all of the data congeal into this one place or having all of the servers orchestrate with one another? You kind of want to have some long-lived computation in very different parts of the DAG.

[00:33:55] MR: That's right. I think Spark expects a lot of uniformity of your computation, and that uniformity isn't always present. Look at Airflow, for example, right? In airflow, it also has a DAG, bit it's a very different kind of DAG where every node in that graph is a single execution that happened on one node. It's not necessarily a thousand node computation.

Airflow DAGs look of them tend to be a little bit more complicated. You could not run airflow on Spark. Certainly, you might have a Spark that was being one node in Airflow graph, but the Spark execution engine does not something you're going to run a very complicated not embarrassingly parallel workload on. Dask is kind of like a mix between the competition nature of Spark, but the flexible nature of Airflow.

[00:34:47] JM: Cool. From a design perspective, like is there any comparison you can draw between Dask and Tensorflow?

[00:34:58] MR: To a certain extent. I mean, all these systems, Spark, Tensorflow, Database, Flink. Internally, they all have a task scheduler. Internally, they've got – Extremely they've got some user API they give to users, some abstraction [inaudible 00:35:13], the data frame, the tensor, and then they do a bunch of work. Internally, they break that down to lots of tasks. They need to run in different machines. Now, there's the sort of scheduling problems that they all solve.

Dask fundamentally is just that lower level piece that does the scheduling. Dask is kind of like the shared component that Spark or Tensorflow or Flink or Database would have in common. Dask handles all of the bits like networking, like load-balancing, like resilience, we've talked before, but without having lots of opinions, but the kind of workloads that it's running.

As a result, you can use Dask to build things like Spark and we see that in Dask data frame or Dask ML. You can use Dask to build things like Tensorflow. You can use Dask to build completely other things that we've never thought of before.

Actually, when we're designing Dask, our original goal was to parallelize NumPy or just to give this sort of parallel multidimensional data structures. In order to do that, we had to build this really complicated task scheduler, which how this really complicated algorithm use C and multidimensional arrays.

Then we went to clients and we said, "Hey, great. Here's a multidimensional way. Isn't this awesome?" A lot of them said like, "Ah, that's cool, but actually we just want the task scheduler." You've built a really cool car, but I actually just want the engine of that car, because I am

building a rocket ship. We actually see a lot of this in the Python space, because in Python people tend to build lots of really strange things. Think of like hedge funds and quantitative finance.

Really, Dask is just the internal execution engine that you would find inside of something like Spark or Tensorflow, but we've made that accessible to the end user so they can build their own systems. Dask is everything you need to build your own Spark if Spark isn't enough for you.

[00:37:06] JM: Right. One, or a few examples of this, scikit-learn and XGBoost both the use Dask as a building block. Describe how one of these uses Dask as a building block, the scikit-learn or XGBoost, or if you want to talk about some other library.

[00:37:28] MR: Yeah. Sure. Those are both good examples. To be clear, Dask is not core to either library. It is an optional dependency. Yeah, XGBoost does this parallelism with a library called joblib, which does embarrassingly parallel computations. Sort of doing a grid search of a lot of different models. Joblib is running under the hood inside of scikit-learn. Kind of use N-jobs parameter. That's what it's using. It usually dispatches out to a thread pool or a process pool or because Dask kind of implements those same interfaces. It can dispatch out to Dask. You can wrap your scikit-learn code with a use Dask context manager and then scikit-learn will then dispatch these little tasks off to a Dask cluster running on thousands nodes in the cloud rather than on to your local machine. For scikit-learn algorithms, you can scale them up pretty easily.

XGBoost, they have their own system for parallel computing, but they can launch themselves on top of a Dask cluster. If you are using Dask data frame and you are on the cloud somewhere, you can just import XGBoost, run it on your Dask data frame and it will – We hand all the data off to XGBoost. We set up XGBoost. We give them the right network permissions. XGBoost does its thing, and then it hands data back to Dask.

[00:38:46] JM: How would the experience of somebody using a Dask data frame and XGBoost compare to what ever data structure they will be using otherwise? Whatever native data structure to XGBoost they would be using otherwise?

[00:39:01] MR: It's the same. I mean, we're not doing a different data structure. We're just using XGBoost. Yeah, sorry. XGBoost has a distributed XGBoost built into it. They built out their own parallel XGBoost system. We're not reinventing that. We're just making it very easy for Dask users to set up XGBoost in parallel, hand data off to XGBoost and then let XGBoost do its thing.

[00:39:23] JM: Oh right. Okay. Of course.

[00:39:24] MR: This is maybe a big point about Dask. Dask does not try to own the world. Spark made its entire ecosystem. They made everything. In Dask, we're working with existing tools. We have no desire to make gradient boosting trees condition. We just want to make it very easy to be able to connect those things together.

This gets back to the community aspect of Dask, right? We're taking all these pre-existing tools, and as you said, just sanding off the edges. That's really how we operate. We're actually a really small team and we've done a huge amount of work in the last few years because we're just partnering with all of the existing open source tools.

[SPONSOR MESSAGE]

[00:40:09] JM: I've recently started working with X-Team. X-Team is a company that can help you scale your team with new engineers. X-Team has been helping me out with softwaredaily.com and they have thousands of proven developers in over 50 countries ready to join your team and they can provide an immediate positive impact and lets you get back to focusing on what's most important, which is moving your team forward.

X-Team is able to support a wide range of needs. If you need DevOps, or mobile engineers, or backend architecture, or ecommerce, or frontend development, X-Team can help you with what you need. They've got a full-range of technologists who can help with AWS, and Go lang, and Shopify, and JavaScript, and Java. Whatever your engineering team needs to get to the points of scale that you want to get to, X-Team can help you grow your team. They offer flexible options if you're looking to grow your team efficiently, and their model allows for seamless integration with companies and teams of all sizes. Whether you're a gigantic company like Riot Games, or Coinbase, or Google, or if you're a tiny company like Software Daily. You can get help with the technologies that you need. If you're interested, you can go to x-team.com/sedaily.

That's x-team.com/sedaily to learn about getting some help with your engineering projects from X-Team.

Thank you to X-Team for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:41:54] JM: Dask is a way of creating these Dask data structures, the distributed data structures. If I am using XGBoost to run computations across those data structures, am I using the Dask scheduler to run those computations?

[00:42:15] MR: No. When you're using the XGBoost bit, the Dask scheduler has control off to XGBoost and it lets it do its thing. I mean, XGBoost devs are really, really smart. They built a nice system to an extra boost of parallel way better than we could. We have no desire to replace that work. It works nicely.

[00:42:32] JM: What are the instances where the Dask scheduler would be used?

[00:42:39] MR: Yeah. Let's look at another example, maybe Prefect. Prefect is like an Airflow competitor. Some of the Airflow devs left. Airflow is great, but it doesn't evolved in the last few years. Lots of features people would like to have. They made a new product called Prefect and they wanted Prefect to be scalable, responsive, handle in a worker communication. They're like, "Hey, let's just build this on to of Dask."

Dask handled half of their work for them, all the execution coordination. Well, they get to build lots of business logic like the sort of the cron aspects of Airflow or the various niceties that Airflow provides.

That's a good example where someone is using the Dask scheduler in a more intimate way. If you look at – There are like lots of large banks today. They do run a credit risk application every time you apply for a credit card. You should go to a webpage, you apply for a credit card and they want to then figure out what should your credit limit be. How much should they value you? They run a bunch of little models on you. Are you a student? If so, what do you major in? Do you own a home? What's your ZIP code? What's the risk in that ZIP code? What's the flood risk

in that ZIP code? There's this massive web of thousands of little machine learning models that's really complicated. That's run on top of Dask today in I think like three or four major banks. There's a chance that your credit card in your wallet runs on Dask every time you apply for it.

This is the case where people are using Dask not to build a big Spark-like thing. They're building it actually just integrate it into their existing business logic. Many people have for four-loopy Python code where there's clearly some possible parallelism, but it's not just the big data frame. It's not as big array. It's not a big embarrassingly parallel problem. It's more complicated. People – Again, a lot of Dask used is to parallelize custom bespoke systems often that have been around for years.

[00:44:39] JM: That have been around for years written in Python.

[00:44:42] MR: Right, or written in C, C++ but have links to Python. That's quite common.

[00:44:48] JM: Their options for parallelizing those things are – Well, I guess you described some of these options at the beginning of the show, but if they were to try to use Spark, what would be the bottleneck of the impediment or the issue?

[00:45:02] MR: Spark doesn't handle that, right? It's not like they're running a machine learning model on lots of data. They're running lots of different kinds of machine learning models on this data in a really complicated web. It's not this really structured problem. It's a really unstructured problem. There's lots of concurrency in that problem. There are lots of opportunities to run things at the same time, but it's not this massive run this function across all of my data situation. It's way more complicated. Dask really thrives in those really complicated situation. Does that make sense?

[00:45:35] JM: Yeah. I think so. I mean, I'm kind of imagining the way that Dask operates is it just gives me access to thinking of this distributed system as a more fine-grained way of allocating memory and performing computations across the objects in that space rather then I guess the Spark system, which would require me to think of really large datasets that.

[00:46:05] MR: We often find is that people who have these sort of complicated workflows, they try to rewrite it in such a way that it fits into the Spark way of thinking. They say, “Oh! Great! I can make this RDD. I can filter it out. I can do some group by operations,” but changing your way of thinking about doing that, you end up actually losing a lot of the problems you were trying to solve. Dask sort of doesn't force people to fit their problem into the certain paradigm. Dask is much more flexible. You can use a little bit of Dask to wrap around your existing system, and Dask will give you lots of tools. I mean, Dask has distributed queues, it’s got contributed locks, it has distributed task scheduling framework. It has all of these systems that you can use to build out your own system. That, again, ends up being useful when you get a little bit outside of the sort of traditional business analytics workflows.

Another way to think of this is that maybe most problems can be solved by a database. If you can write your problem in SQL, you’re set. Now, just outside of the database, those problems that don't quite fit, and that’s where Spark is really useful. It's almost a database probably, but not quite. You want to do a little bit of ETL, you want to do a little bit of lightweight machine learning. Great! That’s where Spark is useful.

You go outside of that again, there is even like another periphery of problems where it's not a bulk problem, where it’s more complicated. That again is where is Dask is useful. Then outside of that stuff, there’s lots of problems where Dask is not going to be useful and you have to sling your own code. Here you’re dealing with sockets and you’re dealing with your own custom systems. Dask sort of fits that problem of building out stuff where Spark isn't really flexible enough.

[00:47:53] JM: As far as building the scheduler, I can imagine a difference between operations that you want to want to schedule on, embarrassingly parallel kind of situations where you don't need to do as much coordination between the different intermediate datasets. I can also imagine stateful computations where you have more intermediate coordination that you need to perform between different datasets that are being processed along the DAG. Were there any specific design decisions that you made in building the scheduler that pertained to stateful versus stateless computation?

[00:48:38] MR: If you are doing complicated data science competitions, you have to deal with state. I mean, stateless, embarrassingly parallel problems are really common, a really important use case. Any system will do them well. You can just Dask. You can use Spark. You can use MPI. You can do anything. [inaudible 00:48:55]. You can use lots of things. Yeah. When you hit, again, more complicated algorithms, if you require more flexibility, you definitely need to think about state. Any system that Dask [inaudible 00:49:06] think about state.

I may have lost your question though. You said are there any considerations?

[00:49:11] JM: It was a muddy question. I guess I'd like to know little bit more about the scheduler implementation and just the coordination side of things. What's the master versus replica or as a master, like coordinator node kind of thing? What's you what's your set up for kind of orchestrating and talking to all these different nodes that are doing operations there?

[00:49:33] MR: Yeah. Sure. Yeah, Dask is a centrally controlled distributed system. There are a bunch of workers, which – Who still are many of them and they do work and they hold on to data and they communicate peer-to-peer. They're all usually TCP servers talking to each other. Then there's also the scheduler, which is the central coordination point.

The scheduler talks to all of them and tells them what to do. Fundamentally, what the scheduler does is it has a task graph inside of it where it is tracking all the task has to do. It might be millions of these little tasks, each of which is a Python function. There is some frontier across that task graph that is currently running on all the workers. Some worker will say, “Hey, scheduler. I just finished task 10,000. It was this size, it returned this data type. It took me this long to run. What should I do next?”

The schedule says, “Oh great! It updates this little – Its understanding of the graph,” and says, “Ah! Two more tasks have been made available.” Then it needs to quickly figure out where to run those tasks. It says, “Oh! Well, wait. [inaudible 00:50:39] just done. It’s got some free capacity, and also it has the dependency that this task requires.” That would be a good worker to run this task on. It sends that task off to that worker. It might say, “Hey, this new task actually requires two pieces of data.” It needs to make a choice about which worker to send that task too based on the amount of data that we need to just move around, the backlog in each worker. If

there are any dependencies on that task, like a GPU or high-memory. The scheduler is just making these very, very rapid decisions about getting new information that the graph has been updated and then deciding where to send tasks. It has to do that in sort of the like few hundred microsecond time window.

The schedule is just this state machine living in one machine that listening for all these signals from all the workers and from clients that's sending it work, and then updating that graph at sending up messages to everyone else to make sure that everyone is in sync. It's kind of like a switchboard operator. Making sure that everyone or a foreman at a job site makes sure that everyone is working on the right thing that are all – They're not moving too much data around. If one of them gets hurts, it's swapping someone else in. That's maybe like what a task scheduler does. I think the foreman at a job site is probably a good analogy.

[00:51:59] JM: We had a show recently about the Ray Project and the company around that, Anyscale. How does Dask compare to the Ray framework?

[00:52:10] MR: Yeah. Ray is really cool. Ray is similar. They're both similar that they're both libraries that target Python users mainly that provide flexible parallelism. They're trying to do trendy things that are more complex than Spark.

I would say the biggest difference comes maybe about where they were started or kind of problems they work on. I think Ray really focused originally on reinforcement learning, which is sort of like close to the deep learning space. They're focused on doing really advanced machine learning things at scale. I think they're probably trying to replace tools like Horovod or sort of distributed Tensorflow.

Dask started off more with sort of more of a data science focus. How do we get parallel NumPy? Parallel Pandas? Parallel scikit-learn? I think Ray is also maybe trying to like build its own ecosystem. There is all – A lot of things built on top of Ray to be able the buy the Anyscale. Dask is really much more of a components of a larger sort of Python data science ecosystem. We're trying to be sort of minimally creative and reuse a lot of components.

We talked about XGBoost before. We're not making our own XGBoost library. There's no reason to do that. We're just reusing – Seeing how we can sort of give XGBoost a little bit of a boost so that it's easier to use with the existing parallel ecosystem. On a technical level, [inaudible 00:53:27]. Ray is a distributed scheduler, which means that it makes the scheduling operations happen everywhere rather than one centralized location. It's like a job site where there's no foreman, where all the workers are the foreman, which like has pros and cons depending on how you want to schedule something. Yeah, those maybe are the differences. You shouldn't believe me. I'm super biased towards Dask.

[00:53:51] JM: That's fine. Speaking of which, you started Dask company. What you can about the company?

[00:53:57] MR: Yeah. The company is coiled computing, coiled.io. Yeah. What can I say about the company? Yeah, for a long time, I do whatever is necessary to make like the Python ecosystem grow better. That's kind of been my purpose of life for the last 10 years. What I found is that we were limited not by technical ability, but by the ability for companies to interact with some company.

Often, we will go to like NASA, for example, and say, "Hey, NASA, you should maybe using Dask," or they're coming to us saying, "Hey, we want to use Dask." Then they'll say, "Who can we buy this from?" You say, "Oh, you don't need to. It's open source." They say, "No, we need to buy this from someone. We need to have enterprise support. We need to be able to go on the phone call with someone, or we don't want to set this up ourselves. We don't have enough Kubernetes technologists internally. Can you just make something that makes it easy for us to manage these things internally?" That conversation happened enough times that it was clearly necessary to make a company that supported Dask.

Today, Coiled does a few things. We are making a cloud deployment product. Something maybe like Amazon EMR or like a lightweight Databricks that makes it easy to deploy Dask on the cloud. That should be up hopefully in the next few weeks as a beta version. We also do enterprise support where companies that need to make sure that Dask will always fit their needs, pay us some amount of money. It's just as sort of an insurance policy. If something

breaks that's critical, we fix it very quickly for them. Then, three, we're also partnering with groups that want to parallelize other open source projects.

I was talking to a genomics group the other day. There's a lot of genomics data that's growing. They want to make sure that there's an easy way to process that. Dask provide 80% of what they need. They want to help us build out the extra 20%.

Again, we sell deployment products. We sell enterprise support, and we sell services if it's building out open source technologies.

[00:55:57] JM: Very cool. I guess last question, what did you learn about managing at a project like this? Managing a large open source project in your time at – You spent time at both Anaconda and Nvidia, right?

[00:56:10] MR: That's correct.

[00:56:11] JM: Just give your macro perspectives on managing a project like this. Now that you've done so within two different companies and as an independent person and starting your own company.

[00:56:25] MR: Yeah. That's a whole podcast of content right there.

[00:56:27] JM: I sure.

[00:56:28] MR: First of all, I don't really manage Dask. Dask is made by individuals, four or five different companies right now. There is no centralized control. I'm kind of the figurehead, because I has been around for a long time, but lots people work on Dask that are not me. I probably do the minority of work today. Also Dask, it's not just Dask, it's the community. I spent as much time talking to the NumPy developers or the scikit-learn developers as I do the Dask developers. There's not like one team that works on Dask. It's a bunch of disparate people. It's much more of an anarchy. I'm more like a cat herder than a sheepherder maybe is the right term here. It's not management. It's seeing what people care about and seeing how to support their needs.

That being said, the Dask developer community is very remote, very distributed. I think there's maybe two developers live in the same city out of 20. Actually, there's probably a few people in Paris, but it's fun. It's great. It's a great community. Everyone does Dask as like 25% to 50% of their time. It's really trying to figure out what their interests are and how to engage them and how to make sure that people talking to each other in a way that makes them productive.

Yeah, there wasn't really an answer for you. That was with like management. It's very soft management. It's only in the last sort of few years I've actually had employees that I could direct do things. For the last many, many years most of Dask development is not quite a volunteer. Everyone's paid. But it's definitely like convincing their bosses that they should work on Dask for a bit, and trying to align those perspectives is interesting.

This is true the whole Py data stack, right? The Python data science stack is novel and that it's not made by a company, right? In Spark, there's Databricks; and Hadoop, there's Cloudera. For Python, I guess there's Anaconda, but they don't actually control NumPy, Pandas, scikit-learn. There's actually a bunch of University professors and a bunch of interesting individuals. The Python community is great because it is actually a community with all of the wonderful and wordy parts of being a community. Yeah, I guess maybe my answer is that saying that I manage Dask is a strong statement.

[00:58:39] JM: Okay. Fair enough. Well, Matthew, thanks for coming on the show. It's been really great talking to you.

[00:58:44] MR: Yeah, my pleasure. It's been an awesome conversation.

[END OF INTERVIEW]

[00:58:55] JM: Vettery makes it easier to find a job. If you are listening to this podcast, you are probably serious about software. You are continually learning and updating your skills, which means you are staying competitive in the job market. Vettery is for people like you. Vettery is an online hiring marketplace that connects highly qualified workers with top companies. Workers

and companies on the platform are vetted, and this vetting process keeps the whole market high-quality.

Access is exclusive and you can apply to find a job through Vettery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily. Once you are accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. If you have the right skills, you have access to a better hiring process. You have access to Vettery. So check out vetter.com/sedaily and get $300 signup bonus if you accept a job through Vettery. Vettery is changing the way that people hire and the way that people get hired. Check out vetter.com/sedaily an get a $300 signup bonus if you accept a job through Vettery.

Thanks to Vettery for being a sponsor of Software Engineering Daily.

[END]