EPISODE 809

[INTRODUCTION]

[00:00:00] JM: Distributed stream processing allows developers to build applications on top of large sets of data that are being rapidly created. Stream processing is often described as an alternative to batch processing. In batch processing, a single large computation is performed over a large static dataset. In stream processing, a computation is performed repeatedly and continuously over a dataset that is being appended to.

A distributed stream is often stored in a distributed queue, such as Kafka, Kinesis, Pulsar or Google Pub/Sub. A stream is often processed with a stream processing tool such as Spark, Flink, Storm or Google Cloud Dataflow.

Holden Karau is an engineer who works on open source projects at Google. She returns to the show to describe the state of stream processing and to discuss modern best practices for stream processing.

Before ever show we often mention some goings-on in Software Engineering Daily. I'm just going to start mentioning this as recent updates so that this space in the episode gets condensed more. We have a lot of updates that are happening in our ecosystem, and you can always find these updates in a given episode. So today, our recent updates are that the new version of software daily has recently come out. This is our app and ad-free subscription service. We've built out the new version of softwaredaily.com, which is a nice UI, a much nicer UI than we had before, and a lot of kind of new little finishing touches that make Software Daily little bit nicer to use. So I hope you like that.

We are looking for help with Android engineering, QA, machine learning and more. You can find those kind of open roles on FindCollabs. Again, the link is in this episode. The FindCollabs $5,000 hackathon ends Saturday, April 15th, 2019. So there's still a week to get in your interesting – Or a little bit of time, not quite a week, to get in your interesting projects and to hack on some stuff. It's definitely not a finished race quite yet.

So, again, all those links are in this episode, and I hope you enjoy today's show with Holden.

[SPONSOR MESSAGE]

[00:02:26] JM: As a software engineer, chances are you've crossed paths with MongoDB at some point, whether you're building an app for millions of users, or just figuring out a side business. As the most popular non-relational database, MongoDB is intuitive and incredibly easy for development teams to use. Now with MongoDB Atlas, you can take advantage of MongoDB's flexible document data model as a fully automated cloud service. MongoDB Atlas handles all of the costly database operations and administration tasks that you'd rather not spend time on, like security, and high availability, and data recovery, and monitoring, and elastic scaling.

Try MongoDB Atlas today for free by going to mongodb.com/se to learn more. Go to mongodb.com/se and you can learn more about MongoDB Atlas as well as support Software Engineering Daily by checking out by the new MongoDB Atlas serverless solution for MongoDB. That's mongodb.com/se.

Thank you to MongoDB for being a sponsor.

[INTERVIEW]

[00:03:48] JM: Holden Karau, you are an open source big data engineer. You're currently at Google. Welcome back to Software Engineering Daily.

[00:03:54] HK: Thanks.

[00:03:55] JM: I want to start by just giving some historical context, because we're at Strata and I think of this as kind of a historical event. It happens every year. It's the data conference. Describe the evolution of streaming data since the world of Hadoop began.

[00:04:12] HK: So I think when Hadoop first started streaming was very much an afterthought, if it was thought of at all, right? You look at the early big data tools, and they are all purely batch-

focused. Like MapReduce has no concept of really processing streaming data, right? We started to see specialized streaming tools become introduced with things like Samza and Storm. So we got very specialized tools to handle just our streaming use case, and that's good. It gave us something to work with. But that was really frustrating, because for a lot of people what they wanted to do with streaming data, they wanted to do really similar things with their batch data as well. So they ended up having to rewrite their code to be able to compute the same thing on different pieces of data, and that's kind of a waste. So we saw the introduction of design patterns to sort of simplify that so that you could follow a common pattern and reuse much of, if not all of your code. So it was a very important step.

Then the parts that we started to see afterwards is we started to see the batch systems add streaming support, and the streaming systems add batch support, and that was just the realization that like, realistically, people don't want to write their code twice. It's not a very reasonable thing to ask people to do. It's a lot of work. When you ask people to do that, they make mistakes, and those mistakes are so difficult to find and so difficult to debug that you really want a unified system.

So Samza has batch support now. Spark has streaming support now, and we're sort of at the point where we're seeing things more commonly offer a unified system, because that's what people expect and need.

[00:05:55] JM: When I look at these different streaming frameworks, I've always had trouble distinguishing what the tradeoffs they're making are. I'm not sure what the best analogy is, but would you say a fair analogy to the range of streaming frameworks is like the range of programming languages? Because there are so many different programming languages. All of the make different tradeoffs, there's subjective difference. Is that a fair analogy?

[00:06:18] HK: I think that's not wrong. I think that's pretty accurate. In some special ways, it actually has some extra layers to it that we can think of, right? For example, if we think of the C programming language, the tradeoffs that we get with C are actually different depending on which compilers we choose. Actually, so for systems like Spark, well, we don't have different compilers that we choose. We can choose different streaming runtimes for Spark.

For systems like Apache Beam, that's also even more true. We write the same code. We get one set of general tradeoffs, and then we get to fine tune our specific tradeoffs based on which execution environment we enable. So I think programming languages is a pretty good comparison, and in some ways the cost of switching programming languages is pretty similar to the cost of switching those frameworks around it is it is really painful to do.

[00:07:09] JM: Have you done that?

[00:07:10] HK: I have.

[00:07:10] JM: Migration?

[00:07:11] HK: Oh God! I've done migrations. I will do them for money. I will do a lot of things for money. System migrations is one of those things that, yes, I will do for enough money, or a green card. One of the two.

[00:07:25] JM: To get into some more history, there was this period of time where the Lambda architecture was a big point of discussion, which as I recall, the Lambda architecture is this idea that you have a slow leg of data and a fast leg of data. Why did we have that and how did we move beyond it?

[00:07:42] HK: Well, I think in some ways we actually really haven't moved beyond it completely. We've just made it so that you don't have to consciously think about it as much. So, I mean, we got that because, realistically, people didn’t want to write their code multiple times. You have this idea where I have this common transformation that I want to apply to my fast data that's coming in, but then I've also got this slow previous historical data that I want to apply the same transformations on top of. But the characteristics of processing that data is very different. I think we have the same thing today. It's just now we are using the same systems to do both, and so we don't give it a special name anymore. Some people may disagree with me. That's a matter of personal opinion, and if you disagree with me, that's perfectly all right. I'm not strongly bought into that, but it’s where I’m at.

[00:08:32] JM: So now that you're at Google, you probably have gotten some interesting historical contexts as to how things looked within Google around the time that the open source community was sort of dealing with these problems, probably like 5 to 10 years after Google had dealt with them.

[00:08:51] HK: Totally. Actually, maybe I didn't mention this the last time we talked, but I left Google to go join the open source community.

[00:08:56] JM: Oh, that's true! That's true. You had been at Google before that.

[00:08:59] HK: Yeah. I love to go solve the same problems again, and now I'm back to Google solving the same problems again in open source. So Google solutions are very great. There's nothing particularly wrong with them, but I think that we've learned a lot in the meantime, even though we are perhaps solving problems that Google has solved before internally in open source. The structure of Google's data center – The characteristics of that is different than that of a standard commodity hardware that you're going to get.

So some of the solutions and techniques that Google uses internally, it doesn't make sense to just port those into open source. We actually need to sit down and reevaluate the design choices as we're going forward. I mean, this shows an Apache Beam, right? It's taken a lot of time to – Even though it's from Google, catch up to some of the same features that are in Google's own internal systems, because it's not just a matter of copying it over and tweaking a few things. It's a matter of fundamentally having a different core architecture that we're building on top of.

[00:10:04] JM: It's interesting seeing the contrast in the Google open source communities, like Apache Beam, Tensorflow, Kubernetes. There's a really different community developments across those different projects. What do you think drives that?

[00:10:20] HK: Totally. So one of the things is there're different foundations behind those projects, and foundations, to a degree, they provide sort of a general framework that you work within, right? So Apache Beam is part of the Apache Software Foundation, which is actually just celebrating its 20th anniversary this year.

The Apache Software Foundation is very flexible, but it still has some general guiding principles, and that sort of shaped that Apache Beam community in some important ways. I think Kubernetes definitely has CNCF, and that's very different in terms of their history and where the CNCF comes from, and their view is the right way to do open source. I don't think anyone's right or wrong. I think it's all just different approaches of getting to the same place. It's about finding ways to collaborate.

I think one of the challenges for a company like Google is when you take things that have been developed for a long time internally and then you turn them into open source projects. One of the big challenges is getting the engineers who've been working on the problem before to sort of change their habits, because they're good engineers. But now they need to remember to work with the community so that the community can become involved, and that's not necessarily a thing that they have as much experience doing.

So there's sort of a transition period as these projects move from being internal projects to open source community projects that people are a part of. I think the level of support provided by the foundations there is really important.

[00:11:51] JM: What's the significance of the Google Dataflow Paper?

[00:11:54] HK: It has a lot of significance. It's changed, I think, how people architect systems. I think it certainly spurred a lot of innovation in open source. Of course, I'm biased. I work at Google. So I think that the things that we've done are very good and useful. I think it's a very common pattern where you see Google releasing very high quality papers describing our systems and then seeing the open source world re-implement them.

I think one of the things which we've done differently this time is releasing Apache Beam not at the same time, but in the same sort of general, larger time window, so that people can see some of our ideas of how we think implementations like this can be built. I think doing that together, like not at the same time, but broadly together with the release of the paper has certainly helped, did have a larger impact than it could have otherwise.

[00:12:47] JM: When the dataflow paper came out, you were working on Spark, right?

[00:12:51] HK: I was. Yes.

[00:12:52] JM: You're focused on Spark. What takeaways from the paper did you have around that time? Was there anything in it that was like mind-blowing to you that changed your perspective on Spark?

[00:13:03] HK: So not really. So the problem for me is I just left Google. So it was like, "Cool! Here's a paper about the things that I already knew about," and that was really good, because it meant I could talk with people about those things and they weren't secret anymore. I think that was useful.

I think, realistically though, a lot of my focus at that time was on trying to come up with better support for multi-language pipelines. So the dataflow paper really didn't have a lot of impact on that. I think the people who probably felt the most impact of seeing the dataflow people were [inaudible 00:13:40] and some of the Flink team, who are more for more focused on building a unified batch streaming system at that time than I was.

[SPONSOR MESSAGE]

[00:13:59] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a

bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:16:07] JM: Give an example of common streaming data problem that you see in typical organizations.

[00:16:14] HK: Yeah. I mean, I think probably the most common streaming data problem that I see is just ETL, but ETL is boring. So let's talk about streaming machine learning predictions, because that's more fun and cooler. Actually, we can talk about streaming machine learning training, because that's even cooler. I don't see it as often as the streaming predictions. So, in practice, ETL predictions, and then training at the bottom in terms of frequency, but I think it's a really cool thing and it's going to be really challenging for us to do right.

I've talked with people who have different approaches to how they want to do online updating of their models. You need a parameter server so you can have your parameter somewhere, but one of the challenges is with like traditional machine learning, we have this idea that like I train a new model and then I A-B test it. I compare it. I'm like, "Yeah, okay. My new model is good. It's better."

But if I'm doing online streaming updates, it's a lot harder to pick the time when I want to cut over to my new model and it's also harder if I'm always automatically using my latest model. I can have drift happen much more rapidly than I'm used to, and I can get into a really bad state really quickly. So I see a lot of people exploring different solutions to that. Some people who are

training more classical linear regression type algorithms have sort of bounds which their parameters are allowed to play in from the last batch train. So they only allow so much change to happen during streaming with the idea that like, “Yeah, today is a little different than yesterday, and we should take that into account. But today is probably not drastically different than yesterday. If it is, we probably want to get a human operator involved.” So I think it's a really interesting problem that people are working on right now.

[00:17:59] JM: Can you talk about it more architecturally? Like if I'm implementing a system for streaming machine learning training, online learning, what are the different components that are going to be going into this thing? What's the – Perhaps data lake, the queuing system, the machine learning framework, streaming systems? What do I need?

[00:18:19] HK: Totally. So you’re going to need some streaming data. A lot of times it's coming in on Kafka. Kafka is very solid at this point, and so it is a very good –

[00:18:28] JM: Production data is being written to a Kafka queue. I'm reading from it via a stream.

[00:18:31] HK: Yup. I’m picking out via stream. I’m picking it up in Spark, or Flink, or Beam, my streaming platform of choice. I’m applying some kind of data cleaning on top of it, because all data is garbage, and I'm throwing away some of it, because some data is really truly – Just very bad garbage. So I'm trying to get rid of the truly egregious ones and I try and do that as quickly as I can.

Then I get into the place where I'm now going to try and actually train my machine learning model. So I have often a distinct process, not always. In some places, I see it where the driver program for your traditional distributed computing thing also serves as the parameter server. So I have some idea, I've some collection of weights or parameters that I'm going to be fitting for my model.

So I have my previous good weights. Those are probably from a batch train on my data lake when I'm starting my stream. Then I'm going to update those weights as the new data is coming

in. Each of the workers is going to compute some loss function and I'm going to use the results of that to update the weights.

Often, you see the parameter server distinct from the driver, because if you put it on the driver, that is a lot of coordination work to be happening on the driver, which is already sort of a natural choke point. So you really – I think putting it there is often the thing which people do for like a V1 to get it out the door, but then having to refactor it into a separate parameter server that's able to get the updates separately and distinctly from the regular driver.

Then your parameter server might do a few different things. It might just periodically write the new model out into a GCS, or S3, or HDFS bucket, or it may actually produce another stream. You may produce a Kafka stream of your new models that are then picked up downstream by serving components to do actual predications.

I've seen both architectures, and I think they're both fine. I think the Kafka stream one is definitely feels more like a streaming system, and that's something that people care about. Personally, I tend to be of the opinion that if you're not okay waiting for one minute for a new model, you probably need to hire some very specialized engineers anyways, and these off-the-shelf components probably aren't for you. But it's a personal opinion there.

Yes. So you'll have some set of new models, and then they're going to get picked up by whatever you're using to do your predictions. Sometimes you're going to do your predictions also on another Kafka stream. That's a very nice, easy situation, and I think that's one of the situations where we see the models coming out to the Kafka stream as really making sense, because we see the predictors are already Kafka clients anyways.

In other situations, we have things where we don't want the latency of putting Kafka in the middle. We're serving request on the hot path to a client. So then we have traditional API servers with the rest or other interface which are then being hit to serve my model. I think in those cases you more commonly see people use traditional distributed file system or ObjectStore to put their updated model into, because you don't want to have every one of those be a Kafka client necessarily.

[00:21:50] JM: These days, is it any harder to develop streaming machine learning applications than batch-based –

[00:21:57] HK: Oh, God! Yes. Yes. Yes. Don't worry. Everything is still terrible, but we've solved some of the problems, but most of them are still there. So there're a lot of really lovely tools to make your batch machine learning easier, and there are not really the same tools for online. Data validation is really hard to do when you've got streaming ingest data, because a lot of the data validation tools are about what the distribution of your data looks like. Computing those distributions in small windows leads to false positives and negatives at an unacceptably high rate. So you have a lot of challenges around how to do data validation, which is the key part of machine learning in my opinion. If you don't have a good data validation, you might as well just return a random number. So I think that's there.

Also, realistically, the scope of algorithms which are supported for online learning are smaller than the algorithms that are supported for batch learning often. That's okay, right? But that's the thing which may or may not limit your use case. You may not find the right models that you want out-of-the-box support online learning, and having to write that yourself is going to be a lot of work.

At the very least, you'll have like a guide, because other models support online learning. But adding support for online learning to an existing batch algorithm is not trivial. That is a substantial engineering and possibly research task.

[00:23:21] JM: The point about the algorithms that you made, I guess that's because a lot of these algorithms that are like implemented in – What? The Python libraries or whatever? It's like they're built to process all the examples. They're not built to process all the examples, and then one more example.

[00:23:37] HK: Yes. They do not have the idea of and just one more, as fun as that is. So it's difficult, because they use very good optimization algorithms, but these same optimization algorithms do not work as well, and you only have partial views of the data time. So you have to pick different approaches to optimizing your data. You can't do the same [inaudible 00:23:55] of algorithms that you're used to doing.

[00:23:58] JM: What is Apache Beam? I know you've answered that question a number of times. I've asked that question a number of times. I'm still a little confused, but please tell me.

[00:24:06] HK: So there are a lot of different ways of thinking about Apache Beam. One of the ways that you can think about it is a translation layer. That's a limited way of thinking about it. Another way is to think of it as more like a compiler, except it doesn't really quite work like a compiler, because it doesn't produce a new thing, rather ships a runtime and a bunch of other things.

So thinking of it as somewhere between a new interpreter and a compiler, and a translation layer, and thinking of it as a weird hybrid of all of those things. Somehow working on data is the way that I would say is the fast approach to thinking about it.

It's challenging though, because Apache Beam is also changing a lot. There're some very interesting new features around multi-language pipelines in Apache Beam, which is pretty cool. Traditionally, Apache Beam really had only first-class support for Java and JVM languages, and then added second-class support for Python is often the way. But now it has a more general approach, which allows for things like Go to be written as Beam pipelines, and that's pretty cool. This is actually one of the situations where we see the open source world got there first, and then the Google Apache Beam, which is also open source, but I guess really what I meant say, the world outside of Google about their first. Then Google is able to learn from some of those experiences and come back with a different solution for handling non-JVM languages for big data.

[00:25:39] JM: So going back to the streaming machine learning application, how would we potentially use Apache Beam in the application? Where would it fit in?

[00:25:47] HK: Totally. So we could use Apache Beam in a few different ways. One of the things is there's like a suite of Tensorflow tools that run on top of Apache Beam, and we could try and use some of those tools with some streaming data. A lot of those tools are designed more to work with batch data. So we'd have to do a little bit of refactoring probably to succeed.

We'd use Apache Beam most likely to do our ETL, do our data cleaning, and then get our data over into something like a distributed Tensorflow job to update for these new records. That's not perfect yet, but it's getting a lot better. That's where Beam would fit in to the picture.

[00:26:28] JM: So Beam actually is a streaming framework. It can be used like a streaming framework, or if you are running a Beam job, is it necessarily being turned into like a Spark job, or a –

[00:26:41] HK: Beam itself does not have its own distributed data processing runner. So if you are running a Beam job, by its nature, it's getting transformed into a Samza job, or a Flink job, or a Spark job, or a data flow job. In some of those backends, you can use your Beam job as a streaming job. So I could definitely use Apache Beam for that streaming data.

In some of those backends, it doesn't currently support streaming. So Apache Beam on Spark is not a thing that you would do for streaming data. That would not be a great success. That may change by the time this podcast gets published. Who knows?

[00:27:19] JM: So why is that?

[00:27:21] HK: Well, that's a good question, and the true answer to that probably involves a bottle of scotch to get the answer out of me. But the short version is essentially that support for the different backhands are developed independently. So there hasn't been as much motivation to work on the Beam Spark runner as there has been to work on, for example, the Beam Flink runner. So the variety of features available on the Beam Spark runner are not as broad.

[00:27:52] JM: But let's say like we built our streaming machine learning system. We used Flink. Is there reason to reimplement it in Beam?

[00:28:02] HK: In my opinion, no. I'm sure if you ask someone who works on the Beam project more actively than I do, they may give you a different answer. But if you already have code that works on Flink, there's no particular reason to switch it to another system. I think – The same is true, like if I build streaming machine learning system on Spark, if it's working for me – Oh dear

God! Don't just rewrite it because it's new. That's such a waste of time. Working code is worth its weight in Macbooks.

[00:28:29] JM: Yeah. But if I reimplemented it from Flink to Apache Beam, what I might get is like forward compatibility, right? If somebody came out with a new streaming framework and it solves my problem better, I can magically have the Apache Beam code that I have now be automatically swapped over to that other runtime.

[00:28:49] HK: Yeah, that is definitely a possibility. I am suspicious of the possibility, because the new magical streaming framework will probably not have Beam support for a while, right? We don't see Beam as something that is a must-have feature for new processing frameworks in the data space. It's often added later on. But, certainly, it could make my code have a longer shelf life, and that can certainly be important especially if you're the kind of person who gets stuck maintaining legacy systems could be a way out of having to maintain quite as many legacy systems.

[00:29:28] JM: Is the motivation for Beam to provide people a way to run their streaming jobs on Google Dataflow without being locked into the APIs of Google Dataflow?

[00:29:40] HK: That's certainly a large part of the motivation, right? Google Dataflow is an amazing product, but realistically, a lot of people do not want to be locked in. So offering the ability to move to Apache Flink if Dataflow becomes too expensive or not for you is certainly a very important motivation of the Apache Beam development work I think. Certainly not the only reason, but I think that's a nontrivial portion of why that development occurs.

[00:30:09] JM: What would be some other reasons?

[00:30:13] HK: Well, so I think if we think back to your question about that machine learning pipeline that you said we wrote in Flink and then if we want it to move it to Beam, I don't think that's really the use case that Beam is going after. But if I'm writing something and I'm not sure where I wanted to land. I think using Beam from the start makes sense, right? I think doing rewrites to Beam is probably overkill, just as I think doing rewrites to Spark is probably overkill.

But I think if I start out in the Beam land, I get this flexibility at not that high a cost to me, right? It's not the cost of having to rewrite my things. I think Beam just provides that level of flexibility that people –

[00:30:51] JM: So you focused on Spark even since joining Google. How has the Spark ecosystem evolved in last year?

[00:30:58] HK: Yeah. I mean, Spark had the 2.4 release. It's added support for Kubernetes, and it's been really key to driving Spark I think. We're starting to work on Spark 3, which is really exciting. We're starting to get rid of our old deprecated APIs. Unfortunately, it looks like I will not succeed in my quest to get rid of Python 2 support. But I think soon we will.

We've seen a multitude of streaming options in Spark, and this is not that it's going to use like a non-Spark streaming things. It's just Spark streaming now supports many different kinds of execution. So, for example, if I don't care about my data all that much, but I really, really care about getting most of my data processed really, really quickly, that's now one of the tradeoffs that I can select in Spark streaming, and there's other option sort of depending on where you want to sit in the streaming reliability performance tradeoffs, and that's pretty cool that it's now inside of one system as supposed to necessarily having to change between systems to get a different set of streaming tradeoffs.

[00:32:07] JM: What are some patterns for how Spark fits into a machine learning developer's workflow?

[00:32:13] HK: So there's the classic one where I ETL my data in Spark. I write it out and then I pick it up in a machine learning tool. It's not a bad one, to be honest. One of the things which has happened in the new versions of Spark is this thing called gang scheduler, and this is designed to allow Spark to be more cooperative with traditional machine learning tools like Tensorflow, so that what you can do is you can do your data preparation in Spark and then actually fire up Tensorflow or whatever deep learning or machine learning tool you want to use and then have that pick up the data directly from Spark, be scheduled and control by Spark and produce the result and then go back to Spark's traditional scheduler afterwards.

I think that one – That's not a common pattern that people use today, but I think that's going to be a more common pattern as the bugs in that new scheduler get ironed out. New software always takes a little bit of time to catch on and for good reasons, and I think that's a pretty common pattern. There's another one where Spark itself has its own machine learning libraries inside of it. We do see people use those. I think that's going to happen less. I think we see Spark moving in the direction of allowing you to plug-in the specific machine learning tools that you want and just giving you a way to get your data ready to use with those machine learning tools and giving you a way to use those machine learning tools in a distributed nature. I think that's going to be sort of the future usage patterns that we see around Spark and machine learning.

[00:33:41] JM: So do you mostly think of Spark as this way to materialize large datasets into memory and then just perform operations on those in-memory datasets?

[00:33:55] HK: I think from machine learning, that's not a bad way of thinking about it. For ETL, it's a little different. But I think from machine learning, realistically, that kind of Spark's sweet spot, right? It's really good at getting the data together. But, realistically, there are not like a large group of machine learning engineers working on Spark's core machine learning. Instead, what we see is we see the machine learning engineers working on core abstractions in Spark. So then different people can implement this abstractions, and we see this with people from all sorts of different companies working on those abstractions. I'm not going to mention them by name, because I will forget someone and they'll get angry with me. But that's a really important thing, because then it means that you're not like locked in to just one set of machine learning tools.

[00:34:44] JM: What are some common mistakes that people make writing Spark applications?

[00:34:47] HK: Oh! So many. So many. There's the classic where people call group by key, because it sounds like it's going to group your data to get the right key. The problem is it does that, and then your job fails, because your data, when grouped by key, it turns out that a lot of humans live in New York and all the computers live in this place called [inaudible 00:35:06], and then your data just gets very weirdly shaped and your job fails.

I think, more broadly though, these problems come from people not understanding the shape of their data, and that's completely reasonable. We don't normally have to think about the shape of our data as much when we're writing local non-distributed applications. But I think, really broadly, the problem facing most Spark developers is not having a good grasp of how their data looks, what the distribution of their keys and values look like.

[00:35:39] JM: How does Spark relate to Tensorflow?

[00:35:42] HK: That's a good question. So you can use Spark to get data ready to use with Tensorflow. You can use Tensorflow on data from Spark with Tensorflow on Spark, which is by the Yahoo folks I believe, and all sorts of different libraries by different people. That's probably the closest way that I think Spark relates with Tensorflow.

[SPONSOR MESSAGE]

[00:36:08] JM: This episode of Software Engineering Daily is sponsored by Datadog. Datadog integrates seamlessly with container technologies like Docker and Kubernetes so you can monitor your entire container cluster in real-time. See across all of your servers, containers, apps and services in one place with powerful visualizations, sophisticated alerting, distributed tracing and APM. Now, Datadog has application performance monitoring for Java.

Start monitoring your microservices today with a free trial, and as a bonus, Datadog will send you a free t-shirt. You can get both of those things by going to softwareengineeringdaily.com/datadog. That's softwareengineeringdaily.com/datadog.

Thank you, Datadog.

[INTERVIEW CONTINUED]

[00:37:03] JM: How does the MLlib set of tools within Spark compare to what kinds of machine learning you'd be doing with Tensorflow?

[00:37:13] HK: They've very different. Spark's machine learning tools are very much classical machine learning tools. There is of course neural networks and things like this, but they're not really flashed at to the same degree as Tensorflow, and that is what it is.

I think Spark's machine learning realistically just isn't developed as aggressively as Tensorflow is. There are a lot of people working on Tensorflow versus the number of people working on Spark machine learning. So I don't think we're going to see Spark ML becoming competitive there. I think we'll see – As I was saying, common APIs or systems like that become the way that people from Spark access machine learning tools like Tensorflow.

[00:37:56] JM: What areas of the Spark open source project have you contributed to?

[00:38:00] HK: Sure. That's a great question. I think I have contributed to technically every area except for graphing, I want to say. I've definitely contributed to core machine learning Python. I have contributed to R very, very tiny, very tiny. I don't actually know R. So it took me a long time. Let's see here –

[00:38:23] JM: [inaudible 00:38:23] the project.

[00:38:24] HK: I have. It's a fun project. I mean, I've worked on it for longer than I thought it was ever going to work on any tool.

[00:38:31] JM: Why is that?

[00:38:32] HK: I like open source, and Spark is open source that people are willing to pay me to work on. So that's worked out pretty well for me, I think.

[00:38:40] JM: But you could work on Kafka. You could go to –

[00:38:44] HK: I could, actually. Occasionally, I talk to people about rules working on things like Kafka. That would be cool too. Honestly, actually, I think one of my goals for the next five years is to make sure that I don't just become the Spark lady, because I think if I keep going down the path I am, I'm going to just end up this Spark lady.

[00:39:02] JM: Not a bad place to be.

[00:39:04] HK: Oh, no! Certainly not. I think that's a perfectly reasonable thing to be. But I think knowing myself, I'd get bored doing that. So I want to preempt my boredom and find a solution.

[00:39:15] JM: Okay. Well, on that note, we've been focusing on streaming, but the entirety of the data platform world is expansive, and you've got your data warehouse, your data queue, your data lake, all these different areas, multitude of databases. How does the streaming platform relate to the multitude of other aspects of the data platform?

[00:39:41] HK: Totally.

[00:39:44] JM: That's kind of a vague question. Explore on how you will

[00:39:47] HK: So streaming tends to relate to these things as I start if we look an organization which is just adding streaming support. It normally comes in the form of ingestion, right? That's where streaming often first makes its presence felt, is getting my data ingested, because I have real-time, or like real-time data being produced that I want to ingest into something like HDFS or my traditional data lake.

Then from there once we start to see that, we start to get people who ask for near real-time results, and that's the next place where it sort of starts to play a role, is empowering people to get answers instead of having to wait long periods of time for the data to hit disk and then be re-exported in the correct formats and so on.

So I think, realistically, streaming represents both ingestion into traditional ETL jobs, but also it is answering business questions that we just didn't have before and allowing us to take action faster, right? Previously, if I did my fraud analytics in a MapReduce job, I might not notice fraud before I already shipped out the packages. But with streaming, I can take these same things and answer those questions sooner. I still use my traditional ETL and MapReduce or Spark jobs to train my models, and I think online learning is definitely a thing where I will see people using

updated models in closer to real-time, but I still think we're going to see people training traditional models in batch and then iteratively updating them.

So I think streaming very much builds on top of the existing ecosystem that's there. It doesn't replace it or supplant it. There are certainly people who just let their data live in Kafka and they don't persist it somewhere else. But that's definitely not a thing that you see very often, and that only really makes sense in corner case use cases in my opinion. Now, of course, the Kafka folks may give you a different answer than that, and that's fine too.

[00:41:55] JM: You think they envision Kafka as a data lake?

[00:41:57] HK: Certainly some of the talks that I've seen from some of the Kafka people definitely seem like that's where they think Kafka will sit eventually, is that it will represent both the fresh and the aggregate data.

I mean, sure, it's possible, I think, realistically though, there's a lot of tooling that exists that's going to be really hard for us to rewrite to get that same sort of support on top of just Kafka data, and I mean that might happen, but I don't see the motivation to it. I don't see what it gets us. I mean, people are willing to do small projects for fun, but I think, for example, rewriting Impala to work on top of Kafka, that sounds like a lot of work for – I don't know what gain, and I don't really see something like that happening.

Of course, knowing my luck, there's a good chance that someone's done that. But it's a question of what do we gain by having all of our data live in Kafka. I mean, certainly less things to manage, but sometimes a unified system is not the best. Sometimes there are unique things about different kinds of data that we want to keep distinct and separate.

[00:43:15] JM: It could be entirely plausible that this stuff ends up composing together to feel more like an operating system. I feel today, it's almost like from talking to people, operating your "data platform" is like operating the different apps on your smartphone. You're like, "Go to the queue. Now I go to the HDFS. Now I go to the streaming framework." But when you are operating with like an Apple laptop, you don't know where your L1 cache is. You don't know where your L2 cache is. These are not apps on your desktop.

[00:43:46] HK: It's true. Yeah. Certainly, I think that we will see these become more composable in the UNIX style philosophy as time goes on.

[00:43:53] JM: Composable and like abstract it away?

[00:43:55] HK: Yeah. Which, honestly, as a Spark developer, is like a thing which is a double-edged sword. There may come a day when people don't think about Spark. They just think about the higher-level things that are built on top of it. By that point, I should find myself another job.

[00:44:11] JM: On that note, you have these companies, many of them are at Strata, that are offering a highly integrated “data platform”. Why is there not like an open source slightly opinionated data platform, that if I'm a big insurance company I can just like get this bucket of open source technologies. Here's my data lake. Here’s my streaming system. Here’s my data warehouse. I get all these and it's like a data platform that collects all of them, but it's all open source instead of like a repurposing for proprietary purposes.

[00:44:51] HK: I think there's a few different reasons, and we do see some pure open source things like Ambari. I think it’s Ambari. I might be wrong about the name. We do see some open source things like that that give us that, but they're not represented here because they don't have the same commercial backing and the same money. Well, realistically, I think a lot of those platforms are sold with the promise of support. What you're buying is the promise that there is someone who understands the different components of that platform at that vendor who, when things go wrong, yeah, it might take you a little while, but you can get an answer.

A thing with the open source tools is like, yeah, there's no promise that anyone’s going to ever answer you. That’s just what it is. Also, it's just not nearly as cool work to do, right? I think a lot of open source work falls into two buckets. It’s either cool and people are doing it for fun, or it's being done to serve a larger commercial purpose and then people are being paid to do it. So I think you see a lot of people working on the individual tools from the different vendors, but their special value add is on top of that, and so you're not going to see people paid to cannibalize the vendor's profit centers in open source.

So it's uncool work that no one really wants to pay for right now I think is the problem. This may change. It may make sense at some point for someone who wishes to disrupt the market or whatever phrase we're using these days to make really good open source platform that has everything together, and then vendors will have to find a new way to sell support contracts and call it software revenue.

[00:46:36] JM: Yeah.

[00:46:38] HK: Why are Jupyter notebooks becoming so popular?

[00:46:41] HK: It's a lovely question. I think notebooks are really amazing way of doing data exploration, and I think a large part of this comes from things like data science boot camps the that have really changed the tools that people are learning, and notebooks give you an accessible interface to do your data science in. I think, very importantly, they're not scary. A lot of the other tools that we see can feel intimidating if it's your first time using them or you don't come from a traditional software engineering background. But Jupyter Notebooks do an excellent job of being accessible to a wide range of people.

Then there's this thing where when you do exploratory work in something, the idea of redoing it somewhere else to put into production, it's a bit of a tough sell. So we're seeing Jupyter Notebooks move up the stack in that same way.

[00:47:36] JM: To wrap up, what are the other themes you're seeing here at Strata?

[00:47:40] HK: It's a good question. To be honest, my Strata has been a bit hectic. So I haven't had as much chance to hang out in the hallway as I hoped it would. But I think Apache Arrow is a really big thing that I hear people talking about a lot, and that's a common format that allows different programming languages and tools in different languages to cooperate together in a more efficient manner, and I think that's probably one of the bigger trends that I see happening here at Strata.

Realistically, I spent most of the rest of my time here trying to get my Kubeflow workshop working. So I heard a lot about Kubeflow, but I was working on a Kubeflow workshop. So I probably should've heard a lot about Kubeflow. I always hear a lot about Spark, but I take that with a grain of salt, because if people like Spark, they come and talk to me sort of regardless of where I am in the world.

[00:48:28] JM: What's the significance of the Kubeflow project?

[00:48:31] HK: So Kubeflow is really cool. Actually, I should have answered that for your data platform question.

[00:48:35] JM: I've done a show on Kubeflow, but I want your perspective on it.

[00:48:38] HK: Sure. So I think Kubeflow is really interesting. It gives you a collection of tools that you can use together in an open source way on top of Kubernetes.

[00:48:47] JM: Oh! I guess that is the open source data platform, perhaps.

[00:48:49] HK: It could be. It's not there yet.

[00:48:51] JM: Not there yet.

[00:48:51] HK: Right now, for example, Kubeflow doesn't have really any good answer to where to store your data. That's just like pick something else and connect it to Kubeflow. So Kubeflow has the potential to a development of something like that. Time will tell if it does or not. But Kubeflow is really good for doing machine learning on Kubernetes and making sure that the pipelines that you build locally on your laptop while you're doing your exploration will be able to scale and work in production, and it also makes it a lot easier to collaborate with folks, because you're not in the situation of having to manually manage dependencies. Things are explicitly stated. You're not having a go track down your good friend Joe who wrote something with some random version that you can't find and he didn't bother to mention in a requirements.txt. So it's Joe's fault always. It helps that you don't work with anyone named Joe, I think. I don't know. If I do, I'm sorry, Joe. But it's your fault.

[00:49:48] JM: Holden Karau, thanks for coming on the show.

[00:49:50] HK: Thanks for having me, yeah.

[END OF INTERVIEW]

[00:49:54] JM: This podcast is brought to you by wix.com. Build your website quickly with Wix. Wix code unites design features with advanced code capabilities, so you can build data-driven websites and professional web apps very quickly. You can store and manage unlimited data, you can create hundreds of dynamic pages, you can add repeating layouts, make custom forms, call external APIs and take full control of your sites functionality using Wix Code APIs and your own JavaScript. You don't need HTML or CSS.

With Wix codes, built-in database and IDE, you've got one click deployment that instantly updates all the content on your site and everything is SEO friendly. What about security and hosting and maintenance? Wix has you covered, so you can spend more time focusing on yourself and your clients.

If you're not a developer, it's not a problem. There's plenty that you can do without writing a lot of code, although of course if you are a developer, then you can do much more. You can explore all the resources on the Wix Code's site to learn more about web development wherever you are in your developer career. You can discover video tutorials, articles, code snippets, API references and a lively forum where you can get advanced tips from Wix Code experts.

Check it out for yourself at wicks.com/sed. That's wix.com/sed. You can get 10% off your premium plan while developing a website quickly for the web. To get that 10% off the premium plan and support Software Engineering Daily, go to wix.com/sed and see what you can do with Wix Code today.

[END]