EPISODE 823

[INTRODUCTION]

[00:00:00] JM: Lyft generates petabytes of data. Driver and rider behavior, pricing information, the movement of cars through the space, all of these data is received by Lyft's backend services buffered into Kafka queues and processed by various stream processing systems. Lyft moves the high-volumes of data into a data lake for different users throughout the company to use offline. Machine learning jobs, batch jobs, streaming jobs and materialized databases can all be created on top of that data lake. Druid and Superset are used for operational analytics and dashboarding.

Li Gao is a data engineer at Lyft. He joins the show to explore the different aspects of Lyft's data platform with particular emphasis on Spark. We also talk about the tradeoffs of streaming frameworks and how to manage machine learning infrastructure. This episode is a great companion to our show about Uber's data platform, which was earlier this year, or last year. It illustrates some fundamental differences in how the two ridesharing companies operate.

The FindCollabs Podcast is out. FindCollabs is a company I started recently, and FindCollabs Podcast is a podcast about the goings on in that community. We are also hiring a React developer for FindCollabs, and the first FindCollabs hackathon has ended. We're booking sponsorships for Q3, if you are looking to reach developers. You can go to softwareengineeringdaily.com/sponsor, and we've got a few other updates in the show notes. You can find the ones I mentioned as well as other information.

Let's get on with today's show.

[SPONSOR MESSAGE]

[00:01:53] JM: When I talk to web developers about building and deploying websites, I keep hearing excitement about Netlify. Netlify is a modern way to build and manage fast, modern websites that run without the need for addressable web servers. Netlify is serverless. Netlify lets you deploy sites directly from git to a worldwide application delivery network for the fastest possible performance.

Netlify's built-in continuous deployment automatically builds and deploys your site, or your application, whenever you push to your git repository. You can even attach deploy previews to your poll requests and turn each branch into its own staging site. Use modern frontend tools and site generators, like React, and Gatsby, or Vue, and Nuxt.

For the backend, Netlify can automatically deploy AWS Lambda functions right alongside the rest of your code. Simply set up a folder and drop in your functions. Everything else is automatic, and there's so much more. There's automatic forms, identity management and tools to manage and transform large images and media.

Go to Netlify.com/sedaily to learn more about Netlify and support Software Engineering Daily. It's a great way to deploy your newest application, or an old application. So go to netlify.com/sedaily and see what the Netlify team is building. Also, you can check out our episode that we did with the Netlify CEO and founder; Matt Billman. That was a really enjoyable episode. I'm happy to have Netlify as a supporter of Software Engineering Daily.

[INTERVIEW]

[00:03:47] JM: Li Gao, welcome to Software Engineering Daily. Thank you for coming on.

[00:03:51] LG: Thank you for having me.

[00:03:53] JM: So today we're talking about Lyft and the data platform of Lyft. Before we get into the technical matters, let's talk about the application requirements. What are the requirements of Lyft's data platform?

[00:04:06] LG: In the beginning, the data platform is started, it's like a data warehousing using Red Shift two, three years ago. Actually, three years ago. In the time, most of the data, highly de-normalized datasets [inaudible 00:04:20] to serve our BI dashboard mostly to [inaudible 00:04:25] weekly and – Most likely it's weekly and there are certain daily reports at our executive and the marketing folks needs to make decision based on those BI dashboards. That's the beginning of this data platform [inaudible 00:04:38]. Second half of 2016, Lyft started building this data platform team and we started building a few pieces. So you can imagine, we start outgrow Red Shift a longtime ago and we started introducing Hive around 16 to 17, around that timeline. Also at 17 introduced Presto to speed up some of these ad hoc BI dashboards, at the same, introduced certain [inaudible 00:05:06] BI tools to enable more use cases not only for the executive dashboards, but also for our internal analyst, the business analyst, and that's a very new data science team at the time in 2017.

In around 2018, we started expanding even more. We introduced faster version of Hive at the time to speed up some of those ETL workloads [inaudible 00:05:33] Hive jobs grow [inaudible 00:05:34]. That's when we started looking at Spark for one of this performance [inaudible 00:05:41]. At the same time, we have explosion of many data science machine learning use cases that can really benefit from Spark. So that's when we start converging on a Spark-based solution for many of our batch use cases, not just for ETL, the dashboard serving, but also for model chain. Spark mostly use for our – At 2018, mostly used for model training and certain batch-based model serving as well. That's where we've been to today's world, where we have batch processing running mostly on Hive and Spark and interactive system running on Presto [inaudible 00:06:24] interactive datasets.

We do also have very fast metrics data storage using Druid, and we use different – We also enriched our user interface tools using Superset mostly to start many of our internal dashboards and metrics virtualization as well. We do have our internal notebook services based on Jupyter to serve our machine learning requirements and their model serving as well. That's the current landscape of our data platform at Lyft.

[00:06:58] JM: I'm excited to get into all those matters, particularly into the Apache Spark area. But going back a little bit historically, you mentioned that you outgrew Red Shift at a certain time. So Red Shift I believe is the data warehousing solution on Amazon Web Services. In what ways did you outgrow Red Shift? Why did Red Shift stopped being a performant enough data warehousing solution for your different applications.

[00:07:25] LG: There are two factors. Red Shift – As a way we use Red Shift, is Red Shift is highly coupled between compute and storage. In our case, we have accumulating. Actually, we accelerated accumulation of historical facts. The datasets is very big. Right now, I cannot say the exact number, but in an order of magnitude, tens of petabytes, less than 100 petabytes of those data in our warehouse in any format, sometimes in raw formats, sometimes in normalized, sometimes in de-normalized format. All those datasets is very expensive to feed into the Red Shift model, because Red Shift, there's a tight coupling between compute and storage.

I know in the older version, Red Shift, they changed that. But in our case, the way we use Red Shift [inaudible 00:08:15] coupled there. That's one thing. Second thing is loading time, the ingest and the query, those two are tied together [inaudible 00:08:25] of our real-time dashboard, Red Shift is too slow for the dashboard to serve our business purposes. That's where we start split. We build a setup data platform. We have a faster ad hoc query engine to serve those [inaudible 00:08:40] time metrics and data to our dashboard, and we have a store data ingestion pipeline to transform the data to Hive and Spark. On top of that, like I said earlier, we also have this machine learning models that's not only depending on [inaudible 00:08:57], but also depending on a whole bunch of Python libraries to enrich data.

[00:09:01] JM: You've worked in several different environments before Lyft. How is Lyft different than the data environments and the data requirements at companies that you worked before?

[00:09:12] LG: Yeah, excellent question. Here, what I see is organic growth of the data and the use case is – I will say is the highest growth I've seen in any of my previous companies in the past. Most of the decisions, what tools to bring, is based on the growth. Not only growth of the dataset, but growth of the pipeline, the complexity of the requirements and the different use cases we need to support.

So it's more like a bottom up organic growth to support different platforms, and everyone in the data platform team try to strive to provide a best tool for the best use cases to enhance and enrich our user experience.

[00:09:55] JM: As we get into a conversation about the modern data platform, we think about the different applications that internal users want to build on that data platform. Let's talk about the different tools that you use to serve those applications. I'd like to just go through a quick run through of these different tools. We've done different shows about each of these different tools. So any listener who is totally unfamiliar with one of these tools can go back and listen to one of these episodes. But I'd like to just run through some of these tools, and for each tool, you can just describe what it is useful for at Lyft. Why it's important? How it fits into the data infrastructure?

So let's start with Presto. How does Apache Presto fit into what you do?

[00:10:40] LG: First of all, Presto is not part of the Apache Foundation yet. So, yeah, we call it PrestoDB and I think recently forked as PrestoCQ. We use PrestoCQ mostly for our ad hoc queries. Right now, primarily data to back the Presto interactive query engine is our Hive datasets, which live on S3. We'll talk about more later on this.

We do have a few other data sources that's back in the Presto query engine. We have certain relational databases, mostly PostgreS and MySQL, and we also have the [inaudible 00:11:17] we are building the connection between Druid and Presto to surface a single query engine for all the different enhancement, or we call the joins, that Presto support to enhance our datasets and visualize that in Superset.

[00:11:32] JM: Presto, just to go a little bit deeper on that. That is a query engine. What is it useful for? If I have these different databases, different databases have their own query engines. Why do I need Prestos?

[00:11:47] LG: Excellent question. So Presto – You can think of Presto as like a federated query engine that [inaudible 00:11:54] different backends. So Presto itself does not have a data storage per se. It has some metadata somewhere, but put that aside. The Presto makeup query, you can join data – You can imagine, when there's a case you want to access the data from where it lives. So you can imagine, you have the majority of your raw datasets on a data lake or data warehouse. In our case, our S3 or Red Shift. Also have some datasets you already have in MySQL, PostgreS, and you may have metrics data in a metric store, for example, Druid.

When you want to make a dashboard, once you look holistically across those different data sources, Presto is an excellent engine that allow you to join [inaudible 00:12:44] different datasets in a single view or a single query result.

[00:12:49] JM: I think I saw on the talk that you gave at Strata, there's another tool listed. I actually was unfamiliar with called Apache Phoenix. What is Phoenix?

[00:12:59] LG: Yes. Apache Phoenix is a SQL layer. Basically, it's an embedded SQL layer hosted inside HBase. We don't use this one yet at Lyft. I use that in my previous place and before as well. So the reason you can use Phoenix on HBase is that this tool allow you to model your HBase data or this called raw-based data in a relational model, and it could provide a nice JDBC API to interact with HBase datasets. This one is basically – It's a SQL abstraction on top of HBase binary data. So the use case is for HBase, not for like MySQL or PostgreS use case.

[00:13:53] JM: HBase is another technology you actually do use. This is actually an older technology that we have not done a show on, and we should do a show on, because of how durable the technology is. I guess it's quite a durable data store as well. So HBase to my understanding is an open source implementation of the Google Big Table database, which is basically a database build on top of the Hadoop distributed file system. Under what conditions are you using HBase as the access layer for your data?

[00:14:29] LG: I mentioned, we don't use HBase here at Lyft, but in my previous place at [inaudible 00:14:33].

[00:14:33] JM: Oh, I'm sorry. I'm sorry. I thought you didn't use Phoenix, but you don't use HBase at all.

[00:14:37] LG: Yes. The answer is yes. We don't use it here. But in my previous place, I use HBase. So I can still talk about this one. The HBase is use case I've seen in the past few companies before I joined Lyft is we use HBase mostly to aggregate or to compute. It's a storage layer for a compute in a tip of stream.

Imagine you have a streaming data coming in somewhere to your batch compute and the data needs to land somewhere for this data to do incremental updates and aggregation. HBase, it's a very fast store to do risky-based insert update and roll up as well. So once the data is aging to a certain point, this no longer need to stay in HBase. There typically will be a second layer of ETL which extract the data out to a longer storage, like Hadoop, or HDFS, or S3 somewhere that can serve in the data. On top, if you have Presto, actually it can have [inaudible 00:15:37] of those datasets across Phoenix, across your data lake, your S3 data lake, across those different layers and to serve both fast and slow data to your metrics.

[00:15:47] JM: In a data platform, can HBase serve as a kind of faster access cache that might be sitting over your old data lake?

[00:15:58] LG: Yes. I use that in my past, which is mostly serving the last – It depends on the size of data. Let's say, last a few 30 minutes, or last a few hours of the data before it's being aged out to slower storage like S3.

[00:16:13] JM: Interesting. Is S3 what you're using for your data lake?

[00:16:16] LG: Yes. Here at Lyft, we use S3 primary to store our raw and de-normalized data as well.

[00:16:23] JM: Is there ever a feeling that perhaps the costs are too prohibitive for managing your data platform in the cloud? Have you ever thought seriously about moving this to some on-prem infrastructure and scaling out a Hadoop distributed file system and managing this all yourself?

[00:16:44] LG: That's an excellent question. In our case, actually our compute cost actually is higher than the storage cost once we decouple the storage and the compute back in 2017. So we do have an explosion of those uses cases needs to access data to compute this data in a different way to serve different machine learning model or different dashboards. At the end of the day, majority of the data, they don't need to persist back into the data lake for that purposes.

So, actually, S3 helped us to save money over the last couple of years. Again, it's still very decent chunk of expense we have on S3. We're actively looking for ways to even optimize our S3 usage mostly through the tiered storage strategy, like in AWS. You can have different tiers of storage. You can have S3, S3IA, which is reduced availability tier and you can even further downgrade that storage tier to called Glacier, in a Glacier deep archive.

So there are ways to structure tiers. We need to build intelligent way to how to age the data. It's similar strategy, like I've done in the past when we aged HBase. We look at how it's being used and whether or not we can predict or when at a certain time the dataset most likely won't be used at all, let's say, next 90 days or so. Let's just age it out.

Same thing on S3, we can do something similar here [inaudible 00:18:29] to a cheaper storage. We can do [inaudible 00:18:32]. We are working on that solution actively at some point where you will have a bigger saving on S3.

[00:18:39] JM: For listeners who are more curious about that, we actually just did a show about Amazon Glacier, basically, about the S3 Glacier thing that they rolled. Deep Archive, Glacier Deep Archive that they rolled up pretty recently. Interesting conversation there. But let's move on.

Airflow. How do you use it? Why is it practical for developers building applications?

[00:18:58] LG: Airflow if our – Very good question. So Airflow has matured a lot in the last three years or so since it's open sourced. Here I joined in early 2018 at Lyft, and it's why I received quite a few use cases of Airflow now exploded to even more use cases. So the way why Airflow is so popular, first of all, it has an intuitive user experience UI, web-based UI. Users can see or visualize the dependencies [inaudible 00:19:33] basically for or just different jobs.

For most of our batch compute, even for some [inaudible 00:19:40] of our machine learning model training workflows, we use Airflow. Because Airflow give you this dependency definition using Python. It's a great way to abstract – It can abstract your dependencies through Python, but you can also code your specific logic in Python as well. On top of that, Airflow provides us different operators so we can abstract away how the code should be executed. For example, we have executers for Hive. Not executers, operators for Hive, operators for Kubernetes, Spark and we have operators for Lambda, AWS Lambda function, Invocation and there are a few others.

So you can imagine, through Airflow, you can orchestrate different unit work through a repeatable [inaudible 00:20:29]. I've seen that Airflow has its – Airflow is a repeatable workflow execution engine. So it can define SRA and a dependency even between different [inaudible 00:20:39] rounds, which is extremely helpful for us to have a predictable result. Whenever there's a stack failure, we can quickly look into [inaudible 00:20:48] Airflow UI to troubleshoot sometimes can jump start those stacks right away.

[00:20:54] JM: Apache Kafka.

[00:20:55] LG: Yes. Kafka is our central message bus. So I think hopefully most of the audience are already familiar what Apache Kafka is. The way we use Kafka is we do have multiple Kafka clusters to serve for research isolation purposes and for our security reasons. But in general, we use Kafka for message parsing for our event streaming. Just two purposes.

When we say message parsing, most of our data platform infrastructure services and the components, we do – A lot of different metrics and the logs also recently. All streaming through Kafka and we have different tools, different consumers, which consume those messages and the metrics pushed to our metrics store, or pushed to our Kibana for log searching.

This is one use case, and Kafka also is used for our event streams. This is one of the larger Kafka clusters event streams. You can imagine, most of our business, Lyft business is based on events. When I write a request, I write, let's say event. When drivers drop off a rider on a given location, that's an event. So that's the user-facing events. But also on the system side, when there's a system up or down or system, or when there's, say, a Spark job failure or Spark job is running, say, running slow. Any of those events, this metric – Those events will be captured or kept and send to Kafka and be processed and analyzed by our downstream consumer pipeline. Kafka is a critical piece for us to move data, the real-time metrics data or an event data around.

[SPONSOR MESSAGE]

[00:22:56] 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]

[00:25:04] JM: So if you're talking about an individual ride. Let's say a user is riding – Or a rider is driving around in a driver's car. Is that session, is that modeled as a Kafka topic?

[00:25:18] LG: Model each of those as an event. Each event you can imagine, it has primary dimension of those different attributes. You've got lots of – Have enough of dimension of enrichment metadata. So every event is received by our front proxy, Envoy, and we capture those events and push to Kafka and then bouncing. We do have many different pipelines to [inaudible 00:25:45] and to further enrich those different events.

Towards the end of this pipeline, you can imagine for a BI or as batch report. We do have a pipeline which aggregate and de-duplicate and maybe enrich those datasets until you reach the final destination, which is in our data lake for most of the events.

[00:26:07] JM: How do you do the topic modeling? I guess what's the schema is kind of what I'm asking for –

[00:26:15] LG: Sorry. So the schema –

[00:26:17] JM: No. No. It's okay.

[00:26:18] LG: Yeah. So in Kafka we do the topic – Most of topic are shared. We do multiplexing, because we have so many different types of events. We have tens of thousands of different event types. In Kafka you can imagine, any given Kafka cluster, it can only have a handful, let's say hundreds of topics for a given Kafka cluster.

So what we did is we do have – For most of the events, we do multiplexing. So most of the events is being enclosed in an envelope event type which has metadata describe what type of the event is or is sent as a protobuff, buffer objects as the bytes and sent over to Kafka. At the downstream, we do have basically a routing logic based on the metadatas ahead or up to envelope to decide how to process or who to process the data and do a further splitting.

[00:27:15] JM: With all these processing that takes place around Kafka, so like you mentioned enrichment, for example. You got to enrich all these data points that are coming in. Just to fill people in on why that might be useful, let's say you get a Kafka event. You get an event into Kafka that's from a user's phone. Let's say it was generated from the user's phone and it includes like their geo location.

You might want to enrich that geo location with perhaps the nearest drivers that are close to that user. Then you kind of have a more rich event or piece of data that's sitting on the Kafka queue. I don't know if that's a good example. But there are plenty of reasons why you might want to enrich this data. It comes in. It's sitting in an event stream and then you enrich it. In order to enrich something, you have to do some sort of processing on it.

Now, there’s a number of ways that you can do stream, distributed stream processing as you want to across these Kafka streams of data. A stream is just kind of an append only log of data that you’re writing to repeatedly. Instead of editing – So editing these different events. You’re kind of taking an event off of this queue and perhaps enriching it and then writing it back to the queue in a different place. So what kind of stream processing tool do you use for that enrichment use case, for example?

[00:28:51] LG: Very good question. So it’s an evolution. So it’s not like a single solution. Lyft started with enrichment written purely in Python a few years back. At the time, there’s a very small Kafka and a much bigger AWS Kinesis. Gradually, the Kinesis got scaled down and the Kafka [inaudible 00:29:15] spin up [inaudible 00:29:17] Kafka clusters to serve different cases and then processing their enrichment pipeline [inaudible 00:29:23] multiple pipelines. Then newer pipelines has been implemented. It’s using Apache Flink since 2018.

So nowadays, most of the newer pipelines, or in Flink, we use Flink 1.7 for most of the recent pipelines. Their enrichment typically, you can imagine, there’s a certain data or a small enrichment dataset [inaudible 00:29:52] for example, where you just mentioned, like geo or layer by driver. So there are different types of joins. We do have join against other streams, like the driver stream that – The example you have mentioned. We have a similar idea.

We also have enrichment against other datasets, more static datasets. For example, we have coupons, or you have something, and those streams – Those data sources or datasets are very static. We’d load into those Flink pipeline as well to do the join.

[00:30:26] JM: There’s a lot there that you just talked about that I want to go into. The first thing is you suggested a migration from Kinesis to Kafka and yet you are so heavily into the AWS game. Are you managing your own Kafka on AWS?

[00:30:44] LG: Excellent question. In Kafka right now, most of our Kafka is managed or provisioned through Confluent Cloud, but we’re still running on AWS.

[00:30:56] JM: Why did you move from Kinesis to Confluent cloud Kafka?

[00:31:00] LG: The reason is there are certain issue with Kinesis limitation in terms of the – We call amplification, but there are too many consumers for consumers for a given topic. The Kinesis behavior is lagging or inconsistent from what I heard from our streaming team.

So Kafka has more predictable performance when you have a large number of consumers. We do have a different – Many different consumer pipelines consume from this handful of topics through Kafka. It's [inaudible 00:31:37] reliable comparing to what we seem in the past.

[00:31:41] JM: Okay. Fair point. Given that you are on Kafka though, why not use Kafka streams instead of Flink? You said you use Flink basically for this stream processing on top of Kafka. Why not use the Kafka streams API?

[00:31:56] LG: Very good question. So the reason for Flink is Flink is, I would say, multi-paradigm stream processor. Kafka stream mostly is strictly for Kafka elements. What we need on the stream processor is not only support – not only need process in the stream enrich a stream through these different data streams or data tables either inside or outside Kafka. But we also – We start building a whole list of online machine learning models and enrichment model-driven enrichment as well on to this pipeline, and Flink provide [inaudible 00:32:39] at least provide capability for us to build and enhance the pipeline even further to using the machine learning model to do better enrichment.

[00:32:52] JM: So you felt that Flink, if you chose Flink over Kafka streams, you might be a little more future proof int eh event that you wanted to do a little bit more dynamic application building stuff on top of this streaming system.

[00:33:07] LG: Correct.

[00:33:08] JM: Now, why is that? I guess Kafka streams is more explicitly made, like I guess the API and what it optimizes for is more explicitly like this is when you want to do this kind of enrichment or this kind of like computation that is close to Kafka.

[00:33:27] LG: Kafka stream is very lightweight. I've used that in my past, now here. So it can process this lightweight enrichment actually with lower overhead compared to Flink. But like I said, most of our enrichment here in Lyft, we overgrown to the simple enrichment. We do have a lot of geo-based model, section-based model. Like you just mentioned, like there are certain sessions.

So we have many different machine learning [inaudible 00:33:59] model that's rolling inside those Flink pipelines that is much easier to maintain and enhance, enrich through Flink than through Kafka streams.

[00:34:10] JM: All right. Well, I think it's a good time to shift our focus to Apache Spark. We've barely talked about Spark at this point. So given all the different tools we've already talked about and their pros and cons, including Flink, why is Apache Spark useful and why have you spent so much time instrumenting your data platform to run Apache Spark effectively?

[00:34:36] LG: Very good question. Actually, you have multiple questions inside a single question.

[00:34:41] JM: Sorry. I didn't mean to do that.

[00:34:43] LG: Sure. Let me breakdown one by one. So why do we need Spark here? Spark, if you still remember what I talked about, the evolution, the organic growth of our use cases at Lyft in the last two, three years. We do have a large use case, a large crowd of different teams and different organization within Lyft depending on the ETLs or the data lake or the data warehouse to serve these speeds.

Traditionally, the Red Shift and Hive, those use cases – Those pipelines can certify certain needs. But as we grow even more, it's becoming harder to build only on SQL and only through the store process either Red Shift or Hive. Then Spark comes in to satisfy [inaudible 00:35:38] use cases, particularly this use case is similar as well where we moved from Kinesis to Kafka, to using Flink. Many of those ETLs is a simple SQL transformation. It's transformed from one data format to another data format with certain [inaudible 00:35:57], simple computation.

Most we're seeing are evolving ETL pipeline to include more and more machine learning models inside those ETL transformation to enrich those fast datasets before the landing on the data lake again. So ion this case, Spark is a perfect tool to support this, because most of our Spark use cases are Pi Spark. The reason for Pi Spark is there's a lot of tooling, Python libraries that can use in terms of like geo model and there's also other model like the – What we say, the routing, like when you route a write. These different models, you can model these different data table – Data relationship in a much richer way in Spark [inaudible 00:36:47] Python library than through a simple SQL.

In those cases, we see many of the ETLs, they're moving to Spark to support this enhanced ETL experience. That's one thing. The second thing is Spark has also have a very good integration, especially the newer version, 2.3, 2.4 versions has a good integration with TensorFlow. We do have a few use cases. They need to even embed certain deep learning models inside this ETL pipeline to work with a dataset, enhance the datasets or classify datasets before they save the result into another – Let's say, another Hive dataset in the data lake. So [inaudible 00:37:32] Spark can purify those processes in a single platform. So those are the two primary reasons we use Spark.

[00:37:41] JM: Why doesn't Flink satisfy these use cases as well as Spark?

[00:37:49] LG: Flink, like most of our audience already know, Flink is purely JVM driven. Certain models – When we can't find certain JVM or Scala, Java-based models that can fit into the Flink model and we do need online model driven enrichment, we can put into a Flink. But majority of our models live in Python, and the Spark has excellent support for Python interaction with those Hive datasets. That's where the strengths Spark come from.

Most of those batch ETLs, called batch enhanced ETLs, those are developed in Python and then you use this Python library to enrich and write. It's much harder to do in Flink. Also, in newer version of Apache Beam, they started adding Python support. We test it. It still have limited support to integrate between the Flink processor tasks and Python processes. The integration is not as rich as what Pi Spark or the Spark has. So most of the batch processing stills stay in Spark land for now, but we can't revisit the issue when Flink has a much surer support for Python.

[00:39:12] JM: Let's take a quick detour, because you mention Apache Beam there. Apache Beam is an API out of Google – Well, it's open source, but basically I believe that the creation Apache Beam was for the purpose as follows. So Google has this system called Dataflow. That's a proprietary system. It's also a paper that came out of Google, but the proprietary data flow, Google Cloud Dataflow system, is supposedly really good for batch and streaming. It's a really good system. It's not open source, but there is an API called Apache Beam, which you can use on top of Dataflow. So it's an open source way of accessing Dataflow, and there are also compliant APIs for Apache Flink. I think the API support for Spark is slightly less there.

But I just love to get your – Because you mentioned it. I would love to get your thoughts on the Apache Beam, I guess you would call it a vision. How viable do you think that vision is and, I guess, in comparison to Dataflow, and that's way too many questions that I've thrown at you and you've already chastised me for that. But you can answer whichever ones you want.

[00:40:27] LG: Sure. Let me start from the Apache Beam perspective. We are here at Lyft. We explored Apache Beam. In the past, if you attend one of the Thomas [inaudible 00:40:38], his meet up talk I think back in January also. We talked about some of the use cases. We use Apache Beam. It's a limited experiment. We do use Beam for certain cases and we try explore the possibility whenever it's appropriate. It has some Python support as of the latest Beam version that you just mentioned. It's mostly focusing on the streaming side. So the integration with Apache Flink is much richer than integration with Apache Spark.

So when there's a case we need Python support for streaming and we explore with [inaudible 00:41:21] Beam can help in those use cases. But in general, for high-performing streaming jobs, we do not use Python to interact with Flink at the individual task level just because of the performance is not there yet.

[00:41:39] JM: Well, we should get back to Spark. So you've mentioned why Spark is uniquely useful, and I want to go into the runtime characteristics of Spark running on your Lyft infrastructure. But real quickly, let's give an example. Give an example service that you would use Spark for that describes or that exemplifies why Spark is so useful for you.

[00:42:05] LG: We have quite a few use cases. Sessions is one of them. You can imagine, like I said, Spark mostly for batch compute. So one of the batch compute use case is annotate those events coming out of Kafka, landing on this S3 just raw events. We need to organize and enrich it using session-based annotation with certain session attributes and then serve to our metrics dashboard and – That also serve into downstream machine learning model inputs.

So in this particular case, Spark is helpful is those two factors. First of all, it can break – Usually in the past, we have a very complex SQL written in either Hive and previously also in Red Shift a few years ago. Breakdown that one to a mutli-stage Spark job. That's one. It helped tremendously to improve the efficiency of individual stage of this job based on the characteristics of the data, the data shape. As you can imagine, data shape sometimes [inaudible 00:43:23] SQL data and using these individual states of transition helps the overall job performance in terms of resource utilization and runtime. That's one factor.

Another factor is why we start this journey of introducing Spark for this particular use case. They also started exploring richer libraries that's available to Spark mostly on the geo side. Also the time dimension. When they want to sessionize a set of events, they can build a model. I don't remember the exact machine learning model name, but it can model this different events based on the characteristics [inaudible 00:44:04] on this event time access.

When they model this one [inaudible 00:44:09] this model, they can have a more precise session boundaries between those events. So they can properly attach these session attributes to those events and the push to the downstream data lake table. Or in those cases, Spark help tremendously to improve our job efficiency and the data richly send accuracy as well.

[SPONSOR MESSAGE]

[00:44:41] JM: Today's episode of Software Engineering Daily is sponsored by Datadog. With infrastructure monitoring, distributed tracing and, now, logging, Datadog provides end-to-end visibility into the health and performance of modern applications. Datadog's distributed tracing and APM generates detailed flame graphs from real requests enabling you to visualize how requests propagate through your distributed infrastructure.

See which services or calls are generating errors or contributing to overall latency so you can troubleshoot faster or identify opportunities for performance optimization. Start monitoring your applications with a free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog and learn more as well as get that free t-shirt. That's softwareengineering.com/datadog.

[INTERVIEW CONTINUED]

[00:45:37] JM: Describe the resources that you need to run a Spark cluster.

[00:45:41] LG: Obviously, in any cluster. So it depends on what kind of clusters you did. We do have a – Since we're multi-talent in data platforms, some of our clusters are shared across different use cases. Some of the clusters a more dedicated for reasons of resource isolation and the dependency management.

This is the case before we introduced Kubernetes. We do have, I would say, close to hundreds of smaller Spark clusters on [inaudible 00:46:12], which is very operational heavy to manage so many different young clusters. Most of the Spark jobs actually is not that big. We do have a certain large spark jobs that leads to have a cluster with hundreds of loads. But most of our Spark job, the data size for each individual job is not that big and they can fit into a, let's say, cluster with less than a hundred load.

Again, when we talk about loads, let's say this is a bigger load. This load is what? R4, 16, X-large type of load. So When we have those many clusters, it's harder for a user to interact and decide which clusters or jobs to run from a debugging perspective. From the deployment perspective, actually we instrument most of the deployment process as long as they have an intention – Declare their intention in the metadata what type of job is and within the ownership of the team. We auto assign a cluster to these jobs and they will run either on their dedicated cluster or on a shared cluster depending on those metadata.

[00:47:28] JM: Describe the runtime of a Spark cluster on Kubernetes in more detail.

[00:47:34] LG: Yes. So Kubernetes, if we did something similar, or some of the audience probably already attend my talk in Strata. The Kubernetes, we do have also many pools of clusters. These clusters tend to be smaller, up to a couple of hundred nodes in each cluster. We pull these cluster together to serve different characteristics of jobs.

For example, certain large batch compute that can tolerate high latency, but they need high throughput as well. For those jobs, typically we send those Spark jobs to this particular pool we call scheduled, and the scheduled pool has a few large clusters, but highly utilized. So Spark jobs over there may run slower, but they may get large amount of executers and memory to run to finish. So that's one case.

For other use cases, like other use cases. For example, our user location calculation. If you want to enrich a user location or a certain – Let's say you have a location-based coupon or rewards or something. Those pipeline, Spark jobs, typically is much smaller. We do have smaller cluster pools to serve those use cases to have optimized costs obviously and also to have a better isolation from those larger jobs. This is from the isolation side, but we also have isolation not only for the job size and job characteristics, but also for compliance reasons.

On Kubernetes, we do have separate cluster pools, for example, our financial infrastructure. They are also run in Spark on Kubernetes. In those cases, they have their dedicated cluster pool just for a reason of security and compliance. Their datasets only accessible to the team that you run in the financial reports and no one else within the company can access that. Although we manage the Spark on Kubernetes infrastructure, but we don't have access to that data. So these are some of the distinction we used to have those different Kubernetes clusters.

[00:50:04] JM: What are the biggest frictions that you encountered in getting Spark to run on Kubernetes?

[00:50:11] LG: Yes, excellent question. Spark on Kubernetes still in this very early stage. [inaudible 00:50:18] very early stage even with the Spark 2.4, the latest [inaudible 00:50:22] 2.4.1. We still see there's a huge gap of while we would like to see how Spark perform and what the Spark 2 actually is doing.

So we have to build components to bridge these gaps mainly for observability, like metrics. That's one thing. The other thing is reliability of those parts. In the Kubernetes, you can have – Parts have different [inaudible 00:50:49] service guarantees for most of our jobs that requires higher SRE. We do need to turn on the quality of service mostly in Kubernetes. The highest tier of the quality of service is called guaranteed tier. We have to tread the Spark parameters significantly to support higher guarantee. That's one thing.

The second thing is there are many different hoops we have to jump through to enable different connectivity, because – For example, the Spark driver, when it runs on Kubernetes, you're exposed to surveys, but it doesn't expose a user accessible UI when you have a long-runing Spark job. User does not have a way to look at the Spark UI to see what the progress, where are the bottleneck and so on and to terminate the job.

In a young world, you can go to the [inaudible 00:51:39] resource manager to determine that a poor job.

So we have to build tools to support that and build tunnels and proxies to support those features. That's another thing. The third thing is Spark in Kubernetes does not have equivalent of the scheduler, we call it customizable scheduler that [inaudible 00:52:03] provides. [inaudible 00:52:04] has this different schedule called fair scheduler, or capacity scheduler or a combination of both to serve the different use cases to maximize both the job performance and the cluster utilization.

In Kubernetes, we don't have any of those. So we have to build our – We call job controller at each individual cluster level to simulate what [inaudible 00:52:28] has to serve those different requirements for fair release of the job scheduling, the cost optimization of the cluster and the optimization of the job runtime. So it's a lot of work to add on top of both Kubernetes and Spark to support our use cases.

[00:52:49] JM: Now, this may be a little too ironic, but the Uber team has built a scheduler, I believe for – I think it was originally for Mesos, but it's abstract enough that they're talking to the Kubernetes team about getting that scheduler into Kubernetes. I believe it's a little more dynamic. So you may have your scheduling problems solved by the Uber team.

[00:53:13] LG: Yes, absolutely. Actually, in many of the past conference, I talked to the Uber guy on this, it's called Peloton, the scheduler which is originally come from Mesos. So it's a similar idea, but in our case, we have multi-tenancies. So at a different cluster pool, we need different settings. Sometimes more fair scheduler [inaudible 00:53:37]. Sometimes the scheduler is more capacity-driven. Sometimes it's more at the nature of like a [inaudible 00:53:45] scheduler or combination of those three.

[00:53:49] JM: These other tools that you've built to overcome the frictions between Kubernetes and Spark, you mentioned like some tunnels and some proxies. Can you tell me more about like basically patching Kubernetes and Spark to work together better?

[00:54:06] LG: Yeah. One of the strategy we currently implement is we try to limit our patches directly to either Spark or Kubernetes just for the reason is we want to keep up to date with the latest versions of Kubernetes or latest versions of oru –

[00:54:26] JM: Sorry. I didn't literally mean patches, but thank you. Thank you for answering –

[00:54:31] LG: What we did is we build components. You can think of this as a glue component, glue those two together. So we call controllers. In Kubernetes, unfortunately, they have a very rich set of APIs. We can build controllers on top. So we build a lot of different controllers to bridge the gap between those. For example, we have a long run in Spark job. You have a service and you have a service endpoint. We can expose that through our dynamic controller which serve your [inaudible 00:55:01] traffic through a tunnel to whoever this user is and through our ingress controller. This is Kubernetes building ingress controller.

But when the job finishes, obviously the job UI is no longer live on the driver pod. It moved on to the Spark history. So the controller will – since everything is event-driven on Kubernetes, the controller can dynamically adjust the ring and redirect their user back to the Spark history. That's an example. We can tunnel those UIs.

As an example – So for example, when Spark try to request a certain executor. Some of our controllers, we need to make sure when the Spark is [inaudible 00:55:46] X-number, let's say a hundred executors in a name space. When we see the namespace is saturated, or actually, we do have a [inaudible 00:55:55] namespace and job controller, which can dynamically adjust those parameters and send to send other namespaces just to ensure the job can run on this cluster. It's unfortunate that Kubernetes right now, when we observe, it can only support thousands of running parts in given namespaces. Comparing to what we see in Yarn, we can have tens of thousands, or even hundreds of thousands of Yarn containers in a given Yarn queue.

So what we did, we build a similar abstraction [inaudible 00:56:29] space group. [inaudible 00:56:31] is, for example, in Yarn can have, say, high priority queue in Yarn. In Kubernetes, we can have a high priority namespace group. Within that can map to many namespaces depending on the utilization of each namespace.

[00:56:47] JM: I know we're beginning to wrap up. I just want to comment on what is amazing is how similar the data platforms between Lyft and Uber are. We did a show in the past about Uber's data platform. I would say the main difference between the two – Well, there're too many differences I notice. One is the fact that Uber's rapid international expansion and expansion into other categories of service has led to a much less even data model, I think, and that unevenness of their data model across different geos and different platforms and stuff led to really fundamental differences in decisions that they had to make early on.

The other main difference I noticed is that I think they very early on decided to go to on-prem infrastructure, where as you have kind of gone all in on the cloud. Any other interesting differences between the two company's data infrastructure that you've noticed?

[00:57:45] LG: Yeah. I think you're correct on this too. Very excellent observation. Yeah. Also, since Lyft started later than Uber. So some of our headaches at Uber thing we bypassed, for example, the HDFS issue. If you went to any of the Uber talk in the past, they talk about a lot of the pain points when they're dealing with large HDFS deployment. So we bypassed those issues to the cloud storage.

[00:58:16] JM: Yes.

[00:58:16] LG: So that's one thing. I think there's difference. One of the strategy we tried to adapt is we tried to adapt as much of the cloud native way as possible, and another distinction we have here is we don't use Mesos here. The reason is when we started looking at container orchestration, it's already in 2018, last year. So when we started looking at Kubernetes, it's already the dominant cloud orchestration – Not cloud orchestration. The container orchestration engine in the open source world.

So we moved beyond what Mesos orchestration can do and we're looking at [inaudible 00:58:52] the best from all three worlds, from the Yarn, from Mesos and from Kubernetes. Combine all those best attributes from each world and put into this new world, which Kubernetes is the underlying OS for the distributed Spark loads or are data compute leads.

[00:59:09] JM: To close off, you spoke at Strata. Usually we're at Strata. That was quite an interesting conference. Let's close off. What are your takeaways from the Strata conference and the state of the data landscape?

[00:59:21] LG: Yeah. I think from what I heard from these many different talks and the keynote. The Strata is most of the platforms or solutions or the product I've seen is moving towards a more cloud-driven or cloud native. Certain cases, it's cloud native way. In some cases, it's a multi-cloud solution or cloud agnostic nature as well. So to move data, and this one attribute.

Another attribute I've seen is there's a rise of data discovery, data lineage. Basically, data governance. More importantly, intelligent, or machine learning, or AI assisted data discovery is on the rise. I can imagine further down the road, discover data and use this machine driven inside to help uncover the value of data will become a more common place across those different products.

[01:00:20] JM: Li Gao, thank you so much for coming on the show. It's been a real pleasure talking to you.

[01:00:23] LG: Thank you.

[END OF INTERVIEW]

[01:00:28] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]