EPISODE 594

[INTRODUCTION]

[0:00:00.3] JM: When a user takes a ride on Uber, the app on the user's phone is communicating with Uber's back-end infrastructure, which is writing to a database that maintains the state of that rider's activity. This database is known as a transactional database, or OLTP for online transaction processing.

Every active user and driver and Uber Eats restaurant is writing data to the transactional data store. As you are taking rides throughout the day, your phone is interacting with that transactional data store. Periodically, that data is copied from the transactional data system to a different data storage system where that data can be queried for large-scale data analysis. For example, if a data scientist at Uber wants to get the average amount of miles that a given user rode in February, that data scientist would issue a query to the analytical data cluster.

Uber uses the Hadoop Distributed File System, or HDFS, to store analytical data. On this file system, Uber has a version history of all the company's useful historical data; trip history, rider activity, driver activity, every data point that you would want as a data scientist, or a machine learning researcher, all of this data is in the online analytic processing database. It's also in the transactional system, but in the transactional system, it's not in the format that is easy to query for large-scale data processing. This file format is known as Parquet. It's a columnar data format that we've done a couple previous shows on.

Data scientists and machine learning engineers and real-time application developers all depend on the massive quantities of data that are stored in these Parquet files on Uber's HDFS cluster. To simplify the access of that data in the HDFS cluster by many different clients, Uber uses Presto, an analytical query engine originally built at Facebook.

Presto translates SQL queries into whatever query language is necessary to access the underlying storage medium. Whether that storage medium is a Elasticsearch cluster, or a set of Parquet files on HDFS, or a relational database, Presto is useful because it simplifies the

relationship between data engineers and the application developers who are building on top of that data engineering infrastructure.

If you're a machine learning researcher, all you need to speak is SQL. You just write SQL queries against Presto and Presto's query planning engine is able to gather that data, translate the SQL query into whatever domain specific query, for example querying against Parquet files, or querying against Elasticsearch, and deliver you the result that some SQL database would. Presto is super useful and that's why it's being used at Netflix and Uber and Facebook and a variety of other companies.

In today's show, Zhenxiao Luo joins to give an end-to-end description of Uber's data infrastructure. Zhenxiao is an engineer at Uber and he explains the full pipeline from the ingest point of the OLTP database, to the OLAP data storage system on HDFS, to the wide range of data systems and applications that run on top of that OLAP data.

This is a really good show. I enjoyed getting the end-to-end description. Thanks to Zhenxiao for giving such a good lesson in data engineering infrastructure.

[SPONSOR MESSAGE]

[0:03:49.2] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform, so you can get end-to-end visibility quickly.

It integrates seamlessly with AWS, so you can start monitoring EC2, RDS, ECS and all of your other AWS services in minutes. Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast. Try it yourself by starting a free 14-day trial today.

Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[0:04:54.4] JM: Zhenxiao Luo is a engineer at Uber. Zhenxiao, welcome to Software Engineering Daily.

[0:04:59.6] ZL: Hi, Jeff. Nice to meet you.

[0:05:01.7] JM: It's great to have you here. Uber has a lot of applications, and some of those applications require large-scale data processing. Then there's all these applications that use that large-scale processing. They use the models that are built on top of it, or you have data scientists that are doing ad hoc queries. What are the requirements for the analytics infrastructure at Uber?

[0:05:28.3] ZL: Yeah, very good question. Actually, we have money requirements and most important one I think is scalability, reliability and the performance. The number one thing is scalability. I mean, for Uber we have so many internal either reports, or our tools into analytics queries. The number of queries is huge. Also the data that each of these query are touching is also huge.

We need to scale both our storage, also the compute stack and the whole data stack. That is number one thing. Also for performance, also we have a very high requirement, or a meaning that some of the query we need almost real-time, some most we need, at least within a few seconds. The performance and also we need to be reliable at almost 24/7. All of these [inaudible 0:06:21.8] pocket.

[0:06:23.3] JM: You've got all this user data that gets generated. For example, a user request a car and then takes a ride in that car. When that data is getting created, this is typically called transactional data. You have an OLTP database. The online transaction processing database, these things that are transactions that effect the business applications of the day-to-day business of Uber. Then you have to transform the data in the transactional database to analytical database systems.

This is obviously a very abstract description. You have a lot of systems that underlie both the transactional infrastructure and the analytical infrastructure. When we think about the requirements for the transactional stack versus the analytical stack, so the analytical stack I think of as it's more important to access in a columnar fashion, because for example you need to aggregate all of the tips, for example, that users gave during a certain time horizon of rides that were given. You want to aggregate and you want to see how much people are tipping, so you only want that row.

For the user that's just requesting a car and then taking the ride and then maybe they cancelled the ride in the middle of it, or maybe they make a star rating, the different rows all need to be accessible. You want a column – sorry, you want a row-based transactional fashion. I think this is one of the important things to note about transactional databases versus analytical databases. Tell me about the transactional requirements versus the analytical requirements of the data infrastructure at Uber.

[0:08:10.1] ZL: Sure. Oh, Jeff seems you know a lot about Uber. Where do you learn all this? Actually we have two stands; one for the transactional one, we call it the online one. Another one is analytical database, or analytical, a big data site. These are two stacks. They have different requirements as well. I mean, transactional-wise model, reliability is more important and the survey online data.

A performance, we have a really real-time happens. Before the analytical side, we have different requirements. I mean, sometimes we do not need real-time analytics, but the scale is usually bigger. How do we do the transformation? Surely you may heard of Uber built its own transactional online database called schemaless, which is serving the online traffic and this scale pretty good, and also is very reliable.

The problem is all the data that use schemaless doesn't have a schema. One other problem you already said is the data study knows all the data. Writing is in row format, not in a columnar format. These are the two things that we want to transform. We have our own pipeline. We call it Commodore and this way, we do a thing called core, which replace the old pipeline.

Number one is we need to cut type schema for this dat. Number two is we need to transform from a row format into a columnar format for the analytical side to have a better performance. Yeah, that is the thing we are doing and now I think we have a decent latency. I mean, for the ingestion transformation. Really most of the data arrive within 30 minutes. Some of them we can – I mean, if we're building super models, or model cables, we last longer.

[0:09:59.1] JM: Data gets written quickly to the transactional database. Then there might be a 30-minute latency getting the transactional data into the analytical database.

[0:10:09.7] ZL: Yes, yes, exactly. Since we need to do the transformation for number one, we need to either type the schemas to the data. Number two is we need to transform between a row format into a columnar format.

[0:10:21.5] JM: We'll go through some of the parts of that transformational infrastructure and we'll eventually dive down into the discussion of Presto and Parquet and how that fits in. I want to give people a better framing for your infrastructure. Talking about the transactional database at first, so you said you use schemaless, but you also use – you have MemSQL, you have MySQL, you have PostgresSQL. Why do you have this wide variety of relational databases?

[0:10:54.6] ZL: A company at Uber scale is we have so many teams. A big number of teams need these data storage. If they just need a small data storage for some data, then they may just launch on MySQL or PostgresSQL instance. The MemSQL party, actually they are in the big data size. We are buying MemSQL license for MemSQL provides us a really fast, very good performance. You too work pretty well at a smaller scale and now we see more and more challenges to MemSQL.

Now there is a team, the streaming side they are developing our own streaming database really fast, real-time database, as well as we have other solutions try to our field, a field to try if we form MemSQL.

[0:11:42.9] JM: What about Kafka? What is the role that Kafka plays in your – if we're just talking about the ingestion side of things?

[0:11:49.8] ZL: Yeah, Kafka is a very important row. I mean, all the data initially landing in schemaless. We're polling them first into Kafka, and then so everything in the Kafka events. Then we polling the event out of Kafka and doing the other side, a row format to columnar format transformation, and also the type inference of all these things.

Data flows first into schemaless then down into – together with some other things. Some other events are logged directly into Kafka, and then both schemaless and the Kafka data are dumping and first one, doing the transformation and type inference and finally dumped into HDFS.

[0:12:32.5] JM: You mention HDFS. Can you talk a little bit more about how data makes its way from the ingested and transactional point in the data infrastructure into HDFS?

[0:12:45.3] ZL: They are a field data sources within a company. I mean, the online data schemaless and log events mostly in Kafka and some other things are also results in Kafka. Then there's so many relational MySQL, or PostgresSQL instances. All this data, we try to build, we call the Hadoop data lake, which means we try to capture all the data within and a company and store them in HDFS.

We are doing the whole pipelines and we are polling from schemaless, infer the type, doing the transformation. Kafka side, the similar thing; infer the type and doing the transformation. Also, we are copying directly from MySQL or PostgresSQL directly dumping the data into Hadoop. Finally, everything, all the data are captured in HDFS. At the same time, since we infer the type, so we are also registering tables, the metadata information in the hive meta store. At this time, the HDFS is holding all the files and hive meta store holding all the metadata. Then the data is ready to be queried by hive meta store.

[0:13:48.2] JM: I see. The hive meta store, so hive is a querying engine on top of Hadoop and you need a meta store, because the data in HDFS is not necessarily you can't access the schema by HDFS?

[0:14:04.2] ZL: Yes, yes. I mean, HDFS just storing everything in the files. Well, how about the types and the table to column, might be how many columns the table has and also the column

types, and also whether the table is partitioned, where is the location, all these things all in the meta store. The meta store is another very important component. Everyone is talking to the meta store.

[0:14:27.5] JM: Right. My understanding of hive, is that hive is also, it can be used to query directly. It can query Hadoop directly by querying hive, but I guess you could also just use it as a store of the metadata around those files that are in HDFS.

[0:14:46.6] ZL: Yes, yes. Hive is really very rich functionality. I mean, hive can be used for many things. Mostly, the meta store you just mentioned is formatted data storage, which is actually shared by high precedence mark. Everyone can use a meta store, but five itself, we can use it, user can either shoot query directly on hive, although this is a slower. All they can use hive for ETL drops and so many other things as well.

[0:15:12.7] JM: I think it's worth adding a comment here about schema. Data schema. I did a show about schemaless a while ago, I think. We touched on it at least, but when I think about situations where you don't know the schema of your data, I can imagine this being important for example, if you have all these different teams that are working in different geo-locations, for example if you've got an Uber team that's working in Singapore, maybe they are maintaining different metadata about rides. Maybe they have some Singapore-specific Uber application, like I don't know, rent a motor scooter, or something like that that you wouldn't have in San Francisco.

If you think about the abstraction of a ride, you can't necessarily impose a schema on that across the entire company, because it depends what geolocation you're talking about, what vehicle you're talking about. If you want to just query for rides, you don't necessarily want to be assuming a certain schema. I just make that one comment as an example of a trade-off between maintaining a schema and not maintaining a schema. Maybe you could talk a little bit more about the trade-offs of maintaining schema over data.

[0:16:29.8] ZL: Sure. Yeah, I think this is a very good point and also that one of the challenges in Uber data. As you said, if we are very strict for the schema, then since we have so many branches operating in so many cities, so many continents, it's difficult to scale. I mean,

whenever there's someone operations in Singapore, they log something, but maybe the column data exist on something. It's challenging.

Another thing is if we do not have no schema at all, that is how schemaless is doing that and that's difficult for the pick the aside to analytics, since we do not know the type of the data, that's challenging. The way is our current solution is a number one, we try to make the schema flexible to all the operations, or anywhere for lock data. We are using a lot of the complex data types. For example the strat and airways. The strat, we have inside stratless, I mean, in Uber, it is called [inaudible 0:17:27.7]. 20 fields inside a strat and the inside that strat you have another strat nesting. Then that has another 100 field and inside that 100 fields, you have a raised maps and strat inside.

The strat address are common concept for one column. Inside strat different themes and different geolocations, they may have different number of fields inside that strat. That is a very flexible schema management. For our analytic side, our first challenge is we need to support all these schema evolutions. I mean, some of the data, although they might as well say it has 10 fields, but actually the data only have eight. Some of them, although the metadata is saying it has eight, but the data has maybe 20.

We need to support the data to be querable as much as possible. We try to infer all the types during the data ingestion. Once the data ingest it, in the search – in the query engine, both hive and Presto we are supporting a very complex schema evolution. I mean, the nesting inside nesting, a different type, a different number fields, the [inaudible 0:18:36.2] tracks a huge level of nesting inside that and the fields are added, or removed the very frequently. Yeah, but now our SQL engine does all that.

[0:18:47.6] JM: I imagine developing this data infrastructure must have required an immense amount of planning.

[0:18:55.7] ZL: Yes. This is I think, as far as I know, the data analyst probably started in 2015 when Uber business really scales. Then we see so many challenges. We were using some enterprise buying expensive enterprise licenses. Now we decided to build our own, and it's

caused almost one year to make it. Sure have a decent performance and then – also, already using that.

[0:19:25.1] JM: Yeah. I talked to Matt Ranney, I talked to Danny Yuan, I talked to some other people in Uber infrastructure and it's just – I mean, it was moving. I mean, it's still moving very fast, but at this point the team has had some time to catch up in terms of infrastructure. It sounded like there was a period of time where even thinking about a consistent translation of analytical data to OLAP data, it sounded imposing that across an entire company would have been almost impossible in the early days, when you're just scaling – right?

[0:20:00.2] ZL: Yes, exactly. Yeah, that's the case. I think starting from two years ago, we started seeing there's so many technical tax. Also it's a one problem. We have money proposed solutions. Now we are thinking more about consolidation, whether we just provide one path solution, instead of so many techniques. Yeah, that is also the one of the most important theme of the company.

[0:20:26.6] JM: Incredible. By the way, do you use HDFS right now at S3?

[0:20:32.3] ZL: Mostly. I mean, the whole Hadoop data lake is on HDFS. Within the company, I mean, the different teams they may use S3 for temporary storage, or some other purposes, so there are still some S3 usage.

[SPONSOR MESSAGE]

[0:20:53.4] JM: At Software Engineering Daily, we have user data coming in from so many sources; mobile apps, podcast players, our website, and it's all to provide you our listener with the best possible experience. To do that, we need to answer key questions, like what content our listeners enjoy, what causes listeners to log out, or unsubscribe, or to share a podcast episode with their friends if they liked it. To answer these questions, we want to be able to use a variety of analytics tools, such as Mixpanel, Google Analytics and Optimizely.

If you have ever built a software product that has gone for any length of time, eventually you have to start answering questions around analytics and you start to realize there are a lot of analytics tools.

Segment allows us to gather customer data from anywhere and send that data to any analytics tool. It's the ultimate in analytics middleware. Segment is the customer data infrastructure that has saved us from writing a duplicate code across all of the different platforms that we want to analyze.

Software Engineering Daily listeners can try Segment free for 90 days by entering SE Daily into the how-did-you-hear- about-us box at sign-up. If you don't have much customer data to analyze, Segment also has a free developer edition. But if you're looking to fully track and utilize all the customer data across your properties to make important customer-first decisions, definitely take advantage of this 90-day free trial exclusively for Software Engineering Daily listeners.

If you're using cloud apps such as MailChimp, Marketo, Intercom, Nexus, Zendesk, you can integrate with all of these different tools and centralize your customer data in one place with Segment. To get that free 90-day trial, sign up for Segment at segment.com and enter SE Daily in the how-did-you-hear-about- us box during signup.

Thanks again to Segment for sponsoring Software Engineering Daily and for producing a product that we needed.

[INTERVIEW CONTINUED]

[0:23:23.2] JM: Was the choice of HDFS instead of S3, was that a cost decision, or what was the – because I think, doesn't Netflix use S3 for their data lake?

[0:23:32.3] ZL: Yes. Netflix is totally purely on S3. The reason we chose HDFS, I mean, we are still discussing our running benchmarks. Now we are evaluating all for the cloud side, so maybe in the future, we will be a mixed architecture. Now the data side is still mostly on HDFS. The reason is I don't think there's a technical, or economical reason, but just we start with HDFS and

that we grow so fast that we have so many data the year and it's really challenging to migrate a whole pipeline to another storage. That's really challenging. We even do not have time to catch up on that side.

[0:24:18.2] JM: Okay. I think we glossed over this, but you have data that is written to the transactional side. At some point, it gets copied over to the analytic side. How does that batching work?

[0:24:30.3] ZL: Yeah. We are doing actually a timestamp-based for that. I mean, the transactional data, yeah, number one is while we are polling the data from transactional side, we do not want to affect the online performance. That is a critical requirement. We are mostly polling from the schemaless locks. We do not touch the online data directly. We are polling from locks. Then from the locks, we can reformulate the data, and then we do the transformation or something like that.
Then we are transforming that, we actually keep different versions of each of the file. Let's say one of the value is the making of fairs, if some of them is updated, we keep different version of that. Inside the analytics, you can query a different version of the data, actually.

[0:25:22.9] JM: That's amazing. You've got the schemaless database for the transactional data, and you don't want to hammer the read frequency of that transactional data. Instead, you –

[0:25:34.8] ZL: Yeah, we do not. Yeah, we have to call the log.

[0:25:39.2] JM: You read the ahead log?

[0:25:40.4] ZL: Yes, yes.

[0:25:41.8] JM: Interesting. Because you can derive all the data necessary to reconstruct the transactional database itself from the right ahead log. I can imagine that's additionally useful, because you'll be able to – well, if there's a change to the database, you're also going to be changing that data point when you rewrite that data in HDFS, right?

[0:26:03.5] ZL: Yes. The way is we are pulling from the logs, so all the changes we got that, but we are not rewriting HDFS files directly. Actually, we are keeping a number of versions for each of the file. Let's say within the last 10 hours, or even longer, all your changes we have one version of that. While we keep different versions of that and based on your query time, you say like, say I want to query some data in let's say 10 hours ago, we can do that. Most of the user just showed the most recent version of the file. Then after a while, we are back rotating these old version of the file.

[0:26:42.1] JM: Not only do you have the current state of Uber as of 30 minutes latency in the OLAP stack, you also have the entire revision history?

[0:26:54.9] ZL: Yes, exactly.

[0:26:56.0] JM: Cool. What is the infrastructure using to get that right ahead log from the schemaless right ahead log copied into HDFS?

[0:27:08.8] ZL: Yeah, this is we develop in-house and a number of projects doing that. One is open source called streamio, and also another partner called the Commodore on the who are, which are doing the transformation on polling. There's for different version of the file, we develop our own library called hoodie, which is open source I think last year, and for how could you keep different versions of the file and want to infer, aggregating that, the whole integration worked perfect.

[0:27:37.7] JM: Awesome. What does streamio deal with?

[0:27:40.7] ZL: Streamio mostly touching the schemaless logs and then cutting out logs and then try to rebuild, reformat the structure inside schemaless. That is for the polling from schemaless.

[0:27:53.1] JM: Okay. It's just reading the raw log file and copying that data over in HDFS?

[0:28:00.7] ZL: Yes. A little bit more matter than that, they are also keeping a very small HBase, keeping the most recent version, the year, and then the copy, carry the data and then doing some transformation there.

[0:28:15.4] JM: There's so many large-scale stream processing systems. It seems like you could use Spark, or Heron, or something like that for copying that right ahead log data into HDFS. Why'd you choose to build your own?

[0:28:32.8] ZL: Number one is the schemaless has its own very distinct format and the architecture. I mean, for the other things, I don't think any of the system could work directly with schemaless. Schemaless is actually is – you can think of that like a shard in MySQL, but actually installing data since it doesn't have types. Using shard in MySQL to start trace on, and actually say some things. We have to develop our own loop holding the thing and doing the recycle mission.

[0:29:05.3] JM: Oh, I see. Really, this is about developing a specific reader for the data.

[0:29:12.4] ZL: Yes, yes, exactly.

[0:29:14.3] JM: Okay. With that reader, so you have to read the data and then you have to somehow port that read data into HDFS. I guess, you read the data from the right ahead log into memory and then does it get copied from memory into HDFS?

[0:29:32.2] ZL: Yes. I'm not sure whether we are there yet. I mean, ideally, we should just – yeah, polling from the log and that everything in memory that we dump as Parquet, or see that the columnar format. Actually, it's more challenging than that. Since the schemaless really has a huge amounts of data and data is updated frequently, so if we are doing these, we are heating the memory threshold again and again.

We used to at least dump once. We're pulling the logs, and then we do some transformation, we dump in sequence file on HDFS, that we read from HDFS again, the sequence file would transform that into Parquet and columnar format. Now we are improving that trying to make the optimize as well. I mean, directly dumping as columnar format.

[0:30:20.7] JM: Right. For a given right ahead log, can you parallelize streamio and be able to – I mean, you could do this. You could get it all into memory if you parallelized it well enough, right?

[0:30:32.9] ZL: Yeah, since it's already sharded and yeah. Still, I mean, since the data volume is really huge, I think that's more challenging. We have ongoing projects trying to do that. I think we may assume that go this year.

[0:30:49.2] JM: Awesome. Congratulations. It sounds like a tough problem. Anyway, let's fast forward. You've got this data that's shuttled from streamio from your schemaless database into Parquet files, but you use Parquet files to store the data on HDFS. The schema of those Parquet files is maintained in hive. Talk a little bit about Parquet files. Why is that the file type that you want to store the data of Uber's transactional history in?

[0:31:24.0] ZL: Yeah. Actually, one thing is we want to have a columnar file format, so all the analytic side that performance will be really bad, so we have to use a columnar file format and open source solution is a ORCR Parquet. Number two thing for Uber, which is unique is we have so many nesting data. It's highly nesting. I mean, it's quite common in Uber, we can see that a table has a strat, the strat maybe not seed over 20 levels, each of them has more than 100 fields inside that. It's a highly nesting.

Then to support is highly nesting feed data structure in columnar format than the – I mean, the idea that Uber Dremel paper are the foundation for open source at Parquet. That's a reason why we chose Parquet.

[0:32:13.3] JM: Nested data. Explain what it means to have a nested table and why do you end up with a nested table?

[0:32:21.5] ZL: Yeah, since we talk about a metadata, it's really challenging. We cannot keep the table schema just flat. I mean, everything is just a long topple offload. The challenging since, I mean, different geolocations, different teams and different department, they may have the kind of they need to update the schema really frequently, so we can now keep just a very strict

schema for one table. I mean, the schema is after they [inaudible 0:32:47.5]. We are choosing a way to we provide a schema, but the schema is tangible. That's the reason why we choose strat. Now we have so many strat.

Since we use as strat, even if the strat has three fields that is strat, if the strat has 300 fields, that's still strat. That's thing we had with schema, but nesting, the fields inside the strats are really complicated. That's a contract between the logging and analytic side. Data one is a banking, your nature it is in nesting.

[0:33:25.5] JM: Right. You could impose some schema, right? For example, a ride, you want to impose the schema of a ride has a start and end time, and perhaps how much it costs, but beyond that you want to give freedom to the application developers within Uber, if they're in Singapore and you have some motorbike application that doesn't exist in the United States, you want to be able to include the data about, that are specific to motorbike rides in that schemaless right, so you might have a more column. Then in the more column, you have an additional table, and that entire table contains data that's specific to the motorbike ride.

[0:34:09.1] ZL: Yes, something like that. We are doing that in one table. I mean, we have a few big tables and these tables has a few primitive types. The long on timestamp and double float. Then yeah, the others column, or more called – we call a message, inside message that we have so many things and these things are updated frequently. When under that, we still have nesting, some other things nesting there.

[0:34:34.8] JM: Yeah, but the nesting structure is consistent in a way that if you transform it from the schemaless format to Parquet files, you can have guarantees around how that transformation is going to end up in that different – because the Parquet nested format is going to be much different than the representation of the schemaless format.

[0:34:56.1] ZL: Yeah, yeah, yeah, exactly. I mean, we have our the metadata or schema service, internal called what power. This is a contract that whenever they log something, they need to specify their types. They can update the schema. I mean, adding fields, remove fields inside a strat, but our contract is you can now change the type for any field. I mean, inside a

strat if you have ABC type of long, you can now change that to float. That is unparable. You can now change that back, but you can add on it.

[0:35:29.6] JM: You get this data into Parquet files. Parquet is useful for the reasons that – I mean, I think we actually explore that in previous episodes. We don't want to go too deeply into that, but I think what's useful to know is the Parquet format makes it really easy to – you basically have, I think it's like a login access pattern for getting to a specific column if you're talking about nested databases. If you think of it that the nesting structure as a tree, you just have a login structure of getting to the specific column you're looking for, and then you can read the entire column instantly. That's the usefulness of the Parquet file format. Would you say that's accurate?

[0:36:11.1] ZL: Yes, yes. The cool thing is, let's say you have a table of 100 columns, but you'll acquire or may just two column. Instead of loading out of 100 columns, in Parquet it's possible you just read an asset or data, the two columns. We say we huge decile.

[0:36:28.8] JM: It's additionally useful for the nesting, because you can still have a quick access to the entire column of a nested table?

[0:36:37.0] ZL: Yes, exactly. I mean, let's say a strat has 100 fields by user, may just touch a few of them, one or two of them. Parquet provides a – there is API. We can just [inaudible 0:36:47.0] that just gave me these two fields, is not loading the whole field. Yeah, that is since Parquet internally for all the nested thing, they are still storing the fields value together.

[0:36:59.6] JM: Excellent. Okay, we've gotten to a good point. We've stored all of our transactional data in Parquet files that are now accessible from an OLAP fashion. Now, all we need to solve is all of the different types of applications that want to actually access that data. You've got people that want to build machine learning applications, you've got people that want to make ad hoc data science queries, you've got real time applications. Is there a common interface that we can build for people to access this online analytics data?

[0:37:36.1] ZL: Good question. Yeah, actually we have one. Internally at Uber we call the query builder, which is a big data portal. The inside of the query builder, you can either query let's say

Presto, or hive, or you can try some other things. We have some other MemSQL, or the other enterprise database if you like. That is a portal for users to access that.

I mean, again accounting Uber scales, I would say that more than 80%, or even 90% of users are accessing Presto or hive through query builder. Still, there are so many other teams that they are either touching Presto hire directly, or they have sending HTTP requests, or some other ways, JDBC, ODBC to accessing the view data as well. We are on our way to do the consolidation, but still not there yet.

[0:38:28.8] JM: I think here, it's worth getting in the Presto. Explain what Presto is.

[0:38:33.7] ZL: Yeah, Presto is a interactive SQL admin for big data. Yeah, that's in shard. I mean, the history is Prestor was created by Facebook in I think 2012. In production in 2013, and then now I think as far as I know many companies are using Presto, yeah, so many of them.

[0:38:54.6] JM: What is an example of a query that you would issue to Presto?

[0:39:00.9] ZL: A typical SQL query. Actually, can I say any SQL query, maybe not a typical SQL query. You can run Slack, you can run wear clouds, or you can Urbuy, you can run window functions, rank all the - then you can run – there's so many other, for example geospatial functions, which is added by Uber as well, and a rich functionality to the SQL standard.

[0:39:26.8] JM: My understanding of Presto is not super strong. I think it's something like, you can issue a SQL query, and then this Presto engine can interpret that query into whatever the domain-specific read pattern of an application is. For example, if you don't know hive, like hive has its own domain-specific querying language. Presto would be able to translate the SQL query, in like a user level SQL query into a hive query. Is that an application of Presto?

[0:40:03.6] ZL: I think you are talking about what other powerful feature of Presto, called the Presto connectors. Yeah, so actually Presto has an engine. I mean, internally it has a standard SQL engine. That's the reason why it’s fast. There’s so many optimizations for that. Also Presto provide the interface called Presto connectors, which a light user was SQL on any storage. Not on a SQL on HDFS.

I mean, if your data is storing, for example you have to search, user can also think each of the index inside Elasticsearch is one table. Then the Presto Elasticsearch connector will translate the SQL into Elasticsearch, the search request. User can run SQL on Elasticsearch. You can also possibly run SQL on Cassandra, SQL on Kafka, there's so many things, so many Presto connectors available. That is another cool feature of Presto.

[SPONSOR MESSAGE]

[0:41:07.5] JM: Nobody becomes a developer to solve bugs. We like to develop software, because we like to be creative. We like to build new things, but debugging is an unavoidable part of most developers' lives. You might as well do it as best as you can. You might as well debug as efficiently as you can. Now you can drastically cut the time that it takes you to debug.

Rookout rapid production debugging allows developers to track down issues in production without any additional coding. Any redeployment, you don't have to restart your app. Classic debuggers can be difficult to set up. With a debugger, you often aren't testing the code in a production environment, you're testing it on your own machine, or in a staging server.

Rookout lets you debug issues as they are occurring in production. Rookout is modern debugging. You can insert rook out non-breaking breakpoints to immediately collect any piece of data from your live code and pipeline it anywhere, even if you never thought about it before or you didn't create instrumentation to collect it. You can insert these non-breaking breakpoints on-the-fly.

Go to rookout.com/sedaily to start a free trial and see how Rookout works. See how much debugging time you can save with this futuristic debugging tool. Rookout integrates with modern tools like Slack, Datadog, Sentry and New Relic. Try the debugger of the future. Try Rookout at rookout/sedaily. That's R-O-O-K-O-U-T.com/sedaily.

Thanks to Rookout for being a new sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[0:43:11.3] JM: I want to try to understand the usefulness of Presto. What I think of it as, if I understand it correctly, is it's middleware that makes it easier for data engineering teams and application development teams to work together, because an application development team may want to access a data set that is in the domain of the data engineering team. The data engineering team only has it in Kafka, or in Elasticsearch and the application developer who's building some data science application just wants to speak SQL. They don't want to write an Elasticsearch query.

Presto allows those two classes of teams, the application developers and the data engineering teams, to communicate, or to well, it allows the application developers to write applications against any piece of data engineering infrastructure with just SQL. Is that right?

[0:44:11.2] ZL: Yes, yes. This is one cool feature. I mean, for example in our case, one team they can write SQL directly on Elasticsearch, for example. The benefit of that is they do now need to copy the data and transform the data from Elasticsearch and dumped into HDFS, which will cost reasons and cause a big latency.

I mean, the data if you copy every day and make cost at least tens of minutes or hours to copy the data. Presto brings the gap that you can shoot SQL on anything. A SQL understand by everyone, so everyone can shoot SQL, run analytics query on any storage. That is a power for this cool feature, the Presto connector.
Other thing is since we have this Hadoop data lake, so most of our information is in the Hadoop data lake, but we still have data scattered around company in so many other places; Cassandra, Elasticsearch, MySQL. It's also provided a possibility that you can join what Hadoop data table with another Elasticsearch table. You can join them based on the [inaudible 0:45:16.5] or something. I mean, this is fantastic. I don’t know, any other can do these. Yeah, we have many use cases using this feature.

[0:45:26.2] JM: Well, that's incredible. You can do cross-database joins, or you could join a HDFS cluster with a Elasticsearch cluster?

[0:45:38.1] ZL: Yes, yes. HDFS table with Elasticsearch table. Since in Presto, at first you'll have the catalog. Catalog means, is a name for the wire, your data source resides in. It could be either HDFS, or Cassandra, MySQL, Elasticsearch. Now you have the database name, now you have the table name. I mean, everything is just a table, just resides in a different catalog, another data source. You can do draw on anything, group by anything in SQL. You can do that for other storage.

[0:46:10.5] JM: How does a Presto MySQL query know how to turn into the queries for those specific domains? If I issue a query that's joint – a SQL query that's going to join a set of Parquet file stored in HDFS with an Elasticsearch cluster, what's going on there? How does the presto query engine figure out how to do that?

[0:46:37.1] ZL: Yeah. I mean, why are specifying a table name. Now you need to say if the data is residing in MySQL, you'll need to refer that as MySQL thought, database store table one. Then join hive, they have a store table two. Now you can join these two tables. Use the catalogs, since you have the MySQL as the first part is now, "Oh, I'm looking for MySQL table. Then internal actually, Presto has a MySQL connector, which is after simple just use a JDBC to connecting to MySQL, not streaming the data from MySQL into Presto engine. The same, I mean, for hive the other side, Presto will scan HDFS files and that's streaming the result back. Then Presto engine will do the join inside the Presto member.

[0:47:22.4] JM: What is required to write one of these connectors to connect the Presto query engine to the Elasticsearch, for example, or to the Parquet files in HDFS? I know you said JDBC, but how complicated is it to write one of those connectors?
[0:47:41.1] ZL: I think the Presto connector interface is very clean. If you read the Presto code, you will allow it. The code is really good, but it is some work. For me, I developed a Elasticsearch connector. It cost me the whole – I mean, from development and then testing, stress testing and then in production cost me a few months, two or three months to develop the whole thing.

It will cost some time, but MySQL may be easier since MySQL, the schema almost the same, just have which JDBC that is easier before something is a long SQL key value thing, for example the Cassandra one, or Kafka one is a not traditional SQL start, then you need to do

some transformation, place a module there so we will cause some time, but not – I mean, it's a reasonable time. Yeah.

[0:48:30.7] JM: I think Presto is such a cool example of open source software, because this is essentially a shared resource for the Presto itself, the code is a shared resource between Facebook, Netflix, Uber, and three companies at least, I'm sure there are other companies using Presto, are able to leverage each other's contributions to this project. I mean, I'm sure that's something that's you're aware of, but I think that's a pretty incredible development that these companies are able to share infrastructure basically.

[0:49:02.3] ZL: Yeah, yeah. I think that's a remarkable thing. I mean, I always feel thankful to the Facebook guys who created Presto. Really awesome engineers. They create a very cool thing, and worry on something. I mean, when I was in Netflix, I started looking at that and then bring that into production, and then started contributing to it. Now is kind of a more and more companies. I mean, besides Facebook, Uber, Netflix, also as far as I know, LinkedIn, Twitter, Pinterest, Airbnb is almost all of them are using Presto. It's a bigger and bigger community. In 2014 when the first Presto meetup happened and just maybe 20 to 50 people attending there, but in recent years there's more than 100 people.

[0:49:54.3] JM: That's cool.

[0:49:56.1] ZL: Yeah. So many companies and so many engineers are jumping into this community, so very nice.

[0:50:04.5] JM: We did do a show on Presto in the past and that show covers some of the query lifecycle of Presto. I'm sure a lot has changed since then, because we did that show like two and a half years ago, or so. What are the developments in Presto? Because it sounds like this is a well-defined project, right? You want to be able to issue SQL and have that SQL query execute against disparate data sources potentially joining them together. It's a well-defined problem set, but it's really hard to optimize. What are the areas that are under a lot of research and development within the Presto community?

[0:50:41.9] ZL: Yeah, I think that Presto is, we can say that it's a typical – just a typical SQL engine, but it's default big data and so many things. The number one thing I mean, so many companies are loving Presto is because of Presto performance. It's so fast. If you have a user just using hive, previously if they shoot their Presto query, they will love it. It's much faster. That is number one reason why so many companies are using that.

Internally is a number of optimizations. Presto will be number one is the in-memory columnar processing. I mean, after reading the data from Parquet ORC, internally Presto is storing the values in-memory. The engine also processing the values converting, not as row by row. If you are doing row by row, you are wasting CPU time, or the memory copy to preserve some unnecessary data. Presto internally, it has the internal data block inside Presto engine is also converting.

Also that is encoded. We have different encoding and so many codings for that that also make it fast. My other thing is that Presto also do the – is written in Java, but Presto also doing the batch code generation. I think that is a very cool feature, making it, I would say is almost fastest engine to really write all the SQL operators into the Java [inaudible 0:52:06.8], including the rewrite follow-up into I rewrite the branches, I rewrite the content zone and even farther.

I mean, so many drama internal thing to using ASM to rewrite that. That is also one very cool organization, but few other engineers can do. The other things are, I mean, coming other engines as well, since Presto doing a pool-based model it has the push backs. The push of a model, like if you are doing something, you'll see some column, maybe five. In hive the MapReduce push-based model, the bottom layer have to scan all the files and then push the result to the upper layer of the search.

In Presto, it's pool-based. The top layers I just asked for the file results, if it's a limit five, as long as my bottom layer gave me five results, my query stop. That is one other model how cool Presto is much faster. Yeah, and let me see. Also, the one of the feature for Presto is besides reading from HDFS, and now we're touch disk, all the processing are in-memory, so that is in nature faster than phase engines.

[0:53:17.8] JM: Uber built a Parquet reader to read Parquet files in a custom fashion, as an example for how Presto works. Explain where this Parquet reader fits into the infrastructure, why you built it and how you got a boost in query speed from that reader?

[0:53:37.5] ZL: Sure. Yeah. I mean, since Uber decided to use Parquet for our, almost all of our raw data, so we try to play Presto and Parquet together. Due to the Uber-specific use case, I mean, our data is highly nested and our query mostly, we have a [inaudible 0:53:55.1], so many of these things and usually Presto and Parquet is not fast enough, so we try to optimize that.

The area we optimize in is number one is that Parquet developed originally for the MapReduce use cases. Parquet library is supporting the nest key as well with a no based. It's quite interesting. I really like how Presto, how the data reading from Parquet. It says like, the Parquet is storing the data in columnar, but the Parquet API provide the data row by row. Then Presto try to convert that node into columns again, so that Presto columnar engine can process that. There's a two times memory copy, which is unnecessary.

We are rewriting a new, we called a new Parquet reader, which we directly reading from the Parquet all the values in columns, that we directly transform that into a personal internal columnar blocks. This saves all this unnecessary memory copy. One other thing we are optimizing is also since Parquet also keeping the stacks, the min/max count, some of these are stack for the file. We can do practically push that.

If you are looking for, let's say wire column greater than 10. We first read the stats saying the max value of this column, I just let's say 5, then we can skip reading the whole thing. Since there's a no possibility that you can find other value greater than that, since the max is just 5. We are doing predicate push down. Further than that, we are doing [inaudible 0:55:30.7]. Parquet internally had one-page storing all the different values. If we just read the min/max and we find that, oh we have to scan this file, that will read the dictionary again to see whether if you are looking for let’s say the column equal to be 8.

We see oh, the max value is 10, but other possible values are 2, 3, 5, 9. Know at there that we still skip reading the decile of Parquet as well. One other thing we did is for the nested in nature since initially I mean, the Presto engine doesn't desire to process the nested thing in nature, so

we are doing many organizations ,including the nested column we need to push down the projection, need to push down, also the nested predicates we try to push down that at a while. That's so many things together, and finally we have a modify as proposed implement using this new Parquet reader for Presto.

[0:56:26.0] JM: Okay. We return to this question of schema. You've got the schemaless core transactional data, you've got data schema that is maintained over your HDFS infrastructure via that hive meta store, you've also got Presto meta store, which maintains the schema of the Presto databases, or the Presto data sources. Does the meta store of Presto update automatically? How reliable is that Presto meta store?

[0:56:59.9] ZL: Actually, Presto is sharing the same meta store as hive. It's one thing. Actually just the hive meta store. Presto using that to – so there's a just one source of metadata that is a hive meta store. Presto using that as a metadata for that.

[0:57:14.5] JM: Does Presto need to have an awareness of the metadata in the other data sources that it might access?

[0:57:21.3] ZL: Oh, good question. Very good question. The presto connector interface has a metadata interface, which is where we implement the metadata there. Yeah, for example the Presto hive connector, the Hadoop connector, we just use the hive meta store, gave us everything, but for example Elasticsearch. Actually we are sending Elasticsearch my ping request for the index that we got out of fields. Then we massage that a little bit, then return to the user. That is a schema. It's we implement in that and based on a different storing solutions.

[0:57:55.9] JM: The connector, in writing the connector ,you are writing and what needs to be –

[0:58:00.9] ZL: Yeah, you need to organize, implementing all of that.
[0:58:03.5] JM: Okay. The Presto meta store will get consistently updated with all of the things that it has a connector to?

[0:58:10.8] ZL: Yes, yes. In hive Hadoop connector, you are relying on the hive meta store. Elasticsearch, you rely on the market request. In MySQL, also hive you can rent this square table, you got everything there, so it's connector-based.

[0:58:25.9] JM: That's awesome. Okay, nearing the end of our time, this has been great, you've seen data infrastructure at Netflix, Facebook and Uber. How do they compare? How do these companies compare?

[0:58:41.1] ZL: Okay. Yeah, as far as I'm not employed by Netflix or Facebook anymore, but I think Facebook is that data unify is very awesome. I mean, one that was the year, the under name culture is very awesome, and all the engineers there are really hard working, so very talented. Facebook, even several years ago, the year they thought unifier has so many cool features, I mean, even now I think very few companies in the Silicon Valley can have the same features as Facebook. I mean, based on scalability and also the functionality. I mean, Facebook the data you have both HDFS, Presto, hive and now they are running Spark. Everything is awesome. I think that's a really awesome team and also copy.

Yeah, Netflix is a very special case, I think. Netflix is everything on AWS, so it's very interesting is you can imagine a company of that size, that scale, they just store everything on S3, but they have their own, either they buy some license from AWS, or they build something their own like Presto to try to query from S3. I mean, Netflix is I think is a very special use case, unlike all the other companies. They believe [inaudible 0:59:59.6], truly believe in that. They pay you money and supply some storage about those cloud, but they are building their own customizing and they make it working perfect. I think that's a very interesting story there.

For Uber, I think we are still early. Yeah. I mean, the Uber we have the potential to grow even bigger, now we are growing very fast. Then for our, the big data stack is started just less than three years ago. I mean, our development speed is really fast, which achieve even more things than the other company, I think, but we have so many years to think of problems. As long as the scale keeps improving, keep increasing. I mean, yeah, our software improving. We have so many things to learn from open source, also the other companies as well.

[1:00:52.2] JM: Zhenxiao, thank you so much for coming on Software Engineering Daily. It's been great talking to you.

[1:00:56.0] ZL: Thank You, Jeffery. Nice chatting with you.

[END OF INTERVIEW]

[1:01:01.0] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you're interested.

With 23,000 people listening Monday through Friday and the content being fairly selective for a technical listener, Software Engineering Daily is a great way to reach top engineers. I know that the listeners of Software Engineering Daily are great engineers, because I talk to them all the time. I hear from CTOs, CEOs, Directors of engineering who listen to the show regularly. I also hear about many newer, hungry software engineers who are looking to level up quickly and prove themselves.

To find out more about sponsoring the show, you can send me an e-mail or tell your marketing director to send me an e-mail jeff@softwareengineeringdaily.com. If you're a listener to the show, thank you so much for supporting it through your audienceship. That is quite enough, but if you're interested in taking your support of the show to the next level, then look at sponsoring the show through your company.

Send me an e-mail at jeff@softwareengineeringdaily.com. Thank you.

[END]