**EPISODE 03**

[INTRODUCTION]

**[0:00:00.3] JM:** LinkedIn is a social network with petabytes of data. In order to store that data, LinkedIn distributes and replicates that data across a large cluster of machines running the Hadoop distributed file system. In order to run calculations across its large data set, LinkedIn needs to split the computation up using MapReduce-style jobs. LinkedIn has been developing its data infrastructure since the early days of the Hadoop ecosystem.

LinkedIn started using Hadoop in 2008. In the last 11 years, the company has adopted streaming frameworks, distributed databases and newer execution runtimes, like Apache Spark. With the popularization of machine learning, there are more applications for data engineering than ever before, but the tooling around data engineering means that it is still hard for developers to find datasets and clean their data and build reliable models.

Carl Steinbach is an engineer at LinkedIn working on tools for data engineering. In today's episode, Carl discusses the data platform inside LinkedIn and the strategies that the company has developed around storing and computing large amounts of data.

Full disclosure, LinkedIn is a sponsor of Software Engineering Daily.

[INTERVIEW]

**[0:01:25.6] JM:** Carl Steinbach, welcome to Software Engineering Daily.

**[0:01:27.6] CS:** Thank you very much. It's great to be here.

**[0:01:29.3] JM:** LinkedIn started using Hadoop back in 2008. This was near the beginning of the age of modern data infrastructure. Give me a brief history of how data infrastructure at LinkedIn has evolved.

**[0:01:44.9] CS:** Yeah, well let's start with the introduction of Hadoop back in 2008. LinkedIn had a big problem at that point in time. The site had a feature called People You May Know, which suggested to members here's someone that you should consider connecting with. It was very clear based on metrics that this was really the engine of growth for LinkedIn. At that point in time when they first introduced this feature, they were using Oracle to actually build the index for People You May Know, using a process called triangle closing. If I know you and you know Steven sitting over here, chances are I also may know Steven, so that would be a good suggestion in my feed.

When the feature was first introduced, it would take a day to build this index on Oracle. It was such a successful feature it drove so much growth towards the site that pretty soon it was taking upwards of a week to complete a single run. At that point, the results were becoming increasingly stale. Scaling Oracle wasn't really an option, so people started looking around for a better approach, something that could be horizontally scaled.

When I joined LinkedIn in 2013, one of the first things I did was to go to Jira and search for the first mention of Hadoop that I could find. I found a couple of tickets, where people are discussing how to set up the first cluster, they're discussing the first use case, which was PYMK. Incidentally, Jay Kreps was heavily involved in this and he's the guy who later went on to start the Kafka project.

The application of Hadoop to the PYMK problem was a huge success. They were able to bring the time required to generate this index down to just a couple of hours. They found that by adding more machines, they could bring it down even lower. That first big success then inspired other people to start leveraging Hadoop for other problems.

Pretty soon, it got to a point where within the company, it was just known that Hadoop was where all of the data was stored, all of the tracking data, all the derive data sets and things like that. We started using it to build search indexes, we started using it for future generation and model training for machine learning and for analytics as well. By the time I arrived, Hadoop had really cemented its position within LinkedIn's broader data infrastructure.

 

One interesting thing about LinkedIn is there's this division between online serving systems, near-line streaming systems and offline batch analytic systems. Hadoop was basically the offline bash analytics solution at LinkedIn.

**[0:04:07.9] JM:** When you joined, was there a standardized process by which somebody stood up a Hadoop cluster, or was given access to Hadoop resources, or were people just spinning up clusters however they could figure out a way to do so?

**[0:04:23.2] CS:** Yes. We actually, or I shouldn't say we, but the people at LinkedIn at that time recruited folks from Yahoo. This would be around the 2008-2009 timeframe, at a time when there were no Hadoop vendors out there that you could consult. I think Tom White published the first Hadoop book through O'Reilly, maybe around 2010. the only place that you could go for help about how to deploy a large Hadoop cluster, how to manage it would be the Apache mailing list, or meetups related to this.

I think the easiest thing then for people at LinkedIn was to recruit some of the people from Yahoo who had been working on Hadoop. That's how the original Hadoop operations team was built out. We then inherited the model that Yahoo had been following for running these clusters; build big clusters, as opposed to many smaller clusters that simplifies management, because you have fewer things to manage, fewer services to manage at least.

It's also good from a data standpoint, because you can just go to one cluster and know that all of your data is there, as opposed to having to think okay, this data set may only exist on this small cluster. That also causes other problems besides discoverability. It also would make it very hard to join datasets together, which are located on different clusters. You wouldn't be able to leverage data locality anymore, you'd get slowed down by the network and things like that. The model at LinkedIn has always been fewer, larger clusters over many smaller clusters.

**[0:05:47.1] JM:** Was there a distinct point at which it went from being a single large cluster to that single large cluster got so big that you had to instantiate additional clusters?

**[0:06:00.1] CS:** I don't know the exact timeline for when a second or third cluster was introduced, but by the time I joined LinkedIn in 2013, it was well established that we had one

large development cluster and one large production cluster. There was a process then for getting your job promoted from development over to production. Since at the time, it was hard to isolate one job from another job, it was important to make sure that the jobs that we're running on the production cluster were well-behaved and had been vetted.

Later on though, that process of vetting individual flows actually became a major problem. People would sometimes wait for up to a month for someone to sit down and review their job. In some cases they were just told okay, well you have to go and fix this and then get back in the queue and wait another month.

One of the things that I did after joining LinkedIn was to try to figure out if we could automate that process. That resulted in Dr. Elephant, which is a service that we run which looks at the exhaust basically from Hadoop jobs. It applies heuristics to those logs and uses those heuristics to diagnose performance pathologies, things like skew in terms of tasks, too much memory, too little memory and stuff like that.

Very quickly, we got to a point where we could take the human reviewer completely out of the loop and we were able to I would say, promote to production probably 80% of things, as well as for the cases where we couldn't promote immediately, we were able to offer actionable advice to the owner of the workflow. These are the things that you need to change in order to get a green signal from Dr. Elephant.

**[0:07:30.5] JM:** LinkedIn was not alone in this problem, even before the open source Hadoop ecosystem. This problem manifested at Google. There's an interview I did a while ago with a guy named Tomasz Tunguz, who wrote a book called *Winning with Data*. I have alluded to this book a couple times, because he talks about what he termed 'data breadlines'.

Basically, the skew that you're referring to where you have to get into some cue either to get your job to run, or to have the data scientists go and write a custom Hadoop job just for you to get the data, to get the nightly report back to you. This was a perennial problem for people building data infrastructure, people who were working as data analysts or data scientists in those nascent days of Hadoop infrastructure.

**[0:08:20.2] CS:** Yeah, that sounds very familiar. I think it also points to a larger issue, which we created Dr. Elephant to help address, which is this inherent tension between developer productivity and infrastructure efficiency. We don't want to require that everyone who uses Hadoop, or Spark needs to be an expert in these systems. They have better things to worry about, right?

They are machine learning experts, or they're analytics experts. To require that they spend a month, or maybe even a year coming up to speed with all of the intricacies of these systems would severely impact their productivity.

Another interesting example of this concept of worse is better, Richard Gabriel wrote this really interesting essay in the late 80s and I think it became well-known in the early 90s, describing this concept of worse is better. He was trying to explain why Lisp had failed in the market, whereas C and C++ were doing really well. He identified what he described as the MIT School and the New Jersey School, where New Jersey was a stand-in for Bell Labs. He compared the MIT School, where everything has to be in a sense perfect, right?

The API should be simple and it's acceptable to push complexity over to the implementation in order to keep the API simple, versus the other approach of saying, "Well actually, we're willing to push responsibilities over to the user for the sake of keeping the implementation very simple."

I think Hadoop is a really good example of the worse is better philosophy in action. A benefit of worse is better is that you're able to get this thing out very quickly. You're able to iterate on it and make it better over time, to a point where it's 95% of what the right thing would have been had you gone for that initially. In the meantime, it spreads like a virus. I think looking at Hadoop, it's a prime example of that.

**[0:10:13.1] JM:** When did the Hadoop infrastructure problems at LinkedIn get alleviated to the point where it was much easier for people to actually just get their jobs, write their jobs in an autonomous fashion without being blocked by – was Dr. Elephant the thing that just solved this problem, or was it more of a progression of additional solutions that alleviated this issue?

**[0:10:35.5] CS:** I think that Dr. Elephant helped us solve this tension right, between personal productivity and infrastructure efficiency. There were other things that it also helped people become more productive. I mean, when Hadoop was first introduced, your programming API was MapReduce, which is basically assembly language for data processing.

**[0:10:53.0] JM:** People were handwriting assembly, essentially?

**[0:10:55.5] CS:** Essentially, right. Or you could also liken it to – I don't have a database, so instead, I have to write the query plan by hand. If I want to optimize it, I have to go and retrieve the statistics by self. It wasn't very productive for people who were coming from more of a database background, or people who wanted to do ad hoc queries, right? If you are writing code and MapReduce, it means you have to compile your code, you have to deploy the JAR files, all of that stuff.

While Hadoop was still incubating at Yahoo, they introduced Pig, which was a new take on SQL. Pig is often described as an imperative language, but that's not true at all. It really is a declarative language with some nice escape hatches for you to insert imperative logic through UDFs and things like that. That helped to provide people with higher level abstractions that they could reason about.

Similarly, at Facebook, they had introduced Hadoop, HDFS and MapReduce and very quickly realized that they needed a higher level programming interface for the majority of people at Facebook, and that's where Hive came from.

It's interesting, right, to consider why did Facebook produce Hive and Yahoo produced Pig. I know for a fact that a lot of the people who worked on Hive at Facebook were ex-Oracle engineers. They I think definitely looked at Hadoop and thought, “Well, what we really want here is the interface that a database provides, but the scalability that Hadoop provides.” I think the people who wrote Pig were coming at it from more a scripting language approach. That's why at least superficially, Pig looks a lot more like an imperative scripting language than it does declarative language like SQL.

 

**[0:12:34.6] JM:** If we take the lineage of query interfaces forward a little bit further, I think that lineage goes from Pig to Hive. Then the next thing along that lineage might arguably be Presto would you say, or is there – I mean, when you think about these higher-level interfaces for querying large datasets in Hadoop, is Presto the next thing in that series, that lineage?

**[0:12:59.7] CS:** Well, I think in a way the lineage of Presto is a little bit different. Presto is very much an MPP database running on top of HDFS, or that's able to access data in HDFS. Comparing it to something like Vertica or Greenplum, I think it's ancestors nice in a way. Whereas something –

**[0:13:18.5] JM:** MPP is massively parallel process?

**[0:13:20.4] CS:** Correct. Yeah. I think, a big distinction to make between Presto and things like Spark, Hive and Pig is with those latter systems, the job that a user triggers runs using their ID. Whether they're able to read data on HDFS, or from a blob store depends on whether they using their account have access to that data. The code basically runs with their privileges, whereas in a system like Presto, the whole service runs as the Presto user is understood that that Presto user has access to all of the underlying data. Then Presto is able to superimpose its own authorization rules on top of that.

One implication of this though is that if you need to write your own user-defined function, you can't just add that to Presto, because that's a code injection vector. I could very easily sneak some code in and use it to access data that I'm not supposed to be able to have access to. When people do add UDFs to Presto, they need to be vetted to make sure that they're okay. Whereas with the Hadoop and Spark model, since everything runs as the user who's actually triggering that job and inherits their permissions, you can allow them to run whatever code they want, inject whatever code they want. In terms of velocity and iteration speed, it's a lot faster in those other systems.

If however you're coming at this from a database background and you're comfortable using SQL, you can solve your problem using SQL in a standard built-in UDFs, then there's no reason not to use Presto.

**[0:14:52.2] JM:** I think I probably jumped the gun here in the evolution of Hadoop usage. Early days of Hadoop usage, you've got an HDFS cluster, the Hadoop distributed file system; that's the place where you're storing all your data, the “data lake.” That HDFS usage is still pretty prevalent today. People are still largely running HDFS clusters, although the large distributed bucket storage systems have displaced some of that HDFS usage if somebody is on AWS, S3 or Azure, blob storage.

The query layer has changed over time, so whether we're talking about Hadoop MapReduce being abstracted into Pig or Hive, or being displaced by something like Apache Spark and people pulling – if I understand the usage of Apache Spark, it's people will pull distributed working sets into a distributed memory system and then they'll query those working sets that are in-memory now, so you can have this ad hoc data science process that's a little bit more of an interactive workflow than the batch Hadoop workflow. My sense is that the Spark usage, that really changed how people saw big data processing. Can you give me your perspective for how Spark changed the data landscape?

**[0:16:17.7] CS:** Yeah. For starters, Spark is more efficient by caching data in-memory, as opposed to materializing it to disk between every stage like MapReduce does. Before I go into details, I'd like to also just say that as soon as MapReduce was released as an open source project and my guess would be even before it was released as an open source project, the people working on it knew about all of these tricks.

They knew that forced materialization between map and view stages and different jobs was a performance problem, but they erred on the side of making sure that they could recover if jobs failed, right? Because they were optimizing for these very long-running jobs.

**[0:17:00.1] JM:** By the way, just to clarify what you're talking about here, this is the fact that in a MapReduce, you often are doing these three operations; map, shuffle and reduce and you're checkpointing the data at each of these operations, which makes it a costly series of operations.

**[0:17:14.3] CS:** Right. If you have a really long-running map task, it makes sense to checkpoint before you hit reduce, right? You don't want one of those transfers to fail and then you have to go and repeat that work. If you're talking about, let's say a fast query, right, which looks at a little

bit of data, you're going to find that that forced materialization actually accounts for most of the runtime, or it's something that slows you down definitely at every stage.

Another issue was that I think everyone understood immediately that forcing people, rather only allowing one reduced stage per map job was an issue. If you wanted to group by multiple keys for example, you would have to chain together multiple MapReduce jobs, even though you really didn't need those map stages in between.

What people really wanted was an MR star model, where you could have multiple reduced stages per map stage. I think paradoxically though, since Hadoop and MapReduce were such a success right off the bat, it meant that the project instantaneously had a very large community, both of users, as well as developers. Finding consensus within that community became a much bigger problem. While everyone, I think was united in understanding these are the next steps for the project, exactly how to accomplish that became a problem.

Similarly, users definitely validate a project, but they also make it very hard for you to evolve APIs. I think with APIs, you are always going to make a handful of mistakes. You want to eliminate later on, but the more users you have, the harder it is to do that. I think that the people who started the Spark project had the advantage of looking at Hadoop, recognizing the problems that Hadoop had and then starting from a clean slate, without the baggage of a large user base and a large community, they were able to iterate very quickly and produce something.

In some ways, I view Spark as Hadoop version 2. Another nice thing about Spark is that the APIs are very elegant, they're very clean, it's very easy to build application verticals on top of Spark in a manner that it isn't really possible with just MapReduce. Another thing that I think has really benefitted Spark, at least in terms of that project's ability to iterate quickly is the fact that it's still located in a single code repository and it's a single project that governs it, whereas with the Hadoop model, it started as a single project, but then you had projects spinning out of it.

Hive and Pig both started as Hadoop projects, but then they spun out into their own projects. What that really ended up doing was creating a problem that then vendors had to solve, right? All of these projects had their own release cycle.

 

They were not necessarily tested, or integration tested against the version that you would want. If you weren't using a vendor, your distribution, your first task was to figure out well, for this version of Hive, what version of Hadoop do I need? What version of Pig do I need? How does this all fit together? Because you could count on the fact that no one from these individual projects had done any integration testing for you.

That really was the value that vendors provided at least in the early days. They were just cleaning up this mess that had been created. I don't know really what the motivations for doing that were, but I think that Spark has avoided those problems just by keeping everything in a single repository.

**[0:20:33.3] JM:** Let's fast forward to today. You work on managing Hadoop infrastructure at LinkedIn. Is that your day-to-day job?

**[0:20:41.4] CS:** Yeah. I would say even though I don't really like this term big data infrastructure in general. Yeah.

**[0:20:45.8] JM:** Why don't you like that term?

**[0:20:47.0] CS:** Because it's a buzzword. People who I think use it too frequently. The sad thing is I can't think of a better term to use.

**[0:20:55.2] JM:** What does your day-to-day consist of?

**[0:20:56.6] CS:** These days, I'm focusing most of my time on a project called Dali, which we've been working on for several years now. Our goal with Dali is really to combine the best aspects of a relational database with the best aspects of existing big data ecosystem. We want to leverage abstractions from the database world, things like tables, views, the decoupling between neurological view of data and the underlying physical details, but combine these with the freedom and flexibility that people are used to in the big data ecosystem.

With big data, I have the option of using more than just SQL to analyze data. I have the ability to leverage different file formats, depending on which format provides the best support for the

engine that I'm using, or the best performance for the query that I'm running. Similarly, I'm able to take advantage of different storage layers and swap those out. In effect, we're trying to introduce a level of indirection at each one of these layers, but also provide the abstractions necessary to really decouple implementations from APIs.

With the goal, both of making things simpler for users so that they don't have to worry about physical details that they shouldn't really need to know about in the first place, but also with the goal of making it easier for the people who provide this infrastructure to make changes underneath without disrupting what's happening above.

**[0:22:21.7] JM:** When you say user, you're talking about an internal application developer at LinkedIn.

**[0:22:27.3] CS:** Correct.

**[0:22:28.8] JM:** Give an example of an application that lets say, maybe I'm building a dashboard, or I'm building some reporting system within LinkedIn. I'm sure you have a prototypical example in your head. Tell me the problems that Dali would solve for a prototypical application.

**[0:22:49.1] CS:** I think one thing it solves is the discoverability problem. We have a dataset catalog that's searchable. That's typically I think easier to discover things in that catalog, then it would be just looking at paths in HDFS. It also allows you to search by column names and annotations and things like that. Just the process of discovering datasets, as well as understanding who produces the dataset and what the contract is between you, the consumer and the producer is another thing that is aided there.

Another thing is that we are able to over time improve the performance of a dataset by looking at the queries that people are running against it and using those to inform how we partition the data set, the format that we encode the data set in and things like that. I think efficiency is one thing.

Another thing that Dali provides, which I don't think any other system at this point has yet provided is the ability for a data set owner to decouple the API of their data set. In other words, the schema of the data set from the actual implementation of the data sets. Over time, right, I can evolve the schema without requiring that consumers migrate in lockstep with me, because I have this level of indirection between the schema that they're consuming, which is provided by the view on top of the data set and the schema of the data that I'm actually materializing on HDFS.

Otherwise, right, when I make a change to the schema, it becomes instantaneously visible to everyone who's consuming it. I think, probably the best way of explaining this is to say that if you think of data sets as services, which have an API, this allows us to support multiple APIs on top of the same data set, in a manner similar to how a service can have a V1, V2 and V3 of the API that it presents to clients.

**[0:24:40.7] JM:** By the way, is it doli, or dolly? Like D-O-L-L-Y?

**[0:24:46.0] CS:** D-A-L-I.

**[0:24:46.7] JM:** D-A-L-I. Okay, like the artist.

**[0:24:48.6] CS:** Right. There's a history of using LI as a suffix on names of projects at LinkedIn.

**[0:24:54.3] JM:** I see.

**[0:24:55.0] CS:** Originally started as data access at LinkedIn, now it just doesn't stand for much of anything, because we've gone beyond just data access to other things, like data management, data catalog, data discovery, issues like that.

**[0:25:09.5] JM:** Okay. What was the motivation for starting this project?

**[0:25:12.9] CS:** Well, I think that one big motivation was simply to make things easier for people developing applications on top of Hadoop and spark by hiding details from them that they shouldn't need to worry about anyway. Things like what file format a dataset is using, or which

cluster a dataset is stored on, or how that data set is partitioned. Also, to give the people who are running that cluster and managing that data the ability to make changes behind the scenes without impacting them.

One interesting thing is that usually when you find a situation where the API is more complicated than it needs to be, or where developers have to manage more details than they should, it's because the people who are providing that API are slacking off, right? They're trying to make things easier for themselves. In this situation, it was bad for both sides. It makes it very hard for infrastructure providers to make any change without talking to every user who's going to be impacted and helping them to migrate. We really want to improve the velocity for both sides.

Then I think in terms of themes or inspirations, what we were really looking at was conventional database, where you have this nice separation between the view that someone writing a query has and the underlying details, right? If I'm using MySQL or PostgreSQL, it's irrelevant to me how MySQL and PostgreSQL encode a table, or how they write it out to disk. Since that's irrelevant to me, since I don't have insight into that, it makes it easy for PostgreSQL and MySQL to change those details without breaking any of my queries.

**[0:26:50.2] JM:** I see. That contrasts with today, because if I'm an application developer at LinkedIn and I'm consuming – let's say I'm consuming a data set and I'm building a dashboard out of it, maybe that data is in a Parquet file, maybe it's in some JSON, maybe it's sitting in a relational database. I have no idea about this. Is that the constraint that we're talking about here, like file formats basically?

**[0:27:19.8] CS:** File formats, but also how you actually organize the data set across different directories. We have a lot of data sets that are date partition, so we have a separate directory for each hour, or for each day. In some of these cases, it would make more sense to let's say, hash partition that data set based on member ID. That would lead to better query plans for the queries that we see being executed against it.

Since people basically have an expectation hard-coded into their logic about how data is organized, we're prevented from ever making that change. Similarly, if the address of a dataset is basically the fully qualified name of the cluster where it's stored, appended with the directory

that it's located in, you can never move that data set to another cluster, because everyone's script assumes that it's on cluster X as opposed to cluster Y.

By adding this additional level of indirection where we have instead a data set view of things, where there's a logical data set name and then we through a service, maintain the mapping from that name to a specific cluster, a specific directory and details about the file format, we're able to make changes without impacting users.

**[0:28:30.8] JM:** This is reminding me of – I had a couple shows about project called Vitess, which is like sharded MySQL. The problem with – historically with sharding MySQL is the client has to be aware of where the shards are. You don't want to be writing cluster addressing logic in your client, because then it puts a hard dependency on where that data set is physically in your data center. That's an anti-pattern. It's a tight coupling that you don't. You're describing that in a different context, but it's the same problem.

Presumably, if you were to implement Dali and you had this consistent access pattern across the average data set at LinkedIn, this would be backward compatible. All the client logic that has been written that is cluster specific, you're still going to have to leave those data sets in those clusters, right? Going forward, people will be able to start addressing those clusters with the new logic, right?

**[0:29:38.3] CS:** Exactly. As we migrate to Azure over the next couple of years, we're enforcing a policy where all public data must only be accessible using Dali APIs, just to make sure that we don't find ourselves in the same problem on Azure.

**[0:29:52.2] JM:** Is Dali – have you gotten it to a point where you can feel comfortable enforcing that?

**[0:29:57.7] CS:** Yeah. We've been there for a couple of years actually.

**[0:30:00.0] JM:** Great.

**[0:30:01.0] CS:** The main problem up to this point has just been the cost of asking people to migrate to it. Another interesting problem there is that many of the benefits of Dali, for example being able to use more efficient file formats, or more efficient partitioning schemes are things that we can only start providing once people migrate. Just one of these unfortunate situations, where we have to ask people to do work upfront before we can start delivering benefits to them that are visible, as opposed to delivering the benefits immediately.

**[0:30:31.4] JM:** This is the same conversation I just had with Nacho in our previous chat about updating Kafka client logic, because he works on the Kafka platform team. They can make all the updates to Kafka client logic that they want, it doesn't matter if people don't actually make those updates to the Kafka client libraries that they're using.

I think this is just a perennial problem for platform engineering teams, where you improve your infrastructure and then you need to go evangelize it throughout the company and get people to actually adopt this stuff.

**[0:31:04.1] CS:** Yeah. I think that that's been a huge problem, especially in these client service models. I think my advice to someone building the next platform would be to think about this problem seriously and come up with a solution where you have an approach to actually managing client-side dependencies in a more controlled fashion.

One way that we've started to do that for Hadoop and Spark is to basically control the image that's running on the VMs, or gateway machines where people are accessing the cluster from and providing client dependencies as system provided dependencies. As long as we don't change the API, but just the implementation, we're able to do that in a transparent fashion. For someone who's trying to access the cluster on a machine that they control, it's up to them really which client version they're using.

I think another approach is to try to make clients as thin as possible and try to push as much of the implementation over to the service side. One thing related to Dali, the Dalis was related to that approach and which relates to HDFS as well, if you know, I could go and redesign HDFS, or go back in time and talk to the people who built HDFS. I always stressed the importance of providing a dataset API, instead of a file API.

Our biggest problem with scaling HDFS has been the file system itself, basically that hierarchy of directories and files. There's a well-known locking issue in HDFS in the name node, where when you want to move a directory or a file from one location to another, there's basically a monolithic write-lock that locks the entire namespace and everything grinds to a halt. A lot of applications built on top of HDFS rely on this atomic move functionality in order to commit changes and make them appear in an atomic fashion.

This is something that you can also – you can provide the same functionality if you just move up one level of abstraction and start thinking about things in terms of datasets and partitions, instead of directories and files.

Another I think interesting implication of the file system-based approach is that you're stuck then with file system type of authorization and ackles. By which I mean, that I can only give you permission to read all of a file, or none of the file. I can't say, for example that you're allowed to read certain sequences of bytes in the file but not others. If you imagine that that file maps to a data set, right, consisting of records and columns, it stands to reason that some people you may want to give access to only a subset of those columns. You can't actually enforce that using file system level permissions.

Rather than exposing that on the data nodes side, if I could go and rewrite things from scratch, I would want to provide a record-oriented API instead. Where instead of reading bytes from a data node, I'm reading records and each record has a fixed schema and set of columns. Then I would be able to actually apply column level and potentially even row level access control policies on the data node side in a secure manner.

**[0:34:17.2] JM:** You mentioned that there is this push to go towards Azure. Do you have plans to start using cloud bucket storage as your data lake, instead of HDFS?

**[0:34:32.1] CS:** Yeah, that is our plan right now. Azure provides Azure blob storage, ABS, as well as Azure data lake storage, ADLS, which is actually a layer built on top of ABS. ADLS supports existing HDFS client APIs. It has a file system layout and supports basically a file system type view of data storage.

For user home directories, we're planning to use ADLS to manage those. For all public data sets, data sets that are shared between people or teams at LinkedIn, we're planning to store those on ABS and use our catalog basically to address those buckets, so that people won't know exactly which bucket data is stored in. Instead, our catalog will provide the mapping from a data set name to the correct bucket and format.

**[0:35:23.7] JM:** Are there any particularly difficult parts of that migration that you anticipate?

**[0:35:31.0] CS:** I think that there's going to be a shift in focus for us. We're no longer going to worry about the implementation details of the storage layer, because we actually don't have access to those. It's now service provided to us. Certain issues, like latency become much more of a problem.

In general, latency using ABS we expect to be higher and then we're able to achieve on-prem using HDFS. We also know that there's more variability in terms of the latency that you can expect. Monitoring that from the client side, seeing what clients observe is going to be very important. Understanding that one abusive client can actually cause all of your clients to be throttled as another problem that we're going to have to look into. That's an issue for us right now on HDFS, where it's possible for an abusive client to overload the name node. Being able to detect those issues early on, figure out which job is doing it and address that issue as quickly as possible is something that we're going to have to focus on.

**[0:36:33.1] JM:** Let's come back to the application developer, somebody building a data application on top of LinkedIn data infrastructure today. Help me understand their user experience. What are the libraries that they're frequently using? How low level is the infrastructure that they're interfacing with? To what degree is it abstracted away from them? Just help me understand the life of a data analyst, or a data scientist, somebody building a data application on top of this quota big data infrastructure.

**[0:37:07.1] CS:** Right. I think that things have gotten basically as high-level as possible. We're seeing a major shift to Spark from Pig and Hive and MapReduce at this point. If you're implementing, let's say a new application or a new workflow, we see people for the most part

using Spark, unless there's at least within their team an established convention of using one of these older tools.

We also see people, like for all new data sets that are being produced, there is a policy in place that they need to use Dali to access those data sets, so they no longer have to worry about a file path and format and things like that. They do need to use our record reader and writer libraries, but we have support for Spark, Pig and other libraries. Then if it's a scheduled workflow, they would typically deploy that application to Azkaban, which is our workflow management system and monitor it using different services that we provide.

One service is called WIMD, for where is my data. That helps people to understand the connections between data sets in terms of lineage, and also who to contact if an SLA is missed. Then from there, typically data once on HDFS, is exported back to nearline and online systems using one of several different transport mechanisms.

**[0:38:23.9] JM:** If I'm a data engineer and I want to use Spark infrastructure, am I provisioning my own Spark cluster? Are there Spark closers that are already built for me? Is there some standard process for spinning up Spark infrastructure?

**[0:38:38.5] CS:** For on-prem Spark users, people are running Spark on the large clusters that we operate, using yarn as the resource management layer. I think one advantage of that approach is that you're able to achieve much better overall utilization of cluster resources than you would if people were spinning up their own individual clusters, which for the most part you would expect to be idle majority of the time.

This way, we're able to achieve utilization rates of 90% on average. Anecdotally, I've heard from other people at other companies that when people use smaller clusters or a cluster per job model, they frequently see utilization in the 20% to 30% range.

**[0:39:22.5] JM:** Is there a lot of back and forth between Kafka and HDFS? Are people using Kafka as an intermediary buffer for data that they're going to eventually write to HDFS, or are these systems pretty disjoint?

**[0:39:38.7] CS:** No, they're very tightly connected, at least in terms of how they're used at LinkedIn. Anything that happens on the site will generate an event. That event is published to Kafka. From Kafka, it's eventually ingested into HDFS. There's a very tight relationship between the schema that an event uses and the dataset schema that it eventually appears in on HDFS. If you want to analyze what's happening on the site, over time you go to HDFS and look at those corresponding datasets.

**[0:40:09.4] JM:** All of these events are stored in HDFS, or some subset, right? You're not storing every event.

**[0:40:15.7] CS:** The best of my knowledge, every event.

**[0:40:17.0] JM:** Really?

**[0:40:17.7] CS:** Yes.

**[0:40:18.3] JM:** Okay. Any numbers on that? Do you have any idea how much data there is that's being –

**[0:40:23.7] CS:** We have thousands of these event topics. Some of them are created as one-offs, or are effectively no longer used past a certain point in time. In terms of let's say, number of events, or total size, I don't think I can really go into many details.

**[0:40:39.1] JM:** No problem. No problem. Are those old – the events that you're saving, are those mostly for recovery purposes, for auditability purposes? Are there reasons why maybe if I'm some random data analyst at LinkedIn, I would want to go into archival events, or was – What I imagine is that these events are by and large getting materialized into more approachable data sets.

They're getting materialized into databases and maybe Parkquet files that are easier to consume, rather than just being used in their raw event format. I guess, what I'm asking is what is the purpose of storing all of these archival events?

**[0:41:25.2] CS:** Well, first of all, I should clarify that we're not storing individual events as individual files. We are aggregating these events into large Avro files. We're using an ingestion framework called Goblin, which is an open source project, Apache project.

We also have plans to start ingesting directly to ORC, which is another columnar file format similar to Parquet. The fact is we wouldn't be able to fit these datasets all into a database. HDFS is really our only option. It's also our only cost-effective option. We could write them to tape of course, but then they wouldn't be accessible.

To answer your question about who makes use of these things, when you are let's say training a machine learning model, you want to be able to go fairly far back in time, as far back as you can go, in order to use that historical information to train the model. We actually see for machine learning use cases, the initial feature generation and model training is done offline on Hadoop, or Spark.

At some point, that model will be deployed to production and used to influence what's happening on the site. The iterative process of improving that model, developing new features, all of that happens offline, simply because that's where all the historical data is.

**[0:42:39.4] JM:** Okay. I think I'm seeing this in a little more color. Let's say, first of all, I want to constantly have a record that is the most popular profile on LinkedIn and by just the number of people who have viewed it. I want to store that in a database somewhere. I want to just have profiles by popularity and specifically the one at the top. Events over time of people viewing profiles on LinkedIn, those are going to be aggregated into the most popular profiles on LinkedIn. If I wanted to build a machine learning model for at what times of day do people visit certain profiles within LinkedIn?

It's entirely possible that the only way I can train that machine learning model is literally by going back into the raw event data. It's entirely possible that there is not some materialized view, some materialized database that gives me that answer. I might have to go into this archival, raw event data in order to train a model.

**[0:43:48.8] CS:** Yeah, exactly. I mean, the whole goal of machine learning is to predict what's going to happen in the future based on what's happened in the past. The more past you have to look at, the more effective your model is probably going to be. It's that whole notion, right, of hard problems become easy if you throw enough data at them.

Similarly, also, let's say that we could in theory have a database, a SQL database large enough to hold all of this information. I think if you go and talk to the people at LinkedIn who are doing machine learning, they'd quickly tell you that SQL would not be their preferred language, or interface for doing machine learning. Well, acknowledged that there are some academics who have actually promoted this approach. Since I'm a database person, I personally think it also has some appeal. I can understand why for someone who's used to using Python, or Scala, it's not a very attractive option.

**[0:44:41.3] JM:** At LinkedIn, have you standardized on any machine learning frameworks?

**[0:44:45.9] CS:** I think that there is an – there's an effort going on right now to standardize on machine learning frameworks. There's a larger initiative called Productive ML, or Pro ML. I think one of the things that that project seeks to address is the fact that over time, many different AI and machine learning groups at LinkedIn have developed their own frameworks for machine learning. There's a lot of heterogeneity in that environment right now. At some point, that heterogeneity becomes more of a problem than an asset.

I think that there was a decision that they did reached that point. They could also look at all the different approaches that have been taken up to that point and try to identify what are the things that we've learned and how can we apply that to building a platform that satisfies our needs in a better manner.

**[0:45:30.8] JM:** A quote from you, “Scaling data infrastructure is hard. Scaling human infrastructure is harder.” What do you mean by that?

**[0:45:37.9] CS:** When I go to a conference, people are always interested in asking how many nodes are in our cluster? How many clusters do we have? How much data do we store? Those

are interesting numbers, but the one that I pay much more attention to is how many people are using the platform at LinkedIn?

I think that coordinating interactions between people, helping people to discover datasets that they may be interested in looking at, helping people to understand these are joined conditions between this data set and this data set, that it will actually return results, all of these things become hard problems. The more data sets you have, the more people you have producing datasets and stuff like that.

To me, this is one of the unaddressed problems, or problem that people so far really haven't paid as much attention to, as how do we scale this system type approach. That to me is the thing that I've tried to focus on in my time at LinkedIn. Making sure also that people who come from many different backgrounds and have many different skill sets are able to be productive on the system.

**[0:46:43.4] JM:** Tell me more about that. What actionable insights have you had?

**[0:46:48.2] CS:** Well, in terms of keeping people, I think productive, one thing is to allow them to use the tools that they feel most productive with. We've made efforts not to restrict people to one query engine, or one query language. We are a poly-system, polyglot environment.

**[0:47:07.9] JM:** I'm just asking for more detail, because I've had a number of conversations with people recently about human scalability of systems. This is a tough topic of conversation, just because it's hard to even know what standard lessons we can extract from problems. You're talking about the human scalability problems of DevOps, like deploying microservices and doing on-call. You can talk about the human scalability problems of data infrastructure. I'm very curious about the common links between these things, or to what extent we can find some common links in making these systems more scalable in terms of the human dynamics.

**[0:47:47.3] CS:** Right. I think that one thing that we've definitely been doing is to draw lessons from software development and apply those to data engineering and data development. Requiring people to use version control and to create a clear link between the workflow that's running on Azkaban and where the source code for this workflow resides is one thing. At

LinkedIn, there's an expectation, right, that no one is going to work here for the rest of their life, which I think is very healthy, right?

**[0:48:14.8] JM:** Tour of duty.

**[0:48:16.0] CS:** Yeah, exactly.

**[0:48:16.7] JM:** I love it.

**[0:48:17.3] CS:** Tour of duty, right? That also means that a lot of people here inherit workflows from other people and need to maintain those over time. Understanding for example, where the code is and the revisions that it's gone through the history, I think is a very important thing. That's also really a form of institutional history and knowledge in how we maintain that. I might add though that I think we've learned about this the hard way when we were doing a migration from Hadoop 1 to Hadoop 2. Some of the public API has changed in very slight ways.

For example, a class became an interface. In Java, that creates an interesting situation where it's a source compatible change, but it's a binary incompatible change. You're in a position where you have to go and ask people to recompile their code after bumping their dependencies from Hadoop 1 to Hadoop 2, because the jars that they produced compiling against Hadoop 1, where this API was a class are incompatible running on a system, where the assistant provided jars think that that class is now an interface.

We found though that in the process of doing this migration, there were workflows running on Azkaban, which we're producing data that people were still consuming, but we could not actually find the source code that was used to produce that workflow. The way we handled it in that case was to actually write a tool called byte ray, which does bytecode inspection and manipulation. We were able to tweak the byte code in these jars to make it work with the new interface in Hadoop 2.

Subsequently, we started using that tool to scan workflows as they're uploaded to Azkaban, looking for dependencies that people shouldn't have, right? There's definitely a tendency for

people to talk to the person sitting next to them and figure out, “Oh, I think you need to add this dependency to make your workflow run.”

In some cases, that information isn't actually accurate. Over time, people would accrue these larger and larger set of dependencies, most of which they didn't really require. In some cases, also, they would include dependencies that were actually provided by the system and the process override what we know what the system needed to run.

Using byte ray, we were able to scan for violations like that. In some cases, just knock out jar files that weren't needed, in other cases, signal a warning to the user that they need to go and fix something.

**[0:50:31.9] JM:** Last question. How will data engineering improve in the next five years?

**[0:50:36.9] CS:** As I mentioned earlier, I think that Hadoop provided a file system when we actually probably needed a dataset system. I think that the migration from distributed file systems to distributed blob stores is going to help to push things in that direction. It will also, I think improve the scalability of the storage there quite a bit, since at least from what we've seen in HDFS, the scale limiting factor is the name node and managing that namespace.

I think the other theme did I expect to see is more ideas and abstractions being borrowed from conventional, relational databases, but combined at the same time with the level of freedom and flexibility that Hadoop and Spark provide, as well as low-latency in terms of being able to read data as soon as it lands and not having to wait for the data to be loaded into the special format that a database requires.

**[0:51:36.2] JM:** Carl Steinbach, thanks for coming on the show.

**[0:51:37.8] CS:** Hey, thank you very much for this opportunity to talk.

**[0:51:39.8] JM:** Awesome.

[END OF INTERVIEW]

 

**[0:51:50.0] JM:** LinkedIn is a software company with the goal of creating economic opportunity for every member of the global workforce. LinkedIn is hiring data scientists, software engineers, researchers and many more roles for its engineering team.

To find out more about the problems that LinkedIn is solving and the teams that are solving these problems, check out engineering.linkedin.com, where you can read about culture, open source projects and hard problems that LinkedIn has worked through and blogged about.

Thank you to LinkedIn for sponsoring the show and for creating software that I use every day.

[END]