EPISODE 1258

[INTRODUCTION]

[00:00:00] JM: Data Management Framework used to simplify incremental data processing and data pipeline development. This framework more efficiently manages business requirements like data lifecycle and improves data quality. Some common use cases for Hudi are record level insert, update and delete, simplified file management and near real-time data access, as well as simplified CDC data pipeline development.

In this episode, we speak to Vinoth Chandar, VP of Apache Hudi. Vinoth is the creator of the Hudi Project at Uber, and he continues to lead its evolution at the Apache Software Foundation. Previously, he was a principal engineer at Confluent, and a senior staff engineer at Uber before that. We discussed building large scale distributed systems and data management systems.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the Club for Software Daily on Clubhouse. It's just Software Daily. And we'll be doing some interesting Clubhouse sessions within the next few weeks. And two, if you're looking for a job, we are hiring a variety of roles. We're looking for a social media manager. We're looking for a graphic designer. And we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster, and you're curious about how to get involved, we are looking for people with interesting backgrounds who can contribute to Software Engineering Daily. Again, mostly we're looking for social media help and design help. But if you're a writer or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com. That's jeff@softwareengineeringdaily.com.

[INTERVIEW]

[00:01:48] JM: Vinoth, welcome to the show.

[00:01:50] VC: Hey, glad to be here. And thanks for having me.

[00:01:54] JM: So today, we should have a pretty riveting discussion about data lakes, and data warehouses, and data infrastructure in general. So I'd like to start off by just setting the stage as I see it. The abstraction of the data lake is this place where you throw all your data into low-cost storage and you can later on do things with that data. And since it's cheap, you really save all your data there. And the data warehouse is a place of more expensive storage. It's maybe closer to being in-memory, and it's typically more expensive, but faster to access. That's a very rough description of how I see those two abstractions. I would like your perspective on those two abstractions and how those terms have evolved over the last couple of years.

[00:02:48] VC: Yeah, I think how you describe them were pretty much how they were till a few years ago. Starting from – If we rewind a bit, what we had was back to the Teradata, the Vertica, the era of on – Like more coupled storage, compute data warehouses, right? They were pretty much MPP databases, right? And then Hadoop came along to power mostly – Bring a lot of horizontally scalable compute power, or the kind of like data that you collect from your operation systems, right?

And then this is kind of like how most of these data lakes and warehouses co-existed. But in the last, I'd say like five years or so, both the data lakes and the warehouse ecosystems have evolved dramatically. Specifically speaking, cloud warehouses are prime time now and they offer actually much – A very different architecture than their predecessors. Specifically, they decouple storage and compute. And then the storage is usually horizontally scalable using cloud storage. That way, they sound similar to data lakes, right?

And then if you take data lakes, data lakes have absorbed a lot of the needs for transactionally managing data, or having abilities to mutate and update your data that you stored in a data lake. Moving away from the sort of, "Okay, let's treat it as cheap dump of all the data to like a more consciously organized, a lot of structured data that is flowing into the data lakes." And then the data lake technologies have started to become more and more like database/data warehousing perimeters as well. So that's the world that we are heading towards, from that I see it.

[00:04:47] JM: What do you know about the different popular data warehouse, data lake abstraction? So I would see – The three primary ones I would see are Snowflake, BigQuery and the Spark data lake house architecture with Delta and Spark. Maybe you would include Redshift plus something. I don't know. But maybe you could lay out the different popular data warehouse, data lake abstractions and how they contrast with one another.

[00:05:24] VC: Yeah. So let's start with cloud data warehouses, right? And I actually would put Redshift in front of the list. I look at Redshift, BigQuery and Snowflake as the cloud that houses when I think of them. They all share some very common characteristics. Like they all have a lot of database-like parameters, right? They let you do foreign keys, da-da-da. The amount of transactional capabilities that you have on them is actually much higher than what you see with – As we we'll see when we talk about the data lake abstractions, right?

And then they all have one internal proprietary format. They're not very open. But then they are able to – They're pretty much like vertically integrated systems, which are one SQL, one file format like execution runtime. And it's a very tightly integrated system that you use for BI and like sort of like your regular analytics, right? So I look at them in a similar sort of like bucket.

And moving on to the data lakes, that's where the context of my involvement with Apache Hudi from 2016 or so. There's Apache Hudi and then there's Delta Lake from Databricks, and then Apache iceberg, which originated Netflix, right? So all of them provide, I would say, a transactional layer on top of what used to be just like managing files on top of S3 or cloud object stores just using the same open file format that we all have come to like love and embrace in the last four years, like Parquet, ORC, right? And we can get into this more, but all these three projects even have very similar – Some similarities as well as huge contrasts as well in their internal design and how they approach the problem. And Lakehouse is talking about that. Personally, for me, when the Lakehouse terminology and everything came out, it wasn't hugely surprising to me, because we are already been building something like that at Uber in production for a few years. I know like several large tech companies have been already doing similar things where the core idea was, “Hey, let's bring some data – Like sort of warehouse primitives to the data lake and kind of tried to do more on the data lake itself.” That was the that was the core idea, right? And I think Databricks did a great job of kind of adding an industry perspective to that kind of architectural pattern, being one of the leading Spark compute providers out there. I think they did a very good job of like articulating that vision. But that's kind of how I see it.

[00:08:08] JM: Since you mentioned Uber, can you give me a little bit more context on data warehousing at Uber or data infrastructure at Uber where you worked?

[00:08:18] VC: Yeah. So let's take a little trip down memory lane. So I showed up at Uber in 2014. And this were like really – Uber was growing really, really fast, right? Which means as we launched more cities, the data volume was growing as well. And Uber is a very real-time business. So there is actually a lot of incentives. And we can like save a lot of money by building in a lot of systems that optimize for data freshness would have the company, the city operators, the engineers, everyone have access to data, right?

So we started with like a cloud – Sorry. Not a cloud. So Uber is all on-prem. So we started with Vertica and like best in class MPP, an on-prem data warehouse. But as the data volume grew, we realized the need for a data lake, right? Going back to how I started our chat. We brought in like HDFS, HDFS-based. We dumped all our initial data into the data lake, use Spark, wrote some ideals and then moved out of the raw data of our on-prem warehouse.

But as we tried to do this exercise, what we realized was adding this extra layer between our operational databases and the warehouse with the data lake was actually adding a lot of latency in between them, because mostly due to everything being done in a bulk, big batch fashion in the Hadoop world. And that's kind of how Hudi was born.

What we tried to do with Hudi was, well, Vertica is guys It already supports updates, deletes. And then we are actually able to take, let's say, database change logs and kind of absurd Vertica, if you will, and then that gives us great query data freshness, and Vertica gives us good query performance. Why don't we replicate something similar by building like a transactional, or a serverless transactional layer right on top of Hadoop file system abstraction so it will work with HDFS, S3, it's future-proof. And we try to solve for updates and deletes for while ingesting operational data into the data lake.

So we could we could go from a 24-hour batch job that runs for like 12, 16 hours, 24 hours to actually something that finishes in 30 minutes, 15 minutes, 5 minutes. And then we can actually tune the latency based on our needs actually, because the Hudi was written as a database using Apache Spark as a runtime. So it can horizontally scale really, really well.

And the second problem that we solved was just solving for updates and deletes wasn't like enough, right? Because normally, in the data lake architecture, you'd have a set of raw tables, and then they would write ETLs to build more derived tables out of them. So what happens now is all these derived tables have no intelligence about what data actually changed. So instead of, for example, like if you have a simple ETL job, which was standardizing currency conversion or some very simple raw base operation, it would have to actually full table scan the entire tips table table to actually understand what changed, right?

So we said, “Okay.” How the stream processing solved this is you give people like a chain string, right? So those are the two fundamental things that we built in Hudi. And then we build transactionality as actually a side effect of having to support like update, deletes and incremental change streams. And that's kind of how Hudi was born by end of – We did this in 2016. And by end of 2016, two years already powering all of the critical kind of business critical data sets at Uber, storing like a few petabytes already. And we open sourced the project in 2017. And we entered the incubator, the Apache Incubator programming 2018 and graduated out of that. And we've been having sort of like growing the community alongside many cloud providers who adopted Hudi in their platforms as well to solve these same problems that exist broadly across the industry.

[00:12:45] JM: So tell me why the problems that you're describing with Hudi – Like just to double down on your explanation. Why did the problems that you're describing that were solved with Hudi not – Why were they not solved by like the existing data infrastructure technology at the time? And have they been solved by these existing data lake, data warehouse products today?

[00:13:14] VC: So I think the core problem was we were kind of stuck between a rock and a hard place, right? We can get – We knew we needed capabilities like transactions and updates and deletes for us to quickly ingest data from like an upstream database into the warehouse. But the warehouse couldn't hold all of the data, right? Because it was the storage – And you can run like tens of nodes of Arrows cluster. But our data is just so huge that it won't fit on any one cluster, right? So that was a limitation of the data of Arrow stack where we had the functionality, but we couldn't scale it out.

But on the Hadoop stack, generally speaking, there is like not very mature systems to ingest data and manage them really well at that time. So most of the work in the Hadoop plan had gone into building HDFS, Yarn, these Hadoop Spark, Hive Spark, Presto, these query engines, right? Not a lot of attention had been paid to actual data management, or the storage layer. Simple things like, for example, sizing the file. So famously, HDFS, you scale HDFS and keep it healthy by writing well-sized files, right? So no library would enforce this uniformly across the entire ecosystem, right? Large companies all tried to build their own solutions and like in different timelines in their company's evolution. So we were actually – We had a clear gap that we're trying to build. And that's kind of how Hudi was born.

If I fast-forward today, I would say the cloud warehouses have done a great job of kind of solving for, let's say, the data scale problem that I mentioned with the old-gen data warehouses, right? Because their storage is on S3, not on GCS or some cloud store, not on the local box. So they do solve the data storage scaling problem. But the issue still remains that they are proprietary formats, right? So if we were to do this again, today, I would still build what we built, because we wouldn't want to, like lock away all of the company's raw data into one proprietary format. The challenge remains.

The second problem is, I think these are the places where the argument that Databricks makes towards Lakehouse resonate with me, is they're not very general purpose, in a sense that we need support for data science and machine learning as a first class citizen. And then we want the data scientists to be building their models and analytics of the same data that the analysts are using for reporting. So if you have a sort of like a stack where the data gets into warehouse, the cloud warehouses first, and then you make copies for in your data lake for your machine learning or something. You run into tons of data quality problems. And architecturally, I think it still makes sense to have the data enter into a raw data lake powered by something like Apache Hudi very quickly. That way, it lands very little overhead for any downstream processing that you want to do. And then you pick and choose what tools you want to kind of curate your data, if you will, for analytics. Or there are lots of other real-time analytic stores, right? Like your Druid, and the Clickhouse, Rockset kind of like world.

So even for real-time analytics, there are other systems, other specialized engines that you can feed this data off of from now. As an organization, you have all the freedom because all of your data, the mother copy, if you will, of all this raw data, well-managed, and it's in an open format on very cheap storage, right? So that's what I see. At least what I think companies should be doing for their data architecture.

[00:17:25] JM: Hudi is an open source project, which implies that this is broadly applicable. This is not just applicable to a company of Uber, Uber scale, which are few and far between. Why is this a widely applicable problem?

[00:17:41] VC: Yeah, so this is actually a very, very good question. So when we actually started Hudi, and even to trace through my own journey about the problem, I was pretty convinced that this is how like we have to build this for Uber. And this is what generally something that people should adopt in general. But the kind of a forcing function wasn't there to get people to n-mass sort of adopt this, right? If you look at some of these technologies, Hudi has been around since 2016 or so. Delta has been around from 2000 – Late '17, or '18, something like that. And it actually took us something like GDPR, some like forcing function like that where suddenly you said, "Okay, you know what? All your all your data that you're simply dumping onto S3 or something, you need to manage them." Now, you need to be able to like delete stuff, you need to be able to like correct or mask things in there. That is essentially something that applied to any global company, right? And then that was like pretty much brought a lot of focus back on to data lakes, because you couldn't just use it as a dump anymore and kind of like ignore a lot of these problems. And that is pretty much where we felt a lot of the Hudi or option really take off. So this problem of updating data transactionally, and then ingesting like the streaming data lake pattern, as I call it, is slowly catching on, where people are realizing that, "Hey, you know what? It looks like I have these laws, and then the data privacy laws are only increasing. So I need to properly manage my data late now. So what's the right architecture?" So they're now engaging with this conversation and trying to actually look at answers for, "It looks like I need a data lake and I need a warehouse. So what's the right architecture?" So we are right at that point in the industry now with these tools. And I think we will settle into a pattern of sorts in the next few years.

[00:19:44] JM: Tell me a little bit about what you see as the ideal data architecture. Like what is the ideal user experience that cuts away as much of the configuration and painful setup work or maintenance work? And what kind of the dream experience for the data engineer, the data science team that's working on top of the data platform?

[00:20:12] VC: That's, that's another fantastic question. So let's think this through from like an organization standpoint, right? So let's say there's a company. It's reasonably successful now. It has hundreds of employees. And then now kind of like the data management problems start to like crop-up, right? Till then you can pretty much use some tools that integrates to do your basic reporting analytics. But now, if you're like a lot of people, you have like two three business functions. You have, let's say, a risk team, a risk kind of fraud team, and there is like a finance team, and then there is a product team. And then each of them hire data engineers, right? And then they are interested in some amounts of data that sits in some operation, like upstream database, like either MySQL, Postgres, or Oracle, or RDBMSs, or the NoSQL stores. And then you have the Kafka data, right? Like the event tracking data. Pretty much every company uses something like that to track a lot of activity that is going on.

So most companies essentially choose a path where they start by hiring data engineers into individual business functions. They curate pick and choose some data sets that they need, right? And then they actually don't end up building like a fully centralized data lake, because it's often hard organizationally to fund that kind of product.

So what ends up happening is, over time, there are these silos. And then a query that a finance team guy writes is not – He can't reconcile the data with someone on the fraud team. And then you need to give like separate kind of like different kind of production data access controls to someone on finance team, as opposed to the fraud team, right? So I'm just like describing how people generally go about today, and why you hear the pain that people express working with data lakes, right?

So I think the first and foremost thing to solve, the dream experience in my opinion would be like if there were like a very easy path for first, just in-raw, replicate a lot of your upstream systems into the data lake, right? And this has to be done in the raw without much transformations. And it has to be done very quickly so that you're not waiting for this data to arrive for like hours together before you can do your things, right? So as long as you can build that like streaming raw data layer, as they call it, and then it free – And with some like tools and controls, access controls, on top of it, then it frees your data engineers from just focusing on the business function, right? They have all the data at their disposal if they want to cross join something to get better data quality, right? If you need to join against one another table to weed out bad rows, it's all there, right? And you don't have to like do something without that and later figure it out.

And then you essentially choose, okay, if you look at Databricks today, Databricks is a Spark runtime with a great sort of data science stack on top, right? And if you look at something like Starburst, or the Presto, the HANA Starburst, the Presto – The query engine companies, they're really good for – Like they have a good BI stack on top, and then so do all of the cloud warehouses, right? So then you're actually – If you do it this way, you can pick and choose based on your use case which of these query engines that you want to use. And then you can actually say, "Okay, you know what? I'll give the data science team the database subscription. While the finance team was mostly running reports, and like dashboards, so I'll give them like a Presto or like a cloud warehouse account. So you have the freedom of choice and you can actually control what data goes where and at what price point. And then you use – Again, like going back to what I was saying before, this initial raw data layer adds very little data latency. So you're not really – A lot of people, given historically the data getting into the data lake itself is a pain and takes like few hours, people usually circumvented. They dumped databases into – Database change logs into Avro sets, reverse copy it into the data lake. They do like a lot of these like convoluted data flows. But with this kind of model, it's very streamlined, right? All operational data flows into the data lake very quickly. And this is the origin point for any derived data computation that you do in your company. So at any point, you can move from one cloud warehouse to another or bring in like your favorite – Deprecate an old real-time analytical engine and bring in a new one. You will be re-compute pretty much anything if you need it to, and you have a lot of freedom in this model. And that's what I think we should be moving towards, in my opinion.

[00:25:10] JM: So given that future that you've just described as a North Star, take me through what needs to happen to get the average data infrastructure to that world? Like what are the other kinds of abstractions? Or what kinds of advancements do we need in infrastructure technology to get to that future?

[00:25:36] VC: Right. So I think a lot of the technology for this is kind of there. And if we now go piece by piece, first things we need to do like get really good at change data capture from – And this is where projects like Debezium are doing a fantastic job. And there are more and more companies which are providing CDC streams from databases. And as that kind of becomes like mainstream, so more standardized tooling towards sort of ingesting these streams and kind of baking them into tables on top of the data lakes, I think that is something that we would we would really need.

With Apache Hudi, we’ve already taken some large steps in this direction with Hudi. That's like something that we've been building Hudi as kind of like a platform, rather than like a transaction layer or something that solves this one update problem. So you get like an ingestion to a lot of box, which you can point at like your Kafka cluster, or if you're doing Debezium, there's a reason Robinhood talked where they did exactly that. You can get these change logs and apply them very quickly. And then you get all your business data there, right.

So the more tooling and a better path towards like ingesting and integrating the data quickly, that's one. The second part I would say, which is in terms of – If I can take a minute to contrast like cloud data warehouses and the data lakes. So the central meta store in the data lake Land is still the Hive Metastore, right? And then in the recent years, the Hive Metastore has its own like scalability problems, right? It cannot track detailed statistics at the file level or things like that. So I feel like we need to – For people to be able to query this data with great performance and also like provide very good usability, we need to either like significantly improve something like the Hive Metastore or build something like that that can now become like – Recognize these formats, this new kind of like systems like Hudi and kind of like abstracted away for the open source query engines. So that is one technical gap I see. And without this, for example, individually your presto query engine may be like really, really good. But without all the statistics being fed into it, you're not going to be able to, for example, get the same level of performance as you would get from like a fully vertically integrated system like a cloud data warehouse. So those are some areas, again, that I think we need to improve on, right?

The third point that I would say, which is very core to actually what Hudi's goals are, and admittedly, actually, this is someplace where I thought as a project we'd have gotten much farther than what we did, which is we have to sort of like think about how we can learn from stream processing and make more of our batch jobs sort of like incremental or running – Not reprocessing so much. Because, again, anytime you have these like bottlenecks around either data freshness or query performance, that leads to like – Because like business is our real needs around them, that can be like risks and threats to the ideal data architecture that we just like discussed before. That's when people start taking shortcuts and like sort of going a different direction with their data architecture. And these are three things I think that we should build.

[00:29:22] JM: Coming back to Apache Hudi, can you go a little bit deeper into the architecture of Hudi? I guess since people listening might be a little confused, we should revisit what Hudi does exactly. And then let's talk through the architecture.

[00:29:40] VC: Yeah, so Hudi at a high-level, the best way to describe something like Hudi is it's a data lake platform. And it has a lot of platform components, which is built around sort of like a database core, if you will, which is optimized for a very streaming friendly way. So I knew I threw in like a lot of words in there. So let me like break it down more, right?

So, fundamentally, at the lowest level, Hudi has no long-running servers. It's a library, and where you can use Spark or Flink to write data to it. Hudi basically organizes like data in Apache Parquet or Apache Avro files. And it provides a lot of metadata. Also tracks a lot of metadata about writes and changes that are happening to this logical data set on top of cloud storage. And then all query engines like Hive, Spark, Presto, Impala, Trino, and even Redshift, they can directly query data that is written in a Hudi table. That's a high-level what Hudi does as a project.

So now if we break down Hudi as like an OSI data layer kind of stack, so you have cloud storage, and on top of that there are these open data file formats, Parque, ORC, Avro file formats, and everything, Hudi basically defines a layout of an organization of files on top and then provides concurrency control and transactions on top of that. And then on top of that provides some capabilities to do indexing of the data, so that you can do fast update deletes. And on top of that, it has like a runtime of what I call as stable services. So these are – think of them as if you're familiar with RocksDB, or BookDB or any of these like databases more, they all have internal daemons, which are optimizing the storage layout and are re-indexing something or compacting or doing several things in the background while the user is blissfully just writing data into the format. So we have a bunch of these things, which are run during writing, or you can run them asynchronously. They can be kind of like be running like a daemon optimizing all your tables. So we do things like size files. You can observe updates into delta logs, and then later compact them. So there's a compacting service. You can fundamentally alter your storage layout and re cluster your data for better query performance, right. So we have a clustering service there. And then there is something that can like clean and purge old files. And all of these services are kind of like they are aware of each other. Like, for example – And that is a core design differentiation of Hudi as opposed to like some of the other systems out there.

And on top of this, this whole like database sort of core, we do have a large number of upper level services. Like there's an ingest service, which can take data from Kafka, turn it into like essentially absurd stream, and just absurd a Hudi table on top of S3, and it can do checkpoint management, it can like recover on its own. Same thing, we have like a bunch of different unstructured data formats that we can kind of like ingest into Hudi table. We also have support for transformations. So you can, for example, write streaming style incremental ETL pipelines where you say, "I'm going to consume change streams from even just an upstream hoody table, right? Just like how will you tell a Kafka topic. And then you get, essentially, let's say, all the records that have been inserted or updated since a point in time. You can simply take them undo your joints and right like ETL jobs.

So again, we do duplicate data. We have tools that can give you commit notifications when data lands on Hudi tables. And we have like a lot of these things. And this is why I was saying essentially, our principle has been not to just build a core, the database core, if you will, but also a set of tools on top that people can simply take it and then solve a real use case very easily. And that's in a nutshell what Hudi let's see do.

[00:34:16] JM: and just to double down and make sure people understand, explain data infrastructure with and without Hudi. Like what are the disadvantages they're going to face if you are using a system that would benefit from Hudi but does not have Hudi?

[00:34:35] VC: Yeah, sounds good. So let's take a data lake built without Hudi. You have an upstream database. Let's say you have a Cassandra cluster or Dynamo, like some database, which has a bunch of change streams that are coming in, right? And you have a Kafka cluster through which your microservices are logging events. And then now, essentially, you have what we want to build as a data lake architecture. But you want to build a set of raw tables, and then write some ETLs and build a sort of derived tables, right? So without Hudi, the way typically people were doing it is they would write like some sort of like Spark job or use a framework like Kafka Connect or Camel or some something to just keep write – Like ingest from Kafka, write these events out as like Avro files and row-oriented. That's how you lay out the raw data. And then they would bulk import their database over a few hours, or they can do change capture from these databases, but they don't know how to apply them, right? So they would like do a bulk merge against the entire table.

So essentially, they will be doing bulk ingestion of databases and they will be doing like incremental sort of ingestion of the events. And if they want to go to ingesting every five minutes of Kafka data, or every one minute of Kafka data, they'd have to do a lot more to control the file sizes and everything, right? So what this leads to is there is poor data freshness for the database data in the raw layer. And then the event data in the raw layer has a lot of small files, or it's row-based formats, which are not very suitable for analytical querying. So when you write these ETLs, they are now delayed as well. Typically, you write a bunch of ETLs using Hive or Spark and then you build a set of derived data tables, right? Those derived data tables also suffer from poor data freshness. And the query efficiency of the raw data is very, very poor as well, because you have to contend with raw data formats, cyto-based formats, and you have to deal with a lot of small files, which typically kill query performance.

So with something like Hudi, what you're able to do is use the Hudi's delta streamer tool, which pretty much can – If you have a CDC log from Debezium or wherever, or any data really in some Kafka cluster, you can keep incrementally ingesting it, continuously keep ingesting it while it can absurd the table directly, which means your data latency is in few minutes for even your database data. And for something like event data, you can use – Hudi will automatically size your files for you so that you're writing like well sized files so that your downstream ETLs and queries are performing really well. And it's in columnar formats. And you can take advantage of features like clustering to say, "Okay, I'm going to ingest data in arrival time." But later, let's say for the case of Uber, if you ingest data every 30 minutes, you're going to like write 10 files. Most of those 10 files will contain a data for all cities, because that's kind of like how data arrives. But with something like Hudi clustering, you are able to reorganize the data and say, “Okay, I'm going to bring all the records that belong to a given city closer to each other. That way, a query has to actually scan very less data.” So you can do a lot of these things where you can cut down on query cost and be more efficient and also improve your data freshness pretty well.

Now, moving on to the derived data pipelines. So Hudi also now gives you change stream for every single table that you write in Hudi, which means you're now able to take the same concepts that you have in stream processing, right? So typically, if you're writing a job, Flink job reading from – and joining a couple of Kafka topics. Similarly, you can write like a Flink or Spark job that is joining change streams to Hudi tables. And it can also join against another Hudi table as a snapshot. And then you are now able to write incremental data pipelines, which again, lead to – Since they run incrementally, they reprocess less amount of data, which means lower costs. And also they lead to like better data freshness, right?

This is I think one of the fascinating things for me when we were like also doing this originally at Uber. You don't usually get opportunities where you can actually reduce the cost and also be faster when building databases. Typically, if you want database to be faster, you all **[inaudible 00:39:34]** throw memory or data, or like you actually increase the cost. But here, since batch processing is pretty – Like it has a lot of scope for improvement and optimizations, Hudi it gives you a framework to actually incrementally ingest and incrementally do ETLs. That's in a nutshell what it will do for your data lake.

[00:39:53] JM: So with Hudi, is everything that is built with Hudi, is that being pulled into memory? Or does Hudi use disk at all?

[00:40:04] VC: Yeah, so when you query a Hudi table, it's no different from querying a Hive table or Presto table. Or like for a Hive table, essentially, whatever these lake engines do is what Hudi does. Hudi is merely like for forms of query layers, it just presents like a table abstraction that they query. So Hudi itself stores everything on top of cloud storage at this point. It does not have any long-running in-memory components. During its execution, it may cache a few things during a given transaction. But that that's about it.

[00:40:41] JM: Gotcha. So, does the application owner, like the person who is querying, who's making the ultimate query like the data scientist, do they know about Hudi in the ideal world? Or is it just transparent to them?

[00:40:56] VC: If they are doing, let's say, batch queries, for example, like if you're just querying a snapshot of a table, oftentimes they don't have to actually care whether it's Hudi or Delta Lake or whatever, or even Hive, that format is in, right? They're typically interested in simply, “Okay, the query is faster and the data is correct.” That's all. So they don't have to be aware. But if you are a data engineer writing incremental ETLs and everything, yeah, then you need to tap into features that are very Hudi-specific, right? You need to understand how Hudi will lay out the exchange logs. What's the format? So those people may be aware.

Once again, if you're writing batch ETL pipelines, you're not even aware that it can be a Hudi table. It just behaves like any other Hive or /data lake table.

[00:41:44] JM: Gotcha. So as we're winding down the conversation, there's a lot more I would have liked to explore with you. But tell me about the state of Hudi today. What is it capable of doing? And what's the vision that you'd like to get to with the project? Like how far are you from that vision?

[00:42:03] VC: So speaking in very broad terms, there is a lot of why – What Hudi is, is a manifestation of a vision that we shared in 2016 to say, “Okay, all of batch processing should be very incremental.” I would say in that vision, through Hudi, we made a lot of progress in pretty much cementing that. Yeah, when you integrate, ingest data for the raw data layer, you need to do it in a very incremental fashion. We build a lot of great software stack in Hudi, which can be very performant and has a lot of capabilities to let you do that, right. So specifically, we have a database core and a set of like services that are all horizontally scalable and easily deployable. If you know how to deploy Spark jobs and Flink jobs. And you can do a lot out of the box at Hudi.

The pieces that we would really like to invest for in the future are actually unlocking true end-to-end incremental ETL pipelines, right? We should be able to write very complex like ETL pipelines. So batch processing is very simple. It's stateless. It's simple, right? Stream processing today is stateful and it takes like even a different set of like engineer to write very good stream processing programs. So we want to actually lower that bar, and then help people write complex incremental ETL jobs and more a lot of the batch ETL workload towards that model. And that's the North Star in terms of like what we want the project to be hitting.

The other part is, an adjacent thing that we need to solve for in the project, is if our vision is – Okay. So why do we do everything incrementally? We're doing everything incrementally because we want to save on costs and we want data freshness to be great. Awesome. But there are a lot of gaps in the query engine. Some fundamental limitations of cloud storage systems that can prevent you from getting real-time query performance on this fresh data, right? So there are two dimensions to this; data latency, which we can solve with the incremental ETL and the incremental ingestion, but interactive and like real-time analytical query performance is something that we need to probably build something like a mutable cache, columnar caching layer in Hudi which can actually absorb a lot of updates, keep it in memory, reduce some merge costs so that people can query it well, our V1, for example, Delta an Iceberg. So most of the systems who have tried to overcome high-use limitations have moved us to like a flat file-based like world where all of the table statistics is now written in one JSON file and Avro file. This is like scalable, but it can like take a lot of time to plan queries this way. So I think a highly-performant and a highly-scalable meta store much like what I'm guessing, what Snowflake, or BigQuery, or redshift like have internally, we need to build something like that, I think. These two put together I think will truly unlock the vision that we have, which is all data should be like arrive very quickly and should be able to be queried very quickly as well, right. So I think this is how bad we want to do most of our work going forward.

[00:45:36] JM: Great. Well, it's been a real pleasure talking to you. And I look forward to seeing what happens in the future with Hudi and your impact on data infrastructure.

[00:45:47] VC: All right. Yeah, once again, thanks for having me here. This was like some really deep questions. I really appreciate the amount of like research and like the depth of understanding that you brought out in this conversation, and I really truly enjoyed this. Thanks. Thanks again.

[00:46:05] JM: Thanks, Vinoth.

[END]