EPISODE 672

[INTRODUCTION]

[0:00:00.3] JM: Modern applications produce large numbers of events. These events can be users clicking, IoT sensors accumulating data, or log messages. The cost of cloud storage and compute continues to drop, so engineers can afford to build applications around these high volumes of events. A variety of tools have been developed to process events.

Apache Kafka is widely used to store and queue these streams of data. and Apache Spark and Apache Flink are stream processing systems that are used to perform general-purpose computations across this event stream data. There are many general-purpose distributed stream processing systems; Kafka, Spark and Flink and Storm and Google Dataflow, these are all great general purpose tools, but there is also room for a more narrow set of distributed systems tools to support high-volume event data.

Apache Druid is an open source database built for high-performance, read-only analytic workloads. Druid has a useful combination of features for these event data workloads, including a column-oriented storage system, automatic search indexing and a horizontally scalable architecture. Druid's feature set allows for new types of analytics applications to be built on top of it, including search applications, dashboards and ad-hoc analytics.

Fangjin Yang is today's guest. He's a core contributor to Druid and the CEO of imply.io; a company that makes a storage querying and visualization tool built on top of Druid. He joins the show to talk about the architecture of Druid and his company Imply. This is an example of an open source company success story, which is something we didn't get into, but I wanted to mention that that it is a company built on open source technology.

The other thing I find interesting about Druid is that it is this domain-specific system for dealing with high volumes of streaming data and that stands in some contrast to these general-purpose stream processing systems like Spark, or Flink, or Storm and it will be interesting to see what other kinds of domain-specific event data workloads, or just data workloads, what kinds of new tools get built in the future to handle these high-volumes of data, because here you see not

© 2018 Software Engineering Daily

exactly a narrow use case because high-volume event data that is read-only is actually there's a large class of data use cases that fit that type of workload, but maybe there are some other use cases that are subsets of these streams of data that are growing in magnitude every year.

I hope you enjoy this episode with Fangjin.

[SPONSOR MESSAGE]

[0:03:10.1] JM: Azure Container Service simplifies the deployment, management and operations of Kubernetes. Eliminate the complicated planning and deployment of fully orchestrated containerized applications with Kubernetes.

You can quickly provision clusters to be up and running in no time, while simplifying your monitoring and cluster management through auto upgrades and a built-in operations console. Avoid being locked-in to any one vendor or resource. You can continue to work with the tools that you already know, so just helm and move applications to any Kubernetes deployment.

Integrate with your choice of container registry, including Azure container registry. Also, quickly and efficiently scale to maximize your resource utilization without having to take your applications offline. Isolate your application from infrastructure failures and transparently scale the underlying infrastructure to meet growing demands, all while increasing the security, reliability and availability of critical business workloads with Azure.

To learn more about Azure Container Service and other Azure services, as well as receive a free e-book by Brendan Burns, go to aka.ms/sedaily. Brendan Burns is the creator of Kubernetes and his e-book is about some of the distributed systems design lessons that he has learned building Kubernetes.

That e-book is available at aka.ms/sedaily.

[INTERVIEW]

[0:04:45.9] JM: Fangjin Yang, you are one of the contributors to the Druid database and you're the Co-Founder and CEO at Imply Data, welcome to Software Engineering Daily.

[0:04:54.4] FY: Hey, Jeff. Great to be on the show and thanks for having me.

[0:04:56.9] JM: Yeah. Last time you were on the show, we talked a little bit about architecting distributed databases. Today, I'd to talk more specifically about Druid. You are one of the core contributors to Druid. What's the aim of the Druid database?

[0:05:11.9] FY: Right. Druid draws inspiration from a couple of different systems, so it has ideas from search systems, so systems like Elasticsearch and Solar. It has ideas from my time series databases, things like Prometheus and influx and it also has ideas from standard analytical databases, things like Vertica and Teradata.

We've been really calling Druid nowadays like an operational analytics database. It's a database that's really designed for time oriented event-driven data and it's designed to do high-performance analytics on that data. Sometimes the analytics are a little bit more search oriented, sometimes they're more standard group buys, more standard set of slice and dice, as you would find in the OLAP world. That's the main use case of Druid, it's commonly also used to power different types of visual applications as well; different types of UI, different types of digital applications.

[0:06:03.6] JM: There's a number of different components of Druid that we will touch on. Let's start with a high-level example of a piece of data, such as maybe a user clicks within a webpage, or perhaps they click on an ad in a webpage, this is a common example for Druid, since it's a high velocity of these kinds of events. Event data is something that's commonly associated with Druid, so describe the ingest process for a piece of data, like an event of a person clicking on a webpage coming into Druid and the different components that it touches as it's being ingested into the system.

[0:06:39.9] FY: Yeah. Clickstreams is a very common use case of Druid. Pretty honestly, what we've seen in enterprises today is there's a more modern data analytic stack that's evolving and drew it as something that's part of that stack, but it's not an end-all be-all solution for

clickstreams. What I mean by that is if you look at data set like clickstreams or any user behavioral data set, these things has tend to be pretty large in volumes, they tend to be pretty complex and most modern data stacks you have a dedicated technology to solve different problems in the lifecycle of this data.
For example, a very common setup I see is there's a message bus, or there's a delivery piece, which is responsible for taking the event from the system where it's created on and delivering it somewhere downstream where it can be further processed, or even queried. This message bus, popular examples are Apache Kafka. Each of the cloud vendors offer something similar. Amazon offers something called Kinesis, which is similar to Kafka, but the examples of these systems are for event delivery, usually after the event is collected from where it's created. There's another system which is designed to enrich or transform the event, so this is your more standard canonical ETL process.

This is generally some stream processor. Examples here include things like Apache Flink, Spark streaming, Kafka streams, they're all examples of systems that are designed to enrich and transform data. Then where Druid lives in this modern architecture is after the event is processed, or after its cleaned and then it's sent to Druid, then Druid both stores the event and also enables users to run pretty complex sophisticated analytic queries on top of the event.

Each of these three pieces event delivery, data transform and also querying and empowering UIs, each of them require dedicated system, because with today's data volumes and data complexity, you really need a dedicated system to solve different parts of the stack. There's something like a clickstream. This is a very common model that I see. The whole clickstream is it's as an event occurs, it’s sent to Kafka, it's processed and then sent to Druid, so it's a continuous stream of events and everything's being done in real-time as well.

[0:08:54.0] JM: The model that you described there where you have a message bus like Kafka or perhaps a data lake and then you have jobs that load these events, these large streams, or large sets or sets of files of data into Druid in an ETL process. As you mentioned, some of them are using stream processing systems like Flink, or Spark. People also use Flink or Spark for doing the kinds of analytic processing that you can also use Druid for. Just to get at the high level a little bit, how does Druid compare in use case to using something like Spark to read your

clickstream data directly from Kafka, or directly from your data lake and doing processing in Flink, or Spark?

[0:09:43.3] FY: Right. I want to make maybe a small distinction between Spark and Spark streaming. Their Spark core, which is really designed for batch processing, and there's Spark streaming which is a separate project that's dedicated for potentially manipulating streams. To talk about the stream processors as a whole, the way that they work is they all work in similar ways that they're working on a continuous stream of data and you can manipulate the data and you can basically apply various types of operations on the data.

Some of these operations can be query-related operations. For example, you can set alerts, you can do some basic SQL queries on top of the data. However, with almost every stream processors out there today, the stream of data is not stored in any optimized data format, so there's there no structure, no storage format really for a lot of the streaming data. As such, there's no framework in place to really make query performance really optimal.

If you're working on a very small window of data, you're not working a lot of data, you don't really need – you don't need a specialized format to make queries optimal. You just work with the very small window of data. You're building a basic alerting systems, you're building a basic alerting system, you don't need something as heavy as Druid.

However, if you want to do more sophisticated analysis on real-time data and especially if you want to now start correlating results with historical data and doing a combined view of what's happening right now and correlating that with what's happened in the past, then you do need dedicated stored formats, you need more optimized data structures and then you need a system like Druid.

Right. Basically, if you look at a system like Spark, it is really the next generation of MapReduce, so it's designed to be a more efficient version of MapReduce that that was no dominant in the early days of Hadoop. The Spark is general compute engine, which means that you can really do anything. It's not particularly optimized to do a lot of things very well, but you do anything that you want with Spark, and I think that's what makes it very powerful, but at the same time you can think of really a database, a canonical data warehouse from Oracle, or Teradata as being

able do a lot with data as well, but there's also a lot of workflows they're not optimized for. I view Spark in that same camp, that there's a lot of types of datasets, there's a lot of type of queries, a lot of type of use cases that Spark could be applied to, but it's not the optimal solution, and there's going to be pretty significant performance differences trade-offs, but a solution is a little bit more dedicated to the use case.

[0:12:09.1] JM: Right. Well, let's go into the architecture of Druid and maybe we can go back out to the high level a little bit later on after we've examined some of the components of Druid. When a piece of data hits Druid, there is an indexing process that it goes through and this reminds me a little bit of Elasticsearch, because when a piece of data goes into Elasticsearch, or oftentimes indexing it on a number of different dimensions. Can you describe the indexing process? What does that term mean, indexing?

[0:12:39.1] FY: Indexing is a term that we really took from the search world. It's a term to describe a process of converting raw data into a more optimized format. In the search world, what it means is you have some raw data, you store that data into a data structure, which is quite a bit more optimized for search. In the analytics world, we use that term indexing to describe a process that Druid does, where it takes raw data and basically converts it into a format that's very optimized for various types for OLAP and slice and dice analytics. It's taking raw data, applying various compressions on it, building various types of data structures that make certain types of queries extremely optimal.

[0:13:31.2] JM: What kinds of queries would those be? As I understand, when you're doing indexing, you might have a piece of data that has some structure to it, or maybe it is less structured like it's a JSON document. It has perhaps fields like name and time and maybe some rich data piece that's like a blob of text, and you can index on any number of those so that people might want to search across one of those fields and be able to rapidly access records from one of those fields. Can you talk a little bit more about what indexing does and what kinds of use cases it solves for?

[0:14:09.0] FY: Yeah. In the Druid world, what indexing means is basically you might have some unstructured or semi-structured piece of data; for example, a blob of JSON. What happens within Druid is that blob of JSON is going to be converted into a columnar format. Each of

different fields and the different values of the fields are going to be mapped into basically a flat column format. Then some of that nesting is preserved in the naming structure, but for the most part it's just a mapping of a complexity of structure into a more basic simple one that exists within Druid.

On top of each of the columns, there's going to be a variety of different types of more standard database indexes that get built. Some of these might be search indexes for very fast search and filter, some of these might be range indexes, or very fast numerical filters. Really at the end of the day, why we convert data in this semi-structured format into a more structured column format within Druid is to optimize for different types of OLAP queries.

Examples of these queries are group bys, or grouping on time, advocating by some value, grouping in time, aggregating by a bunch of values, and then also searching and filtering on a particular set of results. Those at the end of the day are the queries that this indexing process is meant to optimize for.

[0:15:24.6] JM: As you do that indexing, how much does it inflate the size of a record? Because with the addition of indexes, you're taking a record and you're adding this additional data around it that allows faster access to that data, but one of the trade-offs is that in building those indexes, it also takes some time, which we could talk about. You're adding additional data to your overall system. What's the order of magnitude of data that you're adding by indexing?

[0:15:52.9] FY: Yeah. By adding additional data structures, you are adding more information. However, there's a lot of work that has gone into a system like Druid to leverage various types of compression, where these additional data structures are not increasing in data by a huge amount. What I mean by that is if you look at actually the mapping of how raw data converts into Druid data, oftentimes the Druid data is much smaller than the raw data. In instances of Druid clusters out there, the data in Druid compared to the raw data can be almost a thousand times less.

This has to do with ways that Druid pre-aggregates data as it’s getting ingested. It has to do with some of the many levels of compression that are basically leveraged in Druid to reduce the raw data volume. Then the additional indexes, the additional data structures to optimize for queries

almost become trivial at that point, because if your raw data is reduced by a thousand times and you add some, a little bit of additional information to make queries faster, it's really not going to be that impactful at all.

With every data set that we tried to load into Druid, we're actually trying to make sure right off the bat with some of Druid's pre-aggregation and roll-up capabilities and its various compression algorithms that we get a huge savings with in terms of storage, and then we start applying some of the additional indexes and additional data structures.

[0:17:12.5] JM: Yeah, the notion of pre-aggregation and roll-up, I think this can take us back a little bit to the differences in between using a little bit more domain-specific system like Druid, versus something more flexible, or non-domain-specific like Spark, because the roll-ups, the notion of roll-ups, if you know what your Druid system is going to be used for, for example clickstream data, you know that the field that you need rolled up for example is the number of clicks on a specific webpage, or on a specific family of webpages, and so that one form of compression you can take is if you have collections of records that can be grouped together, you can group together most of the fields is just the same thing and compress them as the same thing and then you have an aggregation of just the number of different clicks that have occurred.

This also reflects the fact that it's this analytic database, rather than one where you're going to want to drill into specific records, although it sounds like if you wanted to, you could in many cases drill into one of those specific clickstream records. You just might have to go through perhaps a process of decompression?

[0:18:21.5] FY: Yeah. To answer your question there, is Druid makes it optional, whether or not you want to store all the raw data. You can actually turn off its pre aggregation and roll-up and just store the raw records as is. This is something that people do choose to do.

However, most of the time if you're working with things like clickstreams, or you're working with like network flows, or various types of operational data, if you have billionth of events per hour, now you may not necessarily care about what has occurred in a particular millisecond in time, right? Most the time what you care about is an aggregate view of your data, what it looks in the last five minutes, or in the last hour, or the last day, versus in the last 10 milliseconds, how many

records did I get, because that level of granularity is generally a little bit too much information for most people.

With a lot of use cases, a lot of data sets, that level of pre-aggregation, with roll-up, of compression just ends up in tremendous, tremendous store savings, which then translates to really, really measurable performance benefits as well. It's something that I think makes Druid a little bit unique in this wide of wacky data space.

[SPONSOR MESSAGE]

[0:19:38.2] JM: Cloud computing can get expensive. If you're spending too much money on your cloud infrastructure, check out DoIT International. DoIT International helps startups optimize the cost of their workloads across Google Cloud and AWS, so that the startups can spend more time building their new software and less time reducing their cost.

DoIT International helps clients optimize their costs, and if your cloud bill is over $10,000 per month, you can get a free cost optimization assessment by going to doit-intl.com/sedaily. That's D-O-I-T-I-N-T-L.com/sedaily. This assessment will show you how you can save money on your cloud, and DoIT International is offering it to our listeners for free. They normally charge $5,000 for this assessment, but DoIT International is offering it free to listeners of the show with more than $10,000 in monthly spend.

If you don't know whether or not you're spending $10,000 if your company is that big, there's a good chance you're spending $10,000, so maybe go ask somebody else in the finance department.

DoIT International is a company that's made up of experts in cloud engineering and optimization. They can help you run your infrastructure more efficiently by helping you use commitments, spot instances, right sizing and unique purchasing techniques. This to me sounds extremely domain-specific, so it makes sense to me from that perspective to hire a team of people who can help you figure out how to implement these techniques.

DoIT International can help you write more efficient code, they can help you build more efficient infrastructure. They also have their own custom software that they've written, which is a

complete cost optimization platform for Google Cloud, and that's available at reoptimize.io is a free service, if you want to check out what DoIT International is capable of building.

DoIT International are experts in cloud cost optimization. If you're spending more than $10,000, you can get a free assessment by going to doit-intl.com/sedaily and see how much money you can save on your cloud deployment.

[INTERVIEW CONTINUED]

[0:22:02.0] JM: I want to talk a little bit more about the storage mechanism. There's this indexing process, you build these indexes that can compress the data and make it easier to search over, or do lookups over and the data gets stored in a column-oriented database. It's structured in a way that aggregations are easier. This column orientation – is this similar to a Parquet schema? What's the schema like for the column-oriented storage?

[0:22:29.6] FY: Parquet is an example of a column-oriented format. I think there are similarities between what Druid has and what Parquet does. We started creating the column format within Druid before Parquet existed. I would say the biggest difference is Parquet is very much a file-optimized, like columnar storage format and Druid's format is designed – it's a little bit more optimized to use memory. All the data that Druid stores and loads is very much like memory optimized.

Definitely, I think Parquet, there's a lot of ideas from what Parquet's above through its columnar format and Parquet is column format; they draw inspirations from that original Google Dremel white paper that was published, talking about how they were storing various records and why that format is particularly optimized for analytics.

[0:23:20.4] JM: How does your approach to in-memory storage compare to a disk-like storage format?

[0:23:27.0] FY: Druid creates shards of data that we call segments and these segments are actually memory mapped. What that just means is we leave it to the end-user to basically configure how much of their data is stored in memory at any given time. It could be a 100% of

the data, it can be 5% of the data. It really depends on the configuration and how much they want to spend on hardware.

The way that the memory mapping works is if you're recording some data and that data stored on disk, basically that data is going to get paged into memory and then the query is going to run over it. Obviously, the more data stored in-memory, the more it can avoid that paging time and the faster queries go. If you compare about a disk-based scan versus crunching results in-memory, obviously it's pretty well-known at this point, like a memory is significantly faster.

[0:24:13.3] JM: Pieces of data come in, they are indexed and turned into – you have a cluster of indexing nodes, and then they are indexed and turned into these data structures called segments. The segments are written in collections to these other nodes called historicals. Explain what a historical is and the relationship between historicals and these objects that we're actually reading the records, these segments.

[0:24:39.6] FY: Yeah. A very high-level overview of basically how Druid works, Druid is designed and its overall architecture is structured in such a way that it's very explicit about its core processes. This is a little bit different than a lot of other open source systems out there. For a lot of open source systems, many processes might be co-located together, they might be black boxed. One nice part about the black box model is that it appears simpler the first time you look at it, and it's generally a little bit more smoother to get started with.

The trade-off is of course, once you get in production, we've seen this a million times and I've experienced this many times in my career. You wish you could fine-tune each of the core processes differently. That was an architecture decision made in Druid, that we would explicitly name and provide configurations for each of the main processes. There's a process for indexing data, there's a process for data coordination, there's a process for reading data as well.

Each of the processes can be fine-tuned depending on your production work case. If you have many, many reads of historical data and not much real-time ingestion, you can allocate a lot more data for your reading process and a lot less for your writing process. That's reflecting the funny names that we have in Druid. The data loading process is called the indexing process.

The way that they work is you can fine tune how much resources that you want to allocate basically for your data ingestion. Your ingestion process can be co-located with your data reading process. The data reading process is really a process called a historical process in Druid, so the idea is that you might have real-time data coming in, the indexers or the indexing process is handling both reads and writes for that recent incoming data. Periodically, what the indexing process does is it creates a segment, which is this immutable data file that exists in Druid and encapsulates all the index data, and that segment is loaded then by the historical process.

Really what the historical process is designed to do is serve queries on top of segments. That's pretty much all it's designed to do. It's very simple in its operation, but you can also explicitly tune it, because it's an entirely read-focused process.

[0:27:01.4] JM: Read-focused is important here, because this is not like a operational database where we're managing user sessions and responding to user things like that. It's more like, we're just reading the analytic data in order to update our machine learning models, or do some data science reporting to people, and so you can optimize for reads, explicitly not for writing back to data.

If I understood you correctly again, going back to the beginning, the pieces of data come in, they are indexed on this cluster of nodes, the indexers, the middle managers, they are turned into segments and the segments are the data structures that these segments are written to disk and to on these historical nodes, and the historical nodes hold the segments off of disk into memory in a place where they can be queried quickly, because they're in-memory.

[0:27:58.0] FY: Yeah, basically. Yeah. That's the overall process that exists within Druid. You're exactly right, raw data comes in, it gets indexed, a segment file gets created this segment file is handed off to the historicals, the historicals download the segment, server in-memory and it's a combination of obviously leveraging memory, it’s a combination of leveraging various storage techniques that give Druid its performance.

The segment itself is an extremely read-optimized data structure. It's designed for very high volume concurrent reads. This is why as you start looking out in the wild of what is Druid doing,

most of the time, it's actually powering some interactive UI. It's seen as something users interact with and they're clicking through visualizations, or interacting through different types of visualizations with the underlying data.

It's because that Druid is a system designed to be used, or power applications and for users, that latency becomes very important, multi-tenancy, concurrency, all these factors become very important.

[0:28:57.3] JM: Well, explain more about that concurrent reads. What are the optimizations that you need to make, or what are the bottlenecks that systems who are not read-optimized, who are not optimized for serving concurrent reading users, perhaps users that are looking at the same dashboard on different computers, what are the kinds of things that you can optimize for to serve that concurrent read use case?

[0:29:19.8] FY: Yeah. Druid's segment is basically in a mutable data structure. Once it's created, its immutable. Of course, the trade-off in Druid is if you need to update, or delete off the data, you have to rebuild the entire segment. A segment might contain five million rows of data. It's a little bit more of an expensive process to really update a segment. What that means is you probably wouldn't ever use Druid for an OLTP use case, you would not use it to power your Facebook user profile. If you have a data set where the same fields are updated constantly, Druid is not a great. Druid is very good for analytics and the immutable segment model is one where, because you have immutability, you basically have read consistency throughout your entire system.

When you replicate a segment, one segment and its copy are identical because they're both immutable, that none of them are saving updates. You don't have to really worry about write consistency, you're just basically focused on read consistency. What the segment, once it's loaded in-memory, drew the parallelization model. The way that works is multiple CPUs can scan the same segment really at the same time. If you have a lot of concurrent requests, the log users trying to access the same data, you can basically have multiple CPU access like the same the same segment at the same time, which really helps for concurrency and multi-tenancy.

[0:30:39.5] JM: Define those terms read consistency and the write consistency.

[0:30:43.4] FY: Yeah. The idea is if you – in distributed systems, one of the challenges where if you're trying to support both high-volume reads and high-volume writes is if you have a piece of data and if that data is constantly getting updated, then every time you query it, you might be getting plenty different results. If you also have different replicas of your data spread out across your system. When you read one replica versus another replica, they might not be in the same state, because an update has not been applied to both of them yet.

What I mean by read consistency is with in Druid, if when you read a piece of data, you query it, you query it again, basically you're going to get the same results. If you query a piece of – or you query a segment, or you query its replica, you're basically also getting the same results. There's no contention there. There's I think simplicity in the architecture that you don't have to think about what happens if there's an update that's occurring, there are certain transactional properties you have to ensure for read consistency.

This goes more into database theory that maybe we don't need to get into, but the idea is within Druid, you're not really worried about a piece of data getting updated and what happens and what locks are being provided as that data is getting updated. It's just the system where you have a block of data, read it, have all the different threads scan it, have your different CPUs can it and you're always will get the same result.

[0:32:02.7] JM: You need to be able to manage what segments are in-memory, versus which ones are on disk and perhaps page things out over time. Maybe you want to keep some window of time in-memory, maybe the last hour you're keeping in-memory in, so you're constantly rolling out data and rolling in new data. What's the process for doing that rolling?

[0:32:27.6] FY: Honestly, we don't do anything internally within Druid. Linux operating system does it. All we do is just ensure that as a piece of data is getting queried, it's going to get paged into memory. If it's constantly getting queried, it's going to retain in-memory, and if it doesn't get queried for a period of time, it just gets paged out. Really that's being managed by the operating system. We're not doing anything particularly clever there, but the way it works in practice is it's

not really a time-based retention, it's based on what pieces of data are getting queried the most often, and it just ends up working pretty well in practice.

[0:32:59.5] JM: Cool. What about if a new, I guess the new records are coming in as your data is sitting there and you're reading it and it's just getting appended into your historicals, is there anything interesting you have to do to update the user's view, like how often does the user have to, or how often is the user polling the system for new database records? I guess yeah, tell me more about the process by which I'm a user, I'm sitting in front of my dashboard. I'm expecting new events to be coming in and I'm expecting my UI to be updated over time. What's going on into the hood to address those updates?

[0:33:36.5] FY: Yeah. I'll say for Druid, it's real-time capabilities are pretty strong from usually from when an event occurs to when it's invisible in Druid is a few hundred milliseconds. That's the general latency that we've seen. It can be faster, or slower depending on your particular setup. The idea is on the indexers within Druid, data is always being held in-memory, so there's a giant in-memory buffer on the indexers and that's where all the events, all the recent new incoming events get appended to.

The in-memory buffer that exists on the indexing nodes, it's basically just a key value store in-memory. Key value stores are known for very fast writes. This in-memory buffer is a write optimized data structure. It's designed to intake events as fast as possible and you append them in-memory.

Then what the indexers do is periodically they flush out this in-memory buffer and then they convert that key value-based storage into a Druid segment format. Druid segment as you recall, is that read-optimized data structure. There's purely audit conversion process from a write optimize data structure to read optimize structure that exists within Druid.

Recent new incoming data, append it as fast as possible, that's how Druid supports very fast writes, data is flushed out, converted into a segment format, which is how Druid supports very fast reads. This is occurring constantly on the indexer process and there's a constant handoff process between the thing that's doing the ingesting and the thing that's handling more long-term historical data. That's what's happening behind scenes.

In terms of the UIs that get built on top of druid, usually how those UIs work is that they're pulling Druid at some level of curiosity. It could be every 5 seconds, it could be every second, it really depends on application to application and how soon do users need results. I've seen systems that basically pull once a second. It's not the greatest thing, because if you start pulling once a second, your data is constantly changing. You actually can't make meaningful analysis that way. Yeah, that's Druid's how it combines that real-time view with its historical view.

[SPONSOR MESSAGE]

[0:35:50.2] JM: Your audience is most likely global. Your customers are everywhere. They're in different countries, speaking different languages. For your product or service to reach these new markets, you'll need a reliable solution to localize your digital content quickly.

Transifex is a SaaS-based localization and translation platform that easily integrates with your agile development process. Your software, your websites, your games, apps, video subtitles and more can all be translated with Transifex. You can use Transifex with in-house translation teams, language service providers. You can even crowdsource your translations.

If you're a developer who is ready to reach a global audience, check out Transifex. You can visit transifex.com/sedaily and sign up for a free 15-day trial. With Transifex, source content and translations are automatically synced to a global content repository that's accessible at any time.

Translators work on live content within the development cycle, eliminating the need for freezes, or batched translations. Whether you are translating a website, a game, a mobile app, or even video subtitles, Transifex gives developers the powerful tools needed to manage the software localization process.

Sign up for a free 15-day trial and support Software Engineering Daily by going to transifex.com/sedaily. That's transifex.com/sedaily.

[INTERVIEW CONTINUED]

[0:37:39.0] JM: Let's talk about the query and the lifecycle of a query. Let's say I'm a data scientist and I want to do an ad hoc query, or I am a operations person and I'm sitting in front of a dashboard that is issuing queries to the Druid system, walk me through the lifecycle of one of those queries.

[0:37:57.0] FY: Yeah. I'll say of use cases of Druid out there, an operator using some UI to troubleshoot, or diagnose, or understand why something's happening, or to create dashboards. It's a little bit more common than people writing raw SQL queries on top of Druid, although Druid does support a raw SQL interface. The reason for that is we just see Druid being used a lot more often by operators to diagnose issues and triage use cases, versus doing more standard data like warehousing need stuff.

The lifecycle of a query, how it works is there's – Druid supports two native query languages, JSON over HTTP language, or a SQL-based language, which is based on Apache Calcite. For a person that writes a SQL query, that SQL query gets translated into something native within Druid and that query gets sent to a Druid process called a broker. A broker is really a Druid process that knows how to do a divide-and-conquer based on the parameters of the query.

For example, it looks at the query and sees, "Oh, the query is querying data for a year, years' worth of data." Then it figures out okay, this years' worth of data is spread across an X number of historicals and Y number of indexers and what it does is it fans that query out to the pieces that hold – to the servers that basically hold the pieces of data relevant to that query. Those pieces are computed in parallel, because we're working in a distributed system and then all that data is then returned to the broker. The broker merges the results and then returns in the columnar. It's really divide and conquer algorithm for processing queries, and that's really how Druid's query handling works.

[0:39:39.6] JM: Why is it that people use Druid most, or why has Druid optimized this, or what optimizations does its use case lead to, that the dashboarding versus the SQL-like, I'm a data scientist, I'm just doing ad hoc queries. Why aren't people using it for ad hoc queries, or why isn't it designed for that? Why isn't it a good use case for that?

[0:39:58.6] FY: I'll just say if you look at a more standard data warehouse, there's a lot of complex joins. It can do, there's a lot of complex – look at a system like Spark, right? You can do anything with Spark and Druid is not a system where you can do anything with it. Systems like Elasticsearch, or systems like Solar, they're not designed to do everything. They're designed to do a subset of things really, really well.

Spark can do everything, but it's not optimized for anything. Druid is more similar to how Solar and Elasticsearch work and that it's optimized for a set of things and it's not optimized for the entire breadth of data warehousing use cases. You would not use Druid to transform data, Druid is not particularly good at doing very large-scale drawings that might take two hours to complete.

It depends on when most people are successful with Druid, they're either powering some sort of application, or they really care about the latency of their results. When people really care about the latency of the results, there's usually some UI that's getting powered, there's some application, or latency and consistency of that latency is very, very important. A lot of the architecture decisions within Druid are designed to make certain types of queries commonly issued by UIs very, very fast.

The thing is that there's been these UIs and these analytic dashboards around for a really long time and maybe you could back these things, back these dashboards with a variety of kinds of databases, or something like HDFS, or Kafka with Spark on top. Taking a step back and evaluating Druid in a historical context and looking at the different options out there of the databases and storage systems that could be backing a analytic dashboard, how does Druid compare with those other systems? Where do those systems fall over in certain ways that Druid may outperform them?

[0:41:55.4] FY: Yeah. I would say that the types of applications that Druid most commonly powers are ones where there's features beyond just dashboarding. The way that a lot of classic tools work is you create a dashboard and that dashboard has a bunch of different types of visualizations. Then you can monitor and check the trends of those visualizations. There's many, many tools are able to offer this, and I think there's a set of databases that can do this as well.

The class of applications that Druid is particularly great at powering are ones where you need a significant amount of drill down, where you're actually not just looking at, or monitoring why – you're not monitoring a particular trend, but you're also trying to understand why it happens. If you see a spike, you see anomaly and then you want to explain why that spike or anomaly occurs, and then that process is one where you're heavily slicing and dicing data, you're drilling into it, you're looking at how different attributes are contributing to different trends within the data. That type of workflow is what Druid is really designed for, and where it starts outperforming a lot of its competition.

[0:43:03.6] JM: Here, we can get into Imply, which is a application that is built with Druid in mind. It's the thing that's serving, or it is the applications, the operator level use case for a database like Druid. Now in a world, we think of the pre-Imply world as maybe the tableau, or the power BI, not that these systems are completely outdated or anything like that, but when I think of these systems I think of them as accessing maybe a SQL database, or a CSV file. It's less of other BI tools. It's like their BI tools that are built agnostic of what the underlying data system is. Whereas, Imply this company that you've built on top of the Druid open source database is really built with Druid in mind.

What kinds of advantages do you get out of that fact when you can build a BI tool with, if you want to call it a BI tool, or operational tool, with a specific database in mind? What kinds of things can you optimize? What kinds of new applications can you develop?

[0:44:11.4] FY: Yeah. Imply is an end-to-end solution built around Druid. The UI is really designed for Druid and some of the things that can do very well. I would say one of the things that Druid does very well is it has a ranking engine in place, which is very good at telling you what are the attributes that basically contribute to a particular trend, particular nominally, particular pattern.

We don't think of the visualization tool that Imply as anything really related to BI. It's actually much more operational in nature. Many of the use cases that we see among our customers is the ability to isolate a trend, isolate a pattern, isolate an anomaly, and then very quickly using some of the concept that exists within Druid, find the root cause of that pattern, of that trend and

also have an engine in place that's just pretty honestly one of the problems I think is talked about a lot is one that we solve as well is just dealing with the increasing volume and complexity of data as well. Druid is a system that because of its various compressions and reductions is very efficient at storing high-volume complex data.

Then the app that we build in Imply is surfacing some of the core features in Druid to make it extremely, extremely fast-determined, and to explain really why any pattern, anomaly, or trend exists within a data set. That's really the core value that we provide to a lot of folks out there.

[0:45:33.9] JM: There's a lot of different kinds of operational analytics that somebody might want to do, like looking at different types of fraud, or looking at these this type of clickstream analysis. This seems like more for, there is a human operation component to it. You're sitting in front of Druid, you're using Druid to look at and analyze data and then from there, I guess you would do things like you would build machine learning models in other application platforms. Druid is really more for this, I think this exploratory and this human element.

[0:46:09.2] FY: Yeah, exactly. It's designed to power applications, I should say to be used by humans. It's designed for people to really take action faster. The top use cases, we generally work with different types of operational data, so clickstreams, APM type data, network flow, server metrics, up and down the stack, fraud analysis obviously a pretty big use case, and many others that resemble that words.

Time to action is becomes very important, understanding and being able to explore and understand why patterns occur are also very, very important. It's designed really for that workflow. That’s Druid’s bread-and-butter.

[0:46:43.8] JM: What's been the model – getting to the business side of things, what’s been the model for figuring out what kinds of users you sell this to, or figuring out who in an organization you sell it to?

[0:46:56.1] FY: I think the people that express a lot of love for our product are our operators within our organization. Operators, whether they're within IT, sometimes they're within marketing, like in the advertising world. They're people that need to take some action really

based on results in data. Some of these folks are technical, some of these folks are not. Druid was initially built for online advertising and ad tech use cases.

The operators at that time were people that needed to adjust campaign performance, they needed to look at what are the factors that contribute to a particular level of user engagement broken down by different types of users, and they start adjusting their marketing campaigns based on the results. That's something that's – it's a processing spun very quickly, but you can see the parallel there within IT.

Examples like network trafficking, network traffic data, what you might want to care about is identifying bottlenecks and then very quickly being able to determine that bottleneck and then reaching your network and figure out that bottleneck. Yeah, it's really operators that get the most value out of the product today.

[0:47:57.5] JM: What's the onboarding process like? Does it require an engineer at one of these companies to help the operator set it up, or does the operator take care of everything?
[0:48:07.4] FY: Imply, just Druid aside, Imply deploys one or two modes. It can really deploy on-premise, or it can deploy as part of a cloud offering. Cloud offering is pretty straightforward. All we really need is a source of data to get started with. That source can be something like Kafka, it could be a file system like HDFS, S3, or whatever it is from that source of raw data. Really there's a lot of automation that that gets built from transforming that raw data into a consumable visualization.

[0:48:34.7] JM: What's your work at the company focused on today? We've covered a lot of different areas from the high-level product that these operators are using, to improvements in this disassembled database. Druid, what are you focused on most and where do you see the highest output from input that you can give?

[0:48:53.9] FY: Yeah, based on every aspect of the company, Imply still a relatively early stage company. I'm focused in working with customers, making sure they're successful. I'm focused on defining the broader messaging strategy, creating a lot of content. People understand what we do and understand the value add that we provide, creating use cases of our various customers and also people in the community.

I'm still heavily involved with the community, making sure that we get together, we talk about the roadmap. Druid is an incubation, the Apache Software Foundation, so working at the community on the roadmap, defining what that roadmap would look like and the various new problems that our people are trying to solve. Really, it's cliche to say when you're a startup founder you do everything. Unfortunately right now, you have to do everything just because there's so much to do, and I work with my team on a lot of planning and strategizing as well.

[0:49:48.0] JM: As we draw to a close, are there any open source projects, or developments in the distributed systems community that you're particularly excited about?

[0:49:58.3] FY: There's a lot of technologies I follow very closely. I think there is really this new modern analytic stack that's evolving. The three pieces of that stack I see are a message bus, a stream processor and also a serving inquiry layer. What I believe is what this modern analytic stack will look like, it's something that's end-to-end streaming, it's something that can handle a stream of data, historical data, you can transform the data, you can deliver it into many different systems and you can also power different types of UIs with it.

In that sense, I follow a lot of what the stream processes are up to. I think the Flink, what Flink is doing is pretty cool. I think what Kafka streams is doing is really exciting. Obviously, I follow the message buses as well; Apache Kafka, Kinesis and the various other ones out there. I do look at – I think there's very popular systems that look like Druid, there are also types of operational analytic databases Pinot and ClickHouse and others out there that are part of this general trend of what should the core layer of the stack look like.

[0:51:01.0] JM: Okay. Well Fangjin, I'll let you go. I know you got a lot to do. I want to thank you for coming back on the show. It's been really fun talking to you about all these different topics.

[0:51:08.5] FY: Yeah. I really appreciate you having me on the show. Once again, it was a really great time and just really excited to share all the developments that happened on Druid in the last –

[0:51:16.4] JM: Okay.

[END OF INTERVIEW]

[0:51:20.1] JM: If you are building a product for software engineers, or you are hiring software engineers, Software Engineering Daily is accepting sponsorships for 2018. Send me an e-mail jeff@softwareengineeringdaily.com if you're interested.

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

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

[END]