EPISODE 460

[INTRODUCTION]

[0:00:00.3] JM: Large scale data analysis was pioneered by Google with the MapReduce Paper. Since then, Google's approach to analytics has evolved rapidly. Marked by papers like Dremel and Dataflow.

Dremel combined a column-oriented distributed file system with a novel way of processing queries. A single Dremel query is distributed into a tree of servers, starting with the root server splitting into the intermediate servers and ending with the leaf servers, which talk directly to the file system.

Once the data is pulled from the file system into the leaves, the data propagates back to the root server, and is shuffled along the way so that the root server receives a sorted response. When Google started turning its internal services into customer-facing cloud products, the effort to productize Dremel began, and BigQuery was born.

Jordan Tigani is an engineering lead who works on BigQuery, and he joins the show to discuss the evolution of the data warehouse. Large scale distributed queries still can take a long time, but the queries get faster every year. Queries that required a nightly hadoop job 10 years ago can be viewed in a frequently updated user-facing dashboard today.

Power users of BigQuery talk about the speed and the query interface as being two of the most valuable differentiating features. As the job of a large-scale data analyst becomes less technically intensive, tools like BigQuery will continue to rise in popularity.

We've done some great shows about Google papers and products, like Spanner and Dremel and Dataflow. You could find these old episodes by downloading the free Software Engineering Daily app for iOS and for Android.

In the other podcast players, you can only access the most recent 100 episodes. With these apps, we are building a new way to consume content about software engineering. These apps are open sourced at Github.com/softwareengineeringdaily. If you're looking for an open source project to get involved with, we'd love to get your help.

Shout out to today's featured contributor on the Software Engineering Daily open source project, Shreyans Sheth. Shreyans has worked on the Software Engineering Daily search API and has also helped us to understand some open source best practices, which we are still learning. Thanks again Shreyans for your work, your contributions and your commentary.

Now let's get on with this episode.

[SPONSOR MESSAGE]

[0:02:40.4] JM: Amazon Redshift powers the analytics of your business. Intermix.io powers the analytics of your Redshift. Your dashboards are loading slowly, your queries are getting stuck, your business intelligence tools are choking on data. The problem could be with how you are managing your Redshift cluster.

Intermix.io gives you the tools that you need to analyze your Amazon Redshift performance and improve the tool chain of everyone downstream from your data warehouse. The team at Intermix has seen so many redshift clusters, they are confident that they can solve whatever performance issues you are having.

Go to Intermix.io/sedaily to get a 30-day free trial of Intermix. Intermix.io gives you performance analytics for Amazon Redshift. Intermix collects all your redshift logs and makes it easy to figure out what's wrong, so that you can take action, all in a nice intuitive dashboard.

The alternative is doing that yourself; running a bunch of scripts to get your diagnostic data and then figuring out how to visualize and manage it. What a nightmare and a waste of time. Intermix is used by Postmates, Typeform, Udemy and other data teams who need insights to their redshift cluster.

Go to Intermix.io/sedaily to try out your free 30-day trial of Intermix and get your redshift cluster under better analytics. Thanks to Intermix for being a new sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:25.3] JM: Jordan Tigani is an engineering lead at Google. Jordan, welcome to Software Engineering Daily.

[0:04:30.6] JT: Yeah, thanks for having me.

[0:04:31.7] JM: We're going to talk about BigQuery today. To get people acquainted with what BigQuery is, why don't you explain what a data warehouse is and what people are looking to get out of the idea of a data warehouse?

[0:04:45.9] JT: That's a great question. I think data warehouse has a – somewhat of an old school Pedigree. It comes from people with large data systems that maybe perhaps aren't necessarily large by today's standards, but they tend to have a very specific idea of what type of data they want to store and how they want to store it. They're like the Kimble method of data warehousing of how you structure your data with fact tables, dimension tables.

It's really for that enterprises – enterprises use the data warehouse to analyze reports over their data. It's very - structure is not the right term, but it's very somewhat narrow how they're used. I think BigQuery is a data warehouse. But we're also trying to be a little bit more than a data warehouse.

It's analyzing any structured data without worrying about how it is laid out. You don't need to deviate, you don't need a lot of the database optimization that people use to in their data warehouses, because the way BigQuery is designed that everything is a table scan that we can do things that are usually slow very fast.

[0:06:08.6] JM: I think, data warehouse has a connotation that historically maybe you just use it to get these nightly reports, and it aggregates this data in an offline fashion. With BigQuery, or with other newer more modern data – what we might have called data warehousing systems in the past, now we call them maybe data engineering or something else, we might take data from a production database and put it into the data warehouse. But it might not just be used – this data warehouse might not just be used for nightly reports. It might also be used for actual functionality, application functionality.

How do you think about the usage of a data warehouse for serving using server requests? Is it for that also, or is it just for offline calculations?

[0:06:56.6] JT: I think it absolutely can. I think you get to like one of the – the important part of data warehouse is that it is separate from your online database where you have – your database has the current state of whatever is happening and it's very important to have high numbers of transactions, high numbers of updates per second.

You can almost think of the data warehouse as the log of all the things that happened to that database. You want to be able to analyze that without impacting what's happening on the online side. You build up the nightly reports for the executives to look at it in the morning and they can see how the business is doing.

There is also the ad hoc analysts being able to figure out, they want to ask questions about the data and maybe they're going to do some predictive analytics, or maybe they just want to build some charts, they want to test the hypothesis.

There's also the, can we turn this over to customers? When a customer hits your website, can it run a query? That's also something that's made possible with your data warehouse in the cloud, especially with something that will scale out nicely like BigQuery, where you don't necessarily have to worry about tightly controlling the load or who gets access to it.

[0:08:22.8] JM: The reason I want to do this introduction is to give people a perspective for the past that this data warehouse was largely an offline thing that was hard to deal with, but it's gradually become lower latency, it's become easier to deal with. One of the tools that enables that is BigQuery.

Now that we have that introduction, I want to start to frame what exactly BigQuery is. I think it's a storage system, but it's also a system for querying that storage. Give you explanation for what BigQuery is.

[0:08:57.6] JT: Okay. I mean, I usually say it's a large-scale SQL query engine and a structured data store. One of the nice things about BigQuery is the separation between compute and storage. You can scale your query ability completely separately to how you scale your data storage.

The query engine is this massively parallel, multi-tenant shared pool of servers that basically pick your SQL queries apart, send them out to an individual shards or each one of those shards operates over a small portion of the data, and then then there's a number of aggregation stages depending on how complex your query is.

It's this sort of dynamic shared multi-tenant pool of query resources, and then it's also a structured storage system that scales out to petabytes, many, many petabytes and is designed to manage your data so that it allows fast analysis to a data.

For example, something that – if you have perfected sharded data, you can often can get much faster query performance and it doesn't matter what type of analysis system you're using. That's one of the things that we try to do is people are constantly adding their data over time, maybe in small chunks, maybe in large chunks, maybe you know what, the weekends are the small and the other days are large, or maybe they're streaming an inconsistent rate. We make sure that the data is in the right shape to be fast when the user wants to query it.

[0:10:40.5] JM: The speed of BigQuery is largely enabled by Dremel. Explain what Dremel is.

[0:10:48.1] JT: Okay. Dremel is a internal tool. Actually one of our engineers built when he was essentially tired of waiting for MapReduces to finish. MapReduces are too slow and while he was waiting he was like, "Column stores are cool. I'll build a column store, because I only need access to a few columns out of this wide data set. Then I'll build a SQL parser."

Then of course, since his Google will make it scale out, will make it scale out to you know, as far out as we want. That's sort of just the ethos here is sort of scaling out, versus scaling up. This is sort of a 20% project and he got a couple other people working on it. But it turned into like this – the system that was extremely popular in Google. It's a tree structured analysis.

They mentioned their query, so though it gets picked apart and passed onto the shards, they are these of the lowest level of the tree is where it's sort of directly accesses the data, then aggregations happen in higher levels of the tree. That was the initial version of Dremel.

It since gotten a bit more dynamic, especially since queries don't always just have a single aggregation phase. Dremel is this execution system that, I guess the other key point of it, it also is just this multi-tenant. We can have a much larger tree and devote a lot more resources to each individual query, because we have lots of people using that same tree.

Sometimes one user is going to be using it heavily and sometimes other users are going to be using it heavily, and we can sort of fill in the gaps and end up devoting a lot of computing power per query.

[SPONSOR MESSAGE]

[0:12:46.0] JM: A thank you to our sponsor Datadog, a cloud monitoring platform bringing full visibility to dynamic infrastructure and applications. Create beautiful dashboards, set powerful machine learning-based alerts and collaborate with your team to resolve performance issues.

You can start a free trial today and get a free t-shirt from Datadog, by going to softwareengineeringdaily.com/datadog. Datadog integrates seamlessly with more than 200 technologies, including Google cloud platform, AWS, Docker, PagerDuty and Slack.

With fast installation and setup, plus APIs and open source libraries for custom instrumentation, Datadog makes it easy for teams to monitor every layer of their stack in one place. But don't take our word for it, you can start a free trial today and Datadog will send you a free t-shirt. Visit softwareengineeringdaily.com/datadog to get started.

Thank you to Datadog.

[INTERVIEW CONTINUED]

[0:13:53.3] JM: Let’s save the execution engine and the query serving part for a little bit later. Talk a little bit about the storage. You’re going to store this data in columnar format. Can you explain the advantages of columnar storage?

[0:14:09.4] JT: Sure. Columnar storage, you can think of it as instead of storing in a file, you have – you can store one in row at a time. That’s additional row-based storage. Columnar storage, you could also think of it as storing a single file has one column. We often sort of can catenate those into it into a single file or a single location in a file, but effectively it’s – you have all the data for a single column is contiguous.

The advantage of this are most queries don’t access all the columns in the table. You might have 300 columns in the table and you access four. If you have a row-based storage system, you really have to read all of the columns for every query, which makes it inefficient because often you end up being IO bound, so the speed of which you can read data off of whatever media it’s on is a limiting factor in your query.

You only have to read a few percent of that, that’s going to be – you can get an order of magnitude or two or there’s a magnitude in performance improvement. The next bit is compression, so if we compress the data, that means we have to read less of it, that means we can essentially go – effectively go faster.

The compression of – compression across the row since did not get very good packing, very high compression ratio, because compression works by eliminating redundancy. There’s not a lot of redundancy across a row, where you have a username and a customer ID and an order ID. There’s not a lot of shared information there.

If you look at just at, say the customer – the state in which they live. There’s 50 states in the US, so there’s only 50 fields that that could possibly have, but maybe those are spelled out. Those essentially can be compressed very highly. Our storage system, I can get into say a little bit later, but those are the primary benefits. You can also play interesting games with the storage and get into some more technical details that are opened up by a column store.

[0:16:18.3] JM: In order to put these data in columnar format, are we often taking it from a row-based format like a production database, like my users database and then reformatting in a columnar way in order to store it and serve it in the manner that would be useful to the type of big queries that we're getting?

[0:16:41.9] JT: Yes. Essentially, from the user perspective they don't have to necessarily worry about how we're storing it. They give us the data in the format that's convenient for them, and we support a bunch of formats including CSV and JSON and Avro currently to alpha testing RK as well. All except the latter was our row-based formats. The users are responsible for getting the data out of their database, or wherever they have it stored and into one of those formats.

There is also another option however, which is streaming, and so you can take – Actually long-term, this is going to be probably the most interesting way to write your data. You can basically send a single rose or single clusters of rows and just send an HTV post request to BigQuery. Those immediately get added to your table.

If you send this request, you use 10 rows, you're going to HTV 200 back and then you run a query immediately afterwards. That data will be returned in that query. If you think about large data sets, pretty much all large data sets are created over time. Really they're created – but data comes in some sort of streaming fashion.

What we think of is we often take a days' worth of data and load that at a time. These are just fixed windows on the stream of data that's coming in. I think as we sort of have better and better tools, we're going to just start seeing more people just want to operate over streams and stream their data in and not worry about having to collect it and batch it periodically.

[0:18:24.6] JM: The columnar storage had been used in different ways prior to Dremel being built, but Dremel allowed for some faster and more parallel query response. Let's get into the idea of a Dremel query. Describe what happens in a Dremel query.

[0:18:45.4] JT: Your query comes into BigQuery, we figure out where the table is, where the data is stored, we route it to the right production data center and it starts talking to the – gets assigned a query master. The query master is I guess you can think of it as sort of like a MapReduce master. It's responsible for figuring out how many workers you need, how many partitions are in the table, how to actually plan the query.

It generates a query plan, can talk to the scheduler to basically say, "I need this many shards." Then the query then gets the scheduler assigned some shards that it can run on the shards or the workers. Then the query gets dispatched to the shards.

If you think of, you're just doing like a select sum over some field. The way you can implement that is – way we can do that in parallel is each – you can send that out to a thousand different workers, and each one of those thousand workers reads one file and each one computes a partial sum, and then you basically send it to some sort of aggregator and the aggregator computes the total sum over those partial sums, then you have the final result.

Obviously, that's a very simplistic version. But in general, that's a two-stage query. There is the bottom stage, which is the scan and partial aggregation of the second stage is the final aggregation. In general in BigQuery and in Dremel, in between those stages, the shuffle. It's a fast in-memory simple operation.

You can think of a shuffle as a sort, but you don't always – you generally don't need to do a full sort. It's just a hashings, or the things that look the same go to the same place. In this case, where you have this really simple query, that wasn't needed, but let's say we're doing a group by, we're doing a group by user ID and a sum.

That the group by would mean that all of the values with the same user ID would go to the same – would get shuffled to the same location, and then that one worker could operate over all the values with that ID.

You can do those in parallel and you know that you can operate as if you know all the – you have all the data for that user and that one shard. Then there might – maybe a final aggregation stage as well.

[0:21:19.2] JM: Right. To recap, this Dremel query executes as a tree. The root server is going to talk a collection of intermediate servers, and each intermediate server might be talking with more intermediate servers, and these are called mixers. Then at the bottom of the tree, the leaf servers – they're leaves, because they're at the bottom of the tree, they're querying the distributed storage layer, which is probably Google's file storage layer, the Colossus layer.

Then the results are aggregated back up the tree, then reduced into a simple response. Can we go into the roles of those different node types in more detail? The mixers and the leaf servers and the root server?

[0:22:10.6] JT: What you've described is exactly what treatment the Dremel paper that I think was in 2006.

[0:22:16.8] JM: A little outdated.

[0:22:18.0] JT: We often talk about things in that same way just because they're easier to understand, but for the last I guess three or four years it's been a different system. The original system with the mixtures and the root mixture and the intermediate mixtures and the leaf servers was very fixed.

It worked super well for certain things. It worked super well for just like simple scan filter aggregate queries, but it didn't work very well for things like joins, or things where maybe you want to do an aggregate by – you want to do a group by one key, and then an outer query you do a group by another key, or you are doing analytics queries.

There's a bunch of things that just are required something more complex than this simple tree. You'd end up having a multiple traversals of this tree. The current version of the system were dynamically – I sort of think of it as dynamically building that tree. The schedulers that I mentioned – the scheduler and the root mixture, they figure out how many levels of the tree that you need and we'll build that tree on a per query basis.

We still have the low level, the initial nodes at the root that are directly reading the data and doing the first level of querying. Then the next phases are often doing aggregation. But they also might be doing join, they might be doing a union or they might be doing analytic functions, which often are – would otherwise require a separate traversal of the tree.

It's a much more dynamic system and much more efficient, because you only essentially have to make one pass through the tree. The other thing that is key is instead of – sometimes you can pass data directly from one to the other. In the initial Dremel tree, there is essentially an RPC that was happening from the mix, or the nodes, then are returned with some partial data.

That would cause problems, because if the amount of data that had to be returned was larger, they can be returned in a single RPC and you'd often get these errors, resources exceeded errors.

Now instead of directly passing data back, we pass data back through the shuffle lair, and the shuffle lair lets you expand the amount of data passed in between layers in the node to be much larger.

[0:24:46.8] JM: Now we've talked a little bit more about the distributed query execution engine. What's the ideal way to have this data laid out in the storage system given how we're going to be executing queries against it?

[0:25:02.6] JT: That's one of the things that we try to do is – and sort of the magic of BigQuery, often we have – there is a certain size of data that you want basically each one of these workers to be operating on. If we make that size too small, then we spend all of our time opening files. If we make that size too large, then we don't get much parallelism.

Say if you have a 100 megabyte table and all that data is in a single file and totally operating on by a single worker, you don't get any parallelism. There is the data optimization layer that goes on that tries to figure out, based on query history how should we distribute this data. There's also you can partition your data and you can partition it based on a date field, and that will also determine how your data ends up being laid out on the Colossus.

[SPONSOR MESSAGE]

[0:26:07.2] JM: Dice helps you accelerate your tech career. Whether you're actively looking for a job or you need insights to grow in your current role, Dice has the resources that you need. Dice's mobile app is the fastest and easiest way to get ahead. Search thousands of tech jobs, from software engineering, to UI, to UX, to product management.

Discover your worth with Dice's salary predictor based on your unique skill set. Uncover new opportunities with Dice's new career pathing tool, which can you insights about the best types of roles to transition to and the skills that you'll need to get there.

Manage your tech career and download the Dice Careers app on Android or iOS today. You can check out dice.com/sedaily and support Software Engineering Daily. That way you can find out more about Dice and their products and their services by going to dice.com/sedaily.

Thanks to Dice for being a continued sponsor. Let's get back to this episode.

[INTERVIEW CONTINUED]

[0:27:23.7] JM: Talk more about that file storage layer Colossus and the interaction between the leaf nodes and Colossus. Or I guess, you said not exactly the same naming convention as the leaf nodes anymore, but what's going on in that query layer – the direct layer between the nodes and the storage layer where they're querying it from? You just described the interaction between the leaf nodes and the storage layer.

[0:27:53.8] JT: Sure. Colossus is the successor to Google File System, to GFS. It solves a lot of the – some of the scaling problems that GFS had. Colossus divides up the data access into there's two basically things you want to talk to. One is the curator server, one is the system.

D ends up living – one of the interesting things about the way that the Google data centers are laid out is that we have this distributed storage system and we have this containerized workers, and the same machines that run the workers also in general also host the disks.

We don't really try to do – try to have locality where you keep the data local to the same machine you're operating on. In fact, we started the opposite. We tried to spray it across as many disks as possible. This gives you a super high overall triplet.

If you're reading from all those hundreds of thousands of disks at once, the effective triplet you get is incredibly high, which is how we can – some queries, we can run almost as fast as some in-memory queries, even though we're reading it off of spinning disks.

We have the Colossus curator. The Colossus curator tells us, "Hey, this is where the data lives." We may say we give it a path and has a bunch of files and it says, "Okay, this is where those files live, these are the D-servers to talk to. Then the nodes will generally talk to the D-servers to pull pieces of the capacitor, which is our file format." Pull pieces of the capacitor files back. Data tends to be in originally small stripes.

If you're thinking about the data and a column stored together. One other advantage of a column store is that, let's say we're reading – it's a 100 megabyte file and I'm reading three columns, and each one of those columns is say 1 megabyte.

Each one of those columns is going to come from a different D-server. Instead of having to do three different seeks and three different take the same disk spindle and have it have to read all this data, we could actually read all of that data in parallel.

[0:30:17.2] JM: How does a query on BigQuery compare to a MapReduce query?

[0:30:24.4] JT: It's interesting that they're much a dual of each other. When we had the tree structure of Dremel, it wasn't as clear what the mapping was. But actually, it's quite similar now. MapReduce tends to be map, combine, shuffle, reduce.

The mapper in BigQuery is sort of the shards which you're doing essentially the scanning of the data. We combine and shuffle what happens in the – we actually do have a shuffle involved. Then the reduce would be where the aggregation happens.

Like in MapReduce, when most interesting problems you want to solve, most interesting things you want to ask of your data involve – can't really be represented in a single map in reduced phase. We might end up having map, map, shuffle, join, reduce, and then join again and then reduce again. That's supposed to be a little bit more flexible, but I think you sort of modern MapReduce systems also can do the same thing.

[0:31:34.1] JM: Can you maybe give an example of a query that would be a good fit for Dremel and one that would be a good fit for MapReduce?

[0:31:40.0] JT: I think that anything that will be good for MapReduce would be good for Dremel. I think writing out large amounts of data – that's not even true anymore. It used to be that writing large amounts of data and do better in a MapReduce would be in BigQuery, but we can write out data from the shards. So I think there would be advantage using one versus the other.

MapReduce tends to be more static. Basically you spin up a pool of workers, and you have this sort of static pool that you own completely and it's used entirely in this computation. Especially if it's a large computation, you may only need all of those workers for a small portion of the operation. But yet, you're stuck with them.

You either have to over-allocate the numbers of workers you have, so that the one phase that needs all those – that is super wide can act to do quickly, or you under-allocate in which case like you don't have a lot of idle workers sitting around, but the – your part of the query that's most data intensive may actually get more slowly. That's the other one of the advantages of BigQuery we have in Dremel is this dynamic tree that we will allocate different numbers of shards for each stage of the – each stage of the query and we can actually dynamically adjust those. Then the rest of them, essentially can get used by other queries.

[0:33:05.5] JM: For people who are new to the concept of a data warehouse, or who are new to BigQuery, talk a little bit about the usability and how – the patterns where people are using this. For example, how often – if I have a production user database and it's getting updated with new user data every day, and got a user interactions database as well, and I want to pull this data into BigQuery on some schedule, and then I want to be able to run analytics queries on it. How am I using BigQuery? How do I set it up to pull that data from my production database into essentially my data warehouse or whatever you would call it, the BigQuery storage layer.

[0:33:54.0] JT: That's something that I have a little bit less insight into than perhaps some of the internals. But we see a wide range of how people are doing this. I mean, some people are they're generating events and those events get streams directly to BigQuery. Sometimes they have an hourly or a nightly batch process that extracts the data that they need and loads it into BigQuery.

I think what we often find is that the period decreases over time. People will often start with, "Okay, I'll take my daily dump of my database and load it into BigQuery, or I'll take these large batches." Then over time they realize, "Oh, it would be great if I can have things that were within four hours." We'll have every four hours.

Then they'll be like, "Well, four hours. Why not four minutes?" They'll do smaller batches. Then four minutes, why not four seconds? Then they'll start using a streaming system.

[0:34:57.3] JM: Yeah, makes sense. BigQuery is this public productization of what was an internal tool in Dremel, and at this point Google has done this with several products that deal with Kubernetes, but Kubernetes is a rebuilding of the Borg project and tensorflow, but tensorflow was – I guess tensorflow was a rebuilding of the disk belief project to become open source.

I suppose those open source projects are a little bit different. But in any case, this process of productizing and externalizing internal Google services has become a pattern at this point. What has Google learned or what you learned specifically about productizing internal projects?

[0:35:44.6] JT: I think one of the things that we learned that it's hard. When we first started the BigQuery project we were like, "Yeah, we'll have this done in a quarter." In fact, we've seen that over and over again. I don't want to call anybody out, because you know, but there were a number of projects where they've been like, "Yeah, well this is a very mature internal products. This works very well. We'll just put an external API on it, and then that's it. No problem."

I think the thing that we've learned is that it's hard. I mean, there's a number of reasons for that. One is there is just an impedance mismatch things, just in terms of how Auth is done internally and how Auth is done – authorization and authentication are done externally. Those are the things that are frustrating. But you can't really get her out.

I think we've sort of got better at that at – as our systems at Google are converging, we have better ways for dealing with that. But certainly the time – I used to say there's some quote – this is like the only hard problem in computer science is naming. I've added an authentication. Because it sounds like it's relatively straightforward, but ended up being difficult.

Other hard things though were – At Google, all these internal monitoring and systems are in place so that when something goes wrong, you can figure out what happened. There's a status pages of the board workers and there is all these tools that people are used to digging into what happens when something goes wrong.

That's really hard to replicate for external users, because you don't have that level of detail. Moreover, you don't want to give them that level of detail because – I mean A, there's a security risk of letting people know like map out your data centers. B, you want to give them some higher level representations of what happened.

Partly so they don't have to worry about the details, but also partly so you can change them in the future. That was a major difficulty. We're still sort of dealing with that. We see people and we're using BigQuery, and when something goes wrong, they don't really know how to solve their problem or how to allocate.

[0:38:14.6] JM: It's serverless.

[0:38:16.0] JT: Yeah, exactly. The problem with magic is that when the magic doesn't work, you're left with not knowing what to do or how to make it magic again.

[0:38:26.8] JM: I could just imagine the engineer who joins the BigQuery team and is thinking, "I can't wait to work on the new query execution engine." Then it's day one, here working on authentication for the next three months.

[0:38:42.4] JT: I think one of the cool things is that there are all kinds of engineers, and some people really want to work on the guts of the SQL engine. Some people like solving security problems, and some people like – more customer-facing side of things. But certainly, there are some pieces that are more sexy than others and some stuff that just – has to get done.

[0:39:08.2] JM: Yeah. Well, I guess final question. When you look back at the evolution of how BigQuery has evolved since the Dremel paper, what are the biggest milestones that you would point out and how the – we are talking about the actual – the guts of the file storage and the query execution layer, where do the milestones – what were the big breakthroughs of figuring out new ways to do this more efficiently?

[0:39:33.6] JT: There was Dremel X, which was a major effort to reroute the guts of how execution happens in Dremel. That was really the thing that changed it from being this static tree that's in the Dremel paper to a more dynamic one.

There was capacitor, which is our new columnar storage format. That was – I think we had some 10X improvements from that. One sort of milestone just was the – it was a team one, but we merged the Dremel team and the BigQuery team. That was done several years ago, but instead of thinking of BigQuery as a layer on top of Dremel, it became more of a collaboration.

Especially as the core Dremel folks realized it, "Hey, this is our primary customer, our primary use case now." That enabled us to think much more deeply about how to solve enterprise customer problems as opposed to how do I solve the problems for the ads teams.

One that I think we learn is just how different external customers are from internal ones. There was a lot of disbelief about that in the beginning. Yeah, we can handle multi-padded queries. There's no problem these external customers are much small data. Won't be a problem. It turns out that it isn't always reduced that way.

[0:41:03.3] JM: Jordan Tigani, thanks for coming on Software Engineering Daily.

[0:41:05.2] JT: Thank you very much. Appreciate it.

[0:41:06.4] JM: Wonderful.

[END OF INTERVIEW]

[0:41:11.8] JM Simplify continuous delivery with GoCD, the on-premise, open-source, continuous delivery tool by ThoughtWorks. With GoCD, you can easily model complex deployment workflows using pipelines and visualize them end to end with the value stream map. You get complete visibility into and control over your company's deployments.

At gocd.org/sedaily, find out how to bring continuous delivery to your teams. Say goodbye to deployment panic and hello to consistent predictable deliveries. Visit gocd.org/sedaily to learn more about GoCD. Commercial support and enterprise add-ons, including disaster recovery are available. Thanks to GoCD for being a continued sponsor of Software Engineering Daily.

[END]