EPISODE 774

[INTRODUCTION]

[00:00:00] JM: In the early days of YouTube, there were scalability problems with the MySQL database that hosted the data model for all of YouTube's videos. The state of the art solution to scaling MySQL at the time was known as application level sharding. To scale a database using application level sharding, you break up the database into shards, which are disjoint regions of data.

When you query the database from your application, you need to know which shard to query. In your application code, you have to issue the query to a specific shard. The solution of application level sharding does scale your database. It allows you to have your application and your database scale appropriately. But the downside of this approach is that every application that interfaces with the databases now has to include code that is aware of the sharding schema.

If you're an application engineer, you don't want to have to worry about the way that the database is sharded, because that adds significant complexity to your code. The engineers at YouTube decided to fix this problem with a project called Vitess. Vitess abstracts away the details of sharding by orchestrating the reads and the writes across the distributed database.

In a previous episode we covered the architecture and the read and write path and the story of Vitess in detail. In today's episode, Jiten Vaidya and Dan Kozlowski of PlanetScale Data join the show to give their perspective on MySQL scalability and their work taking Vitess to market as a solution to scaling relational database. PlanetScale Data is a company built around Vitess. So they sure have a lot of context into how Vitess works, and more generally how database scalability works.

It was a great episode, very technical, and also some elements of less technical details. I hope you enjoy it.

[SPONSOR MESSAGE]

[00:02:08] JM: STELLARES is a job recommendation engine for software engineers. STELLARES finds career opportunities that fit you perfectly by taking all your preferences into account. The STELLARES process starts with a conversation. You text with a conversational interface and the chat bot helps you narrow down what you are interested in. Then after you have given STELLARES your preferences, STELLARES uses its machine learning algorithms to factor in the subtle aspects of the job search so that you find your perfect job, from salary, to work-life balance, to team fit and personal learning goals.

To find out more about STELLARES, go to stellares.ai/sedaily. That's stellares.ai/sedaily, S-T-E-L-L-A-R-E-S.ai/sedaily, and after you go through the job matching process, STELLARES gives you a warm introduction to the engineering teams that you're interested in so there's never any pressure, just opportunities to explore what's out there. Check out stellares.ai/sedaily and find a new way to look for a job.

Thanks to STELLARES for supporting Software Engineering Daily.

[INTERVIEW]

[00:03:37] JM: Jiten Vaidya, you are the cofounder and CEO at PlanetScale; and Dan Kozlowski, you are the lead engineer at PlanetScale. Guys, welcome to Software Engineering Daily.

[00:03:48] JV: Thank you.

[00:03:48] DK: Thanks for having us.

[00:03:50] JM: I want to start by talking about MySQL scalability, or more generally, SQL scalability. When a MySQL database is not scaling well, what kinds of performance issues does that lead to?

[00:04:05] JV: Typically, a MySQL database is used for OLTP transactions for apps that a user is interacting with in real-time. What that means is that somebody refreshes their profile and they're sitting there trying to see the changes on the screen of the app or in front of their computer, and it takes a longtime for that little spinning wheel to give them back the changes that they believe that they've just made. That's just one example of how it will manifest in the app.

[00:04:37] JM: And does the scalability of a MySQL OLTP database compare negatively to that of a NoSQL database, like MongoDB?

[00:04:49] JV: So at least in theory, NoSQL databases like MongoDB could be scaled horizontally by adding more shards or more machines, and a single instance MySQL database, you can scale it only by adding more hardware to the node on which that particular instance is running. So there are some practical limitations to how big a node you can make for a MySQL database.

[00:05:17] DK: Yeah, and I would add to that and say that a lot of what you'll see with NoSQL databases is a very similar performance profile for OLAP queries as you would get out of a traditional MySQL database. But like Jiten said, the real problem happens when you scale up the database, you could only buy so much hardware. Disks only scale so much, CPUs only get so fast, and the real thing that NoSQL database took advantage of was an inherently more efficient architecture. It was the fact that it was easier to horizontally shard them, which meant you could add more CPUs, you could add more disks than what a single server would allow you to do.

[00:05:57] JM: Why are NoSQL databases easier to horizontally shard than a MySQL database, and could you very quickly describe what horizontal scalability is in contrast to vertical scalability?

[00:06:09] JV: Correct. So horizontal scalability is when as Dan said, you add more hosts, which means more CPU, more disk to your cluster. The reason that it's easier to do that with NoSQL databases is because they do not provide some of the inherent guarantees that a relational database provides, such as atomicity, consistency, isolation, durability, or as we all know, asset, right? It's really hard to have a distributed system which is running on multiple hosts which can provide you with these properties. Key value stores typically, at least the sharded key value stores, they do not provide you that guarantee.

[00:06:55] DK: Yeah, and I mean a lot of it comes down to how the original NoSQL databases were put together. They were put together on a single, what we would call, primary key, and it's pretty easy given a single primary key to have some mechanism to distribute that across multiple machines. But when you talk about a full relational database where you have multiple tables that have a complex and arbitrary relationships with each other, then you start to get to the question of, "Well, how do I put data that is related to each other in the same place if all I have is a single primary key that is what I'm going to use to send data to different places?"

[00:07:32] JV: Exactly. So what that means in relational parlance is that you don't get secondary indexes. You don't get transactions, and so it's easier to horizontally scale NoSQL key value store databases.

[00:07:45] JM: What are the traditional approaches to scaling a relational MySQL database?

[00:07:51] JV: Typically, when companies have grown out of a single node, what they end up doing is that they shard in app, which means that in their application, they add logic, which says that if there are 100 million customers, if the customer ID is between, say, 0 to 10 million, send it to shard number one. They actually end up having multiple connection pools logic in the app, which looks at – Before the query is even created, it looks at the user ID or whatever your sharding key is, and then sends the query on the correct connection pool. That's what we ended up doing at YouTube before Vitess was invented at YouTube, and it really makes the application more complicated and there is an ongoing cost in terms of – You have to live with that complexity of what every new feature that you're going to add.

[00:08:46] DK: I'll also say the other route that people will take is depending on your workload, you could get away with just read only replicas and dealing with eventual consistency of your data. So if you have a system where it's mostly read only traffic and it's okay if it's eventually consistent, then one of the traditional routes you would take for a relational database is to put a bunch of read only replicas. It's effectively horizontally sharding, but it's only horizontally sharding your read traffic. So that does give you more capacity, but then you are write bound.

So you can only ever write to a single node, and then that would get replicated to a bunch of other instances. But if you ever get into a situation where you have write traffic that is going to exceed the capacity of a single node, then you would have to do, which Jiten said, of start to jump through all sorts of hoops in application to make your single database actually be a set of databases that the app knows how to route it to.

[00:09:47] JM: I want to zoom in on that idea of application level sharding, because in the last episode with Sugu, we talked about how much overhead this adds to the software development team as a whole, because if I am a microservice developer and I'm writing a service that serves YouTube recommendations to a user, first of all, I have to write some logic that says, "Is this user in the first shard or the second shard, the third shard," and you have to write maybe a switch statement, and then if you are gathering data from other people that that person is related to in order to calculate their recommendations, then you have to say, "Okay. First, fetch the three people closest to this person and, oh by the way, that's going to require some sharding logic." Then maybe the further you go out on the person graph, the more sharding logic you're going to have, until your application code looks more like code to navigate through shards of a database than it does to calculate the recommendations of a user.

[00:11:01] JV: Absolutely. I think that was a very good example that you gave that describes the kind of engineering debt that you incur if you decide to shard in app, and not only that, you are always going to have some queries which don't have the user ID in the query. So either a query or a transaction, and in that case you need to have some mechanism to do a scatter/gather across all the shards. So as you described, rather than writing your feature code, you end up writing this code that is juggling database connections and parallel queries and so on.

[00:11:40] DK: Yeah, in addition to all that, the other sort of hidden cost that you incur as the engineering team isn't even about the sharding. It's about the re-sharding, because at some point in time you're going to say, "I'm going to write my applications that it has two shards," and then you're going to outgrow that and you're going to say, "Now I have to do it to four shards," but if you're already spanned all of the users, now you have to do something about those two databases that have data in them, turning those into four databases.

Perhaps there's a way that you can do that without having to migrate a bunch of data, but a lot of times what you then have to do is take a downtime, take a maintenance window where you're migrating the data and deploying your app in parallel at the same time. So now not only have you got extra engineering effort to keep your complex code up-to-date, but you have extra engineering effort to maintain your database cluster and you are probably looking at maintenance windows to migrate data, which is only going to get worse as the amount of data you have increases.

[00:12:41] JM: Jiten, in the previous conversation with Sugu, we talked about the scaling challenges of YouTube and how this application level sharding just started to cause so many problems that you and him – Or I can't remember who started the project, maybe it was somebody else at the company, started basically a project to write middleware to take care of all of these sharding logic and push it down into a middleware database transaction layer. Can you describe more what was going on in that middleware layer, which I think eventually became Vitess?

[00:13:25] JV: Right. Right. So it was Mike Solomon and Sugu who started the project right around 2010, and the first problem – So at that time, I think Sugu might have explained this to you last time, but Mike and Sugu sat down and made a large spreadsheet which described all the problems that we were having around sharding and this database access layer. They sort of took them out of the day-to-day firefighting.

Basically, one thing that I always like to say is that we are too busy mopping the floor to fix the leak. They decided that they will going to exclusively work on fixing the lead rather than mopping the floor. They made this list of the problems that we were having and looked at it and decided to come up with a middleware – I mean, the conclusion that they came to was that there were a few design principles that they could use to write this middleware layer, which could pretty much solve all the problems that we were seeing around sharded MySQL databases, and that's how Vitess was born.

The first problem that they concentrated on solving was connection pooling. We had I think hundreds of app servers at that time. Maybe we were not quite at thousands. So we were running out of connections in MySQL, and MySQL runs poorly if you have created too many connections to it. That was the first problem that they wanted to solve.

I think the very first piece of Vitess code was written to solve that problem, but they had this overall design in their mind right from the beginning.

[00:14:59] JM: How did the YouTube infrastructure have to change in order to get this middleware layer inserted in between the application developers and the database layer?

[00:15:16] JV: So the infrastructure team at Google always owned this layer, which sat in app, but which was used by the developers to access the databases. So it was not that much of a leap to add this middleware in between and have that layer now communicate to the middleware rather than directly to the databases. So I think that's what we ended up doing when Vitess was deployed first.

Initially, YouTube's databases used to run on YouTube's own data centers. So when YouTube was acquired, all of the pieces of the distributed system that serves our YouTube used to run on YouTube data centers. But very early on, the endpoints that had the maximum amount of QPS were moved out to Google's Borg infrastructure. But for a long time, the MySQL databases still continue to run on YouTube's data center, and we were accustomed to treating them as pets rather than cattle, as many times databases are, because a database master going down is something that traditionally needed an intervention by a DBA. That's where we were at at that time.

[00:16:35] JM: When you were at Dropbox, which was a period of four years sometime after YouTube, how did your experience with Dropbox's infrastructure compare to those of your experience at YouTube? Did you feel like, "Oh, Dropbox has problems that relate to my experience at YouTube," like relational database issues."

[00:16:59] JV: Yes and no. Yes, because the funny thing was that there were four folks from that database. So when I left YouTube in 2012, I was managing the site reliability engineering and DBA teams at YouTube, and in between I worked at a small startup called [inaudible 00:17:20].com for about 16 months, and in those 16 months, I think four folks who used to report to me at YouTube had left YouTube and joined Dropbox, and they were solving similar problems at Dropbox.

When I joined Dropbox, I could have joined them and worked on those problems or I could have done something different. I ended up not working on database related problems at Dropbox. But yes, Dropbox had large MySQL installations and they had also sharded in app, but I think they decided to go with larger number of shards and all the metadata related to the files, the distributed file system that Dropbox has actually ran in sharded MySQL clusters, which was managed by one of the senior DBs [inaudible 00:18:16] who used to be on the DB team at YouTube and had joined Dropbox in between.

[SPONSOR MESSAGE]

[00:18:31] JM: Today's episode of Software Engineering Daily is sponsored by Datadog, a monitoring platform for cloud scale infrastructure and applications. Datadog provides dashboarding, alerting, application performance monitoring and log management in one tightly integrated platform so you can get end-to-end visibility quickly, and it integrates seamlessly with AWS so you can start monitoring EC2, RDS, ECS and all of your other AWS services in minutes. Visualize key metrics, set alerts to identify anomalies and collaborate with your team to troubleshoot and fix issues fast.

Try it yourself by starting a free 14-day trial today. Listeners of this podcast will also receive a free Datadog t-shirt. Go to softwareengineeringdaily.com/datadog to get that t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[00:19:37] JM: There is this idea that in the early years of Web 2.0, if you consider we're still in Web 2.0, there were a number of companies that were facing scaling relational databases. They had invested in relational databases early on. They had scale, and all of a sudden now they've got a database scalability problem. They invest in sharding infrastructure, and then they start to have these issues that we talked about earlier with the application level sharding and they just live with it and it's painful, but it's this sort of looming technical debt that is infecting the company.

Then at a certain point, you and Sugu, who was the previous guest, took a step back and you said, "We solved this problem at YouTube and people are still having it, and we should probably start a company around solving this problem. Was there some breaking point where you just said, "Okay, too many people are still encountering this problem of scaling relational databases. We have to start a company around it." Was there some kind of insight you had that changed your mind to starting the company?

[00:20:57] JV: I think the turning point was companies that were facing this problem starting to discover Vitess with very little promotion by anybody and starting to use Vitess in their production infrastructure to solve these problems. So around 2015, Flipkart was the first company to discover Vitess joined the Slack channel and started directing with Sugu so that they could deploy that in their production infrastructure to solve their sharding problems.

Slack came along, Square came along. So both of us actually I remember going to Slack and doing a presentation at Slack. I forget when that was – I think with Sugu early 2017, late 2016, I forget, and they wanted to make a decision about how they should – They were already sharded and they needed to make a decision about whether they should build this middleware layer themselves or whether they should be using something like Vitess.

I think Sugu sort of – I think the pitch that Sugu gave them was it's not a build versus buy. It's open source. So it's buy and contribute, not buy, but use something which is already built and then contribute back. I think that made sense to Slack. Long story short, people like Slack, people like Square were seeing that this is solid production-ready open source project that they could use in their production, and both Sugu and I felt that more and more people needed help doing this. So it was very clear to us from the inbound queries that we were seeing on the Slack channel that people needed help.

[00:22:46] JM: Now, what's worth pointing out is that Vitess is not just a solution to scaling MySQL database. It is potentially a solution for scaling all kinds of things that need sharding. Is that correct?

[00:23:06] JV: That is correct. The Vitess architecture is such that it sits very nicely on top of MySQL. The points which are MySQL specific are very nicely abstracted in pieces of code which can be replaced with other pieces of code that can interact with other relational databases for example that might need sharding. We have looked at what it would take to support something like PostgreS, for example. So we will need to rewrite the binary protocol that the client uses to talk to VTGate, which is our stateless proxy. That's the first point that the app connects to. That's one place that we would need to change. The second is one of the coolest features of Vitess, is this ability to consume the binary replication log, apply sharding logic to it and start writing to new masters, thus allowing you to do resharding without any downtime for your app.

So what that would mean for PostgreS is that I think PostgreS does its replication using write ahead log. So we will need to figure out how to read that and apply sharding logic to that. So as long as – It's 6 to 9-month man months' worth of work would allow us to sort of put Vitess on top of PostgreS and allow us to shard PostgreS. Yes. I mean, to answer your question, it's an architecture that can be used for sharding many different things.

One more thing that I would like to point out is that sharding and horizontal scaling is just one of the benefits of Vitess. There are two other reasons why people decide to use Vitess. One is it allows people to run stateful workloads under an orchestration framework like Kubernetes. The reason that Vitess can do that is because I think earlier in the podcast I described to you how we used to run databases early on in YouTube's own infrastructure.

I think around 2012 or 2013, we decided to move those databases from YouTube infrastructure into Google's infrastructure, which ran on Borg, which is the predecessor to Kubernetes. Now, under an orchestration framework, you cannot take – As I was saying, under the orchestration framework like Kubernetes or Borg, you cannot take the longevity of the container on which your master is running for granted. It can get de-scheduled and you need to have a really good master failure story, a really good service discovery story and a really good observability story to be able to run stateful workloads, like database and orchestration framework like Kubernetes, and all of that was built into Vitess over a period of about 9 months, which Sugu described as a very painful period of 9 months.

But at the end of it, Vitess allowed us to run YouTube database on Borg very, very well, which means that it allows people to run stateful workloads in MySQL on Kubernetes very, very well now. Companies like HubSpot use Vitess entirely for that reason. They actually have – I think they use one MySQL cluster or database per tenant. So they don't need horizontal sharding of databases, but they still use Vitess because it allows them to run Vitess on Kubernetes. So that's the second reason why people use Vitess.

The third reason is that every time YouTube went down because somebody wrote a bad query, it's really easy for an inexperienced developer to write a query that would do, say, a full table scanning using a column that doesn't have an index on it and bring a whole instance down. Every time that happened, we wrote code in VT-Tablet that allows VT-Tablet to protect your MySQL instance against that, and that's the third reason why people use Vitess.

[00:27:20] JM: As a side note, I want to say that we did cover the architecture of Vitess in detail in the episode with Sugu, and also the documentation and some of the YouTube videos about how Vitess serves a query and what it's doing under the hood are quite good. So for people who do want to go into the depths of how this architecture works, you will not be disappointed by the material that's already out there.

So what you just said about managing stateful workloads on Kubernetes, we just did a show about that very topic and people really liked it, because this is a topic where if – I mean, I hear all the time in the Kubernetes community that it is hard to use Kubernetes for stateful workloads, and stateful workloads is a broad statement. Can you explain in more detail what that term means? What is a stateful workload on Kubernetes and why is it hard to do with existing infrastructure aside from Vitess?

[00:28:28] JV: Right. So a stateful workload basically means that any workload where some data needs to persist across the birth and death of the part or container on which that particular process is running, right? Typically, when you have a process started in a pod on something like Kubernetes, it docks to other microservices to initialize itself or it has all the data in the command line to initialize itself. It starts serving when the part goes away, the process goes away, you start another part. It also initializes itself and starts serving. It's not dependent on any information that's inside the pod. So that's what a stateless service or a stateless workload looks like.

As against that, a stateful workload is a workload which is writing something to a volume that is visible inside the pod and there is an expectation that the data written on that volume would outlive the lifecylce of the pod itself. I think that's sort of the fundamental difference between the stateful workload and the stateless workload.

[00:29:45] DK: Well, I mean I think it actually would go – Part of the reason why Kubernetes has such a problem with stateful workloads is because the language that we use when we talk about Kubernetes is things like pods, and volumes, and processes, but at the end of the day, a stateful workload is I have some information. That information is going to be available no matter what happens.

So a user who wants to put – We normally get this on databases by treating them very well, making sure that they are very special, that they have lots of protections against the lesions or against hardware failures. But when we get into Kubernetes, Kubernetes from the ground up was designed as a system where anything could go away.

So as they started building in persistence, you still have this disconnect, where we'll have a persistent volume. Yeah, but that persistent volume has to be attached to a node and it's going to be communicated with over whatever mechanism the pod has to talk to that volume. So you still get into a situation where if the node goes away, you've lost your data and you've sort of violated that unspoken that contract of a stateful system.

One of the things that Vitess did to allow actual stateful systems on Kubernetes is they brought it back sort of completely out of Kubernetes control of how the information was going to remain under any circumstance. So you have protections built in for a volume getting corrupt. You have protections built in for a pod getting terminated, for a node going down. Even for an entire data center going down, you have the ability to continue to have access to your data and continue to be able to serve that traffic. That's really what you need for a stateful system.

So as Kubernetes gets more mature, you're starting to see more solutions where we could tolerate pod failures but still have access to our data. We could tolerate node failures, but still have access to our data. Right now what the marketplace looks like is there's really nothing out there built in to Kubernetes that actually satisfies that contract for a stateful application. You still have to handle stuff manually outside to get reliable and consistent access to your data.

[00:32:02] JM: Tell me if this is a way of boiling what you both just said. With Kubernetes, you are introducing more components that are required to have a persistent transaction fully occur. There're moving parts. There're more abstractions. Therefore, there are more partitions that can occur across a transaction. So in order to have partition tolerance, you have to be able to cover more cases for a given stateful transaction. Is that right?

[00:32:46] DK: Yeah, I think that word partition tolerance, that's a fantastic way of describing it, because that is what you have. You have now created a bunch of extra connections that can fail. You have to be able to handle all of those partitions. It's no longer just a network partition. You have pods that could go down. You have nodes that could go down, and these all represent partitions that you would have to handle.

[00:33:12] JM: Okay. So what has been your interaction as other people have adapted Vitess? As you've seen companies like Square and Slack, these companies have high-volume workloads that are dissimilar from YouTube. They are similar in some ways, but they probably put new strains on Vitess. Were there any interesting changes to Vitess that happened as companies like Slack and Squre and HubSpot adapted it?

[00:33:47] JV: That's a great question. So I think a lot of new features got built. Definitely Vitess has become a more general purpose solution as these companies have started using it, but I cannot think of anything that was – I can't think of things that were built by these companies, because they hardly needed it, and pretty much everybody in the community said that, "Yes, this is a great idea. Let's do it." But the architectures of Vitess is such that I think – I don't think that there is something that was built into Vitess that solved one particular use case for one particular user and was not useful for other users.

[00:34:29] JM: When we talk about stateful management on Kubernetes, what does that include beyond the database layer? I can think of other things like Redis, which is kind of the in-memory data store, which I guess is kind of a database, but maybe there are other kinds of workloads other than databases that put different constraints on Vitess. Is there anything beyond the database as a service workload that you are considering?

[00:35:02] JV: At the moment, no. We are focused on relational OLTP workloads on Kubernetes as database service. So now you're asking as planet scale, right?

[00:35:14] JM: Yes. Right. Yeah.

[00:35:15] JV: So that's what we are focused on right now. So Sugu just recently finished working on a feature called VReplication, and that's a very interesting feature, because that will allow people, that will make some OLAP workloads really easy. Let me just quickly describe what it is. So I think earlier in the podcast I described to you that Vitess has the ability to consume the binary replication logs from the master, apply sharding logic to it and write – Split that replication stream and write it to multiple masters.

So we took this and sort of pushed it one layer below in the architecture and made it composable, and what that allows us to do now is the ability to provide what we call multi-shard materialized views. So what I mean by that, I'll just give you a quick example. Let's take the example of a marketplace, which let's say there are like hundreds of millions of users, tens of millions of merchants and orders which have a user ID and a merchant ID both.

So at that scale you typically probably ended up having user sharded using user ID and merchant sharded using merchant ID and now you need to make – You have a question, "What do I do with my orders? Do I shard them using user ID, or do I shard them using merchant ID?" Typically, let's say you looked at your query pattern, most of your queries were using user ID. You decided to shard the orders using user ID.

So orders for a particular user, orders live within the user shard with the user. Now what happens when a query comes where it says, "Give me all the orders for this particular merchant." This typically ends up becoming a scatter/gather. Vitess deals with it, but it sends the query to all the user shards, gets the results back for that particular merchant ID, combines them altogether and sends it back to the app. So this is not efficient.

So what VReplication allows you to do is if let's say that there are M-shards of using user ID and M-shards using merchant ID, we start M-cross and replication streams. So the application is still writing orders into the user shards, but there is a replication stream which is replicating the same data in the merchant shard using the merchant ID that was just written to the user shard, right?

So what you'd get is that now if you want to do a merchant ID query, which you don't need read after write consistency, but you can live with eventual consistency. Now this query can be sent to merchant shard and would be satisfied by a single merchant shard. It doesn't need to be a scatter/gather. So you basically get a materialized view of orders sharded according to the merchant ID.

Not only this, but this is really useful when you are trying to do rollups, because many OLAP queries, you are basically counting in various intervals and so on, and all of that can also be done as a part of a replication stream and you don't need to have all these bad jobs and so on which are doing aggregations for you or a completely different system for running these OLAP queries. It's a pretty cool functionality.

[SPONSOR MESSAGE]

[00:39:02] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

Go to softwareengineeringdaily.com/HPE to learn about how HPE OneView can improve your infrastructure operations. HPE OneView has easy integrations with Terraform, Kubernetes, Docker and more than 30 other infrastructure management tools. HPE OneView was recently named as CRN's Enterprise Software Product of the Year. To learn more about how HPE

OneView can help you simplify your hybrid operations, go to softwareengineering daily.com/ HPE to learn more and support Software Engineering Daily.

Thanks to HPE for being a sponsor of Software Engineering Daily. We appreciate the support.

[INTERVIEW CONTINUED]

[00:40:26] JM: What you're describing here, because I asked a kind of awkward question around what else you're looking at in terms of stateful workloads. You're talking about OLAP style queries, and there are these kinds of queries, OLTP verus OLAP. These are kind of general industry terms, and we've covered them on previous episodes. The best episode I can refer to understand OLTP versus OLAP is an episode we did with Uber, where they talked about their system for getting transactional data into an analytic data store.

Basically, what a lot of companies have built is they have this OLTP database, which handles the highly consistent up-to-date transactions that people need to engage with, like if I'm getting in an Uber car and my payment needs to process and I need to make sure that like I'm in constant contact with the Uber database so that if I make a trust and safety complaint, it's like very quickly recognized by Uber. That's the OLTP style workload, but then you also have the OLAP style workloads, where you need to calculate large scale analytics across an aggregation of all the users that are in North America, and that is a very different query pattern than this particular user row and update that user's row.

The difficulty of OLTP versus OLAP is that you often times have to take an OLTP database and copy it into an OLAP database, and so then you have another area of inconsistency, because if I have to copy my entire transactional database into the OLAP database, then the time that I'm spending doing that, there's additional OLTP transactions that are going on. The OLAP database is now out of data. So you're going to constantly have this ETL job that is creating a gap inconsistency between OLTP and OLAP.

So some of the "NewSQL companies", which maybe you put yourself in that category, you're building functionality to have the OLAP queries be able to be served by the same database as the OLTP queries. Is that right?

[00:42:47] JV: I think that was a really great summary of what we are building and what this enables.

[00:42:53] JM: So we've talked to a number of these NewSQL companies that are kind of attacking this problem, because this is a gigantic problem, the OLTP versus OLAP problem. So there are companies like Citus Data, which was acquired yesterday, TiDB, which we talked to recently. We've talked to VoltDB. Can you describe how approaches to solving OLTP versus OLAP vary across the different companies that are tackling this?

[00:43:25] JV: TiDB architecture I'm somewhat familiar with. I'm not very familiar with Volt. Dan, do you have an answer to that question?

[00:43:34] DK: Yeah. I mean, so if you look at a lot of the NewSQL databases, like some of the ones you mentioned, a lot of what they have done is taking a very well-proven persistence layer. So in the case of TiDB, they use a system built off of Rocks, I believe, and they have then written the sharding and management layer on top of that.

Vitess has an extremely similar approach solving this problem. We've taken as the base layer MySQL and we have built our sharding layer on top of it. I think the major difference that you'll see is kind of how these products came to market. If you look at the history of Vitess, Vitess was built out of a need at YouTube to shard their data and all of the design decisions and the features that are in Vitess are traceable back to actual things that occurred at YouTube that required some bit of software to solve.

I think a lot of these other systems are solving the same problem, but they're sort of coming at it from the other direction. They see that there is this differentiation in query types and they know the problem can be solved. So they're going out to solve it. It's not a very stark difference, and you'll see that a lot of what is done is done similarly. But I think what you'll get out of Vitess is the major difference is the amount of emphasis that is put on management and maintenance, not necessarily on achieving the highest queries per second or the highest throughput. It's really about a system that can be ran, can be ran at scale and can serve traffic at scale under any circumstance.

[00:45:14] JV: Right, the production readiness of it. As Dan said, I mean, we came to it very clearly from the OLTP side of things. We wanted to build a system that scaled OLTP, and now we are discovering that it can also support OLAP quite nicely. We haven't really figured out how we are going to build – Take it to the user so that it's easy for them to run OLAP queries, but it's going to be fairly straightforward, because [inaudible 00:45:41] interfaces are very well-known and well-understood interface.

Between scaling OLTP and scaling OLAP, I would say that scaling OLAP is an easier problem than scaling OLTP. Now having saw – The tradeoffs [inaudible 00:45:56] are different. Maybe I should not say that it's an easier problem. I should just say that the tradeoffs are different.

We just believe that with this new VReplications opens up the possibility for us to easily support OLAP workloads, which until about six months ago Vitess was not very great at supporting.

[00:46:16] JM: Dan, since you alluded to this, and I also heard Sugu talk about this a little bit in our previous episode, TiDB uses RockDB as its storage engine, and in the previous episode with Sugu, he talked about the fact that MySQL storage engine is InnoDB, and there are some tradeoffs between InnoDB and RocksDB. We actually just did a show about RocksDB that hasn't aired yet, but that was quite interesting, and I was so unfamiliar with the whole area of storage engines.

Could you talk about storage engines? What is a storage engine do for a database and what are the points of comparison between RocksDB and InnoDB?

[00:47:04] DK: Yeah. I mean, the storage for a database is the actual mechanism to take your data, write it to some persistent volume, where persistent means slightly different than what we talked about earlier. But it's going to write it through I/O, through the I/O protocol of whatever operating system it's on to actually have that data stick around for an extended period of time.

[00:47:27] JV: To some non-volatile storage.

[00:47:30] DK: Right, or even depending on the case of the storage engine. Like if you're using a memory engine in MySQL, it's going to memory, but it's writing through I/O to some place where it's going to stick around.

RocksDB and InnoDB are really sort of the two poster childs for the general classification of storage engines of either a B-tree or a log-structured merge tree, and these are the data structures that they actually use to persist this data.

When we say the difference between Vitess and something TiDB is the storage engine that's not exactly true, because TiDB uses Rocks directly, and that is they will call into the storage engine to actually store their data. Whereas Vitess, we actually use MySQL, which then uses a storage engine, which by default is InnoDB.

So what that will give you is Azure going through and doing a set of operations. You're going to get a different performance profile off of either one of those systems, and it really traces back to the backend data structure. Something like an InnoDB is really good at fetching data and really good at updating data. Something like a RocksDB, which is a log-structured merge tree, is really good at appending data and writing data down quickly.

[00:48:45] JV: The cool thing about Vitess is that, I mean, we actually run on top, because our layer of abstraction is MySQL and MySQL can run on top of either RocksDB or InnoDB, we support RocksDB, right? Because MyRocks – So Dan actually showed a very cool demo. Well, he helped build a very cool demo that was shown by [inaudible 00:49:10] of State Street Bank at Reinvent where we showed 1.7 QPS, two-third reads, one-third writes, on top of a multi-sharded Vitess cluster, which was backed MySQL running on top of RocksDB.

[00:49:27] DK: Yeah. Also the biggest use of Vitess is a website called jd.com, which is sort of a Chinese version of Amazon. They actually run either with InnoDB or with another engine called TokuDB.

[00:49:39] JV: TokuDB, yeah.

[00:49:40] DK: Which is another – It's similar to RocksDB. It's a slightly different data structure, but they have it so that you could run it with either of those to sort of select the performance profile that you want out of those engines.

[00:49:51] DK: Correct.

[00:49:52] JM: Are there any users where they have both storage engines running, and then based on the query or based on something like that, like they use both?

[00:50:03] JV: It's possible, but I don't think that anybody is using that. If I remember correctly, right now we don't have a mechanism of routing queries based on engine type. Not that hard to do. It's an interesting idea, but I think at this level, that level of optimization is something – Yeah. Very specific workloads will benefit from it I think.

[00:50:25] DK: Well, if you look at something like what Sugu is doing with VReplication, that kind of workload now actually becomes extremely viable. You could have a log-structured tree as your write endpoint. You could have a B-tree based – InnoDB is B-tree based. You could have that as your read endpoint. Updates would get a little dicey between the two of them, but that's entirely something that's in the realm of possibility.

[00:50:49] JM: Okay. Let's talk a little bit about the market and kind of the business point of view. So these NewSQL databases and these providers, such as yourself, that are outside of the realm of the gigantic cloud providers, the TiDBs of the world, the Rocksets of the world, the planet scale datas of the world. Y'all are in different positions than the cloud providers in terms of your database go-to-market strategy.

What's your perspective on the database market and how willing are buyers to go outside of their cloud provider and purchase a database from someone such as yourself?

[00:51:35] JV: We are in the process of finding out.

[00:51:39] DK: I always remember the quote that says that data has gravity. So wherever your data is, you have a natural affection to put other stuff around it, and cloud providers know this, but consumers know this too. You know that if you store terabytes of data in Amazon, Amazon really is in a very good position to secure the rest of your business, and they know this, and they're aggressively going out and trying to get people to use their services so that they can sort of partake of that ecosystem.

Customers know this too, and we've had customers tell us that as great as some of these cloud providers are, they are aware of what it means to put terabytes and terabytes of data into a specific cloud provider. So having the option of saying, "Yeah, we will give you our data, but we are not going to make sure that you're the only way that we can get to it," is I think something that a lot of customers find value in.

[00:52:39] JV: Yeah, and what we are building is we can actually sig between your application and some of these managed data services, like RDS Aurora an RDS MySQL. So you could be using Vitess to shard a large RDS Aurora database, because as the size of the database increases, even though your write and read throughput might be manageable, things like applying [inaudible 00:53:08] take a longtime. So it's just generally not a good idea to have really, really large databases that are running on a single instance.

We allow you to shard your RDS Aurora databases or RDS MySQL databases, and I think in general people are finding that they don't want to go beyond a particular size for a single instance not only because of the operational ease, but eventual ability to migrate if for some reason that they need to migrate.

One of the sort of selling points that we have, that we have chosen to express is that we are multi-datacenter, multi-region right from the beginning, or rather multi-cloud multi-region right from the beginning. So we want to give our customers the ability to have their masters running in AWS in U.S. west and some replicas possibly in GCP on U.S. east and some in Azure running in Japan.

[00:54:09] JM: Wow!

[00:54:10] JV: Yes, and all of these is very doable. We have POCs where we have done this. The question mark is do people actually want to run their workloads like this in production?

Maybe not, because the latencies are high. But disaster recovery, you might want to have your data in another cloud provider. Not too many replicas, maybe one or two replicas, or course with some replication lag, but now you have some guarantee that your data is out there.

Second is things like GDPR compliance, right? Vitess architecture, because we allow you to shard, the sharding is really flexible. You can actually shard your data in such a way that your existing data gets distributed correctly in data centers in various regions, and some cloud providers might not be available in certain regions. So that might necessarily – You do have replicas or masters running in other data centers.

So there are reasons for which people might want to run these, but there's always of course – One of the reasons that people have told us that they like multi-cloud, is because it makes it easier for them to have those negotiations with their cloud provider when they are renegotiating their contracts. The ability to migrate, just having the possibility is something that is useful.

[00:55:36] DK: I mean, also, you're starting to see a lot more people who want to use a cloud-like deployment model, but don't actually want to be in somebody else's data center. I mean, I know recently at Reinvent, AWS announced they were going to start putting AWS style infrastructure in people's data center. Slightly before that, they announced that they were going to have a version of RDS that runs on top of VMWare.

So it's not as – While the cloud is eating a lot. I think with the popularity you're seeing off of Kubernetes and out of digital transformation in general, you're getting a lot more models than just the standard EC2 instance. I think that's the other place where Vitess plays really well and we at PlanetScale feel there's going to be an opportunity is we don't care what flavor of Kubernetes you're running on. We don't care if it's in Amazon or behind closed doors on-premises. Vitess is going to run the same way on all of those, and it can run in parallel on all of those. It can span Kubernetes clusters and it can span on-prem to the cloud across cloud providers. That's something that no single cloud service is going to be able to offer you.

[00:56:45] JV: Right. So what we are building in a database as a service of course, but the same software stack that we are using to run our own database as a service, we also license them to enterprises so that they can run it inside their own VPCs, or on-prem on Kubernetes in their own data centers. What that allows them to do is to, as Dan said, have clusters where the data doesn't leave their own VPC or their own security boundaries, but they can still spin out multiple database clusters either in their dev staging or production environments, manage them, test them and so on. Have this whole operational scaffolding be provided by PlanetScale and use Vitess for stateful workloads on Kubernetes.

[00:57:37] JM: All right. There were a ton of points in there that we could go much deeper on, but we're basically out of time. Really interesting company you guys are building here. Jiten, just one last question. You have gone from being an engineer to being a CEO. Can you tell me anything you've learned in that switch or things in the company building process that have surprised you?

[00:58:01] JV: I think I have learned a lot in the last year and a half in domains that I didn't even know existed, right? Fundraising is an interesting example. I have managed people before and I know what that entails, but just there are so many aspects to building a company that I really didn't even know existed that I have got to learn in the last year and a half.

[00:58:27] JM: Okay. Well, guys, thanks for coming on Software Engineering Daily. It's been really fun talking to you.

[00:58:32] JV: Likewise. We really enjoyed this.

[00:58:33] DK: Thanks for having us.

[00:58:34] JV: And thanks for having us, yup.

[END OF INTERVIEW]

[00:58:39] JM: GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use, and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plug-ins. Use the value stream map to visualize your end-to-end workflow, and if you use Kubernetes, GoCD is a natural fit to add continuous delivery to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on-the-fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations. You can check it out for yourself at gocd.org/sedaily.

Thank you so much to ThoughtWorks for being a longtime sponsor of Software Engineering Daily. We are proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]