EPISODE 961

[INTRODUCTION]

[00:00:00] JM: Relational databases provide durable transactional systems for storing data. The relational model has existed for decades, but the requirements for a relational database have changed. Modern applications have requirements for high volumes of data that do not fit on to a single machine. When a database gets too big to fit on a single machine, that database needs to be sharded into smaller subsets of the data. These database shards are spread across multiple machines, and as the database grows, the database can be re-sharded to scale to even more machines.

To ensure durability, a database needs to be replicated. The database needs to be able to survive any single machine losing power or getting destroyed. Sharding and replication can allow a relational database to be scalable, durable and highly available. There are many ways to build sharding and replication into a database.

Karthik Ranganathan and Siddharth Choudhury are engineers with YugaByteDB, a distributed SQL database. In today's episode we discuss the modern requirements of a distributed SQL database and compare the applications of distributed SQL to those of other systems, such as Cassandra and Hadoop. We also talk through to the competitive market of cloud-based distributed SQL providers such as Google Cloud, Spanner and Amazon Aurora. YugaByteDB is an open source database that competes with these other relational databases.

Full disclosure, YugaByteDB is a sponsor of Software Engineering Daily. Also, I want to mention that we are hiring a content writer and also an operations lead. Both of these are part-time positions working closely with myself and Erica. If you're interested in working with us, send an email to jeff@softwareengineeringdaily.com.

Also, we are going to be at AWS Reinvent Las Vegas this week, and we're planning a meet-up at Reinvent, which is going to be Wednesday, December 4th. If you're interested in joining us, then you can sign up. There's a link in the show notes for this episode. That's Tuesday,

December 4th, which is contrast to December 3rd, which we originally announced. We had to move the data, but I hope to see you there.

[SPONSOR MESSAGE]

[00:02:15] JM: Today's sponsor is Datadog, a scalable monitoring and analytics platform that unifies metrics, logs, traces and more. Use Datadog's advanced features to monitor and manage SLO performance in real-time. Visualize all your SLOs in one place. Easily search, filter and sort SLOs, and share key information with detailed intuitive dashboards, plus Datadog automatically calculates and displays your error budget so that you can see your progress at a glance.

Sign up for a free 14-day trial and Datadog will send you a complementary t-shirt. Go to softwareengineeringdaily.com/Datadog to sign up, try out Datadog and get a wonderful t-shirt. Your SLOs will be even more on-track when you're in your Datadog t-shirt. Actually, I can't guarantee that, but I recommend going to softwareengineeringdaily.com/datadog to get the free trial and a t-shirt.

[INTERVIEW]

[00:03:25] JM: Karthik and Sid, welcome back to Software Engineering Daily
.
[00:03:27] KR: Thank you. Thanks, Jeff, for having us.

[00:03:28] SC: Thanks, Jeff.

[00:03:30] JM: I'd like to get into talking about distributed SQL databases eventually, but in order to get us there, I'd like to first start by talking about data in a broader context and how data has evolved over the last 12 or 13 years. I think there are some specific areas that exist along the timeline of data over the last 13 years that got us to the point where we are today, and these are things like Hadoop, the Dynamo paper, Cassandra, Kafka, Spark. Then today, we're at this time where everybody is building on the cloud and we have a suite of distributed SQL databases that people are choosing from. People need a distributed SQL database to provide

consistent transactions, and we'll get there, but you guys both have plenty of history in the data world. Give me your perspective on the last 13 years. How did we get to where we are today?

[00:04:35] KR: Yeah, that's a great question, Jeff. Okay. By way of a quick background, I was at Facebook like about 10, 12 years ago, around 2007. So that's like kind of hits the timeframe and have seen the evolution of databases both in the market and at Facebook, like the data technologies at Facebook over the time.

I'm just going to use that as a queue and talk about it. In the beginning, there were monolithic SQL databases, like they were the only database. They were massively popular. They let you do a variety of queries. Now, fast forward a few years, what you saw was data was exploding. The amount of data was getting larger than what a single machine could hold. People started splitting their data or sharding it and putting it across multiple SQL databases.

Now, forward a little more, and you got file systems like Hadoop, which could store massive amounts of data, and the world also saw the rise of certain types of applications, like time series applications, etc., where you actually had to store and scale a lot, like event data or time series data. Naturally it meant more SQL databases and more sharding and dealing with more of how to re-shard databases and scale them and handle failures.

This pattern was so repeatable that NoSQL databases were born. They said like, "Hey, if you're sharding data across SQL databases, you're giving up on transactionality anyway. You're trying to actually do some sort of sharding, fail over replication, and that's what went into NoSQL." The type of workloads then actually did not require strong consistency in transactions, because a lot of the event data does not. That's the design point those databases chose to build to.

Now, the Dynamo paper, as you mentioned, is a landmark, because it represented the first time a database was built cloud native and it happened to be a NoSQL database that has built cloud native. It was obviously to power the high throughput cart, shopping cart use case of Amazon, and Amazon felt like, "Hey, this would be useful for a lot of people for a lot of use cases." That's what Dynamo represents.

Now, if you fast-forward a little more, you see that the newer age databases that are SQL databases and cloud native started appearing. With things like Google Spanner and Amazon's Aurora, what these guys do is they actually replicate their data over a set of machines. They give you strong consistency and transactions and they're able to scale and move data across geographies, but each to a varying degree of functionality. That leads us to today where predominant number of apps are being built for the cloud and for things at Kubernetes, which essentially converts whatever you run it on to a cloud native setup.

So they have some demands of databases, but people still love SQL. Distributed SQL git is kind of born like at least from my perspective where such applications meet the cloud. The transactional applications meet a true cloud-like infrastructure, which requires resilience, scale and geographic distribution.

[00:07:43] JM: Okay. Well, with that said, describe what an engineering team wants out of a distributed SQL database today.

[00:07:52] KR: Great question. An engineering today is tasked with building more features. We can simply it to that. whether it is a new application, a new feature in an existing application, a new microservice or a completely new way of reaching new customers. What are examples? Amazon has brought to us such an all pervasive and memorable shopping experience that we even forget that it was only 5 or 10 years ago when none of that stuff exists.

I actually remember going to Barnes & Nobles to read books, and now none of that stuff exists, because everybody just orders off of Amazon. How did such a compelling experience come to be? It's because it's a constant stream of new applications, features, discovery, ability to track where your shipping goes on, ability to return your shipping, ability to read reviews, so on and so forth.

Every one of these features puts a demand on the engineering team, and that means they have to come up with an architecture that makes it work. Now, the architecture goes in different parts. It goes in what do I use to build my app? The app framework. What do I use as a database to store the data inside and where do I run this cloud, Kubernetes, where do I run this?

Now, if you look at all of these applications overtime, and this is not by no means limited to Amazon, because if the rest of the retail industry does not do this, Amazon is going to disrupt all of them and take their business, and we're seeing that happen in real-time and this is happening not just in retail. It's happening in all the other verticals as well.

They're thinking, "Okay, what are the popular frameworks to build scalable, massive applications? Because if they're not successful, we just ween them away." But if they are successful, you don't want to fail, because a lot of users came to your site to use your service. So they start gravitating towards specifically for databases those that can give them flexibility of queries, which is proven so far to be SQL. SQL is the standard for flexibility. Of course, NoSQL is pretty limiting on that side.

They want resilience, which means that, "Hey, my machine failed. Sorry, I couldn't serve you your request and you couldn't place your order. It's no longer acceptable." Machine failures must be handled in real-time. So ultra-resilience. More people ordered. So, "Sorry, I couldn't take your order. It's no longer acceptable." Because that's a sign up success, you want to be able to scale massively and quickly.

"Yeah, you're in Europe and I have my data center hosted in the U.S. So, sorry, your order is going to take a long-time to place," is also not an acceptable answer. So you want geographic distribution. So people get instant – The feel of instant reaction, and Google has a well-known study where this is very important.

These are table states. Absolutely important. But people are deploying all of these in the cloud, and in multiple clouds at that sometimes in the private data center but wanted to look like a cloud. So deploying in Kubernetes, deploying across multiple clouds is becoming very important. Because it's the cloud, you have noisy neighbor problems. You have all sorts of failures coming at you. You want low-latency. You want it to be high-performance and low-latency. Finally, because you want the flexibility to do all this, people are looking at open source. We're seeing all of these trends happen.

[00:11:05] JM: Now, we're talking about building databases for actual customers. Different customers have different applications. Different applications have different read patterns and

write patterns. You might have applications that make a ton of writes in a time series kind of situation. All those writes, may be they only get read once and then they get rolled up into an aggregation and then they get thrown out. You might have applications where a bunch of users are requesting rides in a ridesharing service and it's just highly important transactions that you never want to lose, but there are perhaps not that many of them and they're not happening too too frequently.

You might have applications where user accounts just frequently get accessed by a single row. My point is just that there are different read patterns and different write patterns, and what I'd like to get from you is how do different read and write patterns affect the choice of a database that I should be making for my application?

[00:12:10] KR: Okay. So this is a great question and is actually quite fundamental in the world of databases. The clear line that divides it is whether a user is interacting with the database or not or a user like response is needed, which is should the reads be real-time? If I ask you a question and I need the answer really quickly, that constitutes what is called an OLTP database, an online transaction processing database.

If I ask you a question that is complex and that needs you to analyze a lot of data and you get back to me, let's say, even in a couple of hours, but that itself a difficult feat, because there's just so much data out there. This constitutes a warehouse pattern and these are called OLAP databases.

OLTP and OLAP is a clear split in access patterns of applications. YugaByteDB and all of the SQL databases, where SQL databases in general actually address both sides of the spectrum. There are certain types of SQL databases really suited towards OLAP. Examples of these would be like Snowflake, Red Shift, etc. In the non-cloud native world, there'd be like – There're a lot of databases, like Teradata and there'll be a lot of these type of databases that do that.

On the OLTP side, again, there are clearly databases that are very good at answering questions quickly. Examples of these would be our Google Spanner, Amazon Aurora and so on. Some databases have had the luxury of time and incredible amount of engineering so they can

actually do well on both sides. An example of this would be Oracle. It actually does both sides of the spectrum well, because so many features have gone in.

Now, in OLTP bucket, like each one of the tradeoffs are different. If you have a lot of writes and you want to analyze most of the data, obviously you go to OLAP. Now, we're going to focus on OLTP databases, because that's what new applications typically need.

On the OLTP side, you have write scalable databases and read scalable. Read scalability in databases in the world of distributed databases is achieved typically by adding more machines. You just increase the number of CPUs available to you and you can now serve more queries. But the expectation here is that your queries are not all on a single row or a very, very small number of rows. That's called the hot row problem.

The expectation is it's over a reasonable set of rows and you're able to scale the aggregate CPUs needed. If you have a single row or a very small set of rows being read over and over again, the dominant pattern to solve that problem is to have a lot of caches, replica of this data in a lot of caches and you serve it from these caches so that you increase the number of caches, which keep data only in-memory in order to increase your read throughput of the few rows.

Now on the write side, there is no free lunch. Increasingly, OLTP databases have to handle the write scalability, and that's what brings up this whole thing about distributed SQL. Whatever I said about the read side is true whether you're a distributed database or a monolithic database. It doesn't matter. The concepts are similar. A monolithic database, you can only increase the size of the machine. A distributed database, you can increase the number of machines. That's the only tradeoff.

On the write side, you just have to have more machines in order to process more writes. Otherwise, you have to give up one your consistency or something else in your application, like batching it or dropping data or rolling up or so on and so forth. At that point, a distributed database becomes a must.

The new paradigm that we see coming in with geographic distribution is the ability to do reads from the nearest data center or region so that you can serve your users quickly, because some subset of data can actually be served a little stale, but with low read latencies.

There're a number of branches in the tree, but I think the simplest way to think about it is what is the app trying to achieve, and if it falls inside, there is a lot of data that needs read and write scale, then a distributed SQL database is great. If there are a few rows that need read or writes happening over and over again, a cache or some other technique is often required.

[00:16:13] JM: Now we've had distributed SQL databases for a long time. We have databases that I assumed had the read and write scalability properties that I need. But presumably, you see some deficiency in the market. There must be some aspects of the read and write characteristics of the existing distributed SQL databases that you find insufficient.

What are the use cases that when you're building YugaByte you think you can improve on?

[00:16:52] KR: Yup, precise question. Great. The existing distributed SQL databases, like the two I'd like to refer to are, first, Amazon Aurora, and second, Google Spanner, and they both have an interesting paradigm trade of architectural tradeoff in what they did. Amazon Aurora is 100% SQL features compliant. It support PostgreS, all of PostgreS and all of MySQL. Anything you run there you can run here. It gives you all the RDBMS features.

It does have high availability, which means if a node fails, another can quickly take its spot. However, it cannot do horizontal write scaling. What that means is you can make this node bigger, but exactly one node of Amazon Aurora can take writes with the guarantee that data is consistent. That's just the way Amazon Aurora is designed.

Now, Google Spanner on the other side does not have all the RDBMS features. For example, it does not support things like foreign keys, stored procedures, triggers. It doesn't support a bunch of that stuff. It gives up – It creates its own SQL language. But it is highly available, just like Aurora. A failure happens, you're not affected. But unlike Aurora, it is horizontally write scalable. That means you need to take more writes. You can add more machines. You'll be able to expand the capacity of this logical database, which is comprised of multiple machines simply by

adding more machines, because Google Spanner can send subset of writes to a set of machines, so to each machine. In aggregate, they do more work.

Now, with YugaByte, what we noticed and we had the luxury of starting after both of these databases. It was obviously, it was our job to do the research and take the best learnings from them. So what we saw was that Aurora was insanely popular, because it supported everything in SQL, all the features. I think these are probably not official numbers, but it looks like Aurora might have a run rate of around $2 billion plus in annual revenue, which is an impressive achievement.

With YugaByte we said like, “Absolutely. 100% SQL. Everything has to get supported. That’s a must.” We said like if we are building the database for the cloud, and this is the future of all applications, then horizontal write scalability is a must. Obviously, both these databases are highly available and fault tolerance. So that just goes in without saying.

We picked those three things as some of the core vectors. In turn, YugaByte supports 100% PostgreS features and is also horizontally write scalable. 100% PostgreS features like Amazon Aurora. Also horizontally write scalable like Google Spanner and highly available and fault tolerant like both, right?

Now, on top of this – So that’s the type of applications that we’re trying to go after. You want all the RDBMS features. You want write scale. Now on top of that, what we noticed was these are great cloud vendor offerings. But a lot of people overtime, a lot of companies are now gravitating to multi-cloud models, and this could be because they have app A running one cloud and app B running in another, or they have customers wanting them to run in different clouds. This happens in retail, for example, or they have acquired companies which run in different clouds, or maybe they’re subject to GDPR and compliance regulations, which forced them to run in different clouds and different companies, or like they just want to run it natively inside Kubernetes in their own data centers. There’re a number of reasons why you would want databases that are cloud independent. So making it 100% open source and without any external dependencies so it can run anywhere is the other big thing that we offer.

[SPONSOR MESSAGE]

[00:20:38] JM: As a programmer, you think an object. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud area. Millions of developers use MongoDB to power the world's most innovative products and services, from crypto currency, to online gaming, IoT and more. Try Mongo DB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. That's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:21:33] JM: Okay. If I understand correctly, you said the primary distributed SQL databases on the market that people are picking for a globally distributed database are Amazon Aurora, which has the deficiency of not being horizontally write scalable. Meaning only a single node can accept writes at a single time. For example, a massively multiplayer online game would probably not work great on Aurora, because you've got all these state synchronization that you need to be done. If you're only accepting writes on a single database node, that's not going to give you enough throughput to synchronize a beautiful widespread game experience. So that's like the weak point of Amazon Aurora that you're going to try to attack.

Then on the Spanner side of things, the Spanner API is its own thing and you want to try to build a database that has PostgreS compliance, because that will be an easier onramp for people. Do I understand correctly the weak points of the globally distributed SQL databases that you're going to try to compete with?

[00:22:44] KR: Yes. That's absolutely right. As a result, our core is Spanner inspired and the API is PostgreS 11.2, which is like one of the most recent releases of PostgreS.

[00:22:54] JM: From the competitive point of view, what happens if Aurora just solves horizontal write scalability?

[00:23:05] SC: I can take that, Jeff. Actually they have been trying to. They are very, very cognizant of the fact that not having horizontal write scalability hurts. Applications that become so much successful once they are on Aurora.

Recently, Aurora introduced a multi-master configuration for Aurora MySQL, which is currently only two write nodes are available, right? But the challenge is that it's what is known as multi-master architecture as supposed to Spanner and YugaByteDB's auto-sharded architecture. Multi-master, each node contains 100% of the data. As a result, if you write to the same – you can write to the same row on both the nodes and that will lead to internal conflicts and deadlock errors, which the application now has to handle.

Instead of solving the core problem, we believe what Aurora is doing is an incremental step forward, but that incremental step then puts the onus on the application development and engineering to handle all the errors and conflicts that come by. That's one.

Secondly, this multi-master architecture is not the right architecture for fully global deployments where you want a portion of the data to be automatically partitioned into the region that is nearby to the users for the GDPR and the low latency use cases, right? In both cases, there is I would say an attempt, but it hasn't gone far to make a dent in the developer experience.

[00:24:45] KR: As any project, there's got to be some competitive motes, right? Some of these motes are time to implement features, and the other motes are just around mindshare and usage. It's just these two. Pretty much any project has these.

To us, building a core that is write scalable with transactional consistency is a strong mote, it's deep IP in terms of how long it takes to implement and make it production ready and actually performant. We have done a lot of that work.

Obviously, the other mote will be on the adaption side, where this entire database is available to you in – 100% open source, like Apache 2.0 license. We offer software that makes it very easy to convert into a managed service and we already have a number of users using it at fairly large scale. Assuming we now what we're doing, this will seem very compelling to the user or today.

That forms the mote. So they start liking the software the fact that we're 100% open source and using more, and that drives the growth.

The goals are also slightly different for the two companies. If you think about what Amazon is trying to do, they're trying to solve how to make Amazon an easier cloud for people to build in, and what are we trying to do? We are trying to build the default database for the cloud irrespective of which cloud that is. The feature set also, I guess, will go accordingly, right?

[00:26:12] JM: I presume that it would be equally difficult at least for Google to build a PostgreS compliant API on top of Spanner.

[00:26:22] KR: Yeah, that's right. I mean, that also represents the other part of our IP. It is actually significantly nontrivial to map – We had to surgically remove the lower half of PostgreS which wrote to a single node and map that on top of a distributed store that we have, which is Google Spanner-like. Putting those two together is also a significant time mote. Obviously, the open source nature and the best database for the cloud irrespective of the cloud, that still stays for us.

[00:26:53] JM: So you ended up doing a source available license, right? So this is not something AWS can potentially grab from your repo and spin up themselves?

[00:27:05] KR: No. The database is 100% open source. AWS can grab our database and run it and it's completely fine. The source available part is – Effectively, Aurora is a managed service, but it offers MySQL and PostgreS. AWS took MySQL and PostgreS as databases and made changes and offered it up as a service.

Our source available license is only the portion that takes the open source YugaByte database and converts it into a managed service. Unlike Aurora, where Amazon owns everything, we ship our software to convert any user's infrastructure into a database as a service so that they can run their own private database as a service. That's what we do. That portion is source available.

I think the question you're trying to allude to is what stops any of these clouds, or specially Amazon, from taking the software and running it. Is that right? Okay. Yeah. Great question. So

we see it this way. Before I even go into our reasoning, if you take how things have played out in the world, both MongoDB and Elastic, like Elastic being the more applicable example here, were 100% open source, and MongoDB made changes to protect themselves. Elastic made some portion of its software closed. It kept the majority of its offering completely Apache 2.0 and made some features proprietary. Whereas MongoDB went the other way and made the bulk majority of its database proprietary so that a cloud provider could not take it.

Now let's look at the results. Amazon offers both Elastic and Mongo. So closing out the codebase did not really help, because they still have that service. I mean, Mongo becoming popular, it's not just Amazon. Azure's CosmosDB also offers a MongoDB API. So it's multiple cloud vendors that are offering it. In the Elastic side, all of the closed features, Amazon rebuilt those features and put them out into a different repo which they said is the true open source Elastic repo. [inaudible 00:29:12] or whatever. But nevertheless, we still see MongoDB and Elastic as billion dollar companies. They really haven't gotten squished and they seem to be doing perfectly fine. All of the heartache aside, they could be doing better, I guess, but they're still doing fine.

Our reasoning and takeaway, and we're a database born squarely in the era when the cloud was dominant. We were built before the cloud and trying to figure out MongoDB or Elastic. We grew in the cloud era. What we've realized is that there are a lot of people running noncritical use cases, and in an open source software, they're never going to pay. They're never going to pay anybody.

Then there are people running maybe very minor or semi-critical use cases. They may care more about the infrastructure cost and all of that stuff than about the database itself. They don't represent the lion share of revenue, but there is some revenue and they may come to us. They may go to Amazon. That's fine. But the most important segment is the set of guys building their business and business critical applications on top of our database, like YugaByteDB, and those guys would probably trust us as the makers of the database. Building a database for mission critical, those are the guys we want to be talking to and monetizing as a business.

[00:30:30] JM: Exactly. That is such a good summary of why this whole relicensing stuff is so hilarious. It's so hilarious. Maybe it gets into different gradations when you're at like – The way

you broke down the market I think is broadly true, but it's maybe a little bit crude, because probably when you have an open source project that's as popular as Kafka or Elastic of Mongo, there's more gradations of the population.

[00:31:00] KR: Absolutely. I think it's also fair – To be fair to them, the total percentage of revenue may not be large, but the total value of the revenue may still be a relatively big number. So then part of that is fair and them asking the question, like them having worked so hard to build this. Should they get it or not? It's a different type of a question. I think the point is over the longer term, it should be possible for a database maker to make enough valuable features in the database to retain their audience. It's what keeps them honest. A cloud vendor should never be able to equalize that unless they are also a database maker.

[00:31:39] JM: I know. That's the crazy thing, is like – You look at Elastic, as like the thing I always wonder is like you're building a search engine. Google still has not solved search and you really feel the need to take a licensing approach to maintain the defensibility of your search engine. Can't you just out-feature Amazon? I don't know. I mean, that's what you're saying, right? Basically, if you're a database provider, you have to build a platform around your database. You have to build a database platform, meaning like added services, whatever, like serverless things, like the MongoDB company has built Stitch. They've built MongoDB Atlas, which are just better ways of interfacing with your database or added ways of interfacing with your databases. Different ways of interfacing with your database with added developer experience.

Everybody knows like Amazon, they're great at offering you the really wide buffet of different developer products, but developer experience in any one of those particular products maybe highly variable. There are certainly opportunity for these database providers to offer experience that is highly designed for their particular data product, which it sounds like that's what you're doing. That's what the YugaByte platform, that's the thing that's just source available.

[00:33:05] KR: Yes, that's right. Yes, it's the stuff that you could do yourself, but it's just a lot of work and it's headache for you to make sure you got it right and you don't want to deal with the consequences of not getting it right. But it's convenience more than IP.

[00:33:19] JM: Like what? What kind of stuff?

[00:33:21] KR: For example, people that want to install it, like say want to do nightly backups. They have to backup this distributed database as a cluster. So you have a bunch of machines that host one cluster. You may have multiple of these clusters and you want to make sure you have nightly backups. If your backup ever failed to happen, you want to get an alert or like you want to know when some machine is running slow, or if you're running close to capacity, or like a number of other things. You may want to put in like your security features, like TLS encryption and encryption at rest with certificates from like a KMS store.

A bunch of these stuff that you would have to actually hire people and get code around and verify its write. Is that valuable or is it just valuable that you'd just pay for it and use that as a platform?

[00:34:07] JM: Got it.

[00:34:07] KR: That said, the backups itself, like the ability to take a backup is free. It's there. You can do it. But do you really want to schedule that every day and copy that out to say an S3 and make sure that the restore works and make sure that you've put it in the right format, in the right location and deleted the backup if it's like 30 days old? All of these stuff is just work that may not be worth the time of a business.

[00:34:30] JM: In terms of the go-to-market strategy, okay, you've outlined where the opportunity is for YugaByte. Go-to-market is still really hard, because you have to find the person at the organization at a given organization that is having that very specific set of distributed database problems and then you have to convince them to work with your database instead of trusting AWS, which everybody trusts, or trusting Google, which most people trust and perhaps sacrificing some of the features that they want.

How do you win the customer in that kind of go-to-market environment?

[00:35:19] KR: Yup! That's the meat of the question, right? So for us, we are a firm believer in the power of open source. Open source does multiple things. Firstly, it gives you knowledge, full

transparency of exactly what you're adapting, what its strengths are, what its weaknesses are, what are the features that are in progress, everything, right?

You get to partake as a community in changing the roadmap or the feature set, asking for new features, even building some of the features yourself if you really need it in a pinch, right? In that sense, you benefit from what others have fixed and others benefit from what you fix. Overall, that is a powerful thing in itself. That's the first thing.

Now, the second thing is it is very easy for people to experience YubaByte as an open source product. It's very easy to download it, try it, you can try it on your laptop. You can try it on Kubernetes. You can try it in the cloud, etc., etc. The third piece is us trying to solve some of the hard frontier leading edge problems in the cloud. We write a lot about it, and our blog is something like we're very proud of, but a lot of our users come and tell us that they find a ton of valuable information in there. We do a bunch of things to educate users as to the best way to build things in the cloud.

Our knowledge is rooted from practical experience, because we were some of the key people, like building or supporting the growth of Facebook on the database side from 2007 through 2013. We're some of the people amongst a bunch of others. Incredibly talented team, right? So our take away from that growth was what should people be thinking about? What are the features that they would need? What would they see 2 years from now, 5 years from now? So on and so forth, right? Because we feel like we've seen a bunch of these journey before and we were also the team that ran the database introduction. What are the tricks and things they should look for there?

A lot of that comes together in our writing that people really love and come to us. This shows in terms of the amount of growth, like our community growth has been explosive. We've grown more than 10X in a number of different dimensions in our community and people just over the last year. We have only mission-critical data app side. We have a number of customers, like paid customers, that are running us for like enormous workloads, like we have some of these guys come and talk at our first user conference. They came – One of our customers, we didn't know the numbers, but we're blown away.

One of our customers is doing 27 billion operations per day on YugaByte, and all of these is mission-critical. We have like about four customers now that are doing over a billion operations per day. We're pretty proud of that achievement. Finally, when people start looking at open source packages, they look at a number of factors. They look at if the license is good, if there's enough activity, if the maintainers of the project are very responsive. I think all of these – I think I [inaudible 00:38:13] just recently wrote in a New Stack article.

Anyways, so we do a bunch of that stuff and it just happens by itself. Building an open source community and open source project just ends up working out that way. We feel these are some of the differentiators. The final thing I will say is that there is – I think it's funny. Even just over the years of YugaByte as a company and talking to so many enterprise customers, we started in 2016 when most of the companies told us they weren't sure about their cloud in the roadmap or architecture. They just weren't sure how it would turn out.

2017, a lot of lifts and shifts into the cloud. I'm just going to move some stuff into the cloud. 2018, they're like, "You know what? That doesn't really work. I need to build an app that is cloud native to get better ROI." Now 2019, multi-cloud is firmly in there. Hybrid cloud, like on-premise deployments that are cloud native, Kubernetes. These are firmly happening. We feel like there is – While the go-to-market looks hard from the outside, when you go in and start peeling the layers of the onion, there are good ways to make impact.

[SPONSOR MESSAGE]

[00:39:26] JM: I love software architecture, and I know that's a strange thing to have an affinity for, but it's something I've liked ever since I started listening to software podcasts, because in a software podcast, it's sometimes harder to discuss the lower level elements of a programming language or microservices. But architecture is something that's philosophical. It's somewhat subjective, and that's why I also love attending conferences that discuss software architecture.

The O'Reilly Software Architecture Conference is coming to New York February 23rd through 26th. The O'Reilly Software Architecture Conference is devoted to covering software architecture, and if you aspire to be an architect or you are an architect, it gives you the training that you need to stay at the forefront of this ever-changing field. You can join senior developers,

engineers, software architects, all kinds of people that are involved in software engineering and learn about machine learning, data analytics, cloud computing, serverless architectures.

It's all happening in New York, February 23rd through 26th, and if you go to oreillysacon.com/sedaily, you can get 20% off your ticket to Software Architecture Conference, and remember to use the discount code SE20, which will give you that 20% discount. That's a great discount that you can get on a conference that probably your manager will let you expense to the company.

You're saving 20% for your company, or if you are an individual contributor or some kind of contractor that doesn't have access to a company that will pay for the conference, pay for yourself. It will probably come back to you in spades in the future. Just go to oreillysacon.com/sedaily and use the discount code SE20 for 20% off.

I think it's a great way to invest in yourself and invest in your career, and it benefits Software Engineering Daily. You can also look at the show notes and see the link, see the information about this advertisement and get the link to oreillysacon.com/sedaily and use discount code SE20 if you want 20% off a ticket to Software Architecture Conference.

Thanks for listening to Software Engineering Daily, and check out the Software Architecture Conference to learn more about software architecture, soft skills. There are trainings. There are networking opportunities. There are lots of networking opportunities. I've networked at lot at O'Reilly Conferences and all kinds of conferences, but particularly O'Reilly Conferences standout, because they've always been friendly and they've been long-time sponsors of Software Engineering Daily at this point.

So check out Software Architecture Conference. Thanks to O'Reilly for being a sponsor.

[INTERVIEW CONTINUED]

[00:42:47] JM: The most brutal competition that I've covered in Software Engineering Daily has been the container orchestration wars, and that was this real vicious, essentially zero-sum panel for supremacy of the container orchestrator. I just remember like asking people, "Okay, Hashicorp Nomad, Kubernetes, Mesos, Docker Swarm. Give me the side-by-side comparison,"

and it was always just like, “Well, uh, this one does this and this one does that,” and then you go talk to somebody else and they’re like, “Oh! No. No. No. That’s not right. This one does this and this one does that.” I’m like, “Okay. Well, I have no idea which one is going to win.” Then of course the Google one won.

Now, that was a winner-take-all environment and there are very obvious network effects that we can see in today’s version of Kubernetes, even back then it felt like there was going to be a winner-take-all dynamic, because you had all these enterprises that were basically biding their time. Nobody was picking a container orchestrator.

Like the banks, the banks were all saying like, “We’re not going to pick any of these things, because we’re just going to wait to see which one wins.” Then Amazon made this very savvy decision to build the ECS, which was like a proprietary thing that like nobody knows what it was under the hood, or I’m sure somebody knows, but maybe with Mesos under the hood, but who knows? The rebranding exercise was actually really good for convincing some customers to use it.

But in any case, the container orchestrator wars was winner-take-all. It seems like maybe that could happen in open source distributed SQL databases, but maybe not.

[00:44:26] KR: Yeah. So I’ll color this with a couple of different incidence and maybe let you make the decision or your opinion on that. Firstly, the thing about orchestration and IS are any of these things where – Is that they’re just not sticky. Let’s think about the origin of IS. It started out from running stuff on a machine on its local disk. Pretty soon it moved to running the same stuff on a machine but on a remote disk, like a [inaudible 00:44:54] or one of those things. Then it moved from running stuff on the same remote disk but on to a VM. Then it moved from the VM to a container.

Now, a single container was not enough, so there’s a bunch of containers and now you need to orchestrate them. If you’ll even look at like – So at different points, like you see that it is easier relatively speaking, like in some sense of the word, of the term, to change the IS layer. However, you still see mainframes and databases on mainframes. You still see those still around. It’s a thing, because we’re fighting it sometimes. We don’t even know. I don’t know. Throughout my

entire education, I know very little about mainframes. I've never dealt with them too much, but I'm happy to learn more about them now than I ever did collectively in my entire career. That's because mainframe databases are still a thing.

Now, Oracle came after mainframe databases. Now that is still a thing. SQL server came after Oracle. Still a thing. MySQL and PostgreS, still things. I'm guessing the rest of the guys is still going to be things. That's because data has incredible gravity and OLTP mission-critical data is like a black hole. It has so much gravity. It's just tough to move. If it does its job right – Because data is just growing. The age we are in is exploding in amounts of data.

I remember, again, back to the Facebook days, I remember that the amount of data we were generating as a company for OLTP applications was like exponential in those growth years. I'm sure it's still the case. I'm sure that the data rate is just simply multiplying even at this point. Not talking about OLAP data. OLAP data is used to drive the business.

In OLAP there is an interesting phenomenon where you don't want your spend to exceed what you earn, because you can always collect a ton of data that you want to analyze. But if the amount of money you're spending to analyze your data to run your business better actually costs more than what you make from your business, that's a bad idea. But OLTP data is a close to the ground signal of your business growing. You absolutely want to spend for OLTP data, right? At least that's the roundabout narrative of saying this feel is a little different.

[00:47:08] SC: One more pointer. If we talk to the fortune 2000, when it comes to OLTP data and databases, I think they will tell you that they would not like a winner-take-all scenario to be clear to the game, because they want the ability to change their database architecture if so the need arises. Rather than recreate more or less Oracle's hedge money again in the modern cloud.

I will say Oracle hedge money was still threatened by SQL server, by DB2, by MySQL, by PostgreS, but there is clearly one leader and there are a bunch of others in there. In the new world, we believe it is not about a winner-take-all market. It's more about ensuring that you become the trusted partner for those engineering teams as they're growing.

[00:48:10] JM: But the winner-take-all phenomenon I guess in the database layer is more the API, like maybe PostgreS is the winner.

[00:48:17] SC: That's right.

[00:48:18] KR: Yes, that is right. Absolutely. Yes. I think that is very true. In fact, as a company that makes databases, we get this question most often compared to all the other questions, which is why the hell are you guys building another database in 2016, 17, 18, 19, whatever?"

[00:48:33] JM: They're like, "No. No. No. It's an architecture. It's not a database."

[00:48:36] KR: Exactly. That's what we tell them. It's like what you really mean is did we create a new database API? We tell them, "No. We did not," and they're like, "Oh! Thank God. Okay. What do you really do?" Then we tell them, "Look, you have this API, but you cannot run it in the cloud. You have to do all these work to run it in the cloud. How about we give you the entirety of that API along with some enhancements? Because you need those to run in the cloud." They're like, "Oh! This is awesome. This is great." I think that's the difference. I think you hit the nail on the head.

[00:49:03] JM: So Citus, Citus Data, the PostgreS – I thought that was PostgreS for the cloud. It got acquired by Microsoft. What did they do differently than your approach?

[00:49:15] KR: I think to explain it in the simplest terms. So Citus kept the entire PostgreS database and then they made a layer of software about, which is an aggregator that sends data to these sub-databases underneath. You put your data in and you tell them how to shard it and they will take your data and shard it accordingly in those sub-PostgreS databases inside and you can now run this thing as a sharded scalable database, right?

However, this is – Firstly, the replication underneath is still asynchronous, which means if a random node dies, then you'd have to promote another node to take its place and you have lost some data. It's unclear how much data or what type of data you've lost. That's always unclear. That's been the Achilles heel of most RDBMSs. That is still retained.

In YugaByte, the replication uses a consensus protocol with Raft. It uses Raft underneath as the way to replicate, which means if you lose a node out there, there's still a guarantee that you don't lose any data. So that's the first difference. The second difference is between these isolated pools of databases, if you are doing a transaction or you're doing a joint, that's going to be either not supported or inefficient, depending on exactly on what you do.

In YugaByte, the sharding is transparent. It happens under the hood. So you can deal with this entire set of machines as a single logical database as supposed to having to know a little bit more about the internals. Right? That's the second thing. Going back to the data loss point, I just want to point out that like – I mean, I don't know the exact year that Citus was started, but I think it was in the early days of the cloud. But if you think about what is one of the – In fact, we had this incident like I think three days ago with one of our paying customers. One of the Amazon zones went out. There was an outage, network outage day where you couldn't reach it, but they had deployed YugaByte across these three zones, and data was synchronously replicated and they were able to survive the failure of the entire zone without any data loss, nothing. No problem.

But in the case of something like Citus, that would be much more complicated. You'd have to figure out – Either the app has to observe the impact of a little bit of recent data being lost or you would have to go figure out how to make it all right, because the data references each other, right? If you just lose a couple of references, it becomes complicated.

[00:51:38] JM: Makes sense. So as we begin to wind down, I guess I'd like to get a little bit more reflection on the past and prediction about the future. So Karthik, you mentioned that you worked at Facebook for five and a half years. Sid, you're at Oracle for five years. So you guys have a lot of perspective on the past and future of databases. So Oracle I would kind of think of as really, really good insights into the past of when databases were extremely consolidated and probably plenty of lessons for how to form a good database strategy. I'm sure you learned those things at the Hegemon and Facebook.

Karthic, I'm sure you have a lot of predictions about where the future of data is going to lie, because I've done a lot of interviews with Facebook, ex-Facebook engineers. The data requirements for Facebook, they're just crazy and very different, honestly, from what I can tell

than what a lot of applications have requirements for today. What Facebook was experiencing six, seven years ago. It's like the data volume, the level of synchronicity that maybe you'll like – The average database will need in five years or something. So maybe Facebook sees things 10 years ahead. I mean, a lot of these at scale companies see this kind of stuff.

I just love to get each of your perspectives on kind of what lessons you took away from your careers up to this point. Just give me some perspective on where you think the database market is headed.

[00:53:25] KR: Okay. Maybe I'll go first. Yes. I think you precisely put it that – At least, for me, I do see Facebook as being ahead of the – Or any of these hyperscales as being about a decade ahead of the general enterprise. It's actually so funny for me that it's like – At Facebook, we built something called Tupperware. It is our own orchestration engine. Now, there's Kubernetes, which is popular in the world. At Facebook, we had reorganized ourselves into – I mean, we didn't have a better term, but nearby data centers and faraway data centers, and now we have zones and regions. I could go on. I think microservices was pretty similar, like any number of things.

On the database side, I think it has been an interesting evolution. I think the reason we started the company is actually to build whatever Facebook was facing as the database problem to bring that to the enterprise market. So that's really where the company comes in, and that is problems of being able to survive across zones.

I mean, I'll say this. Maybe we have time. I would just say this interesting incident. I used to be a part of the H-space Team. We were the team that both built H-space features and ran it in production. There were times that we were going for five 9s of efficiency and there were times when like, for example, a mistake in a cable that gets pulled out, takes out a bunch of machines. All sorts of stuff would happen.

We were still inching out way close to five 9s, but that's when we realized that with billions and billions of users or requests coming in, you could fail a few million for a long time, and when you divide it, you still get a very small number and you're okay. You're still five 9s. That's great, right? But the reality is there's millions of people whose requests are failing, and that's not good,

right? That's when we realized that the real number is the fraction of requests that come in that should fail. You should have a very, very small number of requests fail. That required a completely different thinking and that's when we were – We started thinking about multi-zone deployments and that kind of stuff, right?

So we see that that is slowly starting to happen today. It's not 100% there. So multi-zone, as in use three zones. Don't use two. Use three regions. Don't use two. So we're seeing that some people are starting to do this, but a lot of people still want to stay with two regions because of other reasons, right? But we think that in the future, a prediction is that it is going to firmly move towards an odd number of regions so that you can do consensus and hands-off stuff and ensure that there's no data failure and so on. That's on one side.

On the second side, I believe that data scale and latencies – I mean, this is – A part of this is in the thesis of the company, but data scale and low latency is going to become a stringent requirement. It's going to just keep increasing. There are – For example, 5G is coming up, right? You see that that is going to enable a new set of applications to come through like – I don't know. I've read about remote surgery and all of this kind of stuff that you can do now. Even teleconferencing is becoming – Like we're doing, right? Teleconference is talking over – The distance is becoming common.

There is a number of applications that will start coming up where low latency as in, say, sub-10 millisecond or sub millisecond is all starting to – It's going to start getting very important and so will geographic distribution. Even in the remote surgery example, you just need the database of whatever it is that you are trying synchronize and write and do to live across the geographies that the surgeon and the patient are in not really anywhere else in the world. So don't be constrained by where your data center is. Be constrained by where your data should live, right? So it's that kind of stuff.

It's only getting accelerated by GDPR and all of those type of developments. So really feel that that trend is going to increase. I'll throw out one more thing that like it feels like edge data centers, Kubernetes, which his already becoming de facto. But the edge and like things like serverless are going to gain more adaption.

I mean, unlike what people say. I don't subscribe that everything can be done by serverless. That's kind of little crazy, but there are clear use cases where it can make life easier. A lot of those patterns would start emerging as well.

Looking at the database side of things, we start thinking about it as becoming a CDN with data, like a data network across the world, like that kind of thing, and you may need very low latency reads. Sometimes you may need strong consistency. You may need to invalidate. That type of a paradigm, but in a language as ubiquitous as SQL

[00:57:48] JM: All right. Let's go with Sid now. That was great.

[00:57:51] SC: I want to add that – I mean, if you look at the Oracle database, it's one of the most amazing pieces of technology that I would say mankind has ever seen. Given how much business-critical information is stored and served out of it all the years. Now the way Oracle built that is through – In the original days, it was all about Maniacal focus on developer productivity. How do we build a database that makes the life simple for these new age developers who are just starting to build for the internet-based applications and get them all the tools that are necessary to get them going. Never lose their data, extreme reliability, all that baked into the database.

It so happens that – I mean, that sort of led to a little bit of thinking that they will never lose that developer. They will never lose that customer and user. As a result, innovation has stagnated. But if you look at what the distributed SQL segment is trying to do is essentially take the same ethos of Maniacal developer productivity and bring it to the modern cloud, because cloud actually changes everything, changes – It is already evident in Oracle's results. It changes how Oracle is going to make money or not make money in the future.

We believe that as time progresses, as all these various kinds of applications, users demand from the businesses, cloud as the delivery vehicle and the distributed SQL as the database layer is just going to happen for sure.

[00:59:40] JM: Very interesting. Okay. Well, last thing to wind down, you guys both went to UT, right? UT Austin?

[00:59:48] KR: That's right. Grow a long horn. That's right!

[00:59:51] JM: I'm also an alumn from there, right?

[00:59:54] KR: All right! Fellow longhorn.

[00:59:57] JM: Where did you guys do your engineering homework? What building?

[01:00:02] SC: Most of the time, Taylor Hall.

[01:00:04] KR: Yeah, Taylor Hall. What about you?

[01:00:04] SC: Basement. Basement of Taylor Hall.

[01:00:05] JM: Yeah, basement of Taylor. Oh, yeah. Definitely.

[01:00:08] KR: You were protected against even a nuclear thing.

[01:00:12] JM: Right. And protected from sunlight. Anybody who remembers that, that was just this crazy subterranean place with really nice Linux boxes.

[01:00:24] KR: That's right. 3AM, 3PM, no problem. Looks the same.

[01:00:28] JM: Sis, you were doing EE homework there, right?

[01:00:31] SC: I was in ACS.

[01:00:31] JM: ACS. Okay. You guys were both in masters programs. Okay.

[01:00:36] KR: Yeah, that's right. Yeah.

[01:00:37] JM: Okay. Yeah, great memories at Taylor Hall. I just remember, you go in there, it's like going into a Casino because you just don't have any windows and the time just goes by really fast. I remember going down there with like a bagged lunch and like a really big thermos of coffee and just spending hours and hours.

[01:00:54] KR: If I were to guess right, the assignments were on the corner, because that's what I used to do.

[01:00:59] JM: Yeah, absolutely. Oh, yeah. You're totally grinding on it like hoping not to have to use last slip day. Oh man! Okay. Good memories.

[01:01:08] SC: We did graduate to ASC Hall in I guess the last semester or something — new one. I mean, it was new. It didn't have that feeling, the feeling of –

[01:01:21] KR: You didn't bond with it. It was like a rental versus your own hall.

[01:01:26] JM: Oh, right. Are you talking about the portables? There was like a set of portables, right?

[01:01:32] SC: Right. I mean, I think that the ones at the top.

[01:01:34] KR: Top. Portables, yes. That's right.

[01:01:36] JM: Yeah. It's funny, because then like they finally built a computer – Well, Gates funded the computer science. But I don't know if that was getting built when you guys were there, but the Gates computer science building is just this palatial, gorgeous building – I mean, I was lucky enough to take my sweet time and graduating and I was using the Gates building facilities near the last couple of years. It was gorgeous. Much more beautiful place to do your homework last minute.

[01:02:04] KR: Yeah. It's fun times though. Yeah.

[01:02:06] JM: Good memories. Okay. Well, guys, it's been a real pleasure talking to you. Great discussion of distributed SQL and I feel much more knowledgeable of the distributed SQL landscape at this point.

[01:02:18] KR: Thanks for having us, Jeff. It was great set of questions. I really enjoyed the discussion.

[END OF INTERVIEW]

[01:02:30] JM: Software Engineering Daily reaches 30,000 engineers every week day, and 250,000 engineers every month. If you'd like to sponsor Software Engineering Daily, send us an email, sponsor@softwareengineeringdaily.com. Reaching developers and technical audiences is not easy, and we've spent the last four years developing a trusted relationship with our audience. We don't accept every advertiser, because we work closely with our advertisers and we make sure that the product is something that can be useful to our listeners. Developers are always looking to save time and money, and developers are happy to purchase products that fulfill this goal.

You can send us an email at sponsor@softwareengineering.com even if you're just curious about sponsorships. You can feel free to send us an email with a variety of sponsorship packages and options.

Thanks for listening.

[END]