EPISODE 905

[INTRODUCTION]

[00:00:00] JM: Spanner is a globally distributed, transactionally consistent database. Spanner initially emerged as a paper that came out of Google in 2012. Around this time, database scalability was difficult to solve even for Google. The Spanner paper offered some breakthroughs in distributed systems, which allowed Google to take some of the learnings from Bigtable's eventual consistency and construct a novel system that was transactionally consistent.

Deepti Srivastava has worked on Spanner for more than 6 years. When she initially started working on Spanner, she was involved in the internally-phasing Spanner system that Google engineers used to spin up instances of the database. Today, she works as program manager for Cloud Spanner, a product within Google Cloud.

Deepti joins the show to describe the breakthroughs of Spanner and how her work has evolved since the product has gone from an internal system to an externally-phasing piece of cloud infrastructure. She also gives reflections on the comparison of Spanner to Bigtable. Bigtable was another paper that came out of Google that eventually evolved into a cloud service.

I enjoyed this episode quite a bit, and I hope you do as well.

[SPONSOR MESSAGE]

[00:01:24] JM: This November, the O'Reilly Velocity and Software Architecture Conferences come together in Berlin to give you the latest insight on key trends for cloud native systems, dev ops, domain-driven design and microservices. Join 2000 other engineers, developers and software architects for networking and essential training on how to design, build and manage resilient systems.

I've been going to O'Reilly conferences for the last four years ever since I started Software Engineering Daily, and they're great way to learn. They're a great way to have some food, network with people. And on November 4th through 7th in Berlin, the Velocity and Software Architecture Conferences from O'Reilly are coming together and you can get 20% off your ticket by going to oreilly.com/sedaily and entering discount code SE20.

O'Reilly conferences are a great place to learn about microservices, domain-driven design, management, software frameworks, cloud services. There are lots of opportunities to learn and get better and get yourself to a new job altogether if that's what you're looking for. I've met some great people at O'Reilly conferences, and you can attend an O'Reilly conference by going to oreilly.com/sedaily. Find out more about the Velocity and Software Architecture Conferences, and perhaps go to Berlin. These conferences are highly educational and your company might pay for it if you ask your manager. But whether they do or not, you can get 20% off by going to oreilly.com/sedaily. Use promo code SE20 and get that ticket.

Thank you to O'Reilly for supporting us since the beginning with passes to your conferences and thanks for producing so much great material about software engineering.

[INTERVIEW]

[00:03:24] AS: Deepti Srivastava, welcome to Software Engineering Daily.

[00:03:28] DS: Thank you, Jeff. It's really exciting to be here.

[00:03:30] JM: Yeah, it's great to have you. You work on Cloud Spanner, which is the cloud implementation of Spanner. Spanner is a globally distributed database. It was originally just a paper that came out of Google in 2012. Obviously, Google had implemented that internally. What were the problems in database architecture that Spanner addressed when that Spanner paper came out in 2012?

[00:03:57] DS: Yeah, that's an interesting question. I think I would think about it, and what we actually did at Google was answer a different question, which was we knew how to do large-scale things with indexing and search indexing and web indexing. We had Bigtable already at the time. But then we realized that as the scale of the data we had grew and the requirements changed. We went from a search company, to and ads company, to an apps company. The requirements from that data changed. So we needed to have strongly consistent relational typed transactional data that we could rely upon for the apps logic to be simpler and for the advertising database to actually make sense.

What we've realized is we ended up in a place where we had the AdWords database, which was a large portion of what Google was doing, was on a sharded MySQL system. It was just, for any of your listeners who've managed a sharded database, even with two shards, you can understand that it's just a nightmare to manage, to maintain, to build, to grow.

And as the dataset was growing, we have to reshard it, and every time the dataset grew beyond what we had, we had to reshard and re-implementing the re-add another MySQL instance to it in order to accommodate the data which was getting nightmarish. So Spanner was born out of that requirement of having transactional data at scale.

[]00:05:29JM: So those requirements between the search product and a product like ads or a product like Gmail, why are the requirements for the backing database different in those different applications?

[00:05:47] DS: This goes back to application philosophy. I come from a distributed systems backgrounds. So I think the answer changes depending on who you are. I think people get very religious almost about databases versus storage systems. But I think we've come to a point where they marry really well, where you can always store all data and flat files on disks or in bytes. If you talk to a storage person they'll say, "Well, a database is just a fancy UI on top of disks and block storage or object storage." But the point is that at some level, you need to have the right constructs in order to build and maintain things like applications, such as Gmail or really important datasets that require strong consistency, like ads, campaigns, or user data, or financial data.

The reason databases have existed and never really gone away in spite the huge NoSQL movement is because it turns out that everybody acknowledges that there is value in having strongly typed data with ACID transactions and the ability to query them, right? It's very hard to get rid of those requirements, especially as you create more complex and interesting apps that exist.

[00:07:10] JM: And I can think of the difference between what you would need out of a search index data storage system versus perhaps a Gmail storage system in the sense that if you're building a search index, let's say you build the search index nightly, and when you update the search index, you're going to be swapping out the old version of the index for the new version of the index. So that's essentially a big write operation to the database. And at some point during that write, you may have people that are reading from the database that may be reading different historical versions of the search index. For a search index, that's not the end of the world. It's not the end of the world if I search for an article about software engineering and I'm given a stale index and it has the results in a slightly different order than somebody else across the world who is reading the same search result that is getting fed a different ranking. But if we talk about a Gmail example, if you were to use that same kind of storage system for Gmail, and there are potential for different clients reading different data about what is in your email. That is actually problematic, because then you can have a situation where people are just receiving conflicting results about an important state of reality that we need to agree on.

[00:08:46] DS: Yeah. I think you nailed it exactly, right? The point is that there are certain requirements for which eventful consistency is fine right, and that the way we think about it here is eventual consistency can be thought of as an optimization, rather than the default. Because for a lot of the data that we are talking about here, user data, or your emails, the transactions. I'll give you an example that is I think one I use very commonly for explaining Spanner, which is that if you deposit $50 into your bank account and the system says, “Yup. $50 have been deposited.” And then you call up your mom across the world and say, “Hey mom, I've deposited $50, and you can take them out now and buy me all the cool stuff that I want.”

That $50 better show up. It cannot be 25 and it cannot be like zero, and it cannot be 49. It has to be exactly $50 and it has to be exactly when you’ve said that you deposited it and the system has accepted it. Any user will not be okay with anything apart from the $50 showing up when another person operates that same account. This is exactly what strong consistency is. In fact, what Spanner provides is external consistency, which is strong consistency across – It's not causally ordered. It's actually global strong consistency, where global here means across the system, not just across the world.

There are applications increasingly that demand that level of consistency, because otherwise the application logic breaks. To your point, if a Gmail – If I as Gmail user on my laptop see an email but on my phone don't see that same email, I will freak out. It's so in our nature at least expect our systems to behave a certain way so much, that when they don't, we'll actually freak out. I think I actually might have to call someone and be like, "What's going on?" if I don't see the same thing.

[00:10:41] JM: You used a term there, causal consistency. What is causal consistency?

[00:10:45] DS: So now I'm showing my distributed systems background, and please feel free to stop me if I get too in the weeds here. But, essentially, most distributed systems operate on what is called causal consistency, where the famous implementation of causal consistency is Lamport clocks, designed by Leslie Lamport. There is a very famous paper out there about that too.

But, essentially, because distributed systems are not – Once they're distributed across a network, generally, there is no concept of one ordering that is observed everywhere. So what normally see is causal ordering or causal consistency, where if something happens and a messages is sent to another machine to make something happen, then that event is happening as a result of or because of or as the cause from a cause of the message that was received. So that is causal ordering and essentially is what builds eventual consistency. Whereas with external consistency, what we're seeing is that there is a global or doing that is observed from anyone who is observing the system anywhere. Does that make sense?

[00:11:55] JM: It does. Yes. Now, as you said, there was a predecessor to Spanner inside Google, Bigtable. Not a direct predecessor, but can you describe what Bigtable satisfied out of what we need from a distributed data storage system and how it contrasts with Spanner?

[00:12:15] DS: Yeah, sure. Basically, Google was a search company. So what we were doing was doing web indexing, page ranking. We wanted to have a URL, and we wanted to store the document, the website, that had the contents of that URL. And then we wanted to do something page rank. So that's a pretty simple sort of requirement. All that we're saying is we want to do these things, but now the data, because the web is so huge, that data doesn't fit in one box or one machine anymore. So we have to split it up.

So Bigtable was basically a scalable system that could store these key value pairs and look them up really quickly. And the look them up here was – Point reads was also a bunch of scans and high efficiency indexing so that we could generate these page ranks and update them and update the global metadata across as the world wide web became bigger and bigger. So that's a pretty simple problem.

To your point that you mentioned earlier, if my search for the same terms return something in a slightly different order than yours at slightly different times, it's okay in most cases. So we weren't looking for these transactional guarantees across the dataset, and that is how, as I said earlier, Spanner evolved to meet those requirements, because we went from a simple search of the web, web indexing, page ranking, URL storing company to now we hos ads, and now we can campaigns of ads. Then we have apps, G Suite. So all of the requirements, they're not just consistency requirements. They're latency requirements, like G Suite type ops that have very high end user latency requirements, for the latency to be very low, because they're the user at the end of a triangle look for things inside their Gmail or their drive folder, or their docs, etc. So all of them, all of these applications, as the product suite that we offer grew, our requirements to meet our users and the customer base and their requirements grew as well.

[00:14:24] JM: Continuing with an exploration of some of the things that were available prior to Spanner, there was sharded MySQL, and we've done a few shows about this, some that come to mind are ones we've done with – There's a team behind a product project called Vitess, which is a sharded MySQL database system that came out of YouTube actually. What it actually did was Vitess – It didn't eliminate the fact that you're doing database sharing, but it moved the sharding logic from the client to the internal server. And it sounds like from what I've heard from people who have done sharded MySQL, the problem is not necessarily that you're sharding MySQL. The problem is the fact that you are putting the responsibility on the client to know which shard you should be addressing to find your data.

So could you just tell me the problems with database sharding that existed prior to a system like Spanner and why those led to difficulties in database management?

[00:15:38] DS: So let's start with the simple thing that you said. First of all, before any of the other complications, you have a database with data that's too big or QPS that's too high for one machine to handle regardless of how big that machine is. I mean, MySQL runs on commodity hardware. You can go by specialized hardware, like you can have DP2 machine. At some point, you cannot add more memory or more CPU to that machine or more disks. It just physically runs out of space.

So then what to do you? Well, you have a database that has multiple tables. So what you do is the simplest way to shard is to say, "Okay, I have one table in one machine, and then the other table I put in another machine." So that becomes two MySQL databases. One managing one table and the other one managing the other table. Great!

So let's use a concrete example. Since you talked about Gmail, you have users and you have, let's say, labels, or email per user. So what do you do? If you need to add another message to your emails, then you have to go to the email table and add a new message there, which is a transaction, which is in one part. But now you have to update the user's message account, which is a user metadata, which is in the users table. So now you have to do a transaction outside of the database between these two tables between these two databases.

So now you've taken away all the like ease of use that MySQL provides, for example, and you've put it in the application layer, which means what happens if the transaction fails? You're going to have to rollback that transaction in one or the other side and then you have to acknowledge that the transaction roll backed and left the database in a correct state rate. Then you have to repeat the transaction, because, ultimately, you do want that email to persist and the email count to go up.

So this is just one thing that you have to do that the database handled for you, rollbacks and roll forwards. Now, what happens when you need to add another user? Same thing. You have to add it to the user's table and then you have to create an email table for that user in the email thing. Great. Now, these emails and user tables are so big that you need to create another shard. So now you have to change the application to your point where you were saying earlier, "Go figure out which of the databases does the user exist in, and then figure out how to go from the user table to the email table and where the email table exists." So your application logic has now become extremely complicated and extremely brittle. That's one set of issues.

The second set is schema changes. How do you manage schema changes such that they are consistent across all your shards? Then the third thing is actually adding the shard itself, which means you have to bring down your database. You have to [inaudible 00:18:30] so that there aren't any more writes happening or reads happening while you're resharding this two table into three tables and putting the third table on a third shard. This is like just off the top of my head.

The amount of maintenance and just headache and the brittleness, like the bugs you can imagine, consistency is a very hard problem to solve. That's why databases exist, so that you can abstract away all of that implementation and gory details and corner cases into the database. Now you've exposed the poor application developer who wants to write a Gmail app into all of these like crazy consistency and concurrency control issues. Then all these people now have to carry pagers and then debug these things in real time, because now their application is not just a tiny application sitting in one corner in one warehouse. It is something that is being used by millions of people across the world have very little patience for things going down.

[SPONSOR MESSAGE]

[00:19:35] JM: Looking for a job is painful, and if you were in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that is a good fit for you.

Vettery is an online hiring marketplace to connects highly-qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies. Access is exclusive, and you can apply to find a job through Vttery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. No more of those recruiters sending you blind messages that say they are looking for a Java rock star with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job.

So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. Vettery is changing the way people get hired and the way that people hire. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. That’s V-E-T-T-E-R-Y.com/sedaily. Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:21:24] JM: I'm sure this still happens today. It was definitely an epidemic in the large-scale internet companies. I don't know. What would you say? Like 2009 through 2015 or so? When was this the most acute? When was kind of like relational database sharding a really prevalent problem in the industry?

[00:21:45] DS: I think it still is.

[00:21:47] JM: Okay. Still it.

[00:21:48] DS: Yeah. I mean, the thing – So I think Google hit this problem much earlier than others just because of what we do in the scale of data we had to manage and maintain. And that's kind of why Spanner is so cool in my mind, because we actually addressed these problems to the point where now most of Google critical applications of all sizes and of all types are basically running on Spanner. So it’s a hardened system that's highly available and enterprise grading from an instrument infrastructure point of view that we have exposed on the cloud to everyone who wants to use it.

I think that's what's interestingly exciting about this product. I mean, the technology is cool, but there's a lot of cool technology out there. The question is how can you convert something that’s a cool tech idea to an actual product, to an actual first developed project and system and then to a product that you can back up with a 99999 SLA, for example. That is the journey that I think is very interesting that I have had the distinct privilege to be part of.

But these problems still exist. And, in fact, when we launched Spanner and saw the momentum shift in the market when our competitors sort of all within the next 18 months launched all sorts of global things and stronger consistency things, etc., etc. To me, what was exciting about that is that it proved that this is a requirement for customers today. An ongoing requirement and something that only gets more acute as data grows, as applications grow, as online on-demand types of apps exist across the globe.

The problem is just this is the tip of the iceberg. I think the problem continues to exist and people continue to go down two routes. One is the sharded model, because they understand that transactions are important and databases are very, very cool in terms of being able to store and serve data, especially in a structured manner. The other half of the people took, and this was something that was happening in the sort of last 10 years, was the NoSQL movement, which is, "Oh, scale is more important than consistency and relational semantics. So we'll go with managing scale."

I think now we have come to a point where both sides of the spectrum are meeting where they're like, "We need scale insurance. We need the ability to have both relational semantics as well as the ability to scale with those relational semantics."

[00:24:21] JM: What were the key innovations of Spanner?

[00:24:25] DS: I would say they key thing that Spanner brought to front is that transactions aren't evil, that you as an app developer shouldn't be afraid of transactions. Because what we did was allow transactions to happen at scale with reasonable performance. I think that would be the key in my mind, actual innovation that changes people's lives, especially in the software industry.

[00:24:54] JM: Let's dive into a little more technical detail. So what were the technical innovations of Spanner? How did Spanner make that possible?

[00:25:01] DS: Many things, right? I think we built the Stack from the ground up. I think of Spanner as the three technical pillars here, right? One is Paxos. So we actually wrote our own

Paxos protocols. SO we didn't use the open source version of Paxos, and we optimized for large-scale transactions. What that means is that we optimized for highly available participants. So anybody that’s familiar with Paxos, it's a consensus protocol with participants, and they have to agree on something.

The reason you don't or people didn’t used to use consensus-based protocols is for things like strong consistency and especially database transactions, is because it can take a long time to agree which can degrade performance, degrade throughput and increase latency. The way we’ve implemented our Paxos protocol and the way we’ve designed the system, the Spanner system itself, allows for very highly available participants. And that allows us to commit transactions even in a networked environment with network latencies that can be like RAM latencies. We can commit transactions fast and move the system forward, because after the biggest issue with databases, is that you have to keep moving the system forward by committing transactions. You can have a stuck state, and Spanner allows that with our unique software stack. That's one.

The second thing is True Time. So we actually used True Time to have global ordering or external consistency. That's what True Time provides is ultimately. The paper covers a bunch of this, and actually if you go to cloud.google.com/spanner/docs, we have a bunch of white papers that talk about each of these things in much more detail.

But going back, like True Time and global ordering and global serialization, allowed us to have external consistency, which means we didn't have to rely on causal ordering and message exchanges to move the system forward to create a global system numbers and things like that that other databases have to do. The third thing is Google's network.

We basically have very high-speed and dedicated network that is not as lossy as the internet, obviously. And that again allows us to build a system like Spanner, which is distributed and networked, but requires resilience and guarantees of message delivery to leverage that to build a transactional globally distributed system.

[00:27:31] JM: So those three key aspects of Spanner, the first one was an implementation of Paxos that was faster. That was fast enough to allow for a consistent database that actually would have the speed that people would desire. The second one is the True Time API, which is an API which I believe can be implemented a number of different ways. But essentially the guarantee that you want from it is a notion of time that the different areas of the database, the different physical areas of the database can agree on. Then the third thing is Google's network. Those were the three things.

[00:28:14] DS: Sorry. Just to go back. The Paxos thing is basically highly available participants. That's the key for the technical audience here. The key thing is highly available participants that we get through our implementation of Paxos. The True Time API that guarantees an ordering. So it guarantees time bounds to be very strict. So basically clock drifts and clock skews are guaranteed to be within a very narrow window, which is one of the key problems in distributed systems with time. That's why we don't use time normally to do distributed consensus. But because of True Time’s unique guarantees around clock skewing, clock drift, we can use that. Then the third thing is physical hardware, essentially. We have built a network across the globe that we can rely on, again, which is different from most distributed systems, because you never trust the network and you never trust time, because those are lossy and not guaranteed to be the same.

[00:29:19] JM: Let's talk a little bit more about that Paxos that you mentioned. So you said that the key innovation there is highly available participation. So we don't need to obviously go into detail about how Paxos works. There’s plenty of the other podcasts or other resources. If you want to learn Paxos, probably the best resource is not a podcast. But anybody who has some cursory understanding of Paxos I think knows that there is a kind of a system of agreement. That you need to reach an agreement between the different nodes in the system so that that the nodes can agree on what is the state of the distributed database. You're saying that part of Spanner's unique implementation at the time was that you keep that participation highly available. Could you just describe in more detail what do you mean by that?

[00:30:14] DS: Sure. Again, for the more technically inclined who would want more than what we can offer here in terms of time, we have all of this in our white paper section. But basically what I'm saying is each – To your appointment, it's a consensus protocol where there are N-number of participants that participate to agree or disagree upon a particular state of the system. It doesn't have to do with databases that all. It's basically a consensus protocol for any distributed system, which has nodes, where nodes here are participants.

They have to say, for example, that – In our case, that either a transaction committed or it didn't. And we have to be guaranteed to go way or the other. Why this can be an issue is if you have consensus, basically. You have to have fault tolerance. So if there are three nodes then one of them die, then you have two nodes left, and now you have what is called a split brain, which is if one of them says yes, it committed, and the other one said no, it didn't commit, then how do you resolve that issue?

Having highly available participants means that we make sure that we can commit the transaction regardless of various types of failures in the system, because that is the most important thing in order to keep the system moving forward. In other words, in order for us to keep committing transactions and increased transactional throughput with reasonable performance and latency, and for that, the way we have implemented our Paxos algorithm, which basically takes key ranges into shards and shard them, right? So we have the concept of splits. A split is a key range that has been sharded, and then each shard is part of a group, because we have, let's say, now we're talking about servers, let's say. So if there are five servers, then we will take that split and replicate it five ways. Then that five re-replicated split is called the group. So then the group has to agree within itself on whether a transaction happened on a particular key or not. And we can make sure that by splitting up the data and having different groups, overall, even if one group is unavailable, the rest of the database is available until the rest of the database can keep moving forward. And if there's a failure in one split, which is one replica of the key range, the other replicas can continue to move forward. We're doing various things in order to keep the availability of the key range high so that you can keep doing transactions on the key range in spite of both node failures and replica failures.

[00:32:54] JM: Okay. Let's go through a few simple examples of transactions. Could you describe what happens during a read and a write to Spanner?

[00:33:04] DS: Yes, although it's very hard to do it without diagrams. So I'm going to try and keep it simple here in order to not lose our audience.

[00:33:12] JM: Sure. Thank you.

[00:33:14] DS: So there are two types of reads. There are like strong reads, which are called consistent reads in a lot of databases. And there are stale reads. Now, I just want to emphasize again. Spanner is always consistent in that. It will never return to you inconsistent data no matter what. We die by our consistency, which means that when you say stale reads, it is an MVCC system, which is a multi-version concurrency control system, which means we maintain multiple versions of the data in the database. So what you could do is you could say, “I'm okay with data that was correct up to like an hour ago.” This is when you're trying to load your playlist for your particular Spotify or any other music app playlist.

It's okay if it doesn't show for the first pass when you’re just opening the app, for example, that it shows your last – Like something that was five minutes old playlist. That's not that it's incorrect. It's just that it's five minutes old, for example. That's a stale read. A strong read is give me the latest data. I mean, I just added $50 to my account and I want to read the account. I need to see the $50. That's strongly read.

The strong lead always goes to a leader. Now, I'm using jargon, which again we like don't want to get into too much. But basically a Paxos is a group of nodes, and one of the nodes is a leader at any given point in time. So that leader has to – Because that leader is responsible for committing the transaction, the leader also has them most current committed data. And therefore you have to go to the leader to get a ping of what is the last written known timestamp, and then you can serve the read locally.

A stale read does not need to go to the leader, and it can just be served locally entirely without a ping. So what am I saying? Data is always served from the closest replica to the requester, to the client. Data is never transported across a network from where the Paxos group is, just to be clear. This is one of the things that makes Spanner interesting and fast is that the data is never served from somewhere else. It's always served from the local replica closest to you. We are just checking what the last timestamp when the data was written if you ask for strong reads. Does that make sense?

[00:35:39] JM: It does. Yes.

[00:35:40] DS: For transactions, when you're doing a write, that means it’s a transaction. That means it has to commit the mutation, which is the change that you want to do, the write that you want to do is sent to the leader. And then the leader runs a Paxos algorithm within the group, which is replicas. And then as long as a majority of the replicas agree, you can commit, and then return the acknowledgment back to the client.

Now the interesting thing about Spanner and about the way we implemented it going back to some of the technical details is that as part of the Paxos commit, we’re actually sending that mutation to all the other replicas even though we're not waiting for all of them to respond back. This includes read only replicas. So what we're doing is we’re actually sending out the data and the individual replicas only apply that data to their buffers when the acknowledgment has come back that, yes, commit has gone through and succeeded.

What that means is we’re not replicating the data, which again reduces network bandwidth usage and also reduces latency. So the mutations are sent everywhere, which is saying something like, “Add 10 to my account.” The mutations are applied at each replica individually, which increases both the availability and reduces latency of these applies. Then they’re served by that local replica closest to you.

[00:37:00] JM: So we've given people a taste of the architecture of Spanner. I'd like to talk from a different angle now, because you work on cloud Spanner at Google, and that's slightly different than what we've been talking about. We've been talking about the architecture of the Spanner database, but cloud Spanner is a hosted scalable version of spanner that is built to be accessible to people who are using it in Google cloud. How does the product that you've implemented in the cloud, how does it differ from what was originally described in the Spanner paper?

[00:37:41] DS: So actually the infrastructure is exactly the same. So the thing we run internally is the thing we run externally. It's the same underlying infrastructure, which is why we have as much confidence as we do in it, because it is not a different implementation actually. What we've done is have the API be different. So the way you access cloud Spanner is different from the way you access internal Spanner, because obviously we have to make it available on the

Google cloud platform. We have to follow the API guidelines. The IM permissions, all those kinds of constructs have to be there and make it easier for our customers to use it in the Google cloud platform ecosystem. But the actual servers and what they do and how they run the Paxos, it's all the same software stack.

[00:38:25] JM: So tell me about the experience of getting that to market, because that's an interesting challenge. I think it's a modern challenge that more and more companies are doing, or even just all the different products that cloud providers, a lot of them are things that start as internal tools and then the cloud provider goes through some process of externalizing it, which varies from company to company, from application to application. Tell me about the process of taking that internal product and managing it into something that is externally available.

[00:39:03] DS: Sure, and this is interesting and a little nostalgic for me, when I came to Google, I was basically tasked to launching Spanner internally as an internal product. So we actually uniquely ran Spanner internally as a service as well. And so we had a bunch of lessons that we learned about how to –

[00:39:23] JM: Wait. So just to be clear. You joined Google to turn it into an internally-facing product. So you were like an internal product manager for this thing that was sort of experimental at the time?

[00:39:34] DS: So it was really experimental. We knew we needed to solve the shard and MySQL problem for ads and for other systems for Gmail, for Google Contacts, for Google Play. All of the Google photos. All of these systems actually today run on Spanner. Yeah, I'd like to call it – It’s funny. But my new Google project was to launch Spanner internally as a service.

So the paper was released when the system was already built and F1 was the partner that we worked with. F1 has externally published a paper as well about this, but basically they’re the AdWords database. So they were running sharded MySQL and they were like really in need of something like Spanner. So that's how we built Spanner.

But then I was tasked with, to your point, like launching it internally as a service with documentation, with SREs, all that stuff, cool stuff, and then onboarding all of these internal projects on to Spanner from the very systems they were on, like Bigtable or whatever. Because they needed a relational database that scaled.

So actually our team basically learned a bunch of lessons around how to make something like this, like an internal project, into a service already. By the time we needed to launch it as an external service, most of the how do you productize a project were already sort of known. The unknowns were mostly around how do you implement the API on top of it that is a GCP compliant API? Which are more technical and certainly more easier challenge to some degree.

The other thing that we wanted to make sure was that there was a need for something like this externally. So we certainly went out. One of the things I did with my team was validate that there is a need for a scalable relational database in the market. It turns out there was. So we launched it with a lot of love on February 14th, 2017, I have to say.

[00:41:29] JM: What was the process for testing it or getting it ready to be publicly available as a cloud service?

[00:41:37] DS: So there are many answers here. First of all, as I said, this service had already been running internally for years and it basically hosted a lot, if not most of the critical data by the time we launched it. It was already running AdWords. It was already running Google Photos. It was already running Google Contacts, which is that login when you log in to sort of Gmail or any of the Google properties. It goes through Spanner at this point.

So we knew the system was battle hardened. We did a whole bunch of testing around the entire API. We had actually some external customers that basically tested the system before we launched it publicly. So we did a whole lot of both internal and external validation of the API off the end to end paths before we made it a publicly available product.

In fact, one of the things that I think is super cool about Spanner, and I don't know that may people know, is that because we are a pretty critical to Google system, we take our availability and our reliability and our mission-critical nature very seriously. So when the design features or even bugs or fixes or whatever, we're not just designing for the exact use, the common path.

We’re actually looking at what are the failure scenarios, even the more esoteric ones, does the solution actually solve for all of those various use cases?

So we don't ever sort of – We are not the people. We are so deep in the sort of infrastructure in such – Like our systems background is fairly strong. So we never write code and throw it over the fence to see will it work or not. We had better like be super sort of confident that it has been tested fairly before we release it to anyone.

[SPONSOR MESSAGE]

[00:43:32] JM: Logi Analytics is an embedded business intelligence tool. It allows you to make dashboards and reports embedded in your application. Create, deploy and constantly improve your analytic applications that engage users and drive revenue. You focus on building at the best applications for users while Logi gets you there faster and keeps you competitive.

Logi Analytics is used by over 1,800 teams, including Verizon, Cisco, GoDaddy, and J.P. Morgan Chase. Check it out by going to L-O-G-Ianalytics.com/datascience. That’s logianalytics.com/datascience.

Logi can be used to maintain your brand while keeping a consistent familiar and branded user interface so that your users don't feel like they are out of place. It's an embedded analytics tool. You can extend your application with advanced APIs. You can create custom experiences for all your users and you can deliver a platform that's tailored to meet specific customer needs. And you could do all that with Logi Analytics. Logianalytics.com/datascience to find out more, and thank you to Logi Analytics.

[INTERVIEW CONTINUED]

[00:44:56] JM: As you’ve gone to market, you have obviously found customers that need a distributed consistent SQL database. Tell me about the prototypical users. What are the typical types of applications that you're seeing? Do you have any good examples that come to mind?

[00:45:18] DS: Of typical application that use Spanner?

[00:45:20] JM: Well, yes. How would you describe like the typical customer type that needs a distributed consistent SQL database?

[00:45:28] DS: Yeah, I will say, and this was a little perhaps pleasantly surprising to most of us. When we launch we were like, “Oh! Who will be our typical customer?” IT turns out we, at this point, have customers across all industries and all verticals, which is super great. It turns out that most people really just want skill insurance for the relational database, and the relational database is usually their database of choice because of all the good things about databases. Because of the asset transactions, ease of use of application and ease of building of application, and we provide that as a managed service. So customers are looking for not having to maintain their infrastructure, and easy to use services, which a database isn’t obviously easier to use for the application, and SQL is – Everybody loves SQL, I think.

So it turns out most customers are looking even if they don't have large datasets, and I think this is key for anybody looking at Spanner, is that even if they don't have large datasets or very high-throughput system, most people are looking for scale insurance, which is that you can just write your application and forget about it. If your application becomes the next big unicorn, next big app and goes from 10 megabytes to 100 terabytes tomorrow. You don't have to re-platform your system.

Or if you go from one user to 100 million users, you don't have to re-platform your system rate. That is what Spanner provides ultimately, and that’s a very compelling message and a very compelling use of a system. So that's why we've seen adoption and interest across the board.

[00:47:06] JM: If I run like an e-commerce store and I've got a MySQL database and I’m not a popular e-commerce store yet, but I have dreams of someday becoming a popular e-commerce store and needing a scalable version of my relational database. Is it going to be significantly more expensive if I move on to Spanner today or is the costs going to be the same as just operating Vanilla MySQL?

[00:47:40] DS: That's a great question, first. So the cost depends. Depending on how you look at it. Do you look at it from a total cost of ownership perspective or do you look at it from just the hardware and software license perspective? Because, obviously, MySQL is free, but how many people do you have to hire to operate that MySQL?

Now, I will say, for a lot of people who are just where simple vanilla MySQL is good enough. There are two answers I give. One, yeah, spanner is maybe more expensive. It is a managed service. We have an army of SREs managing it and carrying pagers.

[00:48:12] JM: Sorry. I should have said maybe compared to like a managed MySQL cluster. If there's some managed MySQL version out there.

[00:48:20] DS: Yeah. Sure, which is our cloud SQL offering, and it's very good. So if you really would just want to have MySQL and a managed version of it, then cloud SQL is a great option. There are two answers to Spanner. It depends on the costs. It depends on whether you're using a regional or a multi-region instance. I think people don't realize that we are replicated by default. So there is cost of that to us, and that's obviously part of the service.

Having said all of that, I think we are very, very cost-effective for medium to higher workloads. I think for smaller workers today, we may not be, and that something we're actively working on, and we have some features coming out both later this year and next year as well that further reduce our entry price points, which we are really excited about. But I can tell you right now that we're very cost-effective for the higher end and medium end workloads for sure.

[00:49:14] JM: Well, will the multiregional feature, that seems like it's pretty useful even if I'm just a small e-commerce store, because if a tornado comes through and destroys the a center region, then I lose all my data if I'm not multi-region, right?

[00:49:32] DS: So that's exactly it. Multi-region – So we offer 9999 of availability for our regional product and 99999 availability for multi-region instances, and that's exactly the point, that first of all, we are replicated. So we're going to be sort of highly available even if one GCP zone goes down. So one availability zone goes away or one replica goes away, because –

[00:49:54] JM: Oh, okay.

[00:49:55] DS: Right? So we are more available than just one instance of any database, whether it's managed or not, because we are by default replicated. In the multi-region case, to your point, if something happened that takes out am entire region, you are replicated in other regions. So you continue to function in spite of that problem, which is obviously cool in my opinion.

The thing is that we are also synchronously replicated, which means that there is no failover and recovery, meantime to recovery window that you have to wait for your application to get back online. That’s not something that is user visible at all. Because we can go from serving from one data center and in one region in another region, which is of course great for when you want peace of mind for your really critical application and continuing to have availability and service regardless of what's going on. So we have like him multi-region configurations that are in three different regions all the way up to nine different regions.

[00:51:05] JM: What's the hardest engineering problems that you've encountered while working on the cloud version of Spanner?

[00:51:11] DS: This is something I alluded to earlier. I think trying to solve any problem, technical problem, such that you can meet the needs of the user that has 10 megabytes of data all the way to hundreds of petabytes of data with availability in this highly distributed synchronous environment. Every problem, every feature that we add makes it take more and more complicated, or technically challenging to look at that problem space then explode it out. Because it has to be performant. It has to be highly available. It has to be resiliency failures and it has to be correct. Because we are mission-critical database. We can’t give you inconsistent, incorrect answers. So every problem, every technical problem has to have those aspects to it in order for it to be something that we add to the system, we add to our feature set.

[00:52:13] JM: Okay. Just a few more questions. One on kind of team management and collaboration. So I imagine, the cloud Spanner team is – How many people is it? Well, like 10 to 20, or how big is it?

[00:52:26] DS: It’s a large enough team that we can meet our customers’ needs.

[00:52:30] JM: So tell me about implementing a feature or scoping out a feature and how the team works together. What's the division of responsibility across the team? You're a product manager. There's probably a bunch of different types of roles on the cloud Spanner team. Just tell me about the dynamics of the team.

[]00:52:54 DS: We really take distributed very seriously. We have teams across the country and actually across the globe that work on Spanner, and we obviously, like any other product, have product managers, software engineers, SREs, sort of test engineers, frontend engineers, designers, and there are a bunch of stakeholders around us to just – Again, because we are a critical system to internal Google as well. So there's a lot of people, stakeholders that we have to get in line to their TPMs. So I think any – I would say any job function that is technically related at all, we have that in Spanner, and we have to essentially align all stakeholders and all sort of team members to make sure that we move forward in the right direction. I think that's the fun challenge of any engineering team, to be honest with you. Anybody that works in an engineering team, whether it's a startup or anywhere in the enterprise, whatever I've said is not news to them, and think that's the fun part of my job is actually aligning people and stakeholders to do what is best for our customers.

[00:54:07] JM: The cloud is a very big business, and I think people even today, underestimate how big the cloud is going to be and how much it's going to change software engineering. What kinds of new abstractions do you expect us to see in the cloud market in the next 5 to 10 years?

[00:54:28] DS: That's an interesting question. I think it's interesting you say that, because to your point, and all the analyst firms have said this too. We're going to see more and more and more cloud adaption and there's going to be – The shift has started, but it's just the beginning and most of these enterprise workloads will basically move to the cloud in the next 5 to 10 years and coupled with the already cloud native and cloud digital companies that are, again, they're just starting that curve. So there's going to be an explosion of data and applications on cloud in the next 5 to 10 years again. So I think we are at critical juncture for cloud, to your point.

But in terms of abstractions, I think it's across the board. Basically, everybody's trying to understand how to go from an on-prem environment and all the tooling and support the worked on the on-prem environment in a cloud native environment. So most of the abstractions I can think about are basically going to be around tooling in my opinion. There are already enough high-level languages. There are already a lot of ways of programming on the web, if you will, or all those kinds of things. I think it's mostly around how do you tool around on-prem and cloud transparently to the user. That would be one thing that I would think is something we'll see a shift in in the next few years.

[00:55:48] JM: Okay. Deepti, thank you for coming on the show. It's been fun talking to you.

[00:55:51] DS: Oh! Thanks so much, Jeff. I hope your audience enjoyed our discussion too.

[END OF INTERVIEW]

[00:56:03] JM: Software Engineering Daily reaches 30,000 engineers every weekday, 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. We have a variety of sponsorship packages and options.

Thanks for listening.

[END]