EPISODE 754

[INTRODUCTION]

[00:00:00] JM: When a user interacts with an application to order a ride with a ridesharing app, the data for that user interaction is written to a transactional database. A transactional database is a database where specific rows need to be written to and read from quickly and consistently. Speed and consistency are important for applications like a user ordering a car and riding around in that car, because the user's client is frequently communicating with the backend database to update the session.

Other applications of a transactional database would include a database that backs a messaging system, or a banking application, or document editing software. Generally, when you are the user interacting with a piece of software, that usage is being written to a transactional database. Oftentimes this database is called an OLTP database.

The data from a transactional database is often reused in analytic databases. An analytic database can be used for performing large scale analysis, aggregations, averages and other data science queries. The requirements for an analytic database are different from a transactional database, because the data is not being used for an active user session. To fill the data in an analytic database, the transactional data gets copied from the transactional database in a process called ETL, and then once the ETL has happened from the transactional database to the analytic database, the analytic database can be used for fast queries on entire columns.

For example, maybe you want to sum all the amounts of transactions across all the rides in Uber on a given five-day period. That would be a longer query. It would be a more difficult query if we were just querying, for example, an operational Mongo database than if we're querying an analytic database, where the columns are arranged for fast querying of the entire column.

Because there is this separation of the transactional databases and the analytic databases of many organizations, there is some problems that can occur for data engineering in this ETL process where you have to move data from the transactional data store to the analytic data store, and those problems are things like consistency and just annoying ETL process that nobody likes to do.

To address these problems, some newer databases combine the transactional and analytic functionality in the same database, and these databases are often called NewSQL. These databases are often also built on higher-level abstractions and more modern abstractions, like RocksDB and Kubernetes.

TiDB is an open source database built on RocksDB and Kubernetes. TiDB is widely used in China by high-volume applications, such as bike sharing and massively multiplayer online games. Kevin Xu joins the show to talk about working at PingCAP, the company he works for, which is building TiDB. TiDB is also open source. Kevin talks about modern databases, distributed systems and the architecture for TiDB. This is a good companion show to some previous episodes we've done on Kubernetes, especially Kubernetes in China, which is a recent show we did where we talked about the usage of Kubernetes in China, and also RocksDB, which the show I did on that has not aired yet, but it's a really good one and I'm excited to have a use case of RocksDB that we're covering before I actually air the show about RocksDB itself. But RocksDB is a cool technology and we'll certainly talk about it more in a future episode.

With that said, I hope you enjoy this episode about TiDB with Kevin Xu.

[SPONSOR MESSAGE]

[00:04:22] JM: OpenShift is a Kubernetes platform from Red Hat. OpenShift takes the Kubernetes container orchestration system and adds features that let you build software more quickly. OpenShift includes service discovery, CI/CD built-in monitoring and health management, and scalability. With OpenShift, you can avoid being locked into any of the particular large cloud providers. You can move your workloads easily between public and private cloud infrastructure as well as your own on-prim hardware.

OpenShift from Red Hat gives you Kubernetes without the complication. Security, log management, container networking, configuration management, you can focus on your application instead of complex Kubernetes issues.

OpenShift is open source technology built to enable everyone to launch their big ideas. Whether you're an engineer at a large enterprise, or a developer getting your startup off the ground, you can check out OpenShift from Red Hat by going to softwareengineeringdaily.com/redhat. That's softwareengineeringdaily.com/redhat.

I remember the earliest shows I did about Kubernetes and trying to understand its potential and what it was for, and I remember people saying that this is a platform for building platforms. So Kubernetes was not meant to be used from raw Kubernetes to have a platform as a service. It was meant as a lower level infrastructure piece to build platforms as a service on top of, which is why OpenShift came into manifestation.

So you could check it out by going to softwareengineeringdaily.com/redhat and find out about OpenShift.

[INTERVIEW]

[00:06:30] JM: Kevin Xu, you are with PingCAP. Welcome to Software Engineering Daily.

[00:06:34] KX: Thank you so much for having me, Jeff. I'm a big fan of this pod.

[00:06:37] JM: That's great to hear. I want to explore new databases with you today. There's a brand of database called NewSQL database. Can you explain what that term NewSQL database means?

[00:06:51] KX: Absolutely. There're certainly a lot of different variations of this definition, but NewSQL is essentially a relational database. So that is the most traditional type of database, relational database that actually scales very easily horizontally, just like a NoSQL database, which is how you get this NewSQL versus the NoSQL versus the traditional SQL paradigm. Like you said, NewSQL is this new generation of database that has really just come on the scene at say in the last 5 years or so, maybe even less, that is trying to solve really hard problems in the database world that really hasn't been solved before.

[00:07:32] JM: There are perhaps some trends that contribute to the increased necessity of a NewSQL database. We have trends like cloud computing, big data or machine learning. We have the rise of mobile. Which of these trends are contributing to a necessity of new kinds of SQL databases?

[00:07:54] KX: Right. So I think the two big trends that are really contributing to this is the combination of cloud computing and big data, and if I may just do a quick kind of a tour of history, I guess, very briefly about the database world. So like I mentioned, we first have the relational database starting in the 80s, perhaps even earlier than that, where you just literally have a table, like an Excel Spreadsheet or something else you can visualize along that line that stores all your data, rows, column. Everything is good because there really wasn't that much data to begin with.

Then you did have the explosion of the internet, and then you also have mobile after that where you started to get a lot of data, because everything is becoming digitized, and that gave rise to the NoSQL category of databases that are particularly architecture or designed to solve the scalability problem very easily, because otherwise growing your database just with the table structure originally, the relational database structure, is very, very difficult to do quickly. So that's kind of the use case that NoSQL databases were addressing.

Then after that took its form, and it's forming. NoSQL databases are still being used very, very heavily in a lot of places. What people realized is that the thing that NoSQL databases does not guarantee is what's known as data consistency. Meaning that it's not a sure thing that all the data that's in your database is actually accurate or looks the same at any given time.

So what the NewSQL category of database, like TiDB and others are trying to address is that we want to make sure we guarantee consistency first and foremost, because our data does need to be accurate to do the things it's supposed to do and have that horizontal scalability very elastically, just like a NoSQL database so that people can have kind of the big data scale that everyone needs at this point, but also be assured that the data that you have in your database system is actually strongly consistent as supposed to not consistent, which could have caused a lot of different problems depending on the business that you're in.

[00:10:19] JM: When we started giving up some consistency restrictions on our databases, why were we doing that and how are we able to now reclaim strong consistency?

[00:10:34] KX: Right. I think everything that's in the database world in the technology world, especially in the infrastructure side of things was related to computing, cloud computing or database. There are a lot of tradeoffs involved. There's no one set of solution that kind of just does everything you want for you, and if any project or product is presenting themselves that way, that's a bit of a malpractice. I think what people had to compromise during the NoSQL era is that let's see if we can live without a little bit of consistency tradeoff, because we just have too much data that we have to manage, too much that we have to store and still make some use of it, because the emergence of the internet and mobile technology. That's kind of revolutionary and also unprecedented in the technology world.

Only after a decade or so of say adapting those SQL technology, and frankly probably getting burned in many situations where this consistency is really valuable even in the context of needing scalability that people like our company and others start thinking about, "Hey, we can really make this tradeoff completely in the other direction. Let's figure out ways where we can guarantee consistency and still have the type of horizontal scalability a very easy scalable solution that the NoSQL world is presenting to the market.

[00:12:07] JM: One axis that has changed is we talked about the earlier generations of cloud database and cloud data infrastructure versus today is cost. It has gotten cheaper to buy machines, and physics hasn't changed. I guess you could say bandwidth and the network latency has gotten better overtime, but from the data infrastructure companies that I've talked to, one thing that has improved is this cost structure, and we can use cheaper data systems and cheaper cloud transactions to paper over some of the consistency issues that we had to suffer under in the past. Is that accurate? Are there ways where we can just pay our way out of eventual consistency into strong consistency?

[00:12:56] KX: I think there are certain levels in which you can say throw machines at the problem nor just throw hardware more specifically at the problem, and that is in a way what NoSQL and also NewSQL databases are doing. The whole kind of notion of horizontal scalability as supposed to say vertical scalability is that you just add similarly typed or similarly cost machine into your cluster and you just keep on adding them and adding them and the fact that you just can add them means more or less that your entire capacity is growing to handle your growth or your database need, and that is also in a way what cloud platforms, whether it's private or public, is presenting that for their user, right? Like if you're a cloud user of any scale, small or big, more or less you don't know or you don't really care what machine is actually underneath you to support infrastructure. You just want it to work.

Now I think the flipside of that though is that just throwing the machine at the problem is only part of the puzzle. The other puzzle that our work at PingCAP building TiDB is trying to fill is that there does need to be a more sophisticated level of software, which in form is a database. Database is just a software, it's just an application at the end of the day, can make that new world, this new horizontal world work better, because the problem that's been introduced in that framework is that you not have to make all these different machines work together really well to guarantee the consistency and the scale and have the high-availability of the data so your system or your business never goes down no matter what happens. So that is I think the two pieces of the puzzles that we are both trying to fill either from the software side or from the hardware, the commoditization more precisely of the hardware side.

[00:14:57] JM: As we continue our top down exploration of databases, and we will get into what you're doing at PingCAP and the database that you have helped contribute to, TiDB, but just to go a little bit deeper into the high-level use cases, we did a show a while ago that really stood out to me, which was with something from Uber about OLTP versus OLAP databases. This is online transaction processing versus analytic data processing, online analytic processing. Whatever that acronym actually extends to.

[00:15:35] KX: Industry jargon.

[00:15:36] JM: Industry jargon. But what does this actually meant in the case Uber, Uber was such a good case study for this concept, is if you're summoning in Uber and then you take the ride and then the ride finishes and then you get out of the car and there's all these payment that goes on across the transaction. That's a transaction. First, you summon the Uber and then that's going to alter the Mongo entry of your session that you're starting with this driver and then you're in the car with the driver and maybe the database is getting updated overtime very quickly, this transaction processing, then you get out of the car, and then the Mongo database gets updated again because the ride has ended. Now you're going to get charged for it. This is transactional processing. This transactional nature where a database entry is getting updated on a regular basis and the user is interacting with that data entry.

Then overtime, let's say every 15 minutes, Uber is extracting all of those transactions and loading it into an analytic processing database so that they can do things like aggregate the costs of all of the rides in the last 15 minutes, which is a very different kind of analysis than, for example, let's update this individual user's ride. So you have a very different kind of query as supposed to looking at a specific database transaction, who is in the car right now? Who are they riding with? Versus let's aggregate across all of the data that happened in the last 15 minutes or the last year or something like that.

These are two dramatically different types of queries. How does the fact that we have these very divergent query semantics that we need to ask of our data? How does that affect our database architecture?

[00:17:22] KX: Right. So I think the Uber example is very, very telling in the sense that the transactional workload, OLTP, and the analytical query, OLAP [inaudible 00:17:33] answer to very different questions that actually need two very different sets of either requirements or criteria, and that actually is giving rise, or it has always been the case actually where people who are designing databases would typically either design for one purpose or the other. So the design decision if you're a database builder, that goes into designing a database that processes the OLTP workload, the transactional workload very well, is a very different set of consideration than the OLAP or the analytical processing workload.

The differences, broadly speaking, is that for the transactional side, you have to be able to do these transactions, these back and forth very, very quickly and very, very accurately all the way down to the user level. So all the way at the – If you're in a table, there is a row that probably has just names on it if you're an Uber user. There's another row that has Kevin's name on it, because I'm also an Uber user. Then the updates as we take different rides, as we pay different kinds of drivers, the fairs that we incur on specific rides needs to be updated very quickly and also concurrently when you operate at the scale of not even just Uber. Even something much smaller in terms of the company size needs that concurrency to be done very, very well.

But the nice thing with that workload is that the query length, literally the query length, is very, very short. So the operation is supposed to be very quick, but you have to be able to do that very quickly efficiently at scale and without of course messing up the accuracy of it. Then the analytical side is where you start to aggregate different trends or buckets or portions or this large database that you have accumulated overtime from the transactions that you process to gather business intelligence, or just to generate reporting and things like that, and those type of workloads usually gets extracted based on column of a data. So the transactional side is more row-based and the analytical side tends to be more column-based.

So when you see this more analytical friendly database design, there's a term out there that folks might have seen called like wide column store. That's like a type of database. Because it's designed to help you pull up big amounts of data by column when the query itself doesn't really care that much if it's Jeff transaction, or Kevin's transaction or somebody else's transaction. It just wants to look at the world and the aggregate. Another piece kind of going back to the consistency argument or the issue that we were talking about in the NoSQL world is that in an analytical setting, even if certain data is wrong or even if certain data is missing or just not updated to the most updated status or state that is supposed to be is probably okay given the need of just analyzing and getting [inaudible 00:20:44] on a macro level.

But on the transactional side, you cannot comprise that level of atomic updates and make sure every single cell is accurate, because I, Kevin, I am going to check whether I'm paying this driver accurately. I want that number to be what I'm actually paying him for. I don't want it to be $5 more for some reason, and you want that to be the same, and the driver probably wants is tips and stuff like that, and that has to be accurate too. But that has nothing to do with the analysis side of things. So that gives two different kinds of database design broadly speaking.

[SPONSOR MESSAGE]

[00:21:28] JM: Triplebyte fast-tracks your path to a great new career. Take the Triplebyte quiz and interview and then skip straight to final interview opportunities with over 450 top tech companies, such as Dropbox, Asana and Reddit. After you're in the Triplebyte system, you stay there, saving you tons of time and energy.

We ran an experiment earlier this year and Software Engineering Daily listeners who have taken the test are three times more likely to be in their top bracket of quiz scores. So take the quiz yourself anytime even just for fun at triplebyte.com/sedaily. It's free for engineers, and as you make it through the process, Triplebyte will even cover the cost of your flights and hotels for final interviews at the hiring companies. That's pretty sweet.

Triplebyte helps engineers identify high-growth opportunities, get a foot in the door and negotiate multiple offers. I recommend checking out triplebyte.com/sedaily, because going through the hiring process is really painful and really time-consuming. So Triplebyte saves you a lot of time. I'm a big fan of what they're doing over there and they're also doing a lot of research. You can check out the Triplebyte blog. You can check out some of the episodes we've done with Triplebyte founders. It's just a fascinating company and I think they're doing something that’s really useful to engineers. So check out Triplebyte. That's T-R-I-P-L-E-B-Y-T-E.com/sedaily. Triplebyte. Byte as in 8 bits.

Thanks to Triplebyte, and check it out.

[INTERVIEW CONTINUED]

[00:23:17] JM: To start to get into TiDB, let’s give an example of a company that might use TiDB. So there are a few examples actually I want to run through. First one if a large bikesharing company. So let’s think of a bikesharing company. It’s kind of in some ways like the Uber example we just gave, but TiDB is this multi-model database that can help us – Well, I’m not sure if you want to call it multi-model, but it serves both OLTP and OLAP queries. Just to give people more context on this Uber show that we did, they have a complex data pipeline where they are taking the transactional processing data, the record-wise data. For example, like Mongo documents or it might be a PostgreS database where they have the records of each of the users established in a row-wise fashion, like a row including each column. So you have user, age, location, etc. All of these different fields and they have this regular job that translates all of those rows into columns. So you have all of the data collocated so that you can more easily aggregate, for example, all of the costs of all the – You can aggregate the number of users in the United States, for example, because it's all arranged in one column. You can do these very quick aggregations. So they have this ETL job, extract transfer load, where they take the data from the transactional database and put it into the OLAP database.

The whole idea some of these multi-model databases is that you don't necessarily have to do this. Everything is just in one database. Maybe it's replicated or maybe it's in this other kind of format. But let's talk about TiDB in this context. So when we're talking about TiDB and we're thinking about it in the context of a bikesharing company, why would a bikesharing company want to use TiDB?

[00:25:16] KX: Right. There are two reasons why someone like an Uber or the bikeshare example that you talk about, which is I believe that company is called Mobike, which is one of the largest bikesharing, dockless bikesharing platform in the world. I think they're one of the first that came up with this model to begin with. The reason why they use TiDB is twofold. One is this pure scalability needs that they need to have as this platform or their business just takes off like wildfire, and that accumulates a lot of data. It is not just purely for scalability sake. It's also to make sure that the database is guaranteeing consistency throughout the record essentially, and I'll talk a little bit about why that's important for their use case.

Another element to Mobike using TiDB is also that they are a big MySQL user, and TiDB is designed to be first and foremost MySQL compatible for the reason that MySQL users tend to have scalability problems and traditionally how they are able to get by is through a process called manual sharding, where you basically breakup these tables as they get too big into smaller into tables so you can scale them, but you have to manage basically how these tables relate to each other in a manual sharding policy sort of way. TiDB takes away all that manual operational work, and we do that automatically within the system. I'll talk a bit more about how that works in a little bit.

So MySQL scalability is number one, and number two is that similarly to Uber, Mobike also needs to be able to generate analysis of their entire platform really ongoing throughout the day, and that is what gave rise or gave credence to TiDBs architecture, which really puts both OLTP workload and the OLAP workload that you were talking about into the system, into this hybrid database. I think the marketing jargon is HTAP, so H-T-A-P, or hybrid transactional analytical processing database.

Regardless of the term, is to have both of these workloads in the system while minimizing this ETL thing that you were talking about, because this ETL, extra transform load process, causes a lot of delay actually between the transactional workload and making them available for the analysis to happen. TiDB as a system minimizes those need to have ETL processes running on throughout the day. So you can access your transactional records a lot more quickly so you can do what is really basically a real-time analysis of your transactional records.

So the MySQL compatibility with the scalability and the hybrid structure or the architecture of TiDB is what Mobike is using us for.

[00:28:21] JM: So is TiDB doing replication of data or is it storing the data only once but in a way that is easier to query?

[00:28:36] KX: TiDB is automatically replicating data. So what's happening is that – And I can kind of go over maybe a bit of the TiDB architecture and this will help listeners understand a little bit of how that's actually being done. So on the very top you have your application. Let's just say is an application that uses MySQL as a database, and because TiDB is MySQL compatible, right below that application you have a layer of servers that are called TiDB servers, and these servers are, number one, stateless. So they're actually very similar to microservices if you come from the cloud native world, and all it does is that it translates MySQL queries into a key value format and also sends that query down to another layer underneath the TiDB server, which is called TiKV, and KV again just stands for key value.

So essentially you have a relational layer that speaks MySQL on top of a key value layer, and in this key value layer, that is where the system itself would number one automatically breakdown the data into smaller partitions, or chunks, or shards, these are all kind of interchangeable terms to refer to this chunk. Then we use a consensus protocol, called Raft, that automatically makes copies of this data on put them on to different TiKV machines, and that just depends on how large your TiKV cluster is. It could be three machines, which is the smallest by default. It could go out to hundreds of different machines scaling horizontally. The Raft consensus protocol will distribute or make copies of these data automatically and then put them on to different TiKV machines so that this data will always be available in case certain machines goes down or there are other hardware failures that happen in your cluster, which always happens, and that's why we need to make these extra copies of each pieces of the data and make sure they're all consistent with each other, meaning that they all have to look the same in order to guarantee the data consistency that we were talking about with this NewSQL category of databases.

[00:31:00] JM: You're describing a series of layers. The top layer being this SQL interface, and then the middle layer being this translation interface that translates SQL queries into key value query, that transforms SQL queries into key value queries, and the lowest layer, the storage layer is TiKV, which is a key value store that replicates the different mapped entries. But here, we can wonder, "Well, okay." So we were just talking about OLAP data where you have the data arranged in columnar format. Like maybe I have a column that is all of the values of transactions in Uber so that it's just like dollar amounts in a single column. But if we're talking about a key value store, that sounds like something that is something more like a document-based level at the lowest level.

So how is this key value storage layer at the bottom, how is this going to allow us to do OLAP queries?

[00:32:05] KX: Right. So that's a really, really good question. I'm really glad you asked. So like you pointed out, the bottom storage layer is a key value structured database, and the reason why we chose that structure is that it is very helpful for guaranteeing transactions first and foremost, guaranteeing consistency, strong data consistency first and foremost. That was the first part that we want to make sure we guarantee our system does, which key value pairs do really, really well.

There are two reasons why we can do OLAP also in the system very well. One is that because the storage level, which is TiKV and the, say, compute layer, which is the TiDB server that I mentioned that speaks MySQL. These are two different components. They're two literally different code bases. You can put any other compute engine on top of TiKV and this engine could be more specialized towards OLAP type workloads, and we actually do have one that we support natively in the TiDB ecosystem as well called TiSpark, which is essentially an Apache

Spark plugin that also sits on top of TiKV that would do what Spark does really well, and that has nothing to do with the SQL or MySQL side of things. It's independent of that, but it talks to the same data stores, which is TiKV, and that effectively removes the ETL process, or the ETL delay that we were talking about in a more traditional architecture. That's how we are able to make this hybrid transactional analytical processing architecture work within TiDB.

Now, that answers part of your question. The second part you're talking about is, "Well, it's still key value pair, right? It's not really a column store. It's still not natively amenable to OLAP queries." Something that we're working on, and I'm pretty excited to share this piece of – I guess you can call it news, is that we are working on actually having another column-based storage to essentially be placed side-by-side with the TiKV layer. So you can kind of imagine two things on the top, which is TiDB, TiSpark, and then you can see another layer on the bottom of that, which is TiKV, the key value side that guarantees transactions and strong data consistency and another engine right next to TiKV that is actually column-based.

What we do is that we use the Raft consensus protocol to essentially make an extra copy every time it's doing the automatic replication and put that extra copy into this column store. Then on top of the column store, you can have a Spark or you can have some other kind of engine that will just do the OLAP query in this column store without really interfering or having to talk to a key value store that's just right next to it.

Really with all that components together, you're really completing the so-called HTAP story that we are working on filling. So then you really have a performant OLTP database and an OLAP database all in the same system again without ETL processes actually so that there's very little delay in accessing your live transactional record to do real-time analysis.

[00:35:41] JM: Okay. So if I understand correctly, the state of TiDB today is that it's a strongly consistent transactional data store with kind of a SQL interface on top, a key value storage layer at the bottom. It's most effective for transactional processing, like getting into my Uber or I'm getting my bikesharing ride and I need a globally consistent database for this and it's written to disk. Then if we want to do some faster operations on it in-memory or perhaps some machine learning application in-memory on top of that, we can use TiSpark, and we can pull the data from the key value storage on disk layer into TiSpark and we can utilize Spark's distributed in-memory processing and have some faster querying there.

Then what you just announced what you're working on is that there is going to be a columnar data format, data storage format, alongside TiKV so that the data is not only represented in a transactional friendly key value storage format, but it will also be represented in a columnar format, and then you can also use TiSpark to potentially pull in some of those columns if you wanted to do columnar oriented analysis on a faster basis, because it would be in-memory. Is that right? Did I get everything right or is that anything –

[00:37:14] KX: Absolutely. You got it perfectly.

[00:37:16] JM: Okay. So there are some other databases that are doing this kind of thing. So I know the TiKV layer, for example, is in spired by the Google Spanner project. Spanner has been around for a while. What did the Spanner project do that was so memorable and how does the TiDB, TiKV projects compares to the Spanner world?

[00:37:44] KX: So we, as a company, PingCAP and also as the project itself, TiDB, it really traces back our kind of architectural origin to the Spanner paper that was published a few years ago. That was when the PingCAP founders were all infrastructure engineers working on some large internet companies. They were literally doing their sharing of MySQL databases themselves. Read the paper and thought, "This should be the future of what a database ought to be," which is this globally distributed, consistent and easily scalable database that can guarantee consistency first and foremost. You can have a relational way of using database still with the scalability that we were talking about before.

Another actually Google either project, it is also a paper now, that we have a leveraged as an inspiration for design is called F1. So you can kind of Google the F1 paper from Google as well, and that actually is more akin to the TiDB layer, the MySQL layer that we have built, because that is really the relational interface on top of TiKV. So sort of the mix and match between these two worlds broadly speaking is that TiKV is generally speaking an open source implementation of Spanner, while the TiDB servers is an open source implementation of Google F1. F1 is this thing that they use internally in Google to support their advertisement network, which is of course very, very massive and also needs to be very, very accurate. So that is kind of the side-by-side picture that you can visualize between what we're working on in the TiDB world and what the Spanner, I guess, inspiration or design is coming from. So that is sort of the origin and also how Spanner fits into us. There're a few other companies or products out there too that are also "Spanner inspired". There's almost this thing called a Spanner family of database that of course Google Spanner being one of them, TiDB one of them, and a few others being part of that category as well that is trying to solve perhaps similar problems in different ways. Though I would say one thing that I think TiDB stands out and is a layered architecture or a component-based architecture that lends itself really well to hybrid workloads in ways that others, at least to my knowledge, may not have gotten to that level yet.

[00:40:22] JM: Okay. Spanner – Was the breakthrough in Spanner the global consistency, the emphasis on consistency across all the database replicas in the environment?

[00:40:38] KX: Correct, and that's in a way a very Google-like problem, right? There are very few companies that are at the scale that Google is at. So internally I'm sure when they first started thinking about building Spanner, it is for their own use. The interesting thing now is that with more and more large companies like Uber, Lyft, Mobike and others. Some of them are TiDB users. They all have global footprint now and is not nearly a uniquely Google problem to solve anymore, and that's why I think there is a demand for Spanner-like implementations or databases out there where you do want geo-replication across geography. You want to have that consistency. You want to have that redundancy. Also, be able to have one single database system that could serve users in multiple regions, regions in terms of geography regions, and time zones, and countries and so on and so forth.

[00:41:44] JM: Where does spanner chronologically fall relative to the Amazon Dynamo paper? Was Spanner a few years after Dynamo?

[00:41:53] KX: I believe so. Don't quote me on this off the spot, but I think it was published, the Spanner paper at least, I think 2012 or something around that line. Yeah, maybe a little bit later, maybe 2014.

[00:42:05] JM: Okay. Then the Dynamo paper earlier, I guess Dynamo was like the weekly consistent kind of distributed data system, right?

[00:42:15] KX: Right. It's more like I think like a document store to begin with. It's funny that you mentioned, the weekly consistent framework, right? Which is more representative of the NoSQL category of databases. I think the thing that Spanner that really came out and in a way really got a lot of people thinking is that we don't have to compromise consistency just for the sake of horizontal scalability. We can in a way have our cake and eat it too, and that's what got a lot of people excited.

[00:42:46] JM: How do we have that though? What are we giving up? I think we talked about this a little bit earlier, but what are we specifically trading off? What did Google do in the Spanner paper to accommodate for the obvious – I mean, if you have a globally – If you want a globally consistent database, it seems like every time I make a transaction to this database, if I'm writing to the database, there are many transactions where I'm going to have to do some kind of global communication. There's going to be this really heavy latency penalty as all my database replicas around the world are communicating with each other. It feels like this blocking slowness that's going to be a painful tradeoff. How do you get around that?

[00:43:31] KX: Right. I think tradeoff is the key I think element that I want to focus on, kind of to address your question, which is really, really a hard question to answer. Because when we started talking about this in the beginning, it is all about tradeoffs. The perception that you can have your cake and eat it too is ultimately not accurate when it comes to reality. It's more like what do you need and how can we evolve our technology to get as close to what you need from the business perspective. The Google being its own kind of ecosystem has certain hardwares that could guarantee consistency especially along the line of like time accuracy. There's this thing called True Time API that is part of the Spanner design that is very much specific to Google's own data center essentially that is not available to anybody else, that is outside of Google. So other solutions find other ways to solve this problem.

But the big tradeoff typically is you have this relationship or this balancing act between your throughput, which is the capacity that your system needs to have to process transactions or analytical queries, say, on a per second level, right? That's how you know or that's the measurement that you need to keep track of to get the capacity, which horizontally scalable solution can offer that capacity seemingly on a dime. Just spin up new machines, add it to your cluster, you have more capacity.

The flipside of that, like you said, is latency. So how long can you realistically wait for these transactions to get processed as your capacity grow? And there's always going to be a bit of a degradation on side or the other of this balancing act. If you're going to just over, over optimize for latency – So latency basically meaning speed. How quickly you can process something? Then you want to keep your data small, everything in-memory, and that still is the fastest way to do it. But if you want to have this one globally distributed database that can suck in all the workload in the same system, because it isn't just about speed.

It's also about operational simplicity for the team, whichever team working for a company that needs to manage this system is always a lot more complicated to manage a bunch of different databases as supposed to one single database that could more or less get to what you need with a single system so there's less for you to manage. That's what a lot of people use perhaps Spanner and certainly TiDB for, is the simplicity of the management or the operational cost, which has nothing to do with performance per se. But that's the tradeoffs that teams think about as they think about, "Do we want a database like this in our system? Can we have a meaningful, reasonable balance in the tradeoff between throughput, latency and the operational maintenance that we need to devote to manage our infrastructure?"

That is kind of where we are getting at in terms of the stage or the point that we need to be so people understand the value of this kind of database, which isn't to promise everything in the world, but it does offer quite a bit if you want to embrace it.

[00:47:01] JM: Yeah. Well, it sounds like, as you said, the architecture is a big deal. The way that you've created the architecture, and it's open source, so that people might iterate on the architecture, or the open source community might develop and modulate the architecture more. To get deeper into the architecture, we can talk about the storage layer, which is where these key value regions are being stored. So the bottom layer of TiKV – Well, TiKV, where the key value entries are being stored, which is queryable via the top layer, the TiDB SQL layer. Again, your SQL transactions get translated into transactions that can return these key value entries.

At the bottom layer, in this TiKV layer, you have a storage engine of RocksDB. So you have replicated RocksDB instances. So RocksDB is a storage engine that came out of Facebook. Can you explain what a storage engine is and what RocksDB does for the TiDB database stack?

[00:48:15] KX: Yup, absolutely. So RocksDB or any other storage engine is I think is the last layer between, say, something like a TiDV, which is a distributed storage layer and the actual physical disk. So in this case, SSDs, but it could be any other kinds of physical disk where your data is physically recorded on.

So RocksDB is the last layer between TiKV and the physical disk or the machines or the bear metal that were the data is actually physically stored. Tracing RocksDB's history a little bit, it was actually a fork of another storage engine called LevelDB that was first developed by Google. Then Facebook forked it and decided to do something else with it, which became RocksDB for their own use case. This might be getting into the weeds a little bit, but I do want to explain kind of what is the difference between RocksDB and maybe some other storage engine that is out there that folks might have heard about. But there are two mains that data gets stored in these storage engine, two different kinds of data structure essentially. One is called LSM tree. It stands for lock structured merged tree, and this is a type of structure that is more optimized for write operation performances, and this is what RocksDB has.

Kind of the other type is called binary search tree, or B Tree, and this is another perfectly legitimate good storage structure that is more kind of geared towards read performance optimization and things like that. There are also some elements of compression rate that the LSM tree or the RocksDB way of doing things is much more amenable to more efficient compression, which means you can store more data essentially with less space. That's actually a pretty important consideration, and I think one of the considerations that Facebook had in the beginning is to get more out of their physical resources, because these disks are pretty expensive, and the rate that Facebook grows, it could be a huge investment to spend more money buying these disks, and if they can get more out of it, they should. So that was one of the original motivation of Facebook building RocksDB.

Of course, the whole thing is open source as well. When we started building TiKV, there was always the consideration that you build your own storage engine right below TiKV or you leverage something else that is already in the open source world that is much more mature. The maturity of this piece of component in the layer is critically important. After assessing RocksDB, which was battle tested by Facebook internal usage and also the motivation of RocksDB is actually very similar to a lot of the kind of classic users that TiDB has, which is also big MySQL users. Facebook is also a big MySQL user, and they all have kind of similar needs and demands for their storage engine to be able to optimize for more write performance and also to get more out of their disks or out of their physical resources as the company or the business grows larger and larger. That is in a nutshell our motivation for why we chose to use RocksDB as a storage engine and why it's a good use case or a good fit for the use case that we are trying to fill for our users.

[SPONSOR MESSAGE]

[00:52:00] JM: DigitalOcean is a reliable, easy to use cloud provider. I've used DigitalOcean for years whenever I want to get an application off the ground quickly, and I've always loved the focus on user experience, the great documentation and the simple user interface. More and more people are finding out about DigitalOcean and realizing that DigitalOcean is perfect for their application workloads.

This year, DigitalOcean is making that even easier with new node types. A $15 flexible droplet that can mix and match different configurations of CPU and RAM to get the perfect amount of resources for your application. There are also CPU optimized droplets, perfect for highly active frontend servers or CICD workloads, and running on the cloud can get expensive, which is why DigitalOcean makes it easy to choose the right size instance. The prices on standard instances have gone down too. You can check out all their new deals by going to do.co/sedaily, and as a bonus to our listeners, you will get $100 in credit to use over 60 days. That's a lot of money to experiment with. You can make a hundred dollars go pretty far on DigitalOcean. You can use the credit for hosting, or infrastructure, and that includes load balancers, object storage. DigitalOcean Spaces is a great new product that provides object storage, of course, computation.

Get your free $100 credit at do.co/sedaily, and thanks to DigitalOcean for being a sponsor. The cofounder of DigitalOcean, Moisey Uretsky, was one of the first people I interviewed, and his interview was really inspirational for me. So I've always thought of DigitalOcean as a pretty inspirational company. So thank you, DigitalOcean.

[INTERVIEW CONTINUED]

[00:54:08] JM: If you only had one replica of RocksDB, that would be enough to serve your queries, but of course if that database got blown away in a hurricane or got ruined otherwise, then you would lose all of your data. So you need to replicate your RockDB instances and you need to have these replicas maintain consistency among each other. So you have your TiKV data broken up into regions, and a region is a set of continuous key value pairs, and each of these regions is replicated some number of times. Describe the replication strategy for these regions of data in a TiKV instance.

[00:55:02] KX: Sure. So the regions or you can call it chunks or partition is like you describe how we automatically breakdown the key value pairs into smaller chunks. So then we copy them. The way we replicate them and how we determine the different configurations of copying these pieces of data to ensure that when disasters happen you still have your data is very much dictated by our implantation of the Raft consensus protocol.

Just a quick primer on the consensus protocol itself, Raft is what is known as like a quorum-based consensus protocol, and what that means is that you literally have a group called a Raft group, and this group consists of the same copy of data that live on different machines and they have to vote for a leader. So one of the copy becomes this so-called leader that actually serves their traffic and interacts with the application layer. So in our case, the TiDB server and then the application layer on top of it, while the other copies serve as what is called followers to be there to essentially be ready to become a leader if the leader machine somehow falls down. Then there will be like a reelection process to elect the new leader on a different machine to do the same thing so that your service is always up.

Because it's a [inaudible 00:56:29] based system, the copy of these regions that we make will always be in odd numbers, so either three copies, or five copies, or seven copies, and that's something that you as the user can configure depending on how available essentially that you want to have your data to be. Obviously, more copy will give you more availability, more failure recovery kind of backups. But that does give you a larger data footprint.

So like I mentioned before, everything is kind of a tradeoff. If you want more, nice. Essentially you're going to have to probably have better capacity to store all these data. Some people are okay with it. Some people aren't. The default size for this region currently is 96 megabyte by default, and that is more a trial and error process in which we figured out through all our production usage and testing, that is more or less the optimal size to start with to balance, again, between the copies and the network traffic that needs to make Raft consensus work and also the hotspot that could form with each of the copies.

Hotspot meaning that one particular region or one particular machine is getting all the traffic for some reason, and then other machines aren't doing nearly as much work, these TiKV machines. Then you would suffer performance degradation, because those Hotspots are forming and we do have solutions within the system to actually detect this hotpot as it forms and then automatically redistribute the copies of the data again into different parts of the cluster so that these hotspots gets removed automatically within the system, which is a really nice feature that a lot of users with large scale really, really likes.

[00:58:22] JM: We could go very deep into the replication layer, but we're obviously only have like 10 minutes left or so. I'd like to go through some other elements of TiDB and then hopefully talk a little bit about the business. Let's go through the lifecycle of a query, and this is something we could spend the entire show on, but I issue a query to TiDB. The query goes to the protocol layer, which is managing the connection and the communications with whatever client is making the request. The query gets issued to the SQL layer, and then the SQL layer gets passed through a parser. It gets validated. There's type inference. Because you need to do this translation of a query that is in SQL into a query that can be served by the key value layer, and then eventually you have the key value layer that is doing the serving of the data.

Take me through the lifecycle of a query to the degree to which you think it will be helpful or we have time for listeners, maybe it can be a read or a write or maybe you can contrast the two.

[00:59:30] KX: Sure. I will try to do that in two minutes with an analogy if I can. All the data that's out there, database experts out there, don't shoot me for glossing over details, because like you said, there are a lot of details that goes into this process. But the way I like to explain this is actually a query that you have to navigate in a database system is very similar conceptually to a GPS system. Like if you were to use a Google Map, you want to go somewhere. You type in an address to go. That address essentially is the query that you send out into TiDB. So that is the way you want to get to eventually.

There are a lot of kind of logical layers to, number one, parse through this entry, this address. Number one to make sure is that you're in a proper address, because there are a lot of queries that don't actually make a lot of sense, and the system stops that. It gives you an error. It'd be like, "Yo! This query doesn't actually make a logical sense. You need this. You need that." So that is kind of the logical layer or the logical optimizer that looks at this address.

Once this address is properly formulated, then the GPS system, this Google Map, has to figure out the route and the most efficient route to find this data that you want in this query and extract that from, in our case, the TiKV layer, and that is what so called like physical optimizer, or physical plan is supposed to do, is literally to physically figure out, "Okay, I have this address that I have to get to, or this query I have to process. Where I can find this data in the most efficient way to get it back to the client or to the application layer?"

One particularly unique thing about the TiKV structure as it works with the TiDB server is that because TiKV is a distributed system, a bunch of different machines underneath it. What we have leveraged is the structure of this distributed system to have what is called a co-processor layer, which is essentially is the way to do parallel processing. In the context of a query, that a TiDB layer does as it tries to figure out the best path is to breakdown a relatively complex query into sub-queries and actually send each sub-query into different TiKV machines simultaneously to retrieve that result or like a partial address from those TiKV machines. Then these TiKV machines will give the partial results back to the TiDB server. The TiDB server reassembles it into a more coherent fully result and then sends that back up to the application layer.

So that is in a way the journey of a particular query and what that goes through in the TiDB system, which is hopefully an easy way to understand if you put the framework of just using your Google Map into that context to visualize what actually happens to a query after it gets sent to a database system.

[01:02:42] JM: Okay. That's great. There is a ton of material around TiDB for people who are more curious about this, there are some great diagrams, some great YouTube videos that people can watch to get understanding of the architecture, and I think it's actually – Whatever YouTube video I watched about the architecture was really useful, because just seeing how databases have matured even just over the last couple of years that I've been doing shows about them and seeing how innovations like Spanner evolve, like RocksDB, how they just get pulled into database architecture in every new database that comes along learns from its predecessors, it's useful archeological dig.

Let's talk at a higher level for a bit about the market and how to sell a database. We are in this world where the cloud providers are increasingly dominant in terms of the channels, dominating the channels of selling to developers. Of course, PingCAP, which is the company that has built or built most of TiDB is based in China, and I realized the market for cloud providers there is slightly different. Tell me about the go-to-market strategy for PingCAP.

[01:03:58] KX: Like you mentioned, the company started in China and I think enterprise market in China and the United States or North America broadly speaking is quite different. One of the key differences is the level of adaption of cloud in general, but particularly public cloud. Now there is I think a trend towards moving into a world of so-called multi-cloud, which is that you use two different kind of public clouds or you do a combination of private and public cloud. The way we see ourselves fit into this admittedly a pretty crowded picture is that because our database operates at scale, which is something that I think a lot of company either needs right now or think they will need in the future as they grow is a very good sweet spot for us to be in to be a multi-cloud database solution where you don't have to worry about cloud vendor lock-in, but still have one database that you can operate to, number one, get the performance that you need and also to simplify the operational cost that goes into managing a database, which is no easy task, and that's something that we provide for our customers so they can consume this either as a service similar to an RDS in the AWS world, or you can deploy this in your own private cloud as well. We make that experience consistent. It doesn't matter which platform you're on via Kubernetes, which is kind of the containerization orchestration layer that we leverage to make that experience consistent.

So that's kind of from like a technical angle how we see ourselves fitting in into this entire market. Like you said, there's a lot of – Different cloud vendors have their different database solutions, but because we are very much open source and we're big open source believers and we also try to be as transparent about our technology and also the tradeoffs as possible so that people actually do understand our technology and give their fair shake before they use it as supposed to giving you a black box that just presents everything that you want to hear as supposed to something that you actually need.

That is in a way how we think about strategically given what we want to do in North America, and it's still very early days for PingCAP in North America, but we have gotten a lot of good interest and traction and I think there is to be determined how well we can do in this market, but we feel very strongly from the architecture we talked about previously that this hybrid transactional analytical processing framework is something that more and more company will take up, and that is the space or that is the competitive advantage that we will occupy, because real-time reporting of your transaction data will be more and more of a need for fintech companies, for financial services in general, certainly ecommerce, ridesharing, all these large categories of innovating companies are growing and to be very mature and they would need a database that is architected the way we have to solve these problems right now.

[01:07:22] JM: So I go to these conferences, the Kubernetes Conferences, KubeCons, and I'm seeing a lot of interesting trends around how enterprise adaption of technology is changing. So you've got increased adaption of Kubernetes, which is causing a lot of opportunity for companies that are selling Kubernetes into the enterprise space, because all of these banks and manufacturers are looking for somebody to help them with their Kubernetes. But alongside the migration to Kubernetes, there's an adaption of cloud, which is sort of in some ways the proprietary alternative to Kubernetes, or perhaps the proprietary adjunct depending on how you look at it. So you have companies adapting cloud. They're adapting Kubernetes. They're adapting products like TiDB. You just see this gigantic influx of enterprise dollars as enterprises become more and more willing to pay for this stuff.

How are the buying patterns of large enterprises changing and what opportunities is that creating for you at PingCAP?

[01:08:32] KX: Right. I think the KubeCon experience, I was there. I think you were there as well. It's a very kind of visceral indication that, number one, IT is really sexy now, which is not always the case, and that large companies are willing to spend, honestly, like hundreds of millions of dollars at this point to transform their IT infrastructure perhaps with something like a Kubernetes in mind to really modernize themselves. I think the reason why that's the case is because – And I personally believe that more and more people will see IT revolution or IT advancement in a company become not just a cost saver, but also a money maker.

Because with new technologies like Kubernetes and with new databases that sit really well with Kubernetes, like TiDB on top of your Kubernetes infrastructure, that you can think of new businesses that you can actually make available to your users, new experiences, new products, all sorts of different things in line with your business that you couldn't really do before because your infrastructure was so old and it did not allow you to do real-time analysis, for example, or even real-time machine learning on transactional data that something like a TiDB could perhaps help you do.

I think that's what got people thinking that this transformation as massive or as laborious as it may be is worth the investment, because it doesn't just save you money or is a cost reducer, but could actually help you make more money in your business. That is a very different way or a different psychology that we as vendors are seeing that we are trying to help as well, because we're no longer just the cost savers. We're not just playing on price per se, but we're playing with new possibilities, new revenue streams for our customers because of the way we developed our technology, and I think that's a very exciting direction for the whole IT industry to move into.

[01:10:45] JM: Kevin Xu, thank you for coming on Software Engineering Daily. It's been great talking to you.

[01:10:49] KX: Thank you so much for having me.

[END OF INTERVIEW]

[01:10:53] JM: HPE OneView is a foundation for building a software-defined data center. HPE OneView integrates compute, storage and networking resources across your data center and leverages a unified API to enable IT to manage infrastructure as code. Deploy infrastructure faster. Simplify lifecycle maintenance for your servers. Give IT the ability to deliver infrastructure to developers as a service, like the public cloud.

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

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

[END]