Shengwei Wang:

Good afternoon everyone. Thank you for staying with us. I'm so excited to meet a lot of old friends and new friends because we've been doing that two years ago in New York. Now, happy to be back on stage again. So my name is Shengwei, I'm a software engineer at Netflix.

Ram Srivatsa Kannan:

And I'm Ram. I'm also a software engineer at Netflix.

Shengwei Wang:

Today we are going to talk about the multi-region use case at Netflix. But before that, let me do a kind of quick intro. So who are we? At Netflix, we have a centralized data platform team to handle all the infrastructure engineering things. Specifically, our team is called online datastores team. So we provide multiple data store as a service like Cassandra as service, Elasticsearch, EVCache, that is a Netflix own thing. Also RDBMS. CockroachDB is definitely one of the offering. It's PayPal offering. When we say PayPal, customer... At Netflix, you have a freedom to choose whatever data store you want to use, but if you go with something off the road, you are on your own. The way we did is try to make CockroachDB easier to use for the customer so they can go ahead and do your own things without worry about too much about data storage layer.

So the model actually operate is that we have a self-service UI. Customer, if they want to use CockroachDB, just click some button from the UI, then boom, you have the cluster and you're free to use all the SQL features you want to use. For the platform, we own the operation but we are not DBAs. We rely heavily on automation. We create our own control plan to take care of the major version upgrade, scaling up, scaling down, instance crash or even lost [inaudible 00:01:51] issue. I'm sorry, I don't want to mention about lost [inaudible 00:01:53]. This is a nightmare, but you see that's our daily work. So talking about our use cases, so CODB has been a productionalized PayPal solution since 2020. And then now with couple years involving, we've been realized there are a couple use cases can really get benefited from CODB. One is multi-region capability customer. They really want to have an active multi-region.

And also regional survival goal is definitely something selling point for them. For other customer, they want to use a high availability data store. So Brian talked about the maintenance window is definitely paying in their other traditional SQL database. So it is not the case in CODB because I can always do the rolling upgrade. So when I do the major version upgrade, I just send an email to all the customer, "Hey, I'm going to do that." But almost no one is complaining about the hiccups because we just take care of everything. There are still a handful of customer, they have a huge scale, that normal database they cannot support. The CODB is known for the distributed transaction. You can scale arbitrarily, theoretically of course, and I know many customers they're asking, "Hey, is Netflix storing all this movie data in CODB?" The answer is no. But the CODB is well known to be used by the content engineering, the studio and gaming. So Ram will take care of that.

Ram Srivatsa Kannan:

Thank you, Shengwei, for this great introduction. Although Netflix didn't store movie data, gaming is on the way for Netflix. We are starting to store the gaming data on Netflix. So it's kind of cool in that context. So before I go into the crux of this talk, which is about multi-region, I will give you an overview of our fleet summary. And we've been giving these talks in RoachFest for the past two years and our fleet has grown quite a bit over these time. We have around 380 plus clusters for which Shengwei and I manage on a day-to-day basis. And out of which 160 of them are production cluster. And going to the

topic multi-region, we have around 60 odd multi-region clusters and Netflix operates only primarily on four regions. US-east 1, US-east 2, US-west 1 and US 2. So we give the opportunity for clients to choose how many regions they want actually deploy into. It could be from two to four and that's completely based on how tolerant they are to region failovers.

So now one of the biggest, again going back to a gaming example, one of our biggest CRDB cluster, which is four region cluster is crdb_ngpen. And the gaming team actually owns and operates this cluster and they store metadata about their entire control plane as a part of this cluster. And this cluster has around close to 48 nodes and spread across four regions. And we have three zones per region. So a lot of nodes. So given that the fundamental question that we want to answer as a database platform team is, why do we need multi-region? We kind of try to answer this question bottom up by throwing this question to our customers and ask them, "Hey, why do you guys need multi region?" And the perspective that a database engineer or database automation platform infrastructure engineering teams would have could be different from how a consumer might think about it.

And most consumers thought about this. We want load balancing during traffic evacuation. More nodes can handle more nodes and that's the outside view of what a multi-region database cluster is for them. And so assume there is a scenario where there's a client here for this example, in US-west 1, that's down because they're rolling out a new feature and that has a bug. So what client app teams to do is they'll actually evacuate the traffic from US-west 1 here to US-central 1. And what they think is that there is an additional region in US-central 1 they can handle more load. And that's one of their fundamental reasons from them to have a multi-region cluster. But while that could be true in several NoSQL databases, right? And for CockroachDB [inaudible 00:06:41] because for every write and read, you either go to the leaseholder master for reads or for every write, you actually have a globally consistent view.

So you send your rights to every node throughout every region. So in that case, whether you have one or two regions, the right traffic is going to be pretty similar. In fact, if you have more regions you can end up having cross region traffic which can harm your performance of the database. So this is very counterintuitive to our customers and this is something that we have been trying to advocate and educate as a part of our journey hosting CockroachDB within Netflix. So now, why do we want multi-region databases? And the answer is pretty straightforward. We want database site, region survivability, right? So if you have more than one region and if a region goes down, in most cases the database failover takes down the app along with itself. So all the entire region is down and the app side teams will actually take the app traffic from that region and transfer it to the other region.

But what they could see is from the database side, absolutely no loss in terms of performance or in terms of availability. They are fully available as if the region failover is completely agnostic to them. And that's the major selling point for us within Netflix to actually offer for customers to actually go to CockroachDB. So now, you use this for a better availability, but what about performance, right? And it comes with its own performance impact because we work on an active active setup. So on a default configuration system, there might be some performance hiccups that customers may want to endure as a part of multi-region setup, right? And we, as a infrastructure team, try to work with customers to actually educate them through this process. And there are several things that we do to actually help them with that. So for instance, one key example is that if you have a multi-region setup and what if your leaseholder masters is in another region, right?

So what you end up doing is a cross region call to actually get the latest copy of the data, which is a performance penalty that we'll pay. And one quick win that they can get is by knowing that hey, if my lease holder for this example is in US-central 1, they can pin the leaseholder to a particular region and make sure app traffic that is confined to that region goes through database connections within US-central 1. By that way, they can get the performance while having multi-region too. So that's one thing

that we actually try to advocate our customers to do. And there are some customers having some tables, are part of their databases that are tolerant to weaker levels of consistency. Those situations, what we advocate is to use follow reads where although the database client knows that the data, the updated or the most latest copy of the data is in a different region, it takes a slight hit on the [inaudible 00:10:24] of the data and returns the local copy in order to actually serve reads faster.

And that's a very common sector that we actually advocate within Netflix. So one of the things that we are thinking into which we currently do not use is global tables. Several customers, they cannot choose relational databases because of its strong nature consistency, especially with 24.1. They've introduced [inaudible 00:10:53] committed, so several customers want different levels of consistency and strongly consistent databases, the one sole reason why they actually go to CockroachDB. So in those situations you can take a slow right by design, by optimistic for a local reach knowing that your client rights are going to be very low. And if at all there is a right, it actually goes to every single replica within an entire data center and tries to get an act from each of them. And in that situation, you kind of take a slow ride by design but still get the most latest copy.

And the one of last feature that we've been advocating more recently is to actually use a geo partitioned database and similar to [inaudible 00:11:45] is that you can actually shard your tables based on or create tables based on region affinity and keep client traffic, that requires regional tables within that region so that you have both efficient rates, because rates also goes through the same client traffic. And you get very fast reads, and you still can have a multi-region setup, although a slight hit that you may take is on your set availability goals. So these are some of the different multi-region setup that we advocate within our control plane and I'll hand over the rest of the talk to Shengwei, and he's going to talk about the topology and the future work that we have planned within our control plane.

Shengwei Wang:

Thank you, Ram. Hopefully, Ram, your talk didn't scare everyone because you can see there's so many powerful tuning we can do in CODB to make customer getting better user experience. On other hand, think about that. Without this capability... It's easy for you to tune a single region database, but the problem is all the cross region costs need to be paid. There's no way for you to optimize that. But with CODB compared to the single region things, you have so many ways to improve your production traffic. Before I actually talk about topology, let's talk about the real life scenario, the customer interaction. I do believe this might be a kind of similar apply to everyone who are running platform team.

So in my dream, in my ideal world, customer come to me saying, "This is my use case and this is my traffic pattern across other regions and I want to survive from a failure or DB failure. I'm very clear, I understand what I'm saying and here is my latency expectation and I do understand there's a trade-off, the cap theory. I know the consistency, availability cannot be survive at the same time. I have to pay for that to get this." So this is actually in my dream. Don't get me wrong, at Netflix, I'm surrounded by my stunning colleagues, but they have a deep expertise in their own domain. So they're the gaming expertise. They handle streaming very well. They know how to produce a great content, but they don't have to be a CODB expert. In fact, I think it's my fault if they have to be the expert before using CODB, right? So then in the reality, as you can see, I'm getting the question, say, "Shengwei, I want to have a multi-region RDBMS." And customer, as you know on a site, as you know, Netflix has been using Cassandra for decades.

So people, they're familiar with how Cassandra is set up and the number of replicas these things, and people come to me saying, "Hey, I've been using CODB, this is not how Cassandra looks like." Or Ram talk about the app failure versus DB failure. Trust me, many of the customers, they've been using multi-region setup just for their app failure until we told them single region can't handle your scenario.

So this is like daily things happening, but how do we handle. I don't have the luxury to talk to all the 200 customer across all Netflix. The better way, the rule of thumb is always go with the default setting and do optimization later. And what is our default setting? Before doing that, let's talk about abstractions. I know CODB in later version, we introduce a new way of doing abstraction in super region or regional viral or the surviving goal AZ or region... Sorry, I don't know many of terminology because I'm not using them.

So Netflix, we adapt CODB since 2020 and back in that the best tool for me is zone configurations. So this is, for me, it's more human-readable because I can easily communicate to our customers saying, "Hey, this is how many data copies there are in this region, how many data copies in that region?" It's also machine-readable. We share the control plan with many of data store reach like Cassandra in EVCache, Elasticsearch, such as you mentioned. We have, this control plan is called Antigravity. Weird name, but this is super powerful and useful. So have a single control plan to control all this data store. It's easier for me to do. So zone configuration is definitely one of [inaudible 00:16:16]. So the downside is, it is a little bit harder for customer to do the tuning, but you know what, it's actually a good things because customer tuning, you don't know what they can do.

So typically the nightmare started with customer, "Hey, I know what I'm doing," and turn out to be not the case. This is the exactly example about how we create a multi-region. So I specify the number of replica as nine and for each region we will ping number of replica 3, 3, 3. So losing one region would not cause problem there. And there's a specific lease preference. You can ping the lease holder. I'll talk about that later. So default setup. When you initiate a new multi-region cluster, so if you remember that I talk about the self-service, so in the self-service UI, customer, they can specify, "Hey, this is my traffic RPS and this is a number of region I want to deploy it." And we take this information and provision the cluster.

By the way that this page from our Spinnaker. Spinnaker is our continuous delivery platform, is open source. Netflix has been widely adopted that. So we decide for each region, we'll have three AZ. We will keep one copy in each AZ. So let's say, you have three regions and then in total you'll have nine replica because you have a three region, multiple by three AZ and one replica per AZ. So yeah, as I said, it's a shared control plan with other data stores. So it's not a surprise for me to learn this new thing because Cassandra has been used that for a while.

And it is also easier understand for customers, they don't even need to look at the dashboard or hacking to the node and looking at this Spinnaker page. They know what they to do. We can argue. Of course, many people, they may think, "Hey, why do we need nine replica?" So five probably would be good enough in reality. Yes, I do believe. This is some further optimization we want to do, but for now, having more replica or having more instance is kind of minimal cost implication, because it's easier to... I mean it's cheaper to pay for this hardware than pay the overhead of learning a new thing, so in my own opinion.

So how about expanding to new region? So Netflix, we recently expand three region to US-east 2, then now it's four region. So when we add a new region, the thing is becoming a little bit complicated because if we are following the old pattern, each region keeps three replica. Then when you introduce a new region, you will introduce higher latency or more data. So the strategy we'll be taking is we are only adding the gateway node. By only adding the gateway node, you are not changing the topology of your cluster. So you'll still see the same latency, read-write latency from existing region. But for new region, of course, you have to pay the cross region latency for both read and write. But it's okay because a new region and you can see as the traffic growing, definitely you were hitting some point. I have to do further tuning, creating different partition for a specific region and change the data locality to the organization. But that can happen always later. So for me, it's easy to only just introduce a gateway node and wait until your customer complain to me.

Okay, this is an interesting use case or example. I have a customer, they want to deploy their application in four regions and the data is geo-shardable. They want to have both low latency and regional survival. As you can see, it's not a hundred percent achievable. So I told them, and they're very smart, "Okay, I accept the fact, but can you do a little bit better?" The answer is yes. Then you have to take pay for at least one. But at most one cross switching latency. So this is our setup. Instead of replicating all the data to four region, we firstly shard the data by region. This US-east 2 is one shard. We distribute the data to US-east 1, US-east 2, and US-west 2.

For each region we'll have three replica. So in that case, when you perform a write, then you will be able to achieve the quorum by only talking to one closed region and when you lost a region, that you are able to survive from region failure because the data has been replicated across all the regions. So this is the things that you actually do, hacky things. For the optimization, as I mentioned, we don't have to have nine replica, we can have five. But honestly, again, it's easier for me to communicate to customer, "Hey, for each data you have a nine copy and each region. If you have data in this region, you have three replica at least." But we can always improve if the cost implication becomes significant for me to do further tuning. Yeah.

All right. I want to talk about a little bit about future work. So obstruction level customer coaching, what that action means. So if I open up the CODB, the webpage, the documentation, I think fabulous work. I mean it's very clear for me to understand everything, but on another side, customer, even with that beautiful doc, customer, they tend to not read any doc. They just try to play something with their own and how can we make that easier for a customer? The abstract level is not from the CODB part, but from the Netflix side. How can we abstract the Netflix use case? Like say regional, the traffic evacuation, maybe a specific has different meaning in Netflix, right? Surviving goal has different meaning in Netflix. How can I make this abstract easier? So before the customer doing the self provisioning or during the process of they doing provisioning, they can pick up specific pattern they're going to use and then it's like early optimization. I don't have to do the tuning later because I know this is the Netflix specific traffic pattern.

The second thing is, as we are using the crlabs abstraction, I mean crlabs abstraction has been more mature like year by year, version by version. We are not using that, we're still using zone configuration, but how can we build a bridge? If there's a way for us to building this two bridge, on the line, we can still use known configuration, but in general, we can rely more on this abstraction. And so customer, they can do their own tuning. This is something we want to think about. How can we make it easier for customer and easier for me? The control plan improvement for organization, as I said, because historically, we've been sharing all this control plan with all the data stores. So the three replica make it easier, but how can we make it also easier when we want to reduce that cost and having a little bit customized setting than other data store. Yeah, that's all the thing I want to talk about. I'll stay here for a while. I mean, later for any question. Happy to do the personal interaction. Yeah. Thank you for listening. Yeah.