EPISODE 467

[INTRODUCTION]

[0:00:00.4] JM: Facebook serves interactive content to billions of users. Google serves query request on the world's biggest search engine. Uber handles a significant percentage of the transportation within the United States. These services are handling radically different types of traffic, but many of the techniques that they use to balance the load are similar.

Vivek Panyam is an engineer with Uber and he previously worked at Google and Facebook. In a popular blog post about load-balancing at scale, he described how a large-company scales up a popular service. The methods for scaling up load-balancing are simple, but effective, and they help to illustrate how load-balancing works at different layers of the networking stack.

Let's say you have a simple service where a user makes a request and your service sends them a response with a cat picture. Your services starts to get popular and begins tiring out and failing to send a response to users. When your service starts to get overwhelmed, you can scale up load by creating another service instance that is a copy of your cat picture service. Now you have two service instances and you can use a Layer 7 load balancer to route traffic evenly between those two service instances. You can keep adding service instances as the load scales and have that load distributed among those new instances with just that one Layer 7 load balancer, but eventually your Layer 7 load balancer is handling so much traffic itself that you can't put any more service instances in front of it, so you have to set up another L7 load balancer and put an L4 load balancer, a Layer 4 load balancer in front of those L7 load balancers.

Now you can imagine a tree structure, we have a L4 load balancer that's balancing load among L7 load balancers and you have L7 load balancers that are each distributed load among service instances and this also illustrates why it's important for a service to be stateless, because you want all those services to be, essentially, it's fine that they're all just replicas of one another. Ideally, it shouldn't matter which of those service instances handles any given request. If they're not stateful, they could all handle any of the requests and that would simplify the routing.

The next step after this is that your L4 load balancer starts to be handling too much traffic, and when your L4 load balancer gets overwhelmed with requests for your cat pictures, you have to set up another tier, and this time it's L3 load-balancing. The L3 load balancer is disturbing load amongst L4 load balancers. Those L4 load balance are disturbing load among L7 load balancers, which are disturbing load among service instances.

That's kind of how load-balancing works, and in this episode, Vivek gives a clear description for that system, and we also review the seven networking layers before we discuss why there are different load balancer types associated with those different networking layers. This was an educational episode for me, because I did not know how load-balancing works. So I hope it is the same for you.

[SPONSOR MESSAGE]

[0:03:27.1] JM: Today's episode is sponsored by Datadog, a platform for monitoring your infrastructure and application performance. Datadog provides seamless integrations with more than 200 technologies, including AWS, nginx and Docker, so that you can start collecting and visualizing performance metrics quickly.

Access out-of-the-box dashboards for HA proxy, Amazon ELB, ALP and more, and correlate metrics from your load balancers with application performance data to get full visibility into your web apps. Start monitoring and optimizing performance today with the free trial. Listeners of this podcast will get a super soft Datadog t-shirt too.

Visit softwareengineeringdaily.com/datadog to get started. That's softwareengineeringdaily.com/datadog. Get your free super soft Datadog T-shirt.

[INTERVIEW]

[0:04:30.6] JM: Vivek Panyam is an engineer at Uber. Vivek, welcome to Software Engineering Daily.

[0:04:35.2] VP: Thanks for having me.

[0:04:36.3] JM: I'm inviting you on because you wrote this really good article about load-balancing. How load-balancing would work at a company like Facebook or Google or Uber, these hyper-scale companies, and you have worked at Facebook and Google and you now work at Uber. Did you see some commonalities in the infrastructure across these different companies that you can also generalize to other big companies, like Amazon or Netflix?

[0:05:04.0] VP: Sure. The overall goal of load-balancing is pretty much the same everywhere, so increase reliability and capacity while minimizing costs and additional latency. Basically, being able to scale up without paying a lot more and slowing things down.

[0:05:21.4] JM: What about more generally speaking? What were the commonalities in the infrastructure deployments at those giant companies you've worked at?

[0:05:28.5] VP: A lot of them are using some form of container or containers, like Docker, and then nginx is a tool that's pretty widely used for a Layer 7 load-balancing. Yeah, I think overall a lot of those companies' architectures came out of things breaking and profiling what went wrong and fixing it or just profiling where they could gain the most performance.

[0:06:01.2] JM: I guess at a certain level these are just companies that have tremendous amount of load, and when we're talking about just load, this is undifferentiated bits coming in. Whether you're Facebook trying to calculate connections between different people and create a newsfeed that ranks billions of pieces of content, or you're Uber and you're trying to schedule all these rides for different people and keep the geospatial world up-to-date. All of these things ultimately boil down to load, so it makes sense that these different companies asymptote towards the same load-balancing architecture.

[0:06:42.6] VP: Also, a lot of the engineering teams at these places are fairly well-connected, so good ideas tend to travel around.

[0:06:53.8] JM: When we're talking about modern load-balancing, are we talking about hardware devices that are specifically built for load-balancing or are we talking about software?

[0:07:03.9] VP: Both actually. You can buy hardware load balancers that are usually pretty expensive, but they can have extremely high throughput and low latency, because their device is built specifically to do that job. The benefits of software is that, generally, it's at a much lower cost and you got a lot of flexibility, and lots of places tend to use a combination of both, but you can build pretty much the same architectures using pure software load-balancing.

[0:07:37.6] JM: When we balance the load, we're often balancing it across multiple instances of the same service. If I'm building a load balancer that works across the services in my huge company, I want to be able to have my load-balancing infrastructure work, whether we're talking about my e-commerce service or my database as a service service, because if you're a company like Google, you've got all these different kinds of services that you're running that different clients are accessing. It would be great to have a standardized way that load-balancing works across these heterogeneous setups set of services. What kinds of standardization do I need to impose among the different services and the service owners of those services?

[0:08:36.5] VP: Generally, you want to be able start as many instances of that service as you want, and there are a couple ways of doing that. One is you can enforce a rule, like any request can go to any instance. That's generally not super advisable in production and you can avoid that by doing things like sticking load-balancing , but that basically means that your load balancers can kind of treat all of these services as the same.

If you use sticky load-balancing, then your load balancers will basically send the same type, or a specific type of requests to a specific instance of the service. For example, if you use some sticky load-balancing's scheme based on user ID, you can guarantee that all requests with that user ID will go so one specific instance.

In terms of standardizing across services, generally, if they're all using the same protocol, you could theoretically use the same instance of your load-balancing software to balance across them, but you can also be fairly flexible with how the services operate as long as they kind of fit the two rules I mentioned earlier.

[0:10:07.9] JM: In your article, you talk through several different tactics for doing load-balancing. Why are there a variety of load-balancing techniques? Why isn't there just one way to do load-balancing?

[0:10:20.4] VP: Sure. All of your requests generally go through, at some level, a bunch of different layers of the networking stack. If you just load balance at one layer, you kind of have to have one load balancer go after the others, I guess. In some sense, they're all happening at the same level. If you're load-balancing across different layers of the networking stack, those give you different pros and cons and they allow you to scale in different ways.

[0:10:59.4] JM: Yeah. I think we'll explain it in more detail as we go through your set of examples. Let's say that I'm building a simple e-commerce store and I just want to sell Software Engineering Daily t-shirts, and at first I've got very few users. I've just got a t-shirt store that's just selling a few t-shirts every day, but then my t-shirt store starts to get a little more popular and it gets so popular that I need to scale it with another instance of the service. So I'm just going to copy the service and I'm going to stand it up as another instance.

Now, all the requests that come in need to be balanced between the first service instance and the second service instance, so I use a Layer 7 load balancer. What is a Layer 7 load balancer going to do in this simple instance of having just two services?

[0:11:54.1] VP: Sure. Your load balancer will take the request from your users and forward those on to — Or split those requests basically. Depending on the load-balancing scheme you use, it'll send some requests to one server or one service and the rest of the request to your other service.

[0:12:18.4] JM: What's specifically happening at Layer 7? For people who are listening and they don't really know these networking layers, what is Layer 7 actually mean?

[0:12:28.9] VP: Sure. Do you want me to just s go through all the layers super quickly?

[0:12:31.8] JM: Sure. Yeah. Explain it real quick.

[0:12:33.3] VP: Okay. Sure. Layer 1 is the physical layer. That's generally the hardware that's powering communication between devices, like switches, network interface controllers, the actual wires that the bits are traveling on. Layer 2 is the link layer which is the layer that kind of represents data transfer between nodes. That would be the internet protocol, or 802.11. Layer 3 is the network layer, which as it sounds, is responsible for routing packets in the network. That is like IPV6 or IPV4. Layer 4 is the transport layer, which some examples of are TCP and UDP. Then Layer 5 is the session layer. I'm not going to go into depth on this, because generally you don't really use Layers 5 and 6 a lot in load-balancing, but the session layer's job is to manage sessions, so like requests and responses between nodes. Layer 6 is the presentation layer, which generally handles your utilization, so converting from application level data to a representation that can be sent over the wire. Finally, Layer 7 is the application layer, and these are things like HTTP or web sockets, and that's where all of your application-specific information goes.

[0:14:10.7] JM: If we're just implementing a Layer 7 load balancer, what's the software that we're using to implement that?

[0:14:19.1] VP: Yeah. You can use things like nginx or HAProxy. Those are two very widely used tools for load-balancing in the industry.

[0:14:29.9] JM: How much can I scale that model with just a single L7 load balancer? How many service instances could I stand up and have traffic routed between?

[0:14:42.4] VP: It really depends on the specific service they you're working with. The sizes of the requests, the sizes of the responses, things like that pretty heavily influence your ability to scale with one load balancer. The answer to that is basically just profile it and see how many you can stick behind it before you start losing performance.

[SPONSOR MESSAGE]

[0:15:14.3] JM: The octopus, a sea creature known for its intelligence and flexibility. Octopus Deploy, a friendly deployment automation tool for deploying applications like .NET apps, Java apps and more. Ask any developer and they'll tell you that it's never fun pushing code at 5 p.m.

on a Friday and then crossing your fingers hoping for the best. We've all been there. We've all done that, and that's where Octopus Deploy comes into the picture.

Octopus Deploy is a friendly deployment automation tool taking over where your build or CI server ends. Use Octopus to promote releases on-prem or to the cloud. Octopus integrates with your existing build pipeline, TFS and VSTS, Bamboo, Team City and Jenkins. It integrates with AWS, Azure and on-prem environments. You can reliably and repeatedly deploy your .NET and Java apps and more. If you can package it, Octopus can deploy it.

It's quick and easy to install and you can just go to octopus.com to trial Octopus free for 45 days. That's octopus.com, O-C-T-O-P-U-S.com.

[INTERVIEW CONTINUED]

[0:16:45.4] JM: My site gets really popular, then I'm going to want to add so many instances of service, or maybe I'm breaking up into multiple services. I'm going to need another L7 load balancer, and then I'm going to need to do load-balancing between my two L7 load balancers. What are my options if I want to load balance between multiple load balancers?

[0:17:11.6] VP: You could use Layer 4 load-balancing, which is the transport as I mentioned before. HAProxy is one popular layer for a load balancer, and that would basically route TCP, or it would route at the network layer to both your load balancer or your Level 7 load balancers and then those would continue at the application layer to your services.

[0:17:39.7] JM: Why am I doing that? Why can't I just do another L7 load, like a tier of load balancers in front of my tier of L7 load balancers that are just interfacing with my services?

[0:17:53.7] VP: You could, and that would work, but I think generally, as I said before, you want to profile everything. The right answer usually depends on what your specific environment looks like. In the article, I used a Layer 4 load balancer in front of the two Layer 7s just as an example so we could keep using more and more, I don't want to say complex, but different types load balancers.

[0:18:22.8] JM: Okay. This was more for the sake of example or maybe — Correct me if I'm wrong. Would there be circumstances where it would actually make sense to just have tiers and tiers of L7 load balancers?

[0:18:34.9] JM: I don't think so. I'm sure people have tried it, but at some point, each of these load balancers is just responsible for splitting your requests by some way, or in some way. Like a Layer 7 load balance could do that. It will probably have different performance characteristics than a Level 4. Yeah, you could try either.

[0:19:04.9] JM: Okay. This Layer 4 load balancer, you said you're often using HAProxy. You could use, with the L7 load balancer, nginx or HAProxy, and at L4 you typically want to use HAProxy specifically, or that's just common?

[0:19:24.8] VP: Yeah. That's a common tool I've seen around, and it's very well battle tested. IPVS is another example. I haven't used it, but it's another tool that I've heard being used for this, for a Layer 4 load-balancing.

[0:19:42.4] JM: Okay. So now my requests are coming in. They're all hitting my L4 load balancer. This is — You said it's the transport layer.

[0:19:51.7] VP: L4 is a transport layer. Yeah.

[0:19:52.7] JM: Transport layer. Okay. If I've got all these requests coming in hitting my L4 load balancer, it's getting distributed to the L7 load-balancing tier. It's like a tree. It's breaking up into these — Being distributed across these Layer 7 load balancers, and then those Layer 7 load balancers in front of each of those, you have a bunch of service instances. Does the Layer 4 load balancer become a single point of failure in this case?

[0:20:22.1] VP: It does, and we can fix that with some of the techniques we're going to talk about in a little bit.

[0:20:29.0] JM: Okay. Just to touch a little bit more on those layers. What does that actually mean? When we're talking about different the layers of the networking stack, are these things

that at any — Just when a bit is traveling through the internet, you can look at that bit as existing on all seven layers or do you — How do you think about those layers? Are they all at the same time or do you think about them sequentially?

[0:20:58.4] VP: Sure. Generally, if you're looking a single bit, that's going to be the physical layer, and the second layer, which is the link layer, kind of looks at sequences of bits put together into frames. Then as you keep going up, the level of structure you look at the data with — Or the way you look at the data becomes more structured. Does that make sense?

[0:21:27.4] JM: Yeah. These are just abstractions. They're all things you can examine at the same time, but they're just varying levels of abstraction.

[0:21:39.0] VP: Mm-hmm.

[0:21:39.7] JM: Okay. Now, my t-shirt site is selling like crazy. I am going to need even better load-balancing. Could I just use L4 load-balancing forever? Could I just have a bunch of L4 load balancers now and then put kind of do the same thing we discussed with the L7 load balancers?

[0:21:58.7] VP: Yeah, you could, but if you're at crazy scale, like most people probably don't need to look at putting Layer 3 load balancers in front of your Layer 4 load balancers, but we'll look at it for the sake of example. You're right. You could just keep going with Layer 4 load balancers.

[0:22:21.8] JM: Okay. What am I going to do with a Layer 3 load balancer?

[0:22:25.6] VP: Sure. This is a little bit more complicated than the first two techniques. Again, we're talking about a Layer 3 load balancer, right?

[0:22:35.8] JM: Yes.

[0:22:36.7] VP: Okay. That's the network, again, and that includes IPV4, IPV6, things like that. We could put a Layer 3 load balancer in front of two Layer four load balancers, which are each

in front of another two Layer 7 load balancers, which are in front of however many services that you're load-balancing across.

The Layer 3 load balancers will use ECMP, which is equal cost multipath routing. Generally, ECMP is used why there are multiple equal cost paths to the same destination. We can kind of exploit this to do Layer 3 load-balancing, because from our perspective, every Layer Four load balancer is the same, because they're load-balancing across services that theoretically are identical.

We kind of just tell the Layer 3 load balancer the all of these paths lead you to the same thing, and then the network layer kind of handles of that — Sorry. The Layer 3 load balancer handles routing data appropriately to your Layer 4 load balancers.

[0:23:57.7] JM: Okay. Let's drill into that term ECMP, the equal costs multipath routing a little bit more. Just explain that in more detail.

[0:24:07.5] VP: Sure. Broadly, it lets a router switch send packets to the same destination over different links. If you have multiple paths to the same destination, you can send them over different links for higher throughput. In this case, we're kind shrieking it, or like lying a little bit by saying it's the same destination, when in reality it's actually two separate load balances, but we can treat them as if they're the same destination, because as we said earlier, all of these are the same service.

[0:24:43.6] JM: Do you need to give all those different — If you're routing from L3, an L3 load balancer to a bunch of L4 load balancers, do you have to give me L4 load balancers all the same IP address?

[0:24:55.8] VP: Correct. Yeah. You do that and then use ECMP to spit the traffic across all of those load balancers.

[0:25:04.6] JM: The L3, that layer does what again? What's that layer?

[0:25:09.2] VP: L3 is the network layer.

[0:25:11.3] JM: The network layer. Okay.

[0:25:12.8] VP: Correct. Generally, this is done in hardware, and unless you're at a huge scale or running on your own hardware, you don't really need to do this.

[0:25:23.0] JM: If it's at the network layer, how does it demarcate — Maybe this is a stupid question. How does it demarcate between the different requests, because it needs to — Presumably, it's got these requests coming in and you've got to break up the bits in a way that partitions the different requests, right?

[0:25:41.5] VP: Right. Generally, with each of these types of load balancers, you split them or you split requests different ways. For instance, in your Layer 7 load balancers, they can look at the particulars of a HTTP request, for example, to decide where to send it or which instance of your service to send it to.

The Layer 4 load balancers can't really see the specifics of the requests. For instance, they can't look at the request URL easily. You use different things like — It's sort of horrible phrasing, but use different things to decide what the next step in your load-balancing chain is at each layer.

[0:26:29.2] JM: There's some kind of metadata that's decodable at each layer.

[0:26:35.7] VP: Correct, and usually that's different for different layer load balancers.

[0:26:41.7] JM: Interesting. I'm not judging you, if you don't, because I certainly don't know anything more about that, but this is just something I have never looked into. I never took a networking class.

[0:26:51.3] VP: It's also one of the reasons Layer 4 load balancers are sometimes more efficient than Layer 7 at least based on some of the things I've read. I haven't gone super into depth on the specifics of performance of each of these. Yeah, I mean they all have their pros

and cons and they can all shard your requests by different pieces of information included in the request.

[0:27:21.8] JM: At the L3 load-balancing tier, I want to do this equal cost multipath routing. I want to be able to route this request, or different requests, along different paths and I want to tell these requests, “Hey, you're all going to the same IP. You're just taking different routes,” but I’m actually sending them to different load balancers that I have given the same IP. What kinds of tools are used to do this multipath routing? Can you talk more about how that's implemented?

[0:27:59.1] VP: I’ve usually only seen this done in hardware with. I haven't seen it done using software configuration. I’m sure it's possible, but I just haven’t seen it.

[0:28:13.8] JM: Yeah. At what scale does the L3 load-balancing start to break?

[0:28:17.8] VP: Again, same instance, before profile it. Usually, if you build a system that has L3 load balancers splitting out to L4 load balancers, splitting out to L7s, which split out to multiple instances of a service, you’re usually not going to have. If you do it right, you're not going to have lots of performance issues. There still is a single point of failure here though, which is your L3 load balancer.

[0:28:46.1] JM: I see. When you’re talking about profiling the load, how does that look in practice?

[0:28:52.7] VP: Usually you’d run load tests. One way of doing that is you record a sequence of requests that appear to be your general load or you come up with specific request patterns that you expect and then you have a bunch of machines send requests like that as fast as they can to your system and see how it behaves, and then you can watch CPU usage, memory uses, see if anything crushers, measure a latency for the responses and see what the average latency is or like the 99 percentile, things like that. Then you can kind of dive deep into each level of your request routing to figure out where your bottlenecks are.

[0:29:47.6] JM: How often are companies doing that?

[0:29:49.8] VP: Usually pretty often as they scale. In terms of load testing, generally, a lot to make sure that services perform how they're supposed under load. For instance, like Uber for New Year's Eve, like we generally get a lot of traffic. There's lots of load testing that happens to make sure we can handle it, and things like that.

[0:30:13.9] JM: The next tier of load-balancing that you explore is DNS load-balancing, and DNS load-balancing works a little bit different than the L3, the L4 and the L7 load-balancing schemes. In those schemes, what we've talked about basically is L3 routes to a bunch of different L4 load balancers. The L4 load balancers route to a bunch of different L7 load balancers, and the L7 load balancers each route to a bunch of different services. You're ultimately balancing across a bunch of different instances of the same service, but if this is a service like Gmail, then that's why you need so many load-balancing tears and so many different services in front of it.

By the way, for anybody who's listening to this is confused, you've got some great diagrams in the blog post that will be in the show notes.

[0:31:06.6] VP: Thanks.

[0:31:07.1] JM: Yeah. Next is the DNS layer. You've got users that talk to DNS, and when they talk to DNS, they're finding the IP address of the nearest L3 load balancer and then their request is going to go through this familiar cluster of load-balancing and then the service destinations that we've explored. What's going on in DNS load-balancing?

[0:31:34.4] VP: Sure. DNS is the system that translates names to IP addresses. For example, it might translate google.com to 4.31.115.246 or something like that. DNS servers can return multiple IP addresses for one request and they can kind of decide how they want to — Or what IPs they want to return.

For example, Geo DNS returns different responses based on who requests it. You could say that if you got a request from Washington, D.C. or something, you'll try and give them the IP address of your data center — Or some node in your data center cluster that is closest to them.

Round-robin is another way of returning IP addresses where you basically just cycle through all of the available IP addresses whenever you get a DNS request. Then once your client gets the IP address, then it'll talk to the Layer 3 load balancer in this case and continue down the stack as we've previously discussed.

[SPONSOR MESSAGE]

[0:33:02.1] JM: Do you have a product that is sold to software engineers? Are you looking to hire software engineers? Become a sponsor of Software Engineering Daily and support the show while getting your company into the ears of 24,000 developers around the world. Developers listen to Software Engineering Daily to find out about the latest strategies and tools for building software. Send me an email to find out more, jeff@softwareengineeringdaily.com.

The sponsors of Software Engineering Daily make this show possible, and I have enjoyed advertising for some of the brands that I personally love using in my software projects. If you're curious about becoming a sponsor, send me an email, or email your marketing director and tell them that they should send me an email, jeff@softwareengineeringdaily.com.

Thanks as always for listening and supporting the show, and let's get on with the show.

[INTERVIEW CONTINUED]

[0:34:03.9] JM: We talked at the beginning about how these services that we're load-balancing across, they should basically be the same, but if they're not —Okay. They're statelessness, but there's also the being the same. I guess those kind of statelessness, because you would assume that if they're all stateless, they all have the same lack of state. You gave this description that if you've got different clusters — If you got different courses of the same service and one of those clusters has different information than the other cluster, for example, if you've sharded all of your — If you've sharded your database across different clusters and there's a JPEG, and that's the example you gave. You've got a cat JPEG on one specific cluster, then you're actually going to need to do manual load-balancing and routing, because you need to route — If a user puts in a request to a service that you've got load balanced across multiple data centers and you happen to have — If it's a CDN, for example, or I don't know what

example you would want to use, and you happen to have only an instance of cat.jpeg in one of the specific clusters. You're actually going to need to route a request to that specific cluster that contains the service instance that has cat.jpeg. Can you describe this in more detail? Give an example for how this might work.

[0:35:38.0] VP: Sure. I'll continue along with the cat.jpeg example. Let's say that one of your users uploaded cat.jpeg and that's stored in a cluster in London, but nowhere else. If someone requests cat.jpeg, we don't want them to kind of automatically get routed to some random service instance. We want them to go to the specific one that has cat.jpeg.

One way of doing this is giving each service a unique — Some way to uniquely access each service instance, so you could give it its own subdomain or just directly use the IP address if you have them publicly visible, which isn't recommended, but that's a possibility. Then your application kind of has to do all of the hard work here and figuring out where it's stored and then returning the appropriate URL.

If you're like an image sharing site or something and you want to display this image, the image you want to embed in your page should be a URL that directly goes to the cluster or a service instance you know has cat.jpeg.

[0:37:07.9] JM: Yeah. Okay. That's great. Great expiration. The final load-balancing technique that you explore in your post is Anycast. With Anycast, multiple computers can have the same IP address and routers send requests to the closest one, and this is similar to what we were discussing when we're talking about the L3 load balancer talking to the L4 load-balancing tier where you assign the same IP address to multiple L4 load balancers that the L3 load balance tier is hitting in order to trick it with the ECMP. What is Anycast? Is that any different than what we were discussing there?

[0:37:51.8] VP: Anycast is generally, as you said, multiple computers can have the same IP address and routers send requests to the closest one. It is kind of different in the sense that, in the context I'm using it in, we're doing this on the internet. If you have like five data centers, you can give them all the same IP address and make it so kind of the internet handles load-balancing for you at that level so you don't have to specifically route requests. If you're in

London and the shortest path to — Sorry. If one of your users is in London and their shortest path so one of your data centers is to your data center in London, the request will automatically go there without you having to do work. If your London data center goes down or something, then that request will automatically get routed to another data center. It provides stability in that sense also.

[0:38:58.9] JM: All right, Vivek. We've talked through all these different layers of load-balancing and you have helped me scale my Software Engineering Daily t-shirt sales website and everything is fine, we're all doing good, but I'm worried about the latency and the throughput of the requests that are coming from my users, because now they're going through all these layers of load-balancing. Am I going to suffer on latency and throughput from all these layers of load-balancing?

[0:39:29.7] VP: Generally, your throughput will go up but your latency also will generally go up with each level of load-balancing you add to your stack of load balancers. Generally, what you want to do is if you make your service as low latency as possible and then run multiple instances of it to get high throughput. Instead of trying to build one thing that is both low latency and high throughput, obviously that's ideal. If you can do that, then great. Another approach is to try and get to the smallest latency as possible and then run multiple instances of it so you can feed requests through per second.

One way to help with performance as well is something called direct server return, which basically bypasses all the load balancers on the way back. In a normal load-balancing set up, your request would go through all of these load balancers on the way to your service instance and then the response would go back through all the balancers back to your user. That trip back isn't really necessary. It doesn't have to go through the load balancers. If you bypass that and have your service instance directly send the response back to your user, you can reduce the load on all of your load balancers.

[0:41:03.1] JM: Okay. After the request gets routed through all of these layers, the response is — It's not necessarily have to go back through those layers. I get the request. I don't have to send it back through all these layers. I can just send the response directly to my user, right?

[0:41:24.7] VP: Yeah. You have to do a little work there to make it happen. You don't get that for free, but that's a way of reducing the amount of data flowing through your load balancers.

[0:41:37.3] JM: Okay. What kind of work do I need to do to implement that?

[0:41:41.4] VP: Actually, I have no idea. I just know that you don't get it for free.

[0:41:45.6] JM: All right. In your post you mentioned some other companies. You referred to some blog posts by other companies, like Cloudflare and Fastly and Netflix. These companies have — They operate at serious volume, just like the Google and the Facebook. I think about Cloudflare and sometimes I just think like that actually — It just sounds like the epitome of the most intense load balancer and scalability issues.

What have you heard from these companies about how they do effective load-balancing particularly like the strategic and subjective decisions? Because I used to think of load-balancing, I'm like, "Okay. This is just like this commodity thing that I always get if I just click a button on AWS somewhere," but actually this is a very tactical and a fast evolving area.

[0:42:35.8] VP: Yeah. I mean the first time I actually heard of Anycast was from a Cloudflare blog post, or Anycast in detail. Yeah, all of these companies have lots of requests coming in so they have to be able to handle that load, right?

I think a lot of what we've talked about has actually come from what I've heard and read from all these places. There is another aspect to doing all of these in production that I kind of ignored in my post. Like for instance, if all of these service instances are auto scaling and containerized, all of your load balancers need to know how to actually get to your service instances right. What happens if one of them goes down? We need to deal with restarting them, health checks and things like that.

If your last layer of load-balancing before your service instances are doing health checks onyou're your instances, then they need to be able to remove some of them from rotation when those health checks fail. In theory, these ideas kind of explain it pretty well, but in practice there is a lot of other stuff that you have to mess with to get this working properly.

[0:44:00.8] JM: Cool. You've worked at Google and Uber and Facebook, and at the beginning we did talk about these companies having almost undifferentiated load-balancing requirements, because it's just at such high-volume. Are there any differences that you've noticed and how they look at the high scalability load-balancing issues?

[]0:44:25.5VP: Yeah. The problem everyone is trying to solve is pretty much the same, but everyone tackles it a little bit differently. One of the things Uber does is they use a thing called T-channel for a lot of their services. This is something that has been open-sourced. Instead of using HTTP to communicate between different layers of services and routing, they use T-channel, which is better suited for Uber's networking needs.

[0:45:02.3] JM: Let's shift topic a little bit since we're running out of time, and I wanted to ask you about this other topic. We're discussing before the show about deep learning and scaling deep learning, which kind of relates to load-balancing. It's a little bit different. This is like kind of more cutting edge topic, and we did a show about this with Will Constable from Intel a while ago, and that show was one of the harder ones for me to narrate honestly because deep learning is hard enough to understand. Deep learning distributed is much harder. What is the distributed deep learning? When you're just doing distributed deep learning, because you've got tons of data, you start to get a huge model. Maybe you could just rhapsodize about it a little bit, why is distributed deep learning difficult?

[0:45:54.5] VP: Sure. Generally, one problem you'd run into is if you have some model that you're training and it takes — For the sake of example, we'll say it takes four weeks to train on one GPU. You want to do that much faster in order to increase iteration time and let you experiment more. You want to run it across multiple GPUs. The hard part is figuring out how to split that training across multiple GPUs and handle communication between all of the individual training processes. It kind of turns into a high-performance computing problem, because you need to be able to send information between each of the "independent training processes" as fast as possible.

At a high level, when you're training the deep learning model, you have a bunch of errors that you kind of want to accumulate across all of your individual processes. Doing distributed

average or things like that tend to take up a lot of time relative to your time spent actually training. Cutting down on that is what gets you high performance.

[0:47:27.3] JM: Are there any parallels we can draw between our load-balancing conversation and the scalability of deep learning systems?

[0:47:34.8] VP: Sure. As I said at the beginning, a load balancer is a device that distributes work across many resources. I don't remember if I said that at the beginning.

[0:47:45.9] JM: Let's assume you did. I'm going to cut it back in now.

[0:47:49.8] VP: Okay. Cool. What you're effectively trying to do is distribute the training work across multiple resources, so multiple GPU's. The hard part is that all of these need to communicate. They're not fully independent. So you need to make that communication as fast as possible. You're not really routing requests from something external.

One of these, let's say, 8 GPU's or whatever, or 8 processes running on their own GPU, you're trying to very quickly communicate information between all of these processes.

[0:48:25.8] JM: Okay. That makes sense. Just to close off, people should definitely check out your blog at blog.vivekpanyam.com. You got some great material there. I'm sure there's more on the way, and I look forward to maybe having you on again in the future if maybe you can come out with another article that's as good as your load-balancing one.

[0:48:48.5] VP: Thanks.

[0:48:48.9] JM: To close off, how did your experience at Google and Facebook compare to Uber in terms of the engineering work you've done?

[0:48:57.9] VP: To clarify, I was intern at Facebook and Google. I wasn't there full time, but at Facebook I worked on photos, and specifically photos on android. At Google I worked on Project Fi. Unfortunately, I'm not allowed to go into depth on what I did there. That's a little unfortunate.

At Uber, I worked on surge pricing algorithms, a product that hasn't launched yet. Then I switched to ATG recently where I've been working on deep learning for perception for self-driving cars. To answer your question, I think your experience really depends on what team you are on. At any large company, I think there is some amount of variance between what experiences people have, and I think that just depends on what team you're on in terms of engineering work.

I kind of worked on a bunch of different things, so I don't know if it's entirely fair to compare them, because like frontend at Facebook versus deep learning here. I don't think this really an apples to apples comparison.

[0:50:10.0] JM: Yeah. It totally makes sense. That's pretty cool. You're cutting your teeth at some very difficult projects. I can't wait to hear — I don't know, maybe in five years, what you work on at Project Fi. Project Fi, for people who don't know, is like this thing where you basically can get cellular service through Google Project Fi and it finds the nearest cell phone tower regardless of whether that cell phone towers owned by T-Mobile or Verizon or AT&T, right?

[0:50:43.1] VP: Last I heard in the U.S., I think it's built on top of T-Mobile and Sprint, I want to say. That might be wrong. It can do some interesting things, like switching between those two underlying networks depending on which one is better and seamlessly handing off calls between Wi-Fi and cell. Basically, you get the best of all of the underlying networks.

[0:51:09.6] JM: That's awesome. Okay. Cool. Vivek, I could talk to you for hours. It was great talking about load-balancing.

[0:51:15.8] VP: Yeah, likewise. Thanks for having me.

[END OF INTERVIEW]

[0:51:18.9] JM: Heptio was founded by two creators of the Kubernetes project to support and advance the open Kubernetes ecosystem. Helptio unleashes the technology-driven enterprise with products that help customers realize the full potential of Kubernetes and transform IT into a business accelerator.

Heptio's products improve the overall experience and reduce the cost and complexity of running Kubernetes and related technologies in production environments. They build products, they build open source tools and they provide training and services and support that bring people closer to upstream Kubernetes. You could find out how Heptio can make Kubernetes easier at heptio.com/sedaily.

Joe Beda and Craig McLuckie, who are the founders of Heptio have both been on Software Engineering Daily and they were fantastic when they came on. They're really fluent in what is going on in the Kubernetes ecosystem because they helped build it. To find out more about the company that they're building with Heptio and to find training and resources and products built for Kubernetes, go to heptio.com/sedaily. Thanks to Heptio for being a sponsor of Software Engineering Daily. I'm really proud to have the company onboard.

[END]