Rethinking SLOs

Narayan Desai explains why SLOs can be problematic and proposes alternative methods for monitoring complex, large-scale systems.

Viv: Hello, and welcome to the SRE Prodcast. This is a limited series where we're discussing SRE concepts with experts around Google. I'm Viv, and I'm hosting today with MP.

MP: Hello.

Viv: And our guest is Narayan, who is here to talk about service level objectives (SLOs). Narayan, welcome to the Prodcast.

Narayan: Hi, so I'm Narayan Desai. I'm what we call an Uber Tech Lead. I'm responsible for the reliability of a portfolio of products—GCP's (Google Cloud Platform's) data analytics products. I've been at Google for about five years. And prior to that, I did a lot of work in high performance computing.

Viv: Cool stuff. Thanks for being here. So service level objectives are pretty well-known generally as a primary tool for SREs to understand their services, and we talk about it in Chapter 4 of the SRE Book, if you're not familiar with that concept. But today, I'm curious about how these concepts actually work when you're looking at something at the scale of Google— something like Google Cloud. So what are your thoughts?

Narayan: So, I think it's been a decidedly mixed experience. So if you think about the business scenario that SLOs were designed in, Google's historical core businesses were very large, had large numbers of customers, and could only

afford to approach reliability in the aggregate, right? Not necessarily targeting individual users of high scale services like Search or Gmail. And our situation in GCP is quite different, where each of our customers counts on us to provide infrastructure that they use to build their businesses, right? And so every single one of them needs to have a good experience. And so the whole construct of SLOs as basically an aggregation tool that you can use to combine all the experiences together and set a reasonable threshold of what good performance is. It gets a lot harder in the situation, right? Because all of your customers don't necessarily have the same value to the business. And also, you wanna deliver a good experience to all of them, right?

And so these core notions, like error budgets and so forth that are built on top of SLOs, end up being kind of problematic, right? Because no one wants to be in the error budget, and it's understandable, right? They're paying money for a service that they depend on. And so I think from that perspective, it's been tricky to apply SLOs uniformly across GCP.

And I think the other dimension that ends up being really interesting is if you think about the core Google services that everyone uses, the use cases are pretty narrow by comparison. You know, if you think about a service like GCE (Google Compute Engine), that's basically a data center in a box. There's a really large API surface, there's a large number of degrees of freedom that customers have, and then they end up building very, very, very complex workflows on top of these building blocks that we provide. And so increasingly, in addition to caring about rates of errors, we care about the performance that's delivered to customers. And I think SLOs really struggle to provide meaningful insights when you need to analyze the performance delivered by a service with a wide range of workloads that customers might request from you.

Viv: Yeah. Makes a lot of sense. It's definitely interesting. So, what would you recommend in terms of filling this gap that SLOs have when looking at something this complex and this large?

Narayan: So I think that it's worth considering errors and performance discretely, right? It's a lot easier to analyze errors using SLOs than it is to analyze

performance using SLOs. But I think even more important than that, the first question that you need to have when you're trying to use one of these systems is: what is the outcome? What is the business outcome that you wanna produce? And how well are these tools doing the job?

I think one of the interesting things about the way that SLOs have been adopted is you'll frequently see people saying that they're actually useful for everything, and you should use SLOs to solve lots of different use cases. I personally think this is pretty problematic, because there's no guarantee that multiple use cases that you try to build on top of the same SLOs will be compatible, right?

And so the first thing that you can do when you're using a system like SLOs is you should be sort of skeptical. Try to understand if the SLOs are helpful to you meeting your business objectives or not. And the second thing that you can do is try to make these SLOs as narrow as possible so that they can be used to answer a single question well, as opposed to answering a variety of questions sort of poorly. And I'm a big believer in the need for high fidelity insights increasingly in our services, right? So that we can understand what's going on.

As a next step, it's really important that we figure out effectively what "normal" is in our systems. So SLOs are used as an approximation of normal, right? You know, typically the way that these are formulated is that an expert in the system will go and examine the delivery performance of the system over some historical window— the previous month, previous couple months— and then say, “Oh, well, for this API operation, usually it's some number of nines— you know, 99.99% of requests— should finish in less than 500 milliseconds." I think that's an approximation of normal, but that's a point-in-time approximation of normal.

So one of the ways that this can go wrong: so imagine that you have a storage system that you use, and the storage system consumes 300 of those 500 milliseconds. And like, software hopefully develops over time; imagine your source system speeds up by 30%. Now the estimate that you have normal at 500 milliseconds for 99.99% of requests is suddenly off because 99.99% of requests should actually only take 400 milliseconds now. So there's also a need for reexamining the data and trying to understand how things change over time. You

can see shifts and workload, right? If customers start sending larger requests or more complex requests, that might shift the behavior that you're seeing. And so increasingly, I think that it's important that you add in a verification step as well, right?

So the three top things that I would recommend are: sort of be skeptical. Be very specific about the questions that you're trying to answer, and then validate and iterate over time to make sure that your estimates still make sense and fit the way that the system behaves. Because you don't want a de-tuned alert, or an overly aggressive alert, right? If some sort of subsystem gets slower, you wanna take that into consideration and be triggering actions at the right time.

MP: And that calls attention to one of the biggest pain points with SLOs that I've seen in my own personal experience— is every time you add an SLO, that's another thing that you have to do maintenance on and actually make sure that your objectives are still up to date with the actual performance.

Narayan: Yeah. Well, I think the other thing that is interesting is… so it is clear how to make SLOs that are narrowly useful to answer questions. However, like you mentioned, there needs to be a maintenance lifecycle, and I don't think that this maintenance lifecycle is very well-developed at this point. And the other big problem that you have related to that is: if you think about what happens, if you have specific SLOs to answer particular problems— is that you end up with SLO proliferation, right? You end up with more and more SLOs. So, you know, for example, we have one service internal to Google, that has— it may have grown since the last time I spoke with this particular team, but they have about a quarter million parameterized SLOs.

Viv: Oh, no.

Narayan: And this is one service. That's kind of hard to manage, right? And so I think that, you know, the other place that this gets very difficult is that if you have that many SLOs, you're always out of SLO. What does that mean? You know, how should you act? How do you combine those signals together into something that's meaningful? I think all of these are very complicated questions that we

haven't really settled very well.

The other thing that I think is really tricky is: SLOs kind of codify what is expected, but it doesn't incorporate any model of what the service behavior should be that's flexible in any way. So you end up with these very brittle representations of what performance should look like. And so not only do you need to maintain it, but it's unclear what the cadence you need to maintain it on is, and so you never have good confidence of where you are.

So you know, you end up in the situation where you're getting more and more SLOs and they need to be modified. You don't know how often they need to be modified. And so the precision ends up being really low, because typically, the way that SLOs are deployed is that you sort of tie all of your alerting to it, and you may trigger freezes or other sorts of large-scale actions based on the state of the system with regard to these metrics. And there are a lot of incentives, you know. For example, if you're gonna page someone whenever you're out of SLO, and you have even a moderate—like, let's forget the quarter million case here for a minute—and you just have a service that has little hundreds of SLOs, you will page someone to death if you use SLOs in that way, right? And so there are a lot of reasonable incentives that cause people to make their SLOs less aggressive, as well. And so I think that really, there are a lot of very, very difficult challenges with SLOs, and we really fundamentally need to move onto a more robust framework.

MP: While we're still in this realm, there's something I wanted to jump back to slightly. In your framing of this and the experiences with Google Cloud, a lot of that sounded like a lot of the concerns arise primarily out of the notion of: the user shifts a little bit where it's not this individual user. It's one "user" that is actually a whole organization behind that that's relying on the system. And it sounded a little bit like these problems might arise most in the case of a cloud provider that's indirect, like a more B2B (business-to-business) instead of B2C (business-to-customer) structure. So I'm wondering what your thoughts are in the framing around what types of business questions are you trying to answer with your SLOs? I'm wondering on what your thoughts are on how all of that applies—if that applies the same, or if it applies in a different way to those direct-to-user

monoliths like Search and Gmail?

Narayan: So I would definitely say the details are different. I definitely think that there is a really radical difference between B2C and B2B. And I think that fundamentally, what this comes down to— I think, you know, at the end of the day, the business problems that you're trying to tackle here are actually pretty similar in both cases. We all have limited reliability resources, and we need to figure out where to spend them, right? And so SLOs are a mechanism that we can use to understand where we should engage, and you know, if you think about it, on-caller resources and incident response is expensive, right? So where and when do we engage that? Well, you wanna engage that when it's needed the most. So we've tried to do that with SLOs. In some cases, that works very well; in other cases, it doesn't work so well.

And I think that from that 50,000 foot perspective, the problems are exactly the same whether you're doing B2B or B2C. But then the specifics get to be very different. We have a wide diversity of users in GCP ranging from free tier customers that we want to give a good experience to when we can— you know, we certainly want to invest there to some extent. But then we also have larger customers that pay us more money and expect better service.

And so a lot of the places where I think SLOs have proven problematic in that shift to B2B is that the quintessential SLO calculation is to take all the requests that you had, and count up the number of them that failed and divide that by the number of requests that there were, and that's your insight, right? In B2B, you need to start considering these questions of: what sort of weighting do you wanna use? Do you want to consider all requests the same? Do you wanna consider some requests to be more important? Do you wanna do alternative types of normalizations that might give you different insights out of the data? These things end up being different because I think B2B is fundamentally wrestling with a different set of questions.

MP: That's great. Thank you.

Viv: So I am curious, then—you mentioned the need for better process around

maintenance for managing SLOs or I guess whatever the alternative would be. I know you also mentioned "a more robust framework." So I am curious: if we're still talking about Google Cloud, B2B, what would that more robust framework look like? And you know, say you could start from scratch, right? What would the differences be? And what are we lacking that could hypothetically fix some of the things that we discussed that are problematic?

Narayan: Sure. So we are working on exactly this, because we think that this is a big problem, or an area of need, right? I like the way that you put that question, because I think that you do actually need to start and rebuild from the ground up.

So one of the things that I think is very important is that we need reliability analytics that incorporates some notion of what the invariants we expect to exist in our system be. So for example, SLOs don't say anything deeply about reliability. SLOs say when you should worry. Like, when your rate of errors gets above this particular percentage, you should take additional actions— [that's] effectively what SLOs tell you. And instead, you know, getting down to brass tacks here for rebuilding this, the first thing that you really need to do is you need to get very serious about understanding what it is that you mean by the word reliability, right? Because at the end of the day— I think this is another difference between B2C and B2B— the customer's perception of reliability becomes even more important.

And so you need to understand what it is that the customers mean when they say that a service is or isn't reliable. And historically, we haven't done a great job of defining that. We've ended up with these circular definitions where your service is reliable if it meets its SLOs. And a service meets its SLOs if it's reliable, right? And then you sort of pick some arbitrary sort of numerical thresholds for that, and you end up with this self-referential mess.

And instead, I think that it's really important to start with a core definition of what it is we're talking about when we say that a service is reliable. So the definition that we've begun to use is a combination of effectively stationarity, right? Customers expect that they continue to get the service that they got yesterday. And they expect this continuity— or stationarity, if you think about it from a

statistical perspective— in three dimensions. They expect that the service will be available: that is to say, when they need it, it will be there. They send a request; you will respond and not just provide an error. They expect that these results will be correct, right? That there is some API contract for the service. And they expect that the performance will be consistent, right? If they send you an RPC, and it took roughly 300 milliseconds yesterday, it'll probably take about 300 milliseconds today.

And what that tends to mean, you know, when you start to cast that in mathematical terms, what you're really looking for is a situation where you can make an assertion that the performance that's delivered from a service results in a stationary distribution over time, right? That's not to say that every request will be exactly 200 milliseconds, but that says that the range and distribution of performance that you get or errors that you get is sort of self consistent over time, right? You'll have good days, you'll have bad days, you'll have fast and slow requests, but the distribution will remain roughly fixed.

And I think that this approach really gives a different way of looking through systems, or looking at the behavior of systems. Because fundamentally, if you think about— if I were giving a talk here, I would put up a slide at this point that had two performance graphs. You can imagine a graph that is relatively level with very small variance, as one example. And another graph that has lower— the trendline is low, except there's some occasional large spikes, right? And imagine that both of these graphs have the same mean. If you think about which service is easier to use and would be viewed as more reliable by a user, it's definitely that first one that has low variance, right? So, I think that one of the things that is the most important here is that we need to start taking variance more seriously as a core property of our distributed systems. So I'd start there.

We built some analytics that will do analysis of performance distributions, and we have found that these are good indicators of customer surprise, right? Like, you know, if you think about it, in this kind of infrastructure, surprise is almost always a bad thing, right? And so really, what you wanna do is you want the surprise detector. And so that's what we've been working on building. And it turns out that if you're able to break up your workload into pieces that are self-similar,

then many of these behave consistently over time and you can use rates of unlikely events to spot situations where your service isn't behaving properly, and it is not a function of customer changes to their workloads.

You know, if I had to look into the crystal ball over the next five years, SLOs in my mind really represent the simplest possible solution to producing summary statistics about the reliability of a service, but I see much more complex analytics happening over the next several years. And I think that these complex analytics are really gonna tell us a lot of very important things about our services and give us tools that we'll be able to use in order to solve problems faster and cheaper, and ideally, before customers notice them, right? Like, I think at the end of the day, we know that we'll have outages, but you want to be able to find and fix these things as quickly as you possibly can. And I think that that's the challenge in front of us. And I think it's a really exciting one.

MP: In your definition of reliability, I'm under the impression that performance as you're using it is more or less synonymous with the user's request latency? Or response latency, I guess?

Narayan: Depending on the service. So if you're thinking about a serving system, then yeah, roughly. So I spend most of my time thinking about analytics services, and some of those are focused as RPC response pairs. Others are much more asynchronous, right? So you may have a data flow job that runs over the course of hours or days, right? So that's not a single RPC. But really what— you know, when I'm talking about performance, there is the thing that is valuable to the customer, right? And if it's an app, Engine App, it's the response to a request. if it's BigQuery, it's the response to a query. Each service is going to have its own valuable product that delivers to customers, right? And so figuring out how to measure those and measure those at fine granularity ends up being a big part of the challenge here.

MP: I was also going to ask that— trying to measure correctness sounds very difficult.

Narayan: It's extraordinarily hard, yes.

MP: In my mind, the only way to know what the response to a given request is is the system behavior itself. It's sort of definitional and it gets back to that circular logic.

Narayan: Yeah. There are strategies that you can use in some cases, right? So check sums are an example where you can detect some failure modes, but the more complex the service, the harder that gets to be. So storage systems have a variety of really difficult engineering challenges. But one place that I really envy them is that they are largely responsible with storing literal bits, which means that you have the ability to at least understand the structure of the data and what should be coming out of particular APIs. There's other correctness difficulties that they have there, so this is by no means a panacea. But for analytics, this is very, very difficult. And so you end up going down a variety of complex tasks.

One of the wilder ideas that we aren't experimenting with but have spitballed a little bit is: there is some sort of historical precedent for multiple implementations of core logic, right? And running important things twice. You can even think about doing that with multiple releases of the code. You know, I think there's some precedent for that as well. But yeah, correctness is a fiendishly hard problem.

Viv: I'm curious, too, because earlier, you talked about potentially weighting different priorities, especially from the user perspective. So when you're looking at something like correctness, or reliability, what users expect— users do a lot of different things. I guess the question is: with this approach of minimizing variance and such, is potentially weighting different user needs over others still in the equation? And how is that accounted for?

Narayan: So to be really clear, my team has not been working on statistical methods in this space so much. This is just sort of speculation at this point. But it's definitely one of the things that we're going to need to tackle. So our primary work has been focused on errors and performance.

Viv: Got it. Yeah, there are a lot of aspects to this problem. I'm already feeling in over my head.

Narayan: I am most optimistic at this point about the insights that we'll be able to get out of performance analytics. You know, there's a weird way that you can view a broad class of error modes. And one of the things that we've observed in many of our outages is that prior to large scale mayhem, you often see performance blips, right? There are subsystems or dependencies that start to get slower. And, you know, as they start to load up and bog down, eventually you hit some critical limit where you have deadlines, and then that bad performance gets helpfully translated into an error for you, which allows you to bail out.

But what that means is that errors and performance problems are actually more synonymous than you might think. And I'm not saying that 100% of errors are performance problems, but at the root of many RPC errors that have returned, you actually have some component that is slow. And if you were to take a really maximalist view of this: what is a timed-out RPC, but an RPC that hasn't responded yet? That's sort of a performance problem.

And so, you know, I'm really optimistic. And we have some evidence that you can see a really large percentage of your outages in the performance domain. And so I think that that's actually the most productive place to start here.

Viv: So what is the big takeaway from all of this, and what would the path forward look like from this point, as you see it?

Narayan: So I think the most important takeaway here is that there's a huge opportunity in doing more sophisticated analytics on our reliability data. My personal belief is that this will involve the use of models that incorporate what we expect our systems to be doing, and starting to build invariants for the behaviors that we expect them to exhibit on a consistent basis. And I think we need to start formulating more detailed questions, and trying to answer them with data— not just in the SLO formulation, but in others, as well, and see what works, right?

I think that this is going to be an interesting and difficult process, where nobody

has all the right answers yet. And so I think it's really important for the community to engage and then talk about what they've done and what works and what doesn't. And that's how we're all going to get better at this. Increasingly, the systems that we're building are getting more and more important, and we need them to be more and more reliable. And so, you know, I think that's an exciting challenge for the next 5 to 10 years. Computing platforms are increasingly at the center of lots of important processes, and we need them to work reliably.

Viv: I agree. I think it's a really compelling marathon to be part of right now. So props to that, I guess.

Narayan: Yeah, it's definitely an exciting time.

Viv: MP, do you have any other questions before we wrap up for today?

MP: I don't think so. I really enjoyed chatting with you today.

Narayan: Well, thanks. This was a fun chat.

Viv: So yeah. Thank you so much for being here. It was great to have you. And for all our listeners: tune into the SRE Prodcast next time for Episode 5, which will be on migration.

MP: Thanks so much.