Google Prodcast Season Three Episode Nine

[JAVI BELTRAN, "TELEBOT"]

STEVE MCGHEE: Welcome to season three of The Prodcast, Google's podcast about site reliability engineering and production software. I'm your host, Steve McGhee. This season, we're going to focus on designing and building software in SRE. Our guests come from a variety of roles, both inside and outside of Google. Happy listening, and remember, hope is not a strategy.

—

Hey, everyone, and welcome to another episode of the Prodcast, Google's podcast about SRE and production software. I'm your host, Steve McGhee.

FLORIAN RATHGEBER: And I'm Florian.

STEVE MCGHEE: This season, we're focusing on software engineering and SRE. And today, we have two guests to tell us a little bit more about observability, this time beyond just metrics. There's plenty of other things out there. We have Narayan and Pat. Can you both introduce yourselves?

NARAYAN DESAI: Sure. So I'm Narayan Desai. I'm an SRE at Google. I work on Google Cloud, and I particularly focus on observability and efficiency stuff.

PAT SOMARU: Hi, I'm Pat Somaru. I'm a production engineer at Meta, working on a lot of things, but at the moment, I'm working on a `sched_ext`, which is a cool scheduler thing for Linux.

STEVE MCGHEE: Great. So as I mentioned, this season is about software engineering inside of SRE. And so, in the past, we've talked about operations and culture and different jobs and things like that. But really at the core of it, we're dealing with software. And it turns out the best way to make software better is to write more software, or change the software, and that comes down to software engineering. So it's not just fiddling the things that are already there, but writing new stuff.

And one of the kind of base problems in operating in SRE in general is knowing what to mess with. And this is where observability comes in. So we have to understand the problem before we can fix the problem, as we say. So out in the world, if you look up observability or even monitoring, you'll see the three pillars, as they say.

So we'll see metrics, logs, and spans or traces. Is there more to it than that? Or is that it? Is that the extent of what we've got?

PAT SOMARU: Well, there's also performance data which, like, think about traces. So I guess, when you say traces, I think about elastic APM, that thing where you're doing distributed observability across containers.

But that also kind of applies to the binary level, to, like inside of your application. If you pull out your kit or perf to see where the time is going. And it covers, it traces, but not necessarily, but yeah.

NARAYAN DESAI: So I tend to think about this more from the perspective of use cases as opposed to the data. I'm an analytics guy, so I think about the way that you can analyze the data and the kinds of insights that you can get out of it. And so from that perspective, those are the data products that you have, but there's a lot of different things that you want to do with them. And I think that the interesting aspect of this, from the perspective of those data products, is the semantics that they have.

So you fundamentally have time series data, and time series data has particular properties, in particular, scalability issues. You have logs, which are fundamentally events, and then you've got traces, which are fundamentally graph structures.

And if you think about observability data through that lens, then what that really tells you is that you need three different classes of approaches for analyzing the data. Because if anybody has ever tried to wedge graph data into a relational database,

STEVE MCGHEE: No one's ever tried that.

NARAYAN DESAI: Not so great. Same thing for time series data. Right?

STEVE MCGHEE: Yeah.

PAT SOMARU: So to build on that, actually an interesting observation I had a little while back: you said there's different classes, and you have graph data, yes. And you also have time series. But in a way, it's a time series of graphs, which is, I don't know, that's a bit of an interesting thing. Like, how is it performing that's a graph? Or what's happening that's a graph? But that changes over time series.

STEVE MCGHEE: Like the graph changes over time. Is that what you mean?

PAT SOMARU: Yes. And that's what actually folks care-- at least, I've found that folks care the most about. Like, why did this get slower? Why does this cost more? Well, that graph changed. You're serializing a timestamp versus storing an integer.

STEVE MCGHEE: Mm-hmm. Yeah.

NARAYAN DESAI: Yeah. I think those connections are really important. And that really makes us, from an analytical perspective, a multi-modal kind of problem. Because you have things that happen at particular points in time, then you have things that are happening in metric space, and that's going to change the structure of the graphs that you have or the content of the graphs that you have.

And being able to connect those things actually produces a whole different set of insights, which are fundamentally actually much more, I think, interesting and compact than all of the other data products that we tend to deal with.

STEVE MCGHEE: That's cool. So if you're stuck in a world where all you can do is make a table and an XY graph, maybe you're missing out on some fundamental insights into what your data is trying to tell you.

PAT SOMARU: Well, I think that the hard part is getting to the table with the XY graph with all the data. Ultimately, that's what you want. It's how many servers do we need, how much is this going to cost. But a time series of a graph needs to somehow eventually be translated into something actionable, whether it's a code change, or a number of servers, or something.

NARAYAN DESAI: Yeah. And fundamentally, many of these techniques are really about taking complex data and building it down into an insight that's simple enough to understand. One story from a while back. We were doing some analysis work, and we were using this overly complicated machine-learning approach for clustering to try to make sense of performance data for a service.

We spent a bunch of time tweaking this approach, and building up the infrastructure, and all this. And it was this really fiddly thing. And once we found something in it, we started looking at it different ways. And we found that it was actually visible with the two component PCA, which is pretty much basic stats 101 stuff. It's effectively an XY plot.

And then what we realized is actually also the simple methods are the best sometimes. So when you can use the simple methods, you should use them. But there's this siren song of new and complicated things. But at the end of the day, what we need to do is we need to take this very complicated structured data that's high dimensional, and we need to turn it into a thing that the human brain can understand. And that's, frankly, simple insights.

PAT SOMARU: You mentioned that. And actually, I found a similar one. I guess, trying to query the data, which is hard. It's like there's a lot of data, just asking a query can be tricky. If you want to make it easily and readily usable, that's another problem on top of itself. But those simple insights, it's like, if you simply have a way of representing the data where you can discard most of it, that actually, sort of intuitively, it kind of makes sense when you think about it.

So, like, you write some library, this is your hash function. You're going to use that everywhere. And when you actually look at the graph, it turns out that if you discard like 90% of the data, you're still covering something like 90% of your cost. And then that enables you to do wild integrations with it.

FLORIAN RATHGEBER: So Pat, you just outlined one of the basic truths, I guess, of profiling and figuring out what's the performance of a particular piece of code. So for someone that knows how to profile the inner loop on their workstation, so to speak, as it were, how does that then relate to the distributed systems, the bigger piece that we care about, series, and production engineers.

PAT SOMARU: It kind of does. So I had a talk at SREcon a little while back about some tricks that I came up with and we played with at work and whatnot, to address that. So most folks don't know how to take `perf` and get a flame graph from it, or to optimize it. It's not the concern at the forefront of everyone's mind, but you kind of need everybody to be a bit concerned about it, sort of like with security. Otherwise, problems happen, and they can be expensive.

The way I found, I think I was saying, with make the data accessible, is that if you can reduce the data by removing the parts of it that you know are so small, you don't care about it so much necessarily, and you can find a good way to represent distinct things-- so like, you might think it's your binary. You run perf on your binary, and that's how your binary performs.

Except for your environmental variables. That can change. So you have to consider the configuration of the system and the hash of the source code, and sometimes, even build information. But once you find the correct representation of that, then you can do things like squiggly lines in an IDE, or something like that, once it's an immediate lookup.

It's like, oh, for this line, well, just put a little red here. This line of code you're editing costs a lot of money. Be very careful not to make this slower or not. It costs a lot of money, but it is ran many times per day. So like hotness of the code, kind of.

FLORIAN RATHGEBER: So it's both a data processing, as well as a visualization challenge in a way?

PAT SOMARU: Accessibility. Developer accessibility, I think. Because you can't expect everybody to be able to pull out `perf` and run it on some JVM code. I mean, if you can even do that. Never try to, just use YourKit. But you want to make it accessible to all developers and kind of put it in front of their face.

Unless it is their job, in which case, they're going to use perf because they're not interested in your high-level removing the details they care about. But you want to make it easy so everybody can-- you move the ship in the right direction. I feel. Yeah.

NARAYAN DESAI: So I think one thing that's kind of related to this that's pretty important is-- fundamentally, engineering is all about building up mental models of what it is you're doing. I think gone are the days where someone could expect to understand all levels of the stack deeply. And so you have more specialization, you just have more things to know.

And so one of the key things that I think is very important when you're designing observability is to really think about the mental model that people are using. I think that there's another angle of this that ends up being kind of interesting as well, which is that, we don't have enough meaningful models of how we think about distributed systems.

And so, to the extent that we can build more useful models there, we can then give people a way to think about their systems that are more useful than-- I mean, I feel like a lot of SREs-- the old adage is, just bad things happen all the time? And that's true. It's one perspective, but there are more patterns in the chaos that we can tease out if we start to model systems better.

And that, often actually, ends up resulting in, if not less data, a lot less noise in the data because you're measuring parts of the system that are actually meaningful interacting components in it.

FLORIAN RATHGEBER: Getting better signal is definitely always a win.

NARAYAN DESAI: So one thing that I spend a lot of time thinking about lately, having worked on observability for the last several years, is, one of the key problems that I don't think that we've really tackled well enough is understanding noise and reducing it.

So we have all these measurements. And anyone who's actually tried to ever use an SLO for something, for a complicated system, knows that you have to deal with noise, and it's a huge problem. And there is some level of concession that people have to noise, which is to say, well, we need to deal with the noise somehow. And so, if we set too stringent an SLO, we're just going to get paged to death for things that aren't real. And what that costs us is that there are going to be some things that are real that we don't see if we set the SLO to some less stringent value.

And I actually think that that's kind of wrong. I think that we should really be looking at observability from the perspective of how do we reduce noise in the system? Because if you can reduce noise in the system from a fundamental perspective, that allows you to see more things and see things more accurately.

So to give a really concrete example, we've done a lot of work in the last few years, and a colleague of mine, Brent Bryan, and I gave a talk on this at a SREcon, I think, two years ago in doing workload modeling. So the work that we talk about there is basically doing quantile estimation for performance of discrete RPCs, basically.

Any service has some amount of workload dependency when you send it a request. There are some big ones, there are some small ones. Some of them are fast, some of them are slow. Some of them are complicated. And what this means roughly is, some of your requests should take 50 milliseconds. And if they run in 55 milliseconds, that's not a big deal because that's sort of within your standard deviation. And there's some requests that should take 10 milliseconds. And if they run for 55 milliseconds, then something is horribly, horribly wrong.

And so the experience that we've had with trying to model our performance events that way is that-- I love insights like this-- fundamentally, workloads matter. This is not surprising to anyone when you say it out loud. But when you consider workloads in the context of your performance events, that allows you to normalize statistical scores for different kinds of workloads, and then combine them back together to understand how your service is doing as a whole. And this effectively allows you to control for the effects of workload mix shifts in your service.

STEVE MCGHEE: So just to drill into that, the terminology around it, the workload you're referring to, like, kind of the inputs to the system matter, not just the system itself. So you can have one system that gets used in three different ways based on its input pattern, and its measurable performance will be dramatically different between those three different ways.

But if you just average them all together and be like, this one thing has this one number, then how does that help you? You want to be able to discriminate between the different modes that this one binary or system, or whatever it is is running under. Am I getting that right?

NARAYAN DESAI: Yeah, that's absolutely right. And it's not even necessarily one system. It might even be one RPC call. So think about an RPC call that has a cache path and a go-to-the-database path.

STEVE MCGHEE: Yeah. Totally.

NARAYAN DESAI: So any system will change over time, and the proportion of cache hits will change. And as that occurs, your average latency will change. This may be meaningful. It may not be, right? It's certainly something that you want to know about, but you should measure that explicitly and then try to understand the extent to which, if your cache hit performance drops by 30 milliseconds, that's a thing that you would want to recognize discretely from your workload mix shifting.

PAT SOMARU: Are you saying that you pull out these different, I guess, categories as dimensions? Like, after the fact?

NARAYAN DESAI: Yeah. So basically, what we do is we extract a bunch of parameters that characterize effectively a cohort in the workload. And whether you hit the cache might be a parameter that goes into that cohort, as well as, maybe the customer or some other aspects that capture the differences between the discrete workloads. And then we're able to use that to build historical models that we can calculate quantiles on, for example.

PAT SOMARU: That is very cool. Yeah. I can say I've only ever tried to do it the other way, which is, know the inputs. I think the highest level of granularity I went to is like, a request is cached or not. But that's pretty cool. I have different types of requests. Yeah, that's cool.

STEVE MCGHEE: Yeah. The example that you gave, Narayan, was fascinating to me because you don't know if a request is going to be cached when it enters the door. It depends on the time of day, and the last thousand people, did they make a similar request or not? But ideally, like, hopefully, you can know it after the fact, like, in some sort of log entry or performance like this request was this speed. And also, by the way, like cache equals true.

So having that data point, at least on the other side, is something that you can then take advantage of later. It doesn't help you predict things, I guess, but it does help you classify that there's this one type of request, and this other type of request. That's awesome. That's a good way to think about it.

NARAYAN DESAI: Yeah. And it turns out that this kind of a calculation is really cheap to do. So basically, what we do is-- quantile estimation is sort of calculating a Z-score. And so if on know on a per granular cohort basis what the mean and standard deviation of historical performances, you can get a pretty good sense of what just happened to you, and then you can aggregate that using low cardinality metric systems relatively readily.

And so this gives you a good sense of what's going on. It's not a panacea by any means, but it allows you to answer a particular class of questions. And so, anyway, this technique is sort of interesting. But what we realized after the fact is that, another way of saying what I just said is that mixture effects, in most services, introduce substantial amounts of noise to basically all of the measurements that you do of that service.

The consequence of this is that if you are able to control for those mixture shifts, then you're able to substantially reduce noise. And in fact, that's what we ended up seeing as we used these techniques more and more. So, basically, there's a lot of native volatility that you see in pretty much any metric.

And this smooths a lot of that out. And it gives you a much better ability to understand when things are changing with much more sensitivity. And the increases that we-- so my opinion here is that by controlling for this, you actually greatly reduce the amount of noise in the system, which means that you need to make less of these noise versus precision tradeoffs.

You'll always need to make them to some extent. You need to decide how much work you're going to put into working on problems in a particular domain, or with a particular metric, or whatever. However, if you're able to reduce noise, that means that you have much less of kind of casino effect as you're getting alerted. And that, actually, is incredibly valuable.

And I think that that really fundamentally comes down to treating this as sort of a post hoc description of what we realized after doing this work, this, unfortunately, wasn't going in with a master plan. But think that we actually need to start thinking about noise reduction in a really explicit way in observability, because then, nobody wants to be paged by a thing that's not actually there.

STEVE MCGHEE: It doesn't help to measure the noise. Yeah.

NARAYAN DESAI: Yeah.

FLORIAN RATHGEBER: And I suppose one of the cool aspects of your method is also that you don't actually need to know, if I understood you correctly, you don't need to know the expected performance of your workload cohort, but you actually infer it by observation, and then regress against past performance in some way.

NARAYAN DESAI: Yeah, that's exactly right. And I think, this actually addresses one of the big issues that, practically, we see with SLO type of approaches, which is the calibration is super hard. So what this gives you is the ability to understand variance on a granular basis. And that variance is calibrated against the historical behavior of all of your individual workloads.

So you still have these questions of mixture effects and what do you want to promise, and the broader SLO decision-making framework. But this gives you a very different kind of data that allows you to know much more confidently that things are changing from the way that they have historically been.

STEVE MCGHEE: Cool.

NARAYAN DESAI: And I guess, at the risk of going a little bit down the rabbit hole of trying to address problems in the SLO framework. The need to choose a threshold ends up being a really substantial difficulty. Choosing a single threshold for a service is really hard because you have many workloads and a mixture, and the mixture is constantly shifting.

And then you have different customer requirements. And so, when you force yourself into a position that you need to pick a threshold, well, one, you will always pick wrong. And then the implications of choosing wrong are sort of random, depending on which customer you're talking about.

You may have customers that you've chosen pretty much correctly for. You may have customers that actually need a better level of performance and generally get it, but you can't see when you regress because your threshold is past the regression that's experienced by the customer, and so forth. So moving to a variance model that's historically calibrated actually addresses a lot of those issues.

FLORIAN RATHGEBER: And in particular because, I suppose, the really tricky thresholds to pick are performance and latency thresholds. Because if you have an availability SLO, then, I guess, you can make a more educated guess of, what do your users expect in terms of the reliability of the service, but the performance expectations and requirements of different customers, they have a much higher variance, making that tuning more tricky.

NARAYAN DESAI: So yes and no. So I would say, we started by doing this with performance because performance is clearly-- there are so many orders of magnitude, particularly when you look at analytical services that have everything from very, very short questions or queries, to extremely long-running queries that take minutes or hours to run.

You have so many orders of magnitude of difference in latency that you can't even pretend that you're getting reasonable results when you average these things together. But I think that it turns out that actually you see the exact same behavior in availability metrics. And I think that the fact that the variance is smaller has confused us into thinking that, actually, we don't have the problem there when we actually do.

So let's go back to that case that I was talking about a minute ago with cache hits. So imagine that you've got one RPC, there's a cache codepath, and there's a more complicated codepath that goes out to a database across the network. Which one of those codepaths do you think is going to be more reliable? Almost certainly, it's going to be the cache path. Because if you've got an entry in a cache, that means that you can pull it out of RAM and serve it right back, or roughly.

STEVE MCGHEE: Short hop.

NARAYAN DESAI: At least for that part of the code, you can do a very simple thing as opposed to doing a very complicated thing. It's a shortest number of steps. There's no network involved. There's no other service involved. That's just the thing that's going to naturally be more reliable.

So what this means is that what we have always done is we've always averaged together these different cohorts that exist within a larger service. And what will change over time is, your cache hit rate will change, and so your transient failure rate from just bad things happening in the network or in the database layer will become a little bit more prominent.

Now, generally, they won't be super prominent, and they will probably be below the noise floor because of all of these mixture effects. So we've kind of been able to ignore them. However, let me tell you a fun story about a customer. So several years ago, I was working on a service, and one of the engineers on the service recognized that there was one customer who had amazing, amazing release hygiene and reliability in their services.

So this was the Cloud Pub/Sub service. And if there was ever a backlog on this topic, we were broken. It wasn't the customer. So we did what any self-respecting SRE would do, and we started alerting on backlogs in that topic because it was a good indication that the service was broken.

Now let's go back to this thing that I was just saying about, different granular cohorts having different properties. So if you get past the codepath part of this, and you talk about which customer it is, you may have some customers where their expected success rate is pretty much 100%. And if you could calibrate from an availability perspective to that, you would know that when one erroneous event shows up for that customer, there's a problem.

Whereas if there's a customer workload that is experiencing normal levels of reliability, maybe you need a couple of events to recognize that there's a problem. Or if you have a customer workload and someone sets up a dev workload, and it's just constantly flapping and producing errors--

STEVE MCGHEE: Not helpful.

NARAYAN DESAI: --the likelihood that this thing is more broken does not increase with 1 or 10 or 100 errors, because it's always producing errors. And so you can think about ways that you would fit all of this together, but really, I think that the fundamental idea here is that, these systems are mixtures of workloads with mixtures of outcomes, and you need to model those explicitly.

And so I actually think that this is a major factor in availability, as well as performance. Though it's not quite as profound as it is in performance, but I think it's just as useful and available.

PAT SOMARU: That does kind of feed into something I've seen with SLOs before. So I was dealing with a service once, and it had great stats, very, very high uptime, but also a lot of complaints. And it was just a matter of breakdown.

Like you just break down the endpoints, and then after that, you eventually have to break down what the incoming objects deserialize into, and you have to find the different codepaths, and, yeah, you have to get that dimensionality bubbled up into your SLOs and surface to be able to actually see if there is a path that is a failing 100% of the time, but it's called very infrequently.

NARAYAN DESAI: You know, we call these needle-in-a-haystack kinds of problems. And one of the things that we see, particularly, with infrastructure products is there are so many interacting features that you may have a very, very small number of requests that represent any particular combination of them. But if the combination is broken, that's a thing we need to know about, and find and fix quickly. And so that's exactly on point.

STEVE MCGHEE: I mean, we all heard of microservices, and, sometimes, there's tiny little nano services in there that we don't even know about. They're just absolutely their own thing, and they're tiny, and you do need to care about them when they interact with the rest of the system. And that's hard to see.

PAT SOMARU: Thinking about that more, I was going to say, I think about caching. And if you want to talk about making things fun, what about services with an extremely high cache hit rate? How do you pinpoint when a problem was actually deployed? Yeah, but having that dimensionality present helps a ton there, totally.

NARAYAN DESAI: Yeah, and I think this is actually one of the places where we need a better set of tools. Historically, a lot of the systems that handle time series don't allow high cardinality breakdowns, and that's an issue for sure. I don't know if that means that you necessarily need a time series system that can handle high cardinality, but you definitely need a way to get high cardinality insights.

PAT SOMARU: So the thing I have always done is to ensure at least enough dimensionality is being emitted in the first place. It's never enough, but close enough. We know if it's cache hit, or miss, or just get the exceptions out and just to get enough observability that when you get robodialed, you have an idea of what might be going on.

NARAYAN DESAI: Yeah, and I think to the extent that's possible, that's the right thing to do. I worry that that often will blow right past the scalability limits from a cardinality perspective of a lot of these systems. To do a lot of the modeling that I've been talking about, what you end up with is these vectors that are relatively long.

And if you think about all the combinations that you see of them, you have millions to hundreds of millions of discrete cohorts that you're modeling. And that's what's required to really get the noise down for some kinds of services, at least.

FLORIAN RATHGEBER: So some of the challenges that you were describing, they seem to require a certain amount of scale, and also the methods require a certain amount of data. So for SREs that don't exactly work for one of the FAANGs, do these methods still apply? Should they still use them? Should they adapt them somehow? What's your recommendation for someone that doesn't deal with Google and Facebook scale questions, observability issues?

PAT SOMARU: I do have a little bit of insight there. At smaller places, the problems you're dealing with are similar, but different, kind of. So it's a bit more like, do we have any observability at all? Do we know why this is slow? Do we need to fish around in the code and throw in debug statements and get it to run or not?

I've seen some of the tooling be used, like some of the distributed observability tooling. My knowledge is probably a bit outdated, but I remember elastic APM was pretty nice forever ago. But it was a form of APM that I can hook into services and be like, oh, we have a call in the Rust microservice, we have a call in a Java microservice.

So just being able to see how different things are, like being able to see it at all, the interaction between all these containers and different systems, I would say that's a good first step for a smaller place. That's presuming really small, but I have something. Something is infinitely better than nothing. So I'd say that might be useful to some folks.

NARAYAN DESAI: I think, also, there's some core philosophical things that are pretty important here. So one, whatever organization you're in, the services that you're running have some sort of a business purpose. And, one, don't shy away from the important questions. If a thing doesn't look amenable to analysis, do your best to analyze it anyway. Right?

I think another aspect of this that's really important is it turns out that many of these systems are built because they have some sort of recurring business purpose to serve, and that means that the behavior will be more stable day over day and week over week than you would anticipate it being. Right?

So one of the big mistakes that we made at Google early on was we assumed, well, we're building a system-- this was an analytics system, that takes user code. So we have no idea what it's going to do. Customers can upload whatever code they want and produce very, very different kinds of systems or different kinds of workloads.

Fundamentally, it turns out that there aren't very many businesses that want to do something new and different every day. Many of them want to do the same thing reliably day over day. And this is a standard property that it turns out that a lot of workloads have, period. And so, it turns out that you will have consistent patterns. You will have workloads that should behave consistently over time.

You may or may not be able to get to the level of fidelity that I was talking about, but starting simple and starting to do even offline analysis of your critical workloads, even if they only run a couple of times a day, that will get you very, very far. And so I think that even if the implementation wouldn't necessarily work everywhere, I think that the general concepts would.

PAT SOMARU: Yeah, actually, I would say that a step further, but also that if you're smaller, you just have to dive deeper. You have less data. And not only that, but not that customers aren't important everywhere, but the chance that that one customer, that's the one. Your big deal is on the line right now. It kind of goes up. So you need to dive deep and just make sure that those paths are good. And by dive deep, I mean like `perf`, YourKit, whatever profiler tool it is, you need to use it.

And you need to have an understanding of your code, what it can do, and how it works, both the correctness, of course, but also performance. And I don't want to say cache as much as you can everywhere, but also that. So, yeah.

STEVE MCGHEE: So a theme that I'm gathering from the two big systems that each of you are responsible individually for or as a group, what you've described here, I feel like something that a lot of customers that I talk to get in trouble with is they'll install an observability tool of some kind, and it will come with dashboards.

And they'll be like, well, these are the metrics that are on the dashboard, so this is what we're going to interpret. And so this is like a common fallacy of just measuring what's available to you. Basically, measuring what's easy. Like the numbers are already here. They're already on a graph, so they must be important.

But what I hear you guys saying is, essentially, like, don't fall for it. Like, think about your business, think about your trends, think about your customers, think about what you actually wrote in the code. Like, maybe the things that you enumerated in the code aren't even being shown in the dashboard because it's just showing you CPU heat or whatever that doesn't matter.

And I see a lot of customers every day. They're like, but how do we build an SLO out of CPU heat? And I'm like, no, what are we doing? So it's a really slippery slope to just be attracted to the metric that's already in front of you. Or that it doesn't even have to be a metric, like, the indicator, whatever it is. So does that ring a bell? Like, does that sound like a theme that you guys are coming across?

PAT SOMARU: Absolutely. I actually have two pretty good examples of just that.

STEVE MCGHEE: Great.

PAT SOMARU: One is a service, a service for a while back where it was misusing the verbs, the HTTP verbs.

STEVE MCGHEE: Oh, yeah.

PAT SOMARU: And yeah, the built-in dashboards, a request would go out, some computation would be done, and it would just poll until the result came back. So you could get the most misleading data in the world. I mean, that's one presumption already. Are the verbs being used how they ought to be?

STEVE MCGHEE: Yeah. Code 200 failure, right? That kind of stuff. You see it all the time.

PAT SOMARU: Exactly, yeah. Code 200 failure, or code 200, but no data, and we're just going to poll forever until we have data. And then it works. It's great. It's incredibly low-response times. Yeah.

STEVE MCGHEE: The average is looking wonderful. Yeah.

FLORIAN RATHGEBER: Yeah, surely when you get a 200, surely you want to inspect your response payload to figure out whether it's actually a success or a failure.

STEVE MCGHEE: You think?

PAT SOMARU: And even on top of that, that can compound in systems to the point where it's like, you have interfaces that are actually built upon using things the wrong way. And then you try and introduce observability.

And then you have to start looking at a system that's observing a system and translating incorrect-- your unique usage of things into more standard usages of things, and then, wire that up. So, yeah, you have to just understand it in depth, and look at all components as directly as you can, as possible.

NARAYAN DESAI: So one thing that I've seen from the perspective of trying to use more sophisticated analytics is that, everybody loves really robust insights that can be used in lots of different situations. The problem is, those don't actually exist. And whenever you do that, you're going to get surprised, and you're going to get surprised in weird ways.

And so one of the things that I think that we need to do as a profession is really teach our engineers how to ask more nuanced questions because it turns out that, going back to the example that I was talking about earlier with cache hit rate, for a lot of systems, changes in cache hit rate are really meaningful.

So the fact that the cache hit rate is changing you care about, but you also care about if the underlying performance of the codepaths or reliability of the codepaths is changing. And so I think that what this is going to cause us to do over time is to be asking and answering more and more questions that are very specific, narrow questions, because you care about this thing, and you care about that thing, and you care about the other thing.

And at the end of the day, one of the things that's kind of fascinating about the reliability of systems is, there is this neverending list of things that customers care about, and we haven't articulated them, and we never will articulate all of the properties of a system that our customers care about. We'll get the big ones.

And, you know, we've gotten the big ones in many cases. But there's always more work to do. And as you talk about subtle feature breakage, or you talk about these more complicated properties of the system, we actually need to articulate, this data will answer the following questions for you, and it won't answer these other questions for you. Because I think there is a tendency to go reaching for the dashboard that you have.

PAT SOMARU: Oh. So that one sentence right there, this data will only answer this question. That is a guarantee that that data is going to be used to answer other questions.

STEVE MCGHEE: Yeah.

NARAYAN DESAI: Yeah, absolutely.

STEVE MCGHEE: You can definitely hold a graph wrong, that's for sure.

PAT SOMARU: I feel like that might be one of the biggest open questions. You have this data, and how do you keep it that way? I don't know. But I do know more is better. More data is better because if you understand what it is you have and how to use it, you can gain some insights.

STEVE MCGHEE: OK, so we've talked about reliability and general system speed. But another thing that actually matters is, like, are we getting as much out of the system as we expect, and maybe, that we're paying for. Especially in the post ZIRP era, performance really matters, right? Are we actually spending money on just generating heat, or are we actually generating revenue or something like that? Is that something that SREs focus on, as well? Is that a thing that you guys look into?

PAT SOMARU: Yeah, so I've done that a bit, good bit. And the context of smaller companies, I would say, that's just pull out your profiler and try and understand what's going on. Because, often, the simplest, and smallest, most innocent of changes can cause something huge. Like, a cache starts storing serialized strings, storing data serialized strings instead of storing data's integers.

And then all of a sudden, you have, let's say 40 minute refresh times versus about a minute or 40 minute load times, versus a minute to initially load a program through there. And then there's also at scale, where it's like, there's large numbers attached to minor changes in performance, and even changes across hardware.

If you look at the performance of chips, you can find sometimes across generations of generation how a hash function performs or regress. And that's a wild thing when you see it. It's like, whoa, this test must be broken. This is a very big difference. Then all of a sudden, you look back and, oh, the chips are different, and it performs--

STEVE MCGHEE: Differently.

PAT SOMARU: Yeah, that definitely that. Also, that the-- I guess the hardware is being used the right way. So like a project I've been working on a bit recently. I'm working on this `sched_ext`, and that's a pretty cool thing. But it's, like, be able to get the Linux scheduler to do-- or have a scheduler on Linux that can do what best suits your workload. Yeah, because there are different characteristics. Based on the characteristics of the workload, some things matter more for some workloads and others.

If you have GPUs and multiple sockets, you want to make sure that the CPU most local to your GPU is handling the processing for that GPU. Just keeping things the right place.

STEVE MCGHEE: Yeah, a refrain I've heard many times inside of Google-- and I'm sure it's the same in other giant companies-- is that a small percentage of a very large number is still a very large number.

PAT SOMARU: Yes.

STEVE MCGHEE: And so like taking advantage of the scale and just being, like, as long as we multiply this tiny little gain by like a bazillion, it'll be great. So finding those is always gold, I think. Narayan, how about you? Any good performance suggestions or tales?

NARAYAN DESAI: Well, so we've tried doing some interesting stuff with performance analytics. So one of the things that we think about a lot is that experts are great, but it's really hard to train an expert. And so we really need analytics that will tell you about the things that you should go and send your experts to look at.

And so we tried to build an interesting prototype where we tried to build a profile over time of the busyness of tasks and the amount of time they were able to do real work as opposed to being blocked on something else. And so what that basically meant was that for a service, you could basically build a histogram or a CDF of the amount of time that the CPU spent waiting.

And for a properly provisioned task, this would start low when everything was really uncontended, and then level out at some level, and stay level all the way to be 100, presuming you never got saturated. And if you were under-provisioned, this would basically take off at some earlier point in the CDF. Well, if you were properly provisioned or overprovisioned, you would never tail off at any point, even up to 100, right?

And so the thing that we were looking at there was, could we try to manage this? Manage the band so that you could get some confidence that a service was properly provisioned? So this isn't so much in terms of code performance. It's more a question of how many resources do you throw at a particular workload.

STEVE MCGHEE: Autoscaling problem.

NARAYAN DESAI: It's kind of like an autoscaling problem, but a lot of times, this could potentially be a different metric for autoscaling. And so this ended up being an interesting sort of thing from the perspective of trying to understand when an application's performance had changed for some reason, and you should look at it, right?

So that case that you were talking about, Pat, where you started storing data in a different way and things got a whole lot more idle because you're waiting for more, or that's a time you want to be able to detect.

PAT SOMARU: Yeah. Yeah, I mean, what's wild is what is it? I think in that case, it actually looked like things a lot more was happening. I think it coincided with some other logic rollout, and it's like, whoa, the computer's doing so much work now. That work was a no op. That's how it gets interesting. It's idle, idle can mean very busy.

STEVE MCGHEE: Yeah. We had a system one time that spent most of its time just context switching because all it was doing was managing connections between way too many backends. During the normal time, the thing seemed fine. But as soon as we hit any kind of load, it just fell apart.

We just went, wait, what? And it turned out, it was one parameter in one config file that was just set to too many backend connections. And change that one thing, and everything starts working again. So systems under load can be drastically different from systems that semi-idle.

But having insight into your spending 80% of your time switching between threads, even at low levels, that would have unlocked that. But no one thought to look at that until it was too late and until it was breaking under pressure. So I get it.

Thank you both for your time. This has been awesome. Before we go, if you would like, you can tell our audience, your fans, where they can hear more from you on the internet. If you have some sort of blog or other kind of presence in the world, maybe you go to conferences, maybe you, I don't know, have a WordPress, or I don't know if people still have those things anymore, but how can people find you on the internet?

STEVE MCGHEE: Social media is, what's that? Yeah, I've heard of it.

NARAYAN DESAI: I am, actually, largely, a social media hermit these days. I have a bunch of inactive accounts.

STEVE MCGHEE: Fair. Yeah, I get it. Pat, how about you?

PAT SOMARU: I like listening more than I like talking, so I tend to just be quiet.

STEVE MCGHEE: All right. More podcasts for Pat. We get it. Cool, man. All right. Well, thank you both for hanging out with us today. Thanks to Florian for jumping in as a special guest host, as well. And as always, may the pager be silent and the queries flow. Thank you all. See you next time.

NARYAN DESAI: Thanks

PAT SOMARU: Thanks. Bye.

—

[JAVI BELTRAN, "TELEBOT"]

JORDAN GREENBERG: You've been listening to Podcast, Google's podcast on site reliability engineering.

Visit us on the web at sre.google, where you can find papers, workshops, videos, and more about SRE. This season's host is Steve McGhee with contributions from Jordan Greenberg and Florian Rathgeber. The podcast is produced by Paul Guglielmino, Sunny Hsiao, and Salim Virji.

The podcast theme is “Telebot” by Javi Beltran.

Special thanks to MP English and Jenn Petoff.