EPISODE 1205

[INTRODUCTION]

[00:00:00] JM: Serverless computing refers to an architectural pattern where server-side code is run on-demand by cloud providers who also handles server resource allocation and operations. Of course, there is a server involved on the provider's side, but administrative functions to manage that server such as capacity planning, configuration or management of containers are handled behind the scenes allowing the application developers to focus on business logic. This makes for highly elastic and scalable systems and can reduce development, testing at iteration time due to reduced overhead. Functions as a service platforms are a model of serverless computing where services are decomposed into modular functions and deployed to a serverless platform. These functions are executed only when called and are typically stateless. Despite the benefits of elasticity and modularity that FaaS offers, it has drawbacks as well. Taking disaggregation of functionality to an extreme means the behavior that formerly required a method call now requires a network call to another function increasing latency and making larger scale operations inefficient.

Johann Schleier Smith is an entrepreneur and engineer who is a frequent contributor to work at Berkeley. He's a researcher and he joins the show for a great conversation about serverless and his opinion on the future of computing.

[INTERVIEW]

[00:01:24] JM: Johann, welcome to the show.

[00:01:26] JSS: Jeff, super excited to be here.

[00:01:28] JM: You have been looking at serverless computing from the vantage point of Berkeley. And I've talked to a number of other people from Berkeley about serverless. I've talked to Ion Stoica and Vikram Sreekanti. From your point of view, why has Berkeley taken an interest in serverless computing?

[00:01:51] JSS: Well, Berkeley has a long history of prominent research in computer science and systems in particular. Lots of really cutting edge work was done here. And I think the faculty are always looking for what's that next thing that's coming down the pipe and can we be on top of and ideally ahead of that trend? And in the context of serverless computing, this is something that we latched on to people at Berkeley. I wasn't actually the first one myself. Ion Stoica, Eric Jonas published the PyWren work back in – So Ion Stoica and Eric Jonas, they published the PyWren work back in 2016, 2017 where they were really saying, "Wow! Serverless allows us this access to super computer scale resources for basically anyone."

So I think that people kind of latched on to, "Hey, there's something new. There's something really different that's happening in the cloud, and we should really pay attention and we should try to understand what the implications of this new technology are."

[00:03:03] JM: And what does serverless make easier? What does it make more complicated? What are the tradeoffs in using serverless?

[00:03:12] JSS: From our perspective, serverless computing is really about making life easier for programmers. That's the big change. Now it makes a number of changes. So it certainly makes life easier for operators as well, in some cases, even completely removing the need for certain system administration responsibilities. So everything that's complicated about servers. And by that we mean things like setting them up. Making sure that they are patched for security. Making sure that when they fail, the application is responding in the correct way so it'll continue to deliver service. All of these concerns go away. They get handed to the cloud provider. The cloud provider has ways of automating them away so that for them it's also much, much easier to manage so they can manage for many, many companies at scale.

So the programmer also has this ability to basically write code in their favorite programming language, upload it to the cloud and then it just runs and they don't have to worry about it anymore. And that is, I think in many ways, fulfilling kind of this promise of the cloud to give you that effortless access to scale. So the downside of that is that you do have to change how you program a little bit. So I think that Lambda was successful because it allowed you to bring along your existing libraries. It allowed you to bring along your existing languages. So there's a fair degree of continuity. On the other hand, if you really are going to make programming simpler, you're going to be writing simpler programs, and that means that you're probably going to be rewriting your programs at the same time. So you do have to learn to think a little bit differently.

[00:05:08] JM: You have to think differently. So in what ways does serverless change architecture of applications?

[00:05:16] JSS: One of the big shifts that we've seen with serverless is the adoption of more event-driven and cue-based architectures. The serverless framework is one of the technologies that has helped to make this transition easier for folks. And what we also see is we see a lot more use of high-level services. So, for example, machine learning service that is offered in in a serverless way. Or the queue services, image processing, image resize services, email services, all these sorts of higher level constructs that you end up weaving together as opposed to sort of bringing your own, bring your own open source software or rolling some of your own code. And so I think this is a shift. I think that the way that people design and architect serverless applications isn't fully settled by any means. I think we're going to see continued evolve evolution in that space. I think there's a lot of room for innovation there.

[00:06:28] JM: Is this re-architecture process around serverless, do you see anything any anti-patterns? Is there anything that we're doing wrong in these still relatively early days of architecting serverless applications?

[00:06:46] JSS: Well, certainly there are some serverless anti-patterns. One of them is to take something that was really designed as a serverfull application, as an application, say, that maintains state and to try to just cram it into a cloud function. For example, if you have a codebase that's very large and monolithic, just running that whole codebase in a cloud function before you've broken it up into individual pieces in a more microservices type of architecture is likely to produce a problem. So certainly, one anti-pattern is that, is just this sort of wholesale port.

[00:07:32] JM: Fair enough. Well, if we talk about the shortcomings of serverless applications today, are there some kinds of applications that simply do not work on serverless?

[00:07:43] JSS: Yeah, there are plenty of them. One example that I just referred to that I thought would be very straightforward is I was at Berkeley and I thought for grading student homework submissions for a database course we could use serverless. I thought this would be great because we would get isolation. We could scale up and scale down as student submissions came in. Most of them right before the deadline. And I spent a few weeks working on this and ended up failing and the reason why was sort of twofold. So the one was that the isolation that I wanted, I wanted to isolate the student code from the auto-grader code, and the natural way to do that would have been using containers. But because of some of the restrictions of the serverless environment, that actually wasn't possible to get that isolation. So that was a drawback.

Another problem that we had, which has been largely addressed now by some of the very most recent releases of AWS Lambda is simply the size limit. So being able to install, say, a Postgres instance that you might need to run for grading a student submission on AWS Lambda just simply wouldn't fit. Now they accept Docker images. They've raised that limit from a few hundred megabytes up to 10 gigabytes. So it's really possible to do a lot more with serverless.

Another limitation that has fallen recently in Lambda has been the billing increment. So the minimum billing increment used to be 100 milliseconds, which you think is a fairly short period of time. But if your request is only one millisecond or if it's 10 milliseconds even, you're going to be overpaying for your time by a factor of 10. So by going down to a 1 millisecond billing increment, they really opened it up to a much broader range of applications that were cost prohibitive not for any real fundamental reason, for a business reason really, and that's been corrected.

Another sort of class of applications that don't work well in serverless today with a current stateless model for compute are any applications that are state intensive, and these can vary from analytics applications need to move large amounts of state between steps in a pipeline to more sort of OLTP type applications that need to do a lot of intensive coordination on state. Right now with serverless, there's no way to name an instance, to connect to an instance, to talk to an instance, to run traditional distributed systems protocols like Paxos to get agreement between instances on the outcome of some computation. And I think there are good reasons for that, which is to say that the model where you have a function and it starts and it does some work and then it ends, that's a very good model for scaling it. Has very good operating characteristics. It makes possible many of the advantages of serverless. At the same time if we are going to be able to do all of the things that we do with servers, we need some other pieces, and I think that the form in which those other pieces get provided is still to be determined.

[00:11:12] JM: So you mentioned that there was this problem in one of the applications you were working on where you could not properly manage or isolate the contents of a serverless application. Can you talk about the downsides of the isolation properties of serverless?

[00:11:37] JSS: In order to provide the benefits of serverless, which include this elastic scalability, it really requires making a use of the hardware on a multi-tenant basis. And frankly, that's tricky. It's being done safely today. However, it's not obvious that it should be safe when you think about some of the vulnerabilities that we've seen over the past few years. Things like Spectre and Meltdown, and there are a whole number of other side channel attacks. It's pretty scary to say my code is going to be running with all kinds of other people's code.

And so what the cloud providers have done to address this is to implement very strong security boundaries between the different workloads. The Firecracker project from AWS is an example of this, and Mark Brooker has a great talk at the recent Reinvent where he goes into some of the details of the security isolation mechanisms.

I think that with some of these latest mechanisms we're going to see the operating system be able to get more flexibility in terms of what is permitted. But historically, it's been very locked down. Container-based isolation has been part of it, and nested containers are something that while possible don't necessarily work particularly easily. That is to say it's tricky to get them to work at the version of Lambda that I was working at. That was related to it.

I think also though that from a security perspective the designers of these serverless platforms have really asked, “Well, what are the capabilities that we really need to give people so that they can run their applications including all the libraries that they might want to bring along?” And what are some of the things that we should probably lock down and keep control so that it runs in kind of a well-defined way and that there's less to manage? And I think that the restrictions say on networking where I can't run my own network stack, that's an example of one type of restriction. Researchers, we get to have fun with things. We get to poke around to see what sorts of kernel modules might be available and so forth. A lot of that stuff is just locked down.

And the other thing that locking down some of these capabilities gives you is it gives the cloud provider the opportunity to optimize and to drop in their own implementations. And I can actually see this continuing. So for example, I can imagine a serverless runtime that says, “We only run pure Python code,” or let's say pure JavaScript code. In fact, actually this is what some of the platforms such as the Cloudflare workers platform demand. They take JavaScript or they take WebAssembly as well, which supports more languages. But because they've restricted you in a certain way, this now lets them do all kinds of optimizations because they know what they're working with. You can imagine them, for example, saying, “Well, we know that your workload is JavaScript, and we have a great new processor that is not an x86 processor. It's something else. It's a RISC-V process or an Arm processor. And we can see that your workload would benefit from running on that type of processor.” And then they can completely just shift you over to that and you wouldn't even know. So these are the types of advantages that you get from locking things down a bit.

[00:15:19] JM: Interesting. And do you think these kinds of newer serverless platforms that we might see in the future, can you envision them like as only being offered by the gigantic platforms like AWS? Or do you think like startups could offer domain-specific serverless platforms?

[00:15:39] JSS: I think there's a great opportunity for startups and serverless. I think we look at things like brev.dev, for example. They've really built a serverless platform that's just really, really focused on making that whole life cycle easy for developers which is something that we haven't necessarily seen as much from the large cloud providers that's come. It's just been a little bit slow in coming. They've been focused more on scale. I think there's been a huge amount of innovation actually even between comparing the large cloud providers in terms of how they do things on all number of axes, durable functions from Azure is a good example of that.

In terms of stateful serverless computing, some of the things that I think are pretty interesting have also been done by startups. So for example, you have Lambda Store, which is a Redis API serverless key value store. I haven't evaluated and really looked at it and tested it myself, but I look at that and I say, "Wow! That's something that really makes a lot of sense that somebody should be doing." And if it's done by a startup actually, in some ways that's better for the ecosystem because that suggests that that is a technology that's going to be available across cloud providers. The work that Tim Wagner has done with Vendia, and I know you've had him on your show as well, I think is a great example of how startups are doing out of the box things with serverless, and in that case blockchain, and linking things across clouds and across companies for data services.

So in particular, when you think about stateful serverless, I think there's a huge opportunity for startups. I think that if you simply want to provide function as a service better than a major cloud provider, I think that's going to be a challenging proposition just because they have so much resource behind that and have a track record of making those services better on a consistent basis.

[00:17:43] JM: What do we know about the implementation of functions as a service at gigantic cloud platforms? Like what do we know about how AWS Lambda is actually implemented?

[00:17:56] JSS: There are a number of things that we can measure. We can measure the startup times. We can measure how quickly they scale. So these are things that we can kind of learn from the outside. They have provided us now with container images and their APIs, their runtime API. So we have a sense of how things work sort of from the application perspective and how that gets wired in.

For a long time, AWS was rather tight-lipped about what they were doing, but they do have some patent filings that you can see that are from the early years from around the time when Lambda launched. And they walk through things like the life cycle of these environments and how they recycle the containers. How they do predictive modeling in order to try to figure out when they should leave something running versus when they should shut it down.

And so I don't know whether they've built everything that's described in those patents. Probably they've built some of it and probably they've built a number of other things that aren't described in the patents. But that's one place where we can learn things. There's also some work done, paper, 2018, USENIX paper peeking behind the curtains of serverless where the researchers, the academics, went in and really tried to reverse engineer what was happening by probing those environments and trying to get a sense for how the networking works. So we've learned some things that way, and that's certainly some work that I'd recommend for somebody who's really interested in the technology.

Firecracker from AWS is open source. So that is the platform that allows these lightweight virtual machines to be launched very, very quickly. And I'm hopeful that they will continue to be contributing to the open source ecosystem. Now they're not the only ones who've been doing so. The Azure Functions platform is largely or completely open source and has been from day one. So that's something that people can go and they can see how Microsoft does it. IBM has OpenWhisk. I'm not sure whether the open source project is exactly the same thing that they run in a production on their cloud, but I imagine it's similar. And Google has open sourced a number of pieces of their serverless infrastructure. So for example, gVisor provides isolation for cloud functions as well as for app engine, I believe. And they also have open sourced Knative, which is the serverless platform that can run in a number of settings. It runs on top of Kubernetes and it also runs as a hosted service in Google's Cloud.

[00:20:57] JM: And from what we know about how these functions as a service run, what's the state of the cold start problem? When I summon a Lambda function, for example, is it spinning up a brand new VM? Is it spinning up a brand new container? Is it able to take it from a warm pool? What do we know about the cold start problem?

[00:21:24] JSS: The cold start problem got a lot of attention early on. And I'm happy to say that I think that for a lot of practical purposes is something that people can either consider resolved or worked around sufficiently that they don't need to worry about it so much. But let me go into a little bit more detail on that. So what is the cold start problem? Well, in order to provide these secure execution environments, the cloud provider needs to create a VM for your workload, because that's really how you can guarantee that you're not going to be exposed to other clients, other tenants.

And so booting a VM traditionally means booting and operating system. Operating systems just simply aren't designed to boot up super-fast. It's not something that really matters. You're usually happy, or traditionally you'd be okay if a server booted up within a few minutes, because it's going to run for days. So what does it matter? And traditionally also the things that happen during boot up time involve things like probing for devices and figuring out whether you've upgraded the hardware and other things that have just no role in a serverless environment. You know what the hardware is and you want to get going as quickly as possible because you want to be able to have that ability to expand elastically. And similarly in order to keep costs low you want to have that ability to just shut things off and effectively power down.

And so what the cold start is really about is it's about that time that it takes. And to be clear, also, what's important about a cold start versus a warm start is that when you have a – Once you start it up, what you can do is you can just leave that function instance you. Can just leave it running so that it can do more than one request so you get to amortize your startup cost over many, many requests. So sort of two reasons why this startup time is becoming less of an issue, and they're actually both related to the technology that's in Firecracker.

So Firecracker makes it much, much faster to boot up the VMs in part because it sort of strips down that kernel so that it has just simply has a much faster boot time. And so the boot times are, instead of seconds, they come down to something like 100 milliseconds or so. And there are a number of other techniques. Some of these are in Firecrackers. Some of these are in research papers that are about making these boot up processes much faster. For example, one thing that you can do is once you have booted an image of a virtual machine, what you can actually do is you could just save those pages essentially, save a state of the memory. And then when you need another one, you can simply clone that and you can use sort of copy and write semantics for that as well so that really you're just creating a new set of page tables to reference that underlying image.

And the SOC work from the University of Wisconsin Madison, that it was sort of some early work in this area. So that's the cold start that's related to the setup of the operating system and environment. There are other things that go into that as well, things like configuring the network in order for that virtual environment that's running the cloud function to be able to talk to the resources that it needs to and to have all the right security configuration. So that's another area where work has gone into amortizing those costs and making it so that you can set up the network once and then use that set up, that configuration, those privileges with a number of cloud functions that are similarly configured.

So then the next part of the startup time would be starting up the runtime environment. And this is one where there are some solutions, again, involving snapshots. So for example, if you have a Python program that imports a number of modules. If you try to import something like TensorFlow, it could be a second or two. And so what you'd like to do is you'd like to take a snapshot of the state of the memory after that has loaded. And then if you need to create another instance, then you would like to just simply use that memory snapshot rather than redoing the whole loading process again of software that wasn't necessarily designed for a serverless environment.

Now, the third part of start times, cold start times in particular, is any sort of application code that you may be wanting to load. And this one is one where it's a little bit trickier. So for example, if you have a model, a machine learning model that you want to serve using that TensorFlow that you just loaded. Depending on the size of that model, fetching that from object storage and initializing TensorFlow with that, that might take some period of time if it's a big model, if it's a model let's say several gigabytes. So that's going to be application dependent.

One of the things that is sort of a change that I might like to see to the serverless APIs is the ability to sort of have a user-provided initialization that can be run ahead of time so that you don't have a request that's actually waiting for that. That the cloud function runtime can – The cloud provider can go and say, “I think we're going to need a few more instances because of the way the workload is increasing, the way the requests are coming in. So let's go ahead and run them and let's actually provide this hook, a user-provided initialization hook to take care of some of that.” And I know that the internal serverless platform built by Netflix provided some of this capability. So they rolled their own kind of around the same time that AWS Lambda was coming out.

The final thing I'll say that of that is also a solution to cold start not of the application kind, but certainly of the language run time and operating system, is the provisioned capacity, which this is one that's a little bit tricky. So with AWS Lambda now, you can say I want you to guarantee that I will be able to have a concurrency of at least 200 or 2000 for this particular workload. I'm actually going to pay for that all the time. So in some ways, this is a little bit of a step backwards. I think that from the purity of the serverless model that's really pay-per-use. It's something that we would say is breaking with the model. On the other hand, it seems to be solving a problem for people and perhaps we can see it as sort of a stepping stone and something that we would hope eventually becomes unnecessary. But I think there's definitely still sort of an open question around how people might pay for the ability to burst on-demand, because the truth is, is that the cloud provider does have real costs associated with keeping idle capacity for people to sort of do arbitrary bursts. And I think there are still some things in the business model, in the pricing model, that maybe can get ironed out. Long answer to a simple question, but I hope that it's kind of a useful perspective.

[00:29:57] JM: Definitely. Two of the problems that Vikram talked about in my conversation – I think Ion also referred to these, and they're sort of related, but the problem of sharing state across AWS Lambda functions as well as the problem of I think what you said also is like doing some sort of consensus operation between the different Lambda functions. Can you explain why these problems of sharing state and coming to an agreement on things? Why is this particularly difficult with Lambda functions? I mean can't we just use Zookeeper or Redis or some kind of shared module to have the shared state that we need?

[00:30:53] JSS: One of the things that makes shared state particularly difficult with Lambda functions is that they are – it's in the name. They're stateless. That means that if their state that you care about that needs to be maintained, that state is going to have to live somewhere else outside of the Lambda function, at least in today's model. And of course this stateless paradigm has a long history. You can go back to running a web server 25 years ago and you would have had a similar model where you had a central repository of state and then a lot of stateless PHP or Perl scripts back in the day, these days any one of a number of languages.

And maybe I'll just say that managing state and managing state well, it's just inherently a difficult problem particularly at scale. And serverless has these fantastic scaling characteristics. And part of the way that it achieves those is that it just completely dodges the state problem and just makes it someone else's problem. Now of course we do have stateful serverless services. We think about things like object storage. We think about – Amazon S3, Google and Azure have their similar offerings. We think about things like the databases that are billed per use, that is to say per operation. So that is I think something like, for example, Azure CosmosDB, Google Cloud Data Store. On AWS, there's DynamoDB, which is often used together with cloud functions. And all of these, they work. The challenge is that if you have access, particularly if you have any sort of frequent or fine-grained access, you have a big performance penalty because of the constant round trips. And typically there is also a pretty significant cost associated with that as well. That is to say it just gets expensive. And again, I think this is not just that the cloud provider is putting extortionary pricing on. It's that that mechanism is just a heavyweight and expensive thing to run.

So this is definitely an area ripe for ongoing research. I know you talked to Vikram about the cloudburst system that I also collaborated on. And what we've done there that I think is an interesting step is to really bring that data in close to the compute. And so what we did was we integrated a caching layer with the cloud functions. And the nice thing about Cloudburst is that it has this causal consistency model, which is a relatively weak consistency model compared to, say, what you would get from Redis. And that puts a little bit more burden on the programmer, which is not really what we want to do in serverless computing, but we're working with the tradeoffs here. What it does allow is it does allow the local caches to be really always up to date within the consistency model and it allows changes at different caches to be merged together very nicely at the backend. So it really lets you run fast locally and then sort of slowly globally come to a consistent state, a state that's consistent with the semantics of essentially your global shared memory model.

So that contrasts to other approaches. So for example, I've been doing some work on file systems and looking to see how you can make something like a POSIX file system, which you could compare to, say, NFS or EFS, which is available in AWS Lambda now. How do you make that work with its strongly consistent semantics? How do you make that work in a context where you have clients that are really all over the place? Clients that are sort of coming and going all the time? And frankly, it's hard. There are some cases where we can accelerate things. So for example, if we know that a function is going to have read-only access to the file system, we can play some tricks to take a snapshot and have a locally cached snapshot while you have concurrent rights happening to other places and not needing to coordinate. But fundamentally these are hard problems.

If you have code – And this is actually whether it's serverless or not serverless. If you have a model where you have one common view of state and you have programs that are manipulating that state that are not in the same place, just because of the speed of light, basically, it makes it a hard problem to have those programs make progress quickly. When you look at the time for light to travel across the data centers, it’s many, many clock cycles of a modern processor.

[00:36:45] JM: So you, as you mentioned, collaborated with Vikram on Cloudburst. Can you share any other learnings from that collaboration?

[00:36:58] JSS: One of the things that we did in Cloudburst is we decided to build the serverless stack, the whole serverless stack from the ground up. And I think this is one line of research that contrasts with some other lines of research that are either about understanding how the current cloud provider offerings work or taking as a given what the current cloud providers are doing and building on top of that. Now the advantage of those approaches is that if you come up with something, it can likely go into industrial production and use relatively quickly. At the same time, it also limits the types of questions that you can ask.

And so I think one of the big takeaways for us, at least for research systems, it's really a good idea to build it from the ground up or perhaps using some other open source components, but to really have a hackable platform where you can go and try things and really try and mix and match these components in different ways. And I think that without having that ability to integrate the cloud function runtime with the storage we wouldn't have had that with the ability to do this type of research. I think there are also opportunities in integrating the scheduling with the storage, with the function runtime.

I think some of the things that are also particularly interesting are the potential for doing language level optimizations. That is to say if the cloud runtime knows that it's working with a JavaScript program or knows that it's working with a Python program and it sees those reads and writes to the storage and it understands the storage, it opens up just this whole another world of optimizations for pulling things apart. Running even part of a function on one place in the data center, one, ultimately a server with today's data center architectures, another part of that same function somewhere else just to optimize locality and the access to the data.

So I think that that's certainly a takeaway for me is that you can do exciting stuff when you just start from the basic premise, which is to say people are going to write some code. That code should be code that is sort of as simple and easy to understand as possible. Code that is related to the business problem really that you're trying to solve that doesn't have really that much information in it about what does the underlying machine look like. Does it have servers? What is the sort of architecture of those servers and how they're wired together? Are they in racks? Do they have a whole lot of cores or only a few cores? Just forget about all of that stuff. Just write my program in a friendly language and then figure out how to run it. And so I think if we take that perspective on the research, it can really take us a long way.

And maybe if you'll kind of an indulge me, I'll just share a little bit of additional perspective here on why it really is that taking servers out of the programming model really goes straight to the heart of the complexity and what makes cloud programming difficult. And the analogy that I use here, and really I think it's more than analogy, it's a fairly direct connection is to Fred Brooks and the work that he did in his seminal article, *No Silver Bullet*, where he really breaks down the complexity of programming and he talks about the essential complexity, that is that complexity which is inherent in whatever it is that you're doing. Like the thing that I'm doing is actually really complicated, which is why I have a complicated program, because it's a finance trading application or a very complex set of business rules for insurance or something like that, right? That's complicated. That's complexity. That's never going away.

And then you have this other complexity which comes from your machine. And if you're writing in the 1970s or 1980s and you want to get high performance, sort of the analog of high scale today or taking advantage of cloud today, what you really probably had to do is to write in assembly language, initially at least. And then what changed was of course that the high-level languages caught up and they were able to do things like register allocation. So figure out which data should go in which register memory layouts. Figure out that we want to lay out these – If you have these accesses to an array, it's going to be in this part of memory and take that away from the programmer worrying about it.

So all of this complexity that used to be in programs, and certainly if you look at assembly language programs today, and there's still a reason people still do it today for very specific things. But all that complexity, that's accidental complexity. And really you want to take that away. Now, in the cloud, servers, to me, they represent basically pure accidental complexity. Servers have nothing at all to do with any business problem that anybody is solving. And so a server to me in a programming or in a programming model is it’s basically the same thing as a register, right? It's something that, yeah, it's there, but you really never want to think about what's going in what register. It's the same thing with what's going on what server or where is that server and what is the locality.

And I think what that hints at that I think is really interesting and that is now becoming a focus of some of the work being done at Berkeley including a lot of the work by Joe Hellerstein, whom I have worked with for a number of years. He’s my faculty advisor there. As well as a number of other faculty now, Alvin Cheung and Natasha Crooks, are driving this direction on new directions in cloud programming. They had a recent paper at CIDR, which is a database conference. And really the idea there I think could connect to this notion, which is to say that we need new ways of programming the cloud. And my take on all this is that the server being part of the programming model is sort of exactly the problem. And just like we've had, what excites me about a lot of this work that folks at Berkeley are now starting to embark on is that it really recognizes that it's language techniques that solved it for the registers, in the assembly language program, giving us these high-level languages. That those language techniques are going to likely be a big part of the future of serverless. Now there are a lot of other things that are happening at serverless that are very exciting too. These are perhaps some of the most far-out ones, but I think it's really fun to be around all of this work and it gives me really a lot of optimism and confidence that we will see much, much easier ways of accessing the full power of cloud computing.

[00:45:05] JM: What are your predictions for how serverless changes in the next five years?

[00:45:12] JSS: Five years is an interesting time scale. It's certainly a time scale where anything that's going to come out and be in production is probably in research already today and people are already thinking about it. I think that the integration of state with serverless runtimes, I think we're going to see that. I think we're going to see stateful solutions, whether that's something like a Lambda Store or something else. Perhaps more like Cloudburst that are offered as products that people are really using on a day-to-day basis. Perhaps they'll have interfaces that are more capable than key value stores. Perhaps they'll even be running SQL. I think that these are things that we're going to see.

Another thing that I think we're going to see is we're going to see that the applications, that work in serverless, continue to expand. And those no-go zones, those are really going to come down. They've already come down tremendously because of things like the cold start times, because of things like the container images being supported up to many gigabytes for the co-deployment sizes, run times getting longer being built on shorter intervals. Al these things – Costs too I think are going to be – They're already competitive for actually significant cost savings for almost all applications. But there were some exceptions. I think we're going to start seeing those things fall as well. So I think that the excuse for saying we can't do a serverless. I think those are largely going to be going away. I think we're going to be seeing companies really being serverless-first.

Another area that I think we're going to see for sure is we're going to see more hardware supported in serverless right now, it's x86 architectures. We're going to see both different instruction sets as well as support for GPUs. That's definitely coming in some form or another. We have folks at Berkeley who are working on that, and my guess is that those in industry, probably cloud providers working on it, and we just haven't heard about it yet.

And then I think sort of towards the end of that period, we're certainly going to be seeing in research too really compelling ways to write programs that run at scale with very little of that accidental complexity that is today associated with scale. I'm not quite sure if we're going to have that widely adopted. I don't expect it to be widely adopted, but I expect us to really have clarity on that path. S so those are a few predictions. There are others that I could add as well. I think that the machine learning techniques are going to have a big role in optimizing serverless computing. And I think that we will be well on our way to a world where serverless is the default way that people use the cloud. I think the writing will be on the wall within five years. I think at this point, that is perhaps still a controversial opinion, but I don't think it will be in five years' time.

[00:48:43] JM: All right. Well, we're nearing the end of our time. I just want to ask, as I do of all the Berkeley people, how does your perspective compare to people in industry? How does the academic perspective compare to people in industry? Or is it the same for this topic?

[00:49:03] JSS: I've spent time in both academia and industry, and I think that's definitely been kind of a privilege and a great benefit to have that. When I talk to the academics, unfortunately, when I'm wearing my industry hat, there's a great deal of work or at least ideas where kind of there's this initial reaction, which I think is the industry reaction too, which is that would just never work. It's interesting and it's not practical. There's actually probably less of that at Berkeley because Berkeley has really significant industry collaborations than other universities. But really understanding the pain points of developers is definitely a challenge that we have in academia. And it's a place where there's an opportunity for us to do a lot better in having a dialogue to make sure that we're working on the right problems.

In terms of what blind spot in industry might be, I think that a lot of developers rightly so are very focused on the problems that they have today. And the problems that they have today, in many cases those are dominated by thinking about the problems that they have with their servers. How do they keep their servers up? How do they keep them reliable? How do they manage their application so that it's cut up right so that it runs on servers? I think the big leap that I would encourage people to start taking is how do you think about things if you completely throw that away, right? If you completely throw away? Like you've completely thrown away thinking about register scheduling. Completely throw away thinking about locality in this big data center, and how would things work if the runtime environment? Call it the operating system, the data center, was smart enough to just put all your data where it needed to be for things to be fast so that you could just write code that's simple and make sense.

So certainly, on the vision side, I know there are people in industry who will have these types of visions, for sure, and fully appreciate them. At the same time, I think that a lot of people, again, with good reason, are pretty focused on what they do today.

[00:51:39] JM: Johann, thanks for coming the show. It's been a real pleasure talking to you.

[00:51:42] JSS: I'm very happy to do so, Jeff. You're welcome, and thank you.

[END]