EPISODE 1080

[INTRODUCTION]

[00:00:00] JM: Server infrastructure traditionally consists of monolithic servers containing all of the necessary hardware to run a computer. These different hardware components are located next to each other and do not need to communicate over a network boundary to connect the CPU and memory. LegoOS is a model for disaggregated network-attached hardware. LegoOS disseminates the traditional operating system functionalities into loosely coupled hardware and software components. By disaggregating the data center infrastructure, the overall resource usage and failure rate of server infrastructure can be improved.

Yiying Zhang is an assistant professor of computer science at UCSD. Her research focuses on operating systems, distributed systems and data center networking. She joins the show to discuss her work and its implications for data centers and infrastructure. If you have an idea for a show, you can write about it on softwaredaily.com. We are always looking for new show ideas and you can also post about your company. You can post about the projects that you're working on. You can post jobs that are related to those companies or projects, and we're going to look for the best ideas and we want to cover things that are interesting, as well as share the job postings that people post that are particularly useful to the audience. You can check all that out at softwaredaily.com.

Thanks for listening.

[SPONSOR MESSAGE]

[00:01:39] JM: JFrog Container Registry is a comprehensive registry that supports Docker containers and Helm chart repositories for your Kubernetes deployments. It supports not only local storage for your artifacts, but also proxying remote registries and repositories and virtual repositories to simplify configuration.

Use any number of Docker registries over the same backend providing quality gates and promotion between those different environments. Use JFrog Container Registry to search the

custom metadata of the repositories. You can find out more about JFrog Container Registry by visiting softwareengineeringdaily.com/jfrog. That's softwareengineering.com/jfrog.

[INTERVIEW]

[00:02:33] JM: Yiying Zhang, welcome to the show.

[00:02:36] YZ: Thank you for inviting me.

[00:02:39] JM: You're an assistant professor at UCSD, and a specific focus of yours is the idea of breaking up monolithic, tightly coupled server infrastructure. Can you explain how server infrastructure is monolithic? What does that mean?

[00:02:58] YZ: It means the servers that I use currently in data centers, fundamentally, there's no difference from like a personal computer. In a computer, you have everything that is needed for your computation. That usually includes a CPU, some memory, some storage device, like a disk, and then there's a network card that connects through the outside. Everything is spent out on their motherboard and that's packaged into a server. That's what I mean by a monolithic server model.

[00:03:31] JM: What would be the alternative to that?

[00:03:33] YZ: The alternative is to break these devices apart and you no longer have a server if there's no motherboards. Every device is directly connects to the network. Your CPU directly connects to the network, and then the memory directly connects to network. Storage directly connects to network.

[00:03:52] JM: When we talk about the CPU, for example, being connected to the network versus connected directly to the other components, what exactly does that mean? If you have these different components that are communicating with each other in the classic monolithic server model, how does that compare to a model in which everything is going over a network?

[00:04:18] YZ: A classic model, what you can access before you go to network is just what is any local server. Like CPU can talk to the local servers, memory can talk to the local server storage over the motherboard, but you cannot – To talk to like other machines memory storage, you have to go across network, and then all the software, they are built in a way that it assumes like [inaudible 00:04:49] is local before you go to distributed. That's the traditional way. If you're breaking apart, then there's no notion of what is local and what is remote. To the CPU, if it directly connects, everything directly connects to the same network. CPU can access any memory device in a same way. There's no notion of like I'm accessing the memory that's local through the motherboard, or I'm accessing another machine through the general network. Everything would just be like going through the whole general purpose network. The benefit of doing that is like you get your allocated resource anywhere that is available.

[inaudible 00:05:29] if you have like a thousand memory devices connect to the network, a thousand CPU processors connect to the network, when you're running an application, your code can be executing one of the CPU and then the memory could be – Some of the memory could be on memory one. Some could be hundred. Some could be memory thousand, and there's no – Conceptually, there's no difference of how you access these different memory, and the suddenly you can access a lot more of them.

[00:05:59] JM: Obviously, that sounds advantageous. If I can just have this disaggregated computer, then I could easily bolt on new memory modules and new CPUs or TPUs. I mean, the work that you've done is LegoOS and you can imagine just these Lego building blocks fitting together more harmoniously than a monolithic system. I'd really like to get a better understanding of what that means as far as the connectivity, because the monolithic model, I can imagine, there are advantages to this direct connection. I don't know much about hardware interfaces or the interfaces between these different components and how that compares to what you get if you network all these things together. Can you tell me more about that?

[00:06:54] YZ: Yeah, that's a very good question. What I described is what I call a future version, where everything could connect to the same network and we can assume you can have the same performance whenever you access any component in this network. But that's very different from today's worth, where you have a local motherboard, local bus that is much faster than the general purpose network that connects different machines, different servers. Also, the

interface would be different. This definitely would be a lot of changes before we reach that machine.

But the good news is that like data center networks, they are becoming much faster these days and they are improving at the rate that is faster than CPU. The frequency of network in data center, they increase much faster than CPU frequency, which means, at some point, we'll have a network being fast enough so that like you don't really have to care what's the other end that you're talking to. It's always going through the same very fast network, but there's obviously a lot of problems that need to be solved, and maybe we will not get you that machine and maybe there will still be some hierarchy.

The current model is you have a hierarchy of basically network. You can think of like local bus as the fastest network, but you only get a very small scale. You can only access what local in your server. Then the next level would be like a rack, which is slower than your motherboard, but still quite fast and you get through like a hundred, like some tens of machines. The next level would be slower, you got more machines. There's some hierarchy in the network if we want to access a lot.

This hierarchy may still exist, but like in general, the network speed is improving much faster than the local bus speed and the CPU frequency, and that's why at some point we would be able to both have good performance and have a reasonable amount of scalability.

[00:09:11] JM: The model that you're suggesting, this disaggregated server infrastructure, we had a show recently with AWS and they have these nitro security chips. They have these dedicated I/O chips. It seems like they're doing work that is related to this. I mean, what they're doing as I understand is they're making custom ASICs that dedicate a hardware device to fulfilling certain operations or certain classes of operations. How does that relate to what you're proposing?

[00:09:55] YZ: I think their model is like you will – There's a need, and the need comes from like CPU [inaudible 00:10:02] scaling and more slower. They are slowing down, and that's why people are looking at other type of processor, other type of hardware. Once you do that, you have a lot of heterogeneous types of hardware.

The traditional servers, they're really designed – They're not really designed for handling heterogeneous hardware. They were designed for like a CPU, a memory, a storage, like a standard server architecture. Once you have more heterogeneity, then you need to think about how to connect these heterogeneous devices. Should we still use the standard motherboard?

The other thing is you may have the need to add and remove devices. If we are still have the traditional server model, then the problem – I think this is also a problem that real data centers face, is that they have the need to add new hardware, new types of hardware. But before that, they have to first think, "Oh, do I have enough PCI slots in my server? Will this bus standard work with this new hardware?"

As this need becomes like more and more pressing, I think it's time to rethink how we connect these different devices and think about a way that's more flexible. If everything could have like a standard way of connecting to network and when you want to add, you don't need to think about if I have a slot in my server, you just add them to the general network and then you can just go. But of course, you need a lot of software support, network support to make this happen. I think the whole idea of disaggregation also fits this general trend of having more heterogeneous hardware devices.

[00:11:51] JM: The way that things are done today is if I have an application that needs to scale, I can just create these different components in a cloud provider. I can spin up a dedicated instance for my machine learning. That instance, maybe it has a TPU, a tensor processing unit on it. If I need a bunch of different instances of microservices, then I spin up another certain kind of instance for those different microservices. If I need a caching server, maybe I spin up an instance of a computer that has a lot of memory on it. This works pretty well. Why is there anything wrong with this model of the current way of pulling resources and distributing resources and allocating resources?

[00:12:48] YZ: I think this can be also from two sides. The first side is from the cloud providers' field. For the cloud providers, they have to allocate resources for a virtual machine for a container, and the current way you have to run the virtual machine all container inside one server. Like a virtual machine, you cannot like span them across multiple servers.

What could end up happening is that you have like some machine that has used all of its CPU but still have a lot of memory left, and another machine has all its memory occupied, but still have CPU cost left. Because you have to allocate, run a virtual machine container on the same machine, like the first machine's memory and the second machine's CPU cost, they are wasted. Because of this, data center, they are actually seeing not very good CPU and memory utilization. If you could improve resource utilization by even like 1%, it means like – I don't know how many dollars, but definitely a lot of money for cloud providers.

In the disaggregated view, there's no such waste, because you can – Application basically runs on the distributed set of hardware devices and it can have memory from any device. It can run your threads on any CPU cost and there will not be this like waste of resource. That's from cloud providers' view.

I would say from like the cloud users' view, one trend is, a least certain amount of cloud users, they would want to move to this what people call serverless model where you don't need to care about clusters. You don't need to care about servers. You just run a program. You have some data and you leave all the rest to the cloud providers.

For this type of customers, they don't really want to care, they don't want to manage where exactly the software is running. Is it running on CPU? Is it running in this memory or on that GPU? I would say for this type of trend, it would be well beneficial if we could have like a disaggregated architecture and then the cloud provider will decide what to allocate for this serverless function.

[00:15:24] JM: When you talk to cloud providers or people from cloud providers about this idea, what's their response?

[00:15:33] YZ: I've talked to several. In general, the current status is almost all the cloud major data centers, they're already disaggregating storage. That's quite standard already. Facebook has a separate storage pool or separate MemCached pool, and Amazon has separate EBS, like S3. Then it's same for Alibaba. They also have like a disaggregated storage.

Storage is considered quite standard to be disaggregated. Then, recently, certain data centers, they are thinking about going like one step further into disaggregating persistent memory, [inaudible 00:16:14], like the Intel Off-Chain, or even disaggregating memory. But one of the major concerns, probably I wouldn't say from which major company, but it's a major data center cloud provider, is the failure model of disaggregated memory, because now it makes the whole failure handling more complex.

If one memory fail, this memory device, it could store memory content for different applications that are running on different CPU processors. How do you deal with this failure scenario? If dealing with failure means that you need to use more memory, then it basically defeats one of the original main purpose, that is you could – By tightening the resource packing, you have better result utilization. But if you end up like using more memory just to provide this reliability guarantees, then cost-wise, it basically offset the cost benefit of the original idea. That is one of the major of this fight back of this whole idea I think.

[00:17:25] JM: Could you talk about them in more detail? What are the tradeoffs in that what you just described? I'm not sure I fully understood what you are articulating.

[00:17:36] YZ: One of the main benefits of disaggregating resources, that you can allocate results from any component. You will have least amount of waste. Currently, data centers, they have like almost 50% of the memory they are not used even when there are waiting jobs, and that is the scenario I described to you. If we could disaggregate, then ideally we could include this to close to 100%. That means like close to 2X improvement in resource utilization. If it's 2X improvement, it means 50% cost cut. Then the problem is if we want reliability, usually, the easiest way to do that is through replication. If you keep 2X replication of your memory content, that means you need 2X the memory space, and that offset, the 50% cost that you actually want with this idea.

Definitely, this failure issue is a new problem. I don't think it's unsolvable, but it's definitely a hard problem. There actually have already been some recent research work in reducing this to less than 2X and some other ways. If you could reduce it to less than 2X and then your benefit is like 2X, then overall, you're still having cost saving besides the other benefits that I talked about.

[00:19:14] JM: Are you saying that if you disaggregate, then your memory constraints are – The memory that you have to waste in order to get reliability is going to be less?

[00:19:31] YZ: First, let's say we don't provide any reliability just by disaggregating. Optimally, you could improve results utilization by 2X. The current is around 50%, 60% utilization, and we could improve it to 90%, 100% utilization. That is without replication. The current service, they don't replicate memory. When we disaggregate, let's say if we don't replicate memory, so that's the cost saving that you are going to get, the resource utilization improvement that you're going to get. But the problem is we are making failure handling more complex, because now your program is scattered around many device, many devices. Any of them, if they fail, they could affect your program, your application. To deal with that, the easiest way is to always have a replica. Having a replica means you're doubling the space usage. That's one of the downside. But this is just the easiest way of dealing with failures and it's almost like too much that you need to do. That's like [inaudible 00:20:44]. With better ideas, better designs, it could be less than 2X.

[00:20:52] JM: You're saying you could – How would you replicate something without having 2X the memory utilization? If I spin up some microservice on container A and I want to have that service replicated or a database, same thing, caching server, whatever, how could I avoid having 2X?

[00:21:13] YZ: This has – Actually are already being done. 2X is exact copy. You don't need 2X to prevent when failure, and this idea has been already applied to storage, like Microsoft Azure, and that's called erasure coding. Erasure coding is like a coding CRA that you could use like X+K copies. Not copies. X+K units to start data that is X. The extra K is like some actual information, and K is less than X. X+K is less than 2X. But with smart coding CRA, like [inaudible 00:21:58] or some other coding theory, then you could go down to like 1 point something and still be able to handle one normal device failure.

Traditionally, this has been applied to storage, because storage, you also need to handle failure and Microsoft, and as a company, you want to save cost. They don't want to go to 2X. That's why they are using this coding, erasure coding CRA. But the problem of that is it's usually

there's some software overhead to calculate these codes. If you want to apply that to memory, which is very fast, you need another way. There are like other research groups solving this problem.

[00:22:45] JM: The kinds of erasure coding that you're describing, is that like kind of like what RAIDStorage is?

[00:22:52] YZ: Yeah, it's more complex than RAID 5, RAID 6, but you can think of it as that. 2X is the simplest. That's RAID 1. But if you go to like RAID 5, RAID 6, you get like less than 2X.

[00:23:08] JM: Cool. You're saying to get that in memory speed is more difficult. There're more constraints? Because I think with RAID, if there's a failure, isn't there some time that it takes to reconstruct everything, and maybe that's not permissible with a memory system?

[00:23:24] YZ: It's not like the runtime overheard. You need CPU cycles to calculate these codes. This software overhead just calculating these extra bits, they are horrible in storage, because storage is slow. But now if you're talking about memory speed, this could be significant, and you need that for every read and write if you want to use [inaudible 00:23:47]. This is not the only way to stop this reliability problem, but that's like one of the problem that some of the researchers here are currently solving.

[SPONSOR MESSAGE]

[00:24:04] JM: When you spend your spare time learning, you can accelerate your career. O'Reilly lets you learn through high-quality books, videos, courses and interactive experiences. O'Reilly content has been built over decades. They're a trusted source of effective technology education. If you're an individual leveling up on your own, you can use O'Reilly to chart a course for your career goals. If you manage a team or a company, you can get access to O'Reilly's career development resources for your whole organization.

Go to softwareengineeringdaily.com/oreilly to explore O'Reilly's e-learning experiences. You can build the skills you need to future-proof your career. Check out softwareengineeringdaily.com/

oreilly, and thank you to O'Reilly for being a partner with Software Engineering Daily for many years now.

[INTERVIEW CONTINUED]

[00:25:05] JM: Okay, let's take a step back. If we're trying to – You're laying out a world of disaggregated server infrastructure. What I wonder is since you're coming at it from the research perspective, do you see this as like a thought experiment that you're doing or are you trying to lay out a vision for actual new kind of server infrastructure that you'd like to put into practice and you'd like to build servers around this and see this actually come to production?

[00:25:45] YZ: Yeah, that's a very good question. First, my philosophy of doing research is we should always be with real systems, and that's what my group is proud of. But the original research idea, we usually try to take a more visionary idea that try to push things, like [inaudible 00:26:06] idea. The vision is what I talked about, like everything is [inaudible 00:26:11] completely separated. You have processes directly connecting to network, memory directly connecting to network. That is a very nice vision, but we need to think about how data centers could actually deploy this idea. The problem of that is data centers, they already have like millions of servers inside. You cannot tell them to just throw them all away. That's like a practical problem, but we don't have to.

The second version that we are currently working on is to think about how to practically deploy this idea with today's data centers. The way towards future data, maybe it will become our vision. Maybe it won't. But current data centers, they already have the need. The biggest need is they basically just need more memory to run their application, because if you look at all these big data machine learning, deep learning applications, they do need a lot more memory. That's like one of the starting end goal that we are currently working on to be able to incorporate this disaggregated idea into today's data centers.

Not like changing these servers, but having like another layer that adds up. Let's say if we want to add disaggregated memory, now we just connect these disaggregated memory to the network and then let servers – The servers would be more like compute-focused and they could now access this disaggregated memory layer through a general purpose network. We are

actually building real – We actually are close to finishing building real hardware for this disaggregated memory layer.

[00:28:00] JM: That's interesting. We had another show fairly recently. There's a project called Cloudburst out of Berkley, I think, and I don't know if you saw that paper, but the premise is – I guess the question he was approaching in his research is how do we get to stateful functions as a service? How do we have stateful serverless functions? His answer is basically just give up big shared memory cluster to all the different serverless functions pretty straightforward.

You're talking about implementing some kind of shared memory system in server infrastructure. You're also talking about you're interested in serverless functions. Can you just tell me more about what you're talking about there with the kind of shared memory component that you're exploring?

[00:28:54] YZ: Yeah. That's a very good angle, and that is exactly one of the major usage scenario we think our disaggregated memory could be used for. I'm aware of the serverless and Cloudburst work.

You can build disaggregated memory. You can build like adding like a bunch of memory into like a memory pool and you can use it for anything. First is you could just like use – If you're running out of memory, you could allocate more memory from this pool, and we are providing it in a very fast way. Then the second way to use it is to use it more like a sharing and a state, sharing in a state storage area, a fast one. That would allow different compute servers to save shared state and to communicate. That's a second very common usage that we envision our disaggregated memory pool would be.

The nice thing is like by just managing – One of the major benefits that data centers they like about the idea of disaggregation, besides all the utilization, besides all the conceptual – Like build for conceptual view, is it's manageability. If you're managing things separately, you could basically upgrade your memory service. You could add more memory, like you move more memory. Managing this pool can be completely separated from managing compute pool, from managing storage pool.

Data centers and clouds, they are seeing real benefit when you get storage side by actually managing disaggregated storage separately. When they are changing their storage pool, it's not affecting the compute pool at all. This is more like day-to-day, like an engineering benefit. This is the same benefit we will have when we have disaggregated memory. You could scale this differently from scaling compute. If like next month you find that your data centers have need for more application data centers, they have need for more memory. You just go purchase more memory and enlarge your memory pool.

Then if next month you find need more compute, you just buy more compute service. Then because like when you're adding memory servers into your memory pool, we made it in a way that it's separated from the compute pool. It's transparent and you can manage. Let's say next month you want to deploy a new service into the memory pool that is like p-value store or MemCached, you could just upgrade that, change the service without affecting the compute side. That's the other nice thing about disaggregation.

[00:31:43] JM: Got it. What I'm having a little bit of trouble understanding is, again, this is providing some modularity to the data center operator. From the average user, like the application developer, the average application developer is not thinking can I get a new hardware component to get better memory allocation. They're thinking, "Can I get a better memory allocation?" They're thinking from the application point of view.

The data center operator has different concerns than the application operators. I guess I'm trying to understand what is in it for the data center operator and what is in it for the application developer or who does the disaggregation, the server disaggregation serve most beneficially?

[00:32:49] YZ: That's a good question. I think it's like a combination of all. Some of the benefit is indirect. I would say the direct benefit to data center providers is cost saving. To data center operators, it's more manageability. Then the direct benefit through users is that they don't need to manage servers. They don't need to know how much memory they need to allocate. Like configuring a new instance, you need to know how much memory your application would need, and you need to set that ahead of time, and that's usually set by the IT department of your company that's using the code. Usually, you don't set it a perfect way because you don't like at runtime what would be used.

In our vision, we don't need this like scope of setting pre-settings some fixed amount of resource. It is allocated as the application goes. You are paying for exactly what you are using instead of paying for, let's say, a fixed 16GB of memory. If your application is using 2DB at this time, 3DB at another time, that's exactly what the data centers would allocate for you, and that's what exactly you will be paying for. That's the direct benefit to cloud users. The indirect benefit would be if the cloud providers, as a whole, they could save their whole data centers running cost, then they can offer better price to the users.

[00:34:34] JM: Okay. You're saying that the disaggregation would just lead to better allocation. It would lead to more accurate allocation rather than over-allocation.

[00:34:44] YZ: Yeah. Yes.

[00:34:45] JM: Okay. You work on LegoOS, and in a typical operating system, you have CPU and memory bundled together. The premise of LegoOS is that if you disaggregate the hardware, you need a new kind of operating system, and that's one that you're building. Why couldn't you just have a traditional operating system work in a disaggregated hardware setting? Why do we need a new kind of operating system?

[00:35:20] YZ: First, is you could always change existing one into something that you want, but the change in this case would be very significant. You better as well just build a new one. The reason for that is like all the traditional OS, they assume I have local access to all the resource that I need. When you are allocating memory, the allocation is local. You don't go across network, and that's something that's handled by current OS and similarly to other things.

But now, if we have memory that's across network and who manages that memory? If you still want to manage that memory, like virtualize, like translating physical memory to virtual memory providing address space, all these functionalities, if you still want to manage that. With a traditional OS, it's like you need basically to rewrite the whole virtual memory system. Because now your memory, physical memory scatters across multiple components over the network, and now you want to build a virtual memory at your space out of this. No existing OS actually does

that, and that's just one example, and there are many cases. Your process can be running on different processors, and how does a traditional OS data only manages a single processor?

That's why it becomes both like distributed in the same layer, like this CPU layer, processing layer, compute layer, you're distributing that, memory layer distributing that. You're also distributing like disaggregating the memory layer from the compute layer. If you are doing all that, you need to also distribute the software, which is the OS.

[00:37:09] JM: Can you tell me more about the main design principles of LegoOS?

[00:37:15] YZ: The main design principle is we manage the hardware at where the hardware is. We manage memory at the memory device. We don't manage memory. We don't run the memory management software at the CPU. That's the traditional way. The whole OS runs in CPU. It manages memory. It manages CPU. It manages storage. What we are saying is the management of the device should be local with the device. If you want to manage memory, the management software should run at that memory device if you're managing storage, that management software, which is the file system, should run at a storage device. That's one principle.

The other principle is to have a distributed view of the operating system. To have building support for network communication, to have building support for failure handling. These are the two major design principles. The first principle, why we think that is possible to manage things local with a device? That if you look at devices today, they are becoming what people call smarter. You have smart SSDs and smart mix. That means the device, it has some processing power internally, some small controller that can run some operations. That means it is possible to run a piece of the OS functionality as a device at these smart devices, and that's what made the first principle actually feasible.

[00:38:55] JM: The LegoOS has a split kernel architecture. Can you explain what a split kernel architecture is?

[00:39:05] YZ: That's basically just what I just described. You're splitting OS functionality into different pieces. You run each piece local at a device, and then the whole this is a distributed system.

[00:39:18] JM: Is there any difficulty in deciding what components of a kernel to split up and assign to different devices? Isn't there like overlapping functionality across the kernel that you might need to run on each of those devices?

[00:39:36] YZ: That's an excellent question. If you look at today, Linux, like everything is sort of combined. Even though Linux has a lot of module, modularized a lot of things, but fundamentally, the design principle is a lot of things like intertwined. For example, in Linux, we usually say that everything is just a file handler. This is like the network, when you open the network connection, you get a handler descriptor back, and that part is like mingled with another layer.

But if you look at – If you throw away what existing OS are doing, if you just think about the original purpose of an OS, the original purpose of an OS is to manage hardware and virtualize them. If you think that, and now you think the new hardware architecture, which is separated, you just think about I want to now manage memory. What do I need to manage memory and what do I need to virtualize memory? That's the part I run at memory. If you come from that point of view, then you get a design that could be cleanly separated.

[00:40:52] JM: Got it. The architecture has something called a Vnode. What is a Vnode?

[00:41:02] YZ: That's basically like an abstraction that we provide to users. That is more just like a container or a virtual machine. We want to hide the physical disaggregation nature from the users so that we could continue run existing applications without asking them to change their soft code or even to recompile. The levels currently, it could run in modified Linux binary. Linux binary, it was assuming like everything is in the monolithic service. The virtual node is like a virtual concept that we create, much like a virtual machine or a container to users and they could basically just like getting a Vnode, then underlying. The Vnode would run their application binaries, and then underlying the binary, like the memory, could be on different memory devices and be – The threads, they could also run on different processors.

[SPONSOR MESSAGE]

[00:42:14] JM: Today's sponsor is Datadog, a monitoring and analytics platform for cloud scale infrastructure and applications. Datadog provides seamless integrations with more than 400 technologies, including AWS, Postgres, MySQL and Docker, so you can start collecting and visualizing performance metrics quickly.

Distributed tracing and APM provide end-to-end visibility into requests wherever they go across hosts, containers and service boundaries. With rich dashboards, algorithmic alerts and collaboration tools, Datadog provides your team with the tools that they need to quickly troubleshoot and optimize modern applications. You can see it for yourself. You can start a 14-day free trial and Datadog will send you a free T-shirt if you just go to softwareengineeringdaily.com/datadog. You can start that free trial and get a free T-shirt. Go to softwareengineeringdaily.com/datadog.

[INTERVIEW CONTINUED]

[00:43:18] JM: Programming an operating system even by the traditional standards is pretty challenging. If I'm creating a monolithic operating system, that's just not an easy thing to do. You're designing and programming a completely new type of operating system. What is the biggest challenge that you faced in implementing this thing form the software and hardware side?

[00:43:51] YZ: Yeah. That's very interesting. In the beginning, we just thought, "Oh! This is really quite it. Fits all the [inaudible 00:43:58] trends, application trends we should be with." Then we just start, and we looked at – We did first look at existing OS and think if it's easier to change them. But then we found it's better to just start from scratch and build a new operating system.

Surprisingly, the biggest challenge, at least the ones that we didn't expect building drivers and supporting a modified Linux APIs. Building drivers, because this is traditionally done by the device builders and we have to now build our own drivers. What we did is we actually parted like

only limited amount of drivers from Linux, but we envisioned, if someone is going to build this, they would – Like different devices providers, they would just build their own drivers.

The other thing is our goal, and this goal maybe too aggressive. We want to rent modified Linux binaries. Not even need to recompile. For that, we have to follow exactly the Linux system core interfaces, like all the way to like a parameter, a return value. That actually took a lot of time, and especially because our underlying layer is completely different from Linux. We actually added like a thin layer on top of LegoOS to basically translate Linux syscalls into our system calls, and that actually ended up taking quite a lot of time.

[00:45:34] JM: Wow! You built a full syscall translation interface. That's what you're saying.

[00:45:41] YZ: It's not full. By the time when we published, I think we supported 130 some system call, some common system calls, not like a complete system call interface. Yeah, but we were able to run TensorFlow and some other complex application without any modification of their binaries.

[00:46:03] JM: Let' talk about that, like TensorFlow, for example. If you are running TensorFlow across disaggregated hardware, what are the different hardware components that the TensorFlow runtime is going to be spread across and what do those interfaces look like at those different hardware facets?

[00:46:26] YZ: This is actually the underlying disaggregate nature and what devices the – TensorFlow application is running on and it's actually hidden from the TensorFlow application. To them, they are just running like a normal TensorFlow program. The main thing is we are disaggregating its memory. The nice thing that fits our model is TensorFlow's computation and memory access pattern. It actually fetches what is called a mini-batch, then fetches it from like slower storage and memory. Then after it fetches, it will do like a lot of computation on that mini-batch.

Our model is that – Our processor side, it still has some small memory and we [inaudible 00:47:13] project we actually configure that small memory as another level of cache. The best application model that could fit this architecture is applications that behave locality access,

memory access locality. TensorFlow is one of them, because it's like kind of it is found in this mini-batch. But this is all hidden from the application. The application, they would just load it. Naturally, what you load is the mini-batch, and that would run in the extra level of cache in your processor.

[00:47:49] JM: Okay. Can you tell me more about what you learned when you were running TensorFlow across your disaggregated hardware?

[00:47:58] YZ: Yeah. One this is this mini-batch really helped us. This program behavior, TensorFlow program behavior or mini-batch. If the mini-batch is too big than what your processor can fit, then it needs to make more network round trips to the memory. That's the first thing. Then, in general, when we're evaluating, something surprising to us is the network was not the bottleneck.

The bottleneck was our memory, memory's implementation. When we built LegoOS, we didn't really have real hardware disaggregated memory devices or disaggregated storage device. We were just using CPU cost on the normal server as if they are the memory device. Then the memory device, it needs to process requests that are coming from a lot of processors through the network.

Originally, we thought network would be the bottleneck, but network was not the bottleneck. It's the processing at the memory side. Memory side, you need to have a lot of concurrent accesses coming from the network. Network could be 100GBPS and then basically just like pulling your network card and get these requests and just translating them from virtual to physical memory. It cannot – Just by using like a general purpose CPU, you cannot keep up with the network speed. That eventually limit application performance.

That's why we started the real hardware project that we are currently close to finishing, and that would use another hardware that is not CPU, that is cheaper than CPU, but actually scales better and that could sustain this very high-network line rate.

[00:49:54] JM: I see. You're going to define, you're going to build specific hardware to replace the CPU in this disaggregated setup?

[00:50:06] YZ: The CPU is still CPU. The compute side, compute side could be CPU-based processors. It could be GPU. It could be TPU. That's the compute side. What I was talking about is the memory side. The memory side, I talked about you want to run some management off the memory, local at the memory device. If you're using a CPU to run those memory management software, even though you think that's something that simple to do, just using CPU and limit the number of cost. You are not going to keep up with the high-network line rate. The memory bandwidth is fast. Network could be as fast as 100GBPS already. The bottleneck is your management software of the memory and you don't want to run that in CPU. That's too costly and that's not going to scale. That's the part that we're rebuilding.

[00:50:59] JM: Okay. I mean, there's so much in your research that would be interesting to discuss, and I just want to jump to a different topic, which is machine learning to help build operating systems. Can you talk about this in some detail? How could machine learning be used to improve the development of an operating system?

[00:51:22] YZ: This is like a new direction that we recently started. If you look at OS building, it's a lot of just like human factors in it and a lot of heuristics and you are deciding this before you know the applications. Applications, they can change very rapidly these days. But once you decide, let's say a CPU scheduling a policy, a memory replacement policy, you build that into your OS. It's very difficult and it's rarely changed, although you have different applications running on them. When you're designing these policies, you don't really know what is the best and you can only come from a heuristic point of view and it's just like human factor in this. That's one case where – Traditionally, there's not so much need, because your application doesn't change that often and you just build a general purpose OS, and that's good enough for most applications.

But now, if you really want to build better OS for these new type of applications and to meet the speed of how applications develop, you want to think of a way that like leaves the human factor out and could automatically change or adapt your OS for different applications. That is one of the motivation.

Then the other thing is like after you build an OS, if you're running Linux, you have to configure a lot of things, and this configuration is just like another human factor. Often times you would have some complication problems and you need to go back and all these and that. That's very complex and you don't want to – It will be better for machines to do that. Finally, it does take time to write OS components, but if you could have a way to automatically generate these components, then that's another way of saving human time and using machine to do that. That was the original big vision that we have.

[00:53:31] JM: Are there some specific decisions in an operating system that could be improved by machine learning? I think about scheduling, for example, are there specific areas of operating systems that you think could be improved by machine learning?

[00:53:46] YZ: There are many places like this that could be improved, like scheduling decisions. Instead of one scheduling decision, if you can learn the behavior of programs that are currently running on this specific server and your OS basically just adapt to that – Adapt the CPU scheduling and policy to that application, type of application that is running on this server and another server is running on that server application, you don't need to change the OS. But if it's a mode, then it could learn and adapt the scheduling decision. Similarly, if you want to do like memory allocation or memory replacement, file system disk allocation, a lot of these decision making you could like – If you use a machine to do that, it could be done more like – Fit your application more. But the biggest challenge of using machine learning for OS versus like use machine learning for other domains is that, first, OS needs to run very fast. Second, OS needs to be always correct. Third, well, you need – For other machine learning, you need to plug data. You need to plug data so that you can learn. These are the three main issues, like challenges that we will face when we use machine learning to build OS.

Machine learning, they are traditionally OS, we sacrifice a lot of accuracy basically. You don't go for like a perfect memory replacement policy. You go for like approximate so that it can run very fast. Now, machine learning, if you want to run ma chine learning, it cannot be as fast. For instance, like if you want to run that GPU, you have to cross the bus rom CPU to GPU. Also, if you want to run CPU, then there's some – It won't be run as fast. There are many like detail issues like this, and the other thing is like OS cannot be wrong. If you're accessing memory and if you are using a machine learning model to say, "I have a model. I predict that this virtual

memory address should map to this physical memory address and that have like 90% confidence. But like you cannot access a wrong memory address. OS is like absolute correct, but machine learning is like statistical by nature. How do you solve that problem? You almost like have to have like a failover mechanism to deal with the natural inaccuracy in all the machine learning models.

If you want to like plug the trace, that dataset that you want to learn, like memory access pattern, storage access pattern, like CPU access pattern, then you need to find a very efficient way to do that at runtime. Because at runtime you don't to add any more overheard. Also, you need to think about where to store them. There are all these major challenges that you need to be solving.

[00:56:51] JM: Okay, Yiying. Well, it's been really fun talking to you and I appreciate the depth and the breadth of the work that you're doing. Keep up the good work and I look forward to seeing what else you come out with in the future.

[00:57:04] YZ: Thank you. This is fun. It's a very nice talk.

[00:57:07] JM: Okay.

[END]