Chenggang Wu Speaks Out on his ACM SIGMOD Jim Gray Dissertation Award, Rejection, Believing in Your Work, and More

Marianne Winslett and Vanessa Braganholo

Chenggang Wu

https://cgwu.io/

Welcome to this installment of ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are on Zoom with Chenggang Wu, co-founder and CTO of Aqueduct. Chenggang received the 2022 ACM SIGMOD Jim Gray Dissertation Award for his thesis entitled The Design of Any-scale Serverless Infrastructure with Rich Consistency Guarantees. His PhD is from UC Berkeley. So, Chenggang, welcome!

Thank you for hosting me. It was a great honor to receive this award.

Our runners-up for the award this year were PingCheng Ruan (National University of Singapore) and Kexin Rong (Stanford University).

So, Chenggang, what is the thesis of your thesis?

My dissertation was primarily centered around the topic of serverless. I actually can't believe it's already been two years since I submitted my dissertation. Although these days serverless computing has become a widely used technique, back then, in 2016, it was still a fairly newish concept and became a hot topic in the research community around 2018. So, just to give a little bit of background, serverless is a software design pattern whose main advantage is to offer a higher level of abstraction to program the cloud.

Basically, in the pre-serverless world, to run an application in the cloud, we first need to provision a virtual machine, specify some resource requirements, like how many CPUs, GPUS, RAM, and disc you need and the operating system you want to use, launch that VM, and copy your code into the VM and run it. For seasoned backend engineers, this might be fine, but for folks without deep systems expertise, this is actually a tall order. But, with serverless computing, programmers can now just simply upload the code to the cloud, issue a request to run the code and get the results back, without having to worry about any of these that I mentioned beforehand, which is very convenient.

[...] how do we design a unified architecture that scales well in the distributed setting [...], but also takes the maximum benefit out of a single core [?]

The interesting thing is this all sounds very promising, but the challenge, of course, is that raising the abstraction also means hiding a lot of the knobs that the users can otherwise tune. So, we, the system builders, now need to do a good job of delivering high performance, scalability, fault tolerance, and consistency on behalf of our users. My dissertation was all about how to tackle these challenges.

Can you tell me more about the challenges that you had to tackle?

My first project, called Anna, was centered around exploring the tradeoff between scalability and consistency. The main motivation behind Anna is that these days there are a bunch of highly scalable distributed computing infrastructures being built. But the interesting thing is that even within a single machine, we actually have access to very rich, very beefy compute resources, like lots of CPU cores and a bunch of RAM. And we noticed that people actually aren't making the most use out of these, even for a single instance of compute node.

So, the research question becomes how do we design a unified architecture that scales well in the distributed setting, of course, but also takes the maximum benefit out of a single core. This architecture can then make full use of every CPU core and available RAM to deliver the best performance, even within a single-node system. The solution that we landed on is something called a "Coordination Free Execution Model." Because we noticed the bottleneck that inhibits the scalability usually lies in the coordination between different compute threads when they access shared state.

So, the coordination free model basically means every thread has access to its own local memory without communicating and just does its own work. So, in that way, everybody can proceed in parallel so that it minimizes the coordination between threads. But of course, this is the ideal world, because if everybody just keeps doing their own thing, then eventually, they have to communicate and exchange information to offer a consistent view of the world.

The corollary challenge that emerges from that is how do we design a suitable consistency mechanism such that although different threads are accessing its local copy, eventually, they will find some way to reach an agreement to offer the application a consistent view of the world. So, we introduced a technology called "Lattice-based Conflict Resolution Strategy." We found that carefully composing these different lattices allows us to guarantee that, although everybody may be doing their own things, and although these messages may arrive at different threads in different orders, eventually they will reach a consensus. So, it's a decent design that offers maximum scalability, both within a single node and across multiple nodes while achieving different levels of consistency such that an application can safely run.

That sounds like a great start to your thesis. What was the topic of the rest of it?

So, basically, in the first chapter of my thesis, Anna, we designed a system architecture that performed well at each scale point, be it the single node context or the geo-distributed deployment. But the key promise that serverless computing wanted to deliver is "pay as you go", which means that it must be elastic. As your workload requirement goes up, the system needs to be able to detect your workload shift and then dynamically add nodes to satisfy your compute demand. And also, if your workload is going to shrink, it should be able to reduce the resource allocation to save costs for you.

The second chapter of Anna is to take this architecture that's performing well at each scale and then implement our own scaling mechanism so that we can dynamically adapt to these workload shifts and adjust the resource allocation accordingly. So, fundamentally, it's exploring the design of the underneath autoscaling mechanism, and exploring the tradeoff between performance and cost efficiency.

The first two chapters focus on the storage side of things, which actually lays out a very good foundation for the compute side of things because you can imagine, during the compute, inevitably, each compute agent is going to access all the shared state that's maintained in the underlying storage system. So, the third chapter is basically bringing in all of these core design principles: coordination free execution model, lattice-based conflict resolution scheme, and autoscaling, and applying these principles to the compute layer.

So, now, there are two layers, there's this compute layer, and there's a storage layer, and we may have workloads that exert different amount of tension to each layer: Maybe there's a workload that requires a tremendous amount of compute but only little storage. In this case, the compute layer can be very beefy, and the storage layer could be pretty lean. And you can imagine a workflow that's the other way around. So, this advocates for the design of resource disaggregation – allowing the two layers to scale independently, which is very economical.

But then the challenge is that as we progress through the compute, at some point, it needs to access the storage; it needs to issue a network request to the storage tier to request the data, which is then sent back to the compute layer. This process can introduce high network latency, which is a significant performance challenge. So, the third chapter is more on exploring this concept of logical disaggregation with physical colocation (LDPC). Logical disaggregation allows the compute tier and the storage tier to scale independently. Physical colocation means that in our implementation, we can carefully cache some of the data from the storage tier up one level to the compute layer.

In most of the cases, because we are accessing the cache from the compute layer, it minimizes the latency drastically, and inside each cache, we extend Anna's design principle, so it still offers a suitable level of consistency while eliminating the network latency. This way, the LDPC design allows us to achieve the best of both worlds (performance and consistency). This summarizes the three chapters of my thesis.

That sounds very appropriate for a startup.

Yeah. Exactly. So, that's why after my PhD research, both my colleague (Vikram Sreekanti) and I, and both of our advisors, Joe Hellerstein and Joseph Gonzalez, were very interested in packaging our research into something that the industry can use. The company we founded, Aqueduct, is doing exactly this. I mentioned before that serverless computing is very suitable for folks without deep systems expertise, and that's why our current target audience is more on the data teams, especially data science folks.

[...] you need to be the No.1 fan of your own research.

These groups of people are domain experts in various fields, like biology, some scientific computing fields, and the financial industry. They have deep domain knowledge but only have some basic programming knowledge in Python and SQL, and they don't have deep expertise on how Kubernetes or Docker containerization works, so they're the perfect audience. So, on the API level, we want to offer them a way to easily program their workflows (which involves some machine learning to predict churns, the weather, or financial trends), and they can construct the workflow inside their familiar Python or Jupyter notebook environment. Then, underneath the hood, when they submit this Python workflow definition, we package that into Docker containers and do all of these performance optimizations outlined in the thesis. So, from our user's perspective, they get both ease of use and peace of mind in the sense that all of the scalability, consistency, and fault tolerance aspects are automated and abstracted away from them. That's the ultimate promise that we wanted to deliver to industry folks.

Do you have any words of advice for today's graduate students? Things that you wish you had known when you were a new PhD student?

I think the most important one is to be a believer of your own research and stick with your belief. There are two parts to this. The first part is that you have to believe in your own research, which means you need to know that what you're working on is important and you need to be the No.1 fan of your own research. Actually, I've been through this during my first year. When I was an undergrad, I was working on something that was unrelated to my dissertation field; I was working on interactive data visualization, and I continued this line of research during the first year of my graduate study.

But then, along the way, I realized that's just not really where my core passion was, and I was interested in more "hardcore" systems-oriented topics. So that's why I made a switch from interactive visualization to distributed storage and distributed systems area, and I found myself to be very much enjoying that. I feel this is the prerequisite of me being a believer of my own research.

The second part is to stick with your belief because I found that, along the way, not everybody is going to be believing in your research. Especially when you submit papers, you're probably going to encounter some rejections. In my case, I think my first paper on Anna only got published when I was a third-year PhD student. That paper actually got rejected three times consecutively. So, that was a "dark age" for me. It was definitely a frustrating experience, but I don't actually think I was depressed by that mainly because I was confident that I was doing good work, and there's an old saying, "good work eventually gets published." So, the fundamental belief that I am doing great work, that it will get publicized and recognized by people, is ultimately the source of power that's driving me through these dark ages.

The second lesson is that I found collaborating with your peers is usually a little bit more enjoyable than working alone. When I first joined Berkeley, it just happened that all of my advisors' grad students either already graduated or were about to graduate, so I had to explore the research on my own. That was fine, but later on, I found my peer collaborator, Vikram Sreekanti (actually, now we're running the company together). Once the two of us started working together, there were a few things that I discovered. I think, first of all, it's just more fun to work with people. You have other folks you can talk to, either research-wise or just complaining about things together is always better than dealing with everything by yourself. Also, although I was communicating very frequently with zero issues with my advisors, they tend to give advice at a higher level: the idea generation level or the design level. But regarding the implementation details, it's still very nice to get some feedback from your peers, who are working together with you and know the details of your codebase.

Also, I think collaborating with folks, in the end, leads to a net gain on productivity. Usually, I was leading a project that my collaborator was co-leading, and I was also participating in the project that he was leading, and I was the co-lead of that project. So, in the end, it was a net gain for both our individual growth and the team's growth.

Great advice. Thank you very much for talking to me today.

No problem. It was my great pleasure, Marianne.