Andy Pavlo Speaks Out on Main Memory Database Systems

Marianne Winslett and Vanessa Braganholo



Andy Pavlo

http://www.cs.cmu.edu/~pavlo/

Welcome to ACM SIGMOD Record Series of Interviews with distinguished members of the database community. I'm Marianne Winslett and today we're in Snowbird, Utah, USA, cite of the 2014 SIGMOD and PODS conference. I have here with me Andy Pavlo, who is a professor at Carnegie Mellon University. Andy received the 2014 SIGMOD Jim Gray Doctoral Dissertation Award for his thesis entitled "On Scalable Transaction Execution in Partitioned Main Memory Database Systems". Andy's PhD is from Brown University where he worked with Stan Zdonik.

So Andy, what do you got there? Can you rotate it so that we can read it in the camera?

This is my Jim Gray Doctoral Dissertation Award.

Woah! Look at that! Now, is this body armor?

No, no, this is my belt. I've only won two awards in my life, okay? I won class clown in high school and then I won this. And you know what? I may never win another award ever again, so I'm going to relish this as much as possible. So I don't want to take away from Aditya because he won as well, but he's won best paper award and things like that. So this is it for me. So I'm going to sleep with this every night for the next year and then someone else will win for 2015.

So you just strap it around your waist?

Well, maybe, but when you give a talk you have to hold it like this because people can't see above the podium, right? And then I'll have my students hold it up behind me and walk in with it in a procession whenever I give a talk. Just for one year, and then after that, it will be retired. It will go in a shelf in my office at Carnegie Mellon.

> ***Your job talk needs to have tons of graphs. Lots of jokes, lots of graphs.***
>
> ***Michael Stonebraker***

Okay! And which parts of it did SIGMOD give you?

Well SIGMOD gave me the plaque. The belt part, you know, I made myself.

You made it yourself?

Well, I got it lasered.

Okay. Well it looks pretty spiffy. Maybe we should do that for every award winner.

Again, so Aditya… he worked hard, he got the award too. He had the option to get a belt. He could have made a gold chain to hold his. He decided not to. I decided to do it. So there you go.

Now by saying that you got the award for class clown, you're setting the bar very high for this interview.

I don't know about that much, but okay.

Okay, well let me ask you first. What's your dissertation about?

We started around 2007, 2008 when the NoSQL movement was gaining prominence. They were all out there saying, "Well, the only way that you can scale up a large scale distributed database system to support a large number of concurrent users with concurrent operations is if you give up transactions entirely", right? So you see these in systems like Google's BigTable or Amazon's DynamoDB, and in the open source world there's Cassandra, MongoDB and Riak, sort of implementations based on those ideas. So when we started we said, "Well, let's not give up transactions. Let's see what we can do in a modern architecture. Can we still have a distributed database system that can still support strong ACID guarantees?" So that's sort of what culminated in the system called H-Store that I helped develop with people at Brown, at MIT and Yale and then what eventually became VoltDB, but it was originally out of Vertica.

The basic concept of the system was that we were going to have a main memory execution environment. We were going to have serial execution transactions across multiple nodes. We would support stored procedures only, we would use a real lightweight logging scheme. So that allowed us to be able to support the large number of concurrent users that you need in these modern Internet applications without having to give up transactions. So I think that was a pretty significant contribution.

And did you achieve that totally?

To some extent, yes. I mean, there are certain aspects of applications where H-Store is not the right architecture. I'm totally upfront about that, right? But I think we've seen this in the commercial space with VoltDB. VoltDB is the commercial implementation of H-Store's architecture. There are a large number of applications where the design we used in H-Store is absolutely appropriate and it works really well. But there are a large number of applications, for instance, anything with a social graph like Facebook or Twitter where you have arbitrary users connected together, that don't partition very well, so H-Store is not the right architecture for that.

So what were the key architectural or implementation choices that you made that make it a success?

The main one was that it's main memory storage engine. That's not a new idea. The first work around this concept was in early 1980s. What makes it different this time around is that we're finally at the point where the price and the capacity of DRAM made it possible to store all but the largest OLTP databases (these front end transaction processing databases) entirely in main memory. Once you have everything in main memory, a lot of the design decisions that came out of the original databases from the 1970s don't make sense anymore – the stuff from Ingres and System R. So for example, you don't need a heavy weight concurrency control scheme with locks and latches in order to mask the latency of disk because there is no disk. So in a traditional system, a transaction could stall anytime because you had to touch data that wasn't in main memory, that was in disk, and therefore you had to allow other transactions to run at the same time in order to mask that stall. But now since everything is in main memory, you're never going to have a disk stall, so it doesn't make sense to allow multiple transactions to run at the same time.

Another key concept that we did in H-Store was that we use a lightweight concurrency control scheme based on partition level locks where each partition is going to get assigned a single threaded execution engine for transactions. And so, because it is single threaded, when it executes a transaction it knows that no other transaction and no other thread is running at the same time to touch that same data and therefore you don't need any locks and latches at the lower stuff, more at the fine grained level. So that allows you to run really fast.

What about cache misses?

There are no cache misses. I mean are you talking about L1, L2? Those are so fast. Having to go to DRAM it's significantly faster than having to go to disk, so those aren't really a big issue for us.

And what part of that giant system did your thesis focus on?

The thesis itself wasn't just, "Hey, we built this system, ta da", right? The fundamental part was "Hey, there is this new architecture, we can do better than a traditional system". Then going beyond that was, "What are all the problems where this doesn't work out?". So the rest of the thesis is saying and identifying, "Well here is the issues we have when working in this kind of operating environment and what we can do to fix it".

Three main parts came after the basic system design. The first one was coming up with automatic techniques to take an arbitrary application and figure out the best partitioning scheme and how to split it up across a cluster of nodes so that you maximize the percentage of the transactions that only touch data at a single partition at a single node. Then you avoid any of the slowness involved with two-phase commit like Paxos.

The one thing that I think I realized in grad school that made me successful is really focusing on this single project.

The second part is related to multi-partition transactions. It happens that for some applications you are not able to get rid of these distributed transactions, these multi-partition transactions entirely. This could be either because the application simply does not partition in a good way (the Twitter/Facebook example is a good one), but also might be because of weird legal reasons. So to give an example, we've visited PayPal in their early days and they had this weird legal restriction where customer accounts from different countries could not be on the same physical hardware for some reason. So that means if you were using a system like H-Store, this would never work because that's always going to be a multi-partition transaction. And so the second piece of the thesis resided in using machine leaning techniques to figure out, when a transaction request comes in, if it's a single partition, a distributed transaction and what partitions it actually needs to touch so that we only need to allocate or lock the bare minimum of resources we need for that transaction. Then while it's running, we can identify when it's done with those resources, go ahead and release them and let the next transactions to start running. This is sort of doing an early two-phase commit process. So that was the second piece.

We've tried to minimize the number of distributed transactions, we have identified when the transactions come in and whether they're distributed or not, but we still have these distributed transactions. We have these points where the execution nodes in the cluster are idle because they're locked by some other guy in another node and therefore they're waiting for the next request to work on, and they're sitting and doing nothing. This is because we're using this serial partition level locking scheme. So the third piece of the thesis is a speculative execution technique where we identify when we're idle at a remote node because of a distributed transaction. We go ahead and peek in our queue for that partition and try to pick out transactions that commute with what work the distributed

transactions have done so far. We show that by using these machine learning techniques that we use for the second part, we can identify transactions that will finish in enough time and won't interfere at all with the distributed transactions. So everybody can do a root commit in the end and we'd all claim we don't violate any isolation consistency guarantees.

Do you know anything now that you wish you had known when you were a grad student? Or during your job-hunt?

Well, that's two questions… For grad school, there was this really critical point in my second and a half year where I needed to decide whether I wanted to go and continue building H-Store (the academic version of the system) or whether I should just leverage what the VoltDB guys were doing in the commercial side. The H-Store relationship with VoltDB is kind of incestuous where we were separate projects that came together as one project then we separated again then we came back together. I thought I was going to go forward using everything of VoltDB, but my advisor to his credit, he said, "look I really think that you'd be better off from a research standpoint, if you did another fork, pulled back some changes from VoltDB in the H-Store code but then rewrote a lot of the stuff that you need for research". This was a hard choice, right? Because this means that for two years or so I'd be writing a lot of code and not getting publications done, but he really pushed me to do this. In hindsight it was the right thing to do. To his credit, he really stood by me for those two years when I was not getting any publications done because I was spending all this time writing the system. He just let me go and do my thing and thankfully, it all worked out. So I'm very grateful for his faith in me in pulling this off.

The one thing that I think I realized in grad school that made me successful is really focusing on this single project. A lot of times I see grad students that are focusing on different things, different projects, and when it comes the time to go on the job market, whether it's an academic or industrial position, they have this tenuous or weak connection of trying to say, "I did this project and this project that are all together in the same package", right? I think it's kind of transparent. Whereas in my case, I was able to go on the market and say, "Hey look, I built the H-Store system. I'm the H-Store guy. Here is all the work based on a single system". I can talk to length about any part about the system because I spent so much time working on it. I credit this idea from Dan Abadi, who's now at Yale, but he was at MIT working on the C-Store project, which was the predecessor to H-Store. So when he was in the job market, whether he knows this or not, he was the C-Store guy and he was really successful in that regard. So I try to emulate that or copy that idea of being the H-Store guy when I was in grad student and I think I was pretty successful with it.

As far as what I wish I knew now about being in the job market… Stonebreaker told me when I got invites to go interview at some schools, at IBM Research, Intel Labs, really awesome research places. He basically said, "Look, you are the only database systems person on the market. Your job talk needs to have tons of graphs. Lots of jokes, lots of graphs". So my job talk was essentially about my gambling addiction at the greyhound dog track. So I went this whole thing about how I go gambling, go see the dog track, and from that, on how it gave me ideas on how to make my database run faster. Actually I don't go to the dog track, and greyhound racing is deplorable. It was a joke, right? And only at one place, one guy thought the joke was real. Everyone knew, that guy is just joking that's fine. There was one guy at IBM who came up to me afterwards, "Hey man, I really like going to the dog track. When do you want to go?" I was like, "no, you totally misread that". I didn't start grad school thinking like, "oh, I absolutely need to go to Carnegie Mellon or a top school like that". I didn't set out to do this. I was just around some really smart people who had a lot of good guidance and I think I worked pretty hard to build this thing and everything worked out.

I've only won two awards in my life, okay? I won class clown in high school and then I won this. And you know what? I may never win another award ever again, so I'm going to relish this as much as possible.

You described it as a big project and you're the only systems guy, does that mean you had like a dozen bosses?

No, so the project started off being myself, another PhD student's at Brown, and two PhD students' at MIT. It was a lot of people at the very beginning. This is like 2007, 2008. We all worked for about a year building the core basic system out. Then around 2008, they went out and forked the code and made VoltDB

(the company). Then all the other PhD students went off and did other things. I, myself, went off and did some stuff with the MapReduce with Stonebreaker, David Dewitt and Sam Madden. All the while thinking that VoltDB was going to add the stuff that I needed. They kept saying, "oh, next quarter, next quarter we'll do it don't worry". Then finally when I went back after doing the MapReduce stuff to go work on H-Store again, I asked, "Hey I need *x*, *y* and *z*, are you guys going to do it?" and they said, "No, this is not what customers are asking for. We're not going to do it". I'm not faulting them, that is a business decision and that's fine. That was the point where Stan was like, "look you should go and fork your code and do your own thing". That's when I did that. When I went back to the system, everyone else was gone. I worked with Evan Jones a little bit but he was sort of off doing the relational cloud stuff up at MIT. So about a year or so, I was trying to cobble together whatever resources I could get, masters students, undergrads, all at Brown to try to help me build this thing…but it was a lot of time, a lot of late nights. I did eventually borrow some code from VoltDB but a lot of the core transactional stuff was rewritten (twice actually) from scratch. I don't recommend it. It probably was not a healthy lifestyle and it was certainly not sustainable. I'm not keeping that same pace at Carnegie Mellon because I have other things to work on. But yeah, it was a lot of code in over a two or three year period to make this all work. It wasn't just me, but for that one period I got to get whatever resources I could to make this work.

Thank you very much for talking with me today.

Thanks for having me. It's a blast. Thanks.