Sudipto Das Speaks Out on Scalability and Elasticity of Database Systems

Marianne Winslett and Vanessa Braganholo

Sudipto Das

http://research.microsoft.com/en-us/people/sudiptod/

Welcome to ACM SIGMOD Record's Series of Interviews with distinguished members of the database community. I'm Marianne Winslett and today we're in Snowbird, Utah, site of the 2014 SIGMOD and PODS conference. I have here with me Sudipto Das, who is a researcher at Microsoft and the recipient of the 2013 SIGMOD Jim Gray Doctoral Dissertation Award, which is for his dissertation entitled "Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms". Sudipto's PhD is from the University of California Santa Barbara where he worked with Divy Agrawal and Amr El Abbadi.

So Sudipto, welcome!

Thank you very much!

Tell me about your dissertation

So before I get to the actual dissertation work, let me give you some background of where I started. It was about 2008-2009 when I got started in this area, early in my PhD program. There was a lot of buzz about Cloud Computing, NoSQL, and key-value stores. The Google BigTable paper had come out, Dynamo from Amazon had come out, and there was a lot of buzz about key-value stores and database systems being dead, RDBMS are bad, and whatnot. So my advisors, Divy and Amr, suggested to try to analyze why the key-value stores were so successful and why people were saying that RDBMS were bad. We then realized that you can look at it as a spectrum where on one axis you have consistency guarantees or ACID properties, and on another axis you have scalability. Key-value stores were very high on the scalability axis, but very low in terms of guarantees. RDBMS, on the other hand, have provided very good guarantees in terms of transactions, in terms of access methods, and whatnot, which key-value stores lacked.

The question that we wanted to answer as part of our work was: is there a way to bridge this gap between the key-value stores and relational databases? One of the key insights that we derived by analyzing all these key-value stores, looking through it, and brainstorming about it, is that the key fundamental idea was to limit most of the accesses to a single server or single node, to avoid a lot of distributed synchronization. Distributed synchronization was still being used in a lot of these key-value stores, but they were using it very judiciously. So with this abstraction and with this learning, we tried to see what transactional abstractions could be carried forward while limiting accesses to a single server. So you get all the good properties of key-value stores: you can scale out, you get elasticity, you get high availability, but you can still provide transactions at certain granularity.

Essentially, the first half of my dissertation looks into two different ways of designing such systems. One way is through a statically partitioned database, where you define a specific schema pattern. It is a hierarchical schema pattern that is actually explored in a number of other systems as well. We show that if you have such a hierarchical schema pattern, and if your transactions adhere to and only access a given hierarchy, you can provide efficient transactions while scaling out similar to key-value stores. The other abstraction that we were looking at was, what if these

“The PhD is your entire life compressed in 5 years. You’ll see ups and downs throughout your life. You’ll see ups and downs throughout your PhD”.

Divy Agrawal

partitions weren’t statically defined? So think of it as if an application comes in and dynamically specifies that “here is a bunch of data items on which I want transactional access for a certain period of time.” Once such a declaration is provided, the system takes on the responsibility of the transactional access on this group of items during a certain span of time, after which the application says, “I don’t need it anymore,” and the system is free to do whatever it wants. So we came up with an abstraction called they *Key Group* abstraction, where essentially an application can come and specify a group of data items. We do some distributed synchronization to localize accesses to the data items during the lifetime of what we call the Key Group. Transactions execute on the Key Group. We take on the responsibility of doing it efficiently. Once the application says the group can be deleted, we take on the responsibility of propagating the updates back to the original servers that hosted the key-value pairs, from which the application has formed the group, and life moves on beyond that. So it’s a way of dynamically defining your partitions on which you want transactions.

After doing the transactions part, one part that was still left was elasticity. In the key-value stores, one of the key selling points is elasticity and so is that of cloud. So the next question that we started to answer was that, can we make (classical) databases elastic as well, while executing transactions? One of the key mechanisms that was missing was this concept of live (database) migration. Elasticity essentially means that when the load goes up, you add a new server, your data spreads out to the new server, and your new capacity gets used. And this is all happening while the system is running, causing minimal disruption to the transactions that are executing.

One of the key mechanisms that enable this is what we call a live database migration. A database is on server *A*. At some point in time, a (system) controller decides that it is time to move this database off to do elastic scaling, and it initiates this live migration while transactions are executing with as little disruption as possible. On the fly, this database gets moved from server *A* to server *B*. The new transactions get routed to server *B* and from the user's perspective it's minimal disruption. And this can happen the other way around -- when the load goes down, the servers shrink, you have consolidation. That is, it helps both ways. So essentially what we developed was two different mechanisms for providing live migration in two different database architectures. One was a shared-storage architecture, where the persistent data is stored in a decoupled replicated storage system, like HDFS in our case. In another case we were doing replication for shared-nothing, where the persistent data was stored in a locally attached disk on the server itself. So during the course of migration, in the first part, we were just migrating the hot cache, so the read and write transactions would see minimal impact at the destination, whereas in the second case, we were actually migrating the persistent data as well, along with the state of some of the transactions. So the first part [of the thesis] talks primarily about scalability and the second part talks primarily about elasticity. And it is the transactions and cloud platforms that ties them together.

Have you seen industrial interest in that?

That's a great question. There is definitely industrial interest in this area. When we were developing these ideas, a number of key industrial developments happened concurrently that kind of resonate on similar ideas. If you look at Microsoft Azure (SQL) database, which used to be called SQL Azure back then, they had a similar notion of a hierarchical schema, limiting transactions to a single node, and then being able to scale out to a cluster. Google's Megastore has taken a similar direction, where there was, again, a hierarchical schema statically defined, and then transactions were operating on that hierarchical schema. So all of this happened concurrently while I was working on my dissertation with my advisors and my colleagues. So I wouldn't say it was an impact to this line of work, but at least it was gratifying to see that indeed these ideas really work out in practice, and things which have similar insights are actually being deployed in production.

In terms of academic impact, there has been a lot of follow-up work. Especially in migration and in transactional key-value stores, people have followed up on our work. There are a number of citations that we have received for our papers. As far as I know, I'm not aware of any system that directly implements my ideas, or the ideas that were developed in the dissertation, but we never know what's under the hood for a lot of these commercial systems.

So true. Is there anything that you know now what you wish you would have known during your PhD studies and job search?

That's a very tricky question. Let me try to rephrase it a little bit. What I would try to answer is what I've learned during the course of my PhD that actually helps me out, even today. So, there was one thing my advisor, Divy, told after my first paper got rejected. I was obviously very depressed. As a fresh graduate student, I had very high hopes that if I did good work, who would stop it from being published? So Divy gave me a punch line saying that "The PhD is your entire life compressed in 5 years. You'll see ups and downs throughout your life. You'll see ups and downs throughout your PhD". Failures are a part of it. You have to deal with failures. I think, one of the key things I learned during my PhD was to deal with failures. In research, in different parts for getting my research accepted in the broader community, etc. I see that this is very true in other aspects of life as well. This is something that has really helped me get through many situations, helped me understand different scenarios, and overall improve me professionally. That is more from the philosophical point.

From the technical perspective, what I learned during the different projects that I was working on and also during my internship, which I did with Phil Bernstein at Microsoft Research, was the value of having good experimentation for systems research. I even have done this on some my papers as well, and I try not to repeat that. Sometimes we have an idea for a system. We try to do experiments that validate only our idea. As soon as we have that validation, we often write the paper. It is good in some sense to have a validation of the idea, but it doesn't provide others with the insights that they need to maybe apply the idea in a different context. Or maybe learn from what worked or didn't work. So essentially what I try to do now or what I tried to do in some of my later papers in my PhD is to have a more thorough experimental study that analyzes the different tradeoffs of the system. This helps me, as well as others who read the paper, to understand the key insights in addition to what is actually being publicized in the paper. So it helps me understand the system better. It helps me sometimes to optimize the system even more. I hope it helps the readers of the papers to get a better understanding as well. So this is something that I'm sure I'm not there yet. I probably need a lot more practice, a lot more nurturing, a lot more work.

I think a very critical part of doing systems research is to have a good understanding of the dynamics of the system. There are probably hundreds of different ideas that are being proposed in many different papers. But then if we step back and try to make sense of what can be abstracted out and applied to other contexts, it becomes really hard. That's why I think the VLDB experimental track papers help us a bit in that direction, but I guess the initial papers can also do a better job in that.

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

Thank you! It was a great pleasure. Thank you for having me here.