Andreas Reuter Speaks Out on Transactions, Reinventing things, Creating a University, and more

by Marianne Winslett and Vanessa Braganholo

Andreas Reuter
http://www.h-its.org/english/homes/reuter

Welcome to ACM SIGMOD Record's series of interviews with distinguished members of the database community. I'm Marianne Winslett, and today we are in Indianapolis, site of the 2010 SIGMOD and PODS conferences. I have here with me Andreas Reuter, who is a professor at the University of Heidelberg[1] and the director of two research institutes in Heidelberg. Andreas was a freelance programmer for more than ten years. He is the coauthor with Jim Gray of the famous textbook Transaction Processing Concepts and Techniques[2]. His PhD is from the Technical University of Darmstadt. So, Andreas, welcome!

[1] When this interview was conducted (2010), Andreas was at Kaiserslautern University.

[2] Jim Gray, Andreas Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann 1993, ISBN 1-55860-190-2.

How was it to write that book with Jim Gray?

Oh, it was a once in a lifetime experience. It was very strenuous. We had a lot of fun. We had many debates, arguments and so on. I think the whole atmosphere is best summarized by what Jim said right at the end of the writing experience, "I'm totally stunned that we didn't kill each other!"

How long did it take?

It took about four times longer than we anticipated. We wanted to write a book covering transactions, databases, everything in about 600 pages. We had a set of slides prepared for a course that we taught in Berlin in the late '80s. And we said, well, we've got all these slides, we just have to turn them into prose, it shouldn't be too hard to do, and so, let's do it. So, Jim went back to America, and I stayed in Germany, and nothing happened. After 2 years of nothing having happened, we decided that if we ever wanted to get this book out, we should do something about it. So we arranged to have a writing assignment for two or three months in Italy.

It was a long argument where we should go in early spring, between middle February to April. Jim said, "I don't want to go to Germany. It's cold, it's raining, and it's terrible. I'm from California. I need it warmer. I need sunshine." So we rented a place in northern Italy and we started writing. And we arrived in wonderful weather; it was sunny, bright skies. It was pretty cold, but bright skies. One week after we arrived, it started raining and never stopped again. We wrote. We had a couple of power failures; the power system is not overly reliable. Jim got more and more upset about this whole thing, the unreliable infrastructure, and no telephone, it was the late '80s. So we decided to pack things, and drive back to Germany, and in Germany, it was warm spring. That was the irony. After these three months, we had about half the subjects covered. We decided to have another writing assignment the next year. And Jim said, "I've had it in Europe in spring, this is all nonsense. We do the next thing in California! I'll rent the place." So he rented a place in Bolinas, north of San Francisco, about the same period of time: February to April. We arrived, and it started raining, and it never stopped.

First build the code, and then write the paper.

And it was foggy too, up there?

Yeah, it was foggy, but it was more rainy than foggy. If there is enough rain, then there is no fog. Close to when we finished up our writing in Bolinas, there was an announcement on the news that the drought was over. That sort of added to the general tension of getting this stuff organized, and getting the right angle on things, and so on. We decided by the end of the Bolinas assignment that this was it. Whatever we have, this is the book. And we hadn't covered all of the subjects we wanted to cover, there is no SQL, there is no database design, there is no application design in the book, which originally was part of the plan. But we just said, well, that's it. It's 1100 pages now, and the rest is sort of editing, and polishing, and doing the index, etc.

But back to your original question, how was it? It was, as I said, technically, and from an emotional perspective, from a human perspective, one of the greatest experiences I have ever had. We necessarily grew very close to each other, just through the narrow confinement of this writing closure. There was nobody else. It was Jim and I for about 3 months. Once in a while, Jim's wife would come up, but she came from San Francisco, stayed overnight, and went back. My wife was back in Germany, and so, there was just nothing else going on 18 hours a day. That was roughly the period of time that we spent per day.

Is there life after transactions?[3]

That's a good question. Our motto through the writing in Italy, and then in Bolinas more so, was Jim saying at the end of the day, or around midnight or whenever we stopped writing, he would come to my room and say, "Reuter, stop – let's go out and see if there is life outside transactions!". And we used it in various contexts to make fun of other things. Technically speaking, I think the main reason for writing the book from Jim's perspective was of course, that he wanted to sum up everything he had learned about building distributed systems, building online systems, and without the notion of transactions it would be impossible. So, in that sense, transactions are a necessary elementary particle, if you will, in establishing such systems. If you don't have transactions, then all the architectural principles just fall apart.

We thought about adding a chapter or a subchapter somewhere on comparing transactional approaches to other approaches using non-transactional concepts, but we finally decided against it. Because, again, Jim in some points was very strict, and he said we only write about this stuff, we don't write about that stuff. From his perspective, non-transactional systems were just bad stuff, and he wouldn't mention it. What we tried to do of course was to make the point, from the technical perspective, that transactions facilitate a whole lot of architectural issues that would have to be solved in more tricky ways otherwise: transactional RPC and all these things. So, if you take the perspective of the time the book was written and from the background of the people who wrote the book, and the people whose work we were drawing on (it is not that we had invented these things): No, there wasn't life outside transactions, really.

But of course, transactions weren't all of life. It is like the basic processes enabling biological life; there is metabolism, there is replication, and some essential protocols that have to be implemented in order to make life viable. When you look at a cell, or when you look at a mammal in isolation, you will find, just by looking at them, very little in common between the two. But if you look at the internal plumbing at the cell level, you will find a lot in common. From our perspective, transactions establish this elementary set of protocols, plumbing, that make very complex distributed systems possible.

What about those people who think eventual consistency is good enough?

That's one of the approaches that you have to take when simple transactions don't help you anymore. But then of course the question is, who rings the bell: Now is the time to talk about consistency, or not. Of course, if you use proposals that have been described in the context of more generalized synchronization protocols, absolute serializability and Escrow-type of locking,

[3] Andreas Reuter. Is there life outside transactions?: writing the transaction processing book. SIGMOD Record 37(2): 54-58 (2008).

etc., this is all the flavor of eventual consistency. Let things get a little bit out of hand along the way, as long as correctness is maintained at the end, its good. The question is, what is the end? Who calls the shots? If things go on continuously, and there is no point when you can say, “well, stop”, then it doesn’t really help you. You have to make sure that the differences don’t accumulate. Once you have to introduce these points of control artificially, well, then you have something like intervals of consistency checking and that’s one aspect that transactions have, originally. But, it always comes back to the ACID properties, which in the original transaction model were captured by one system construct. When you wanted to do more complex things, it turns out that you couldn’t get all of them. You could get, maybe, correctness, but then there goes isolation and atomicity. Or you could atomicity but had to sacrifice consistency, etc.

Where will transaction processing be in 20 years?

Well, if the people who say they don’t play any role anymore are correct, then well we will see transactions in the computer history museum, in some niche, in some corner. Otherwise, I think transactions will still be there, very far down in the system, in the form of transactional memories, which have a good chance of becoming part of many future processor architectures, I assume, especially in multicore environments. Because multicore means you have to raise the level of parallelism to levels that have been unheard of and unmanageable, at least so far, without putting too much burden in terms of controlling parallelism in the program. And so you need system artifacts in between – between the application program and the actual hardware – that will take care of all the hairy parallelism issues. And again, some of the ideas that transactions have incorporated will play a role there: making things atomic; making things automatically recoverable within a certain distance of history, by invalidating some cache lines or through some other mechanism – I can’t go into any detail here. But I think that’s where transactions will play a role. Reliable message delivery, that’s where transactions, or transactional protocols and the idea of atomicity will still be needed. If they will still be called transactions, I have no idea – most likely not. People will not be talking about that.

Reuter, stop! Let’s go out and see if there is life outside transactions!

In computer science, are we always reinventing the wheel?

Yes.

Is it a bad thing?

No, it’s not a bad thing. I mean, when you go from a triangular wheel to a quadratic wheel, this is progress. Then you take this idea further, and things get smoother, and smoother, and smoother. It’s fine. Another way of saying we are reinventing the wheel is that we, in many fields, go around in circles: At any point in time, there is a focus on a specific set of questions, mostly dictated by current technology, for example, the balance between memory and I/O, or between CPU and memory. This is what is given, basically. There’s no such thing like a completely virtual machine; in the end it runs on real hardware. So people start trying to use the hardware as best that they can, and that influences architectural decisions all the way up. Once these problems are solved, and we have a reasonably good performance platform, you build applications on top of that, which take into account what the software underneath does, and so on. After some time, when everything works, the hardware is changing, or has been changing all the time, which could be ignored for a while, but now it becomes obvious that you have to do things differently if you want to exploit the resources optimally. Or your performance requirements have changed, you want to do more with your system, you want to do it faster.

Ten years ago, data mining or data analytics or OLAP were offline activities. You created a warehouse from your online database, and then you had these wonderful OLAP tools scanning the warehouse and telling you things. Now people want to do it in real time, on the real online data. They want no warehouse reflecting the world two hours ago. They want to analyze the online data, and not just the data as they are stored, but they want to analyze the data as they are stored plus the incoming data streams, and compare this to other data streams that other people have and they have access to. So that shifts the set of requirements, and in order to make that possible, you have to look at the hardware again. Given the new hardware that we've got now this requires significant changes of the software; we have to reinvent some of the basic algorithms. The search structures that we have come to love and optimize in databases, do they make sense in really huge memories? Or what about the balance between I/O and buffers in databases, just to stay in this field.

When we wrote the book, there was this famous 5 or 10-minute rule. If somebody referenced a piece of data once in every 5 minutes, then keep it in main memory otherwise, put it on disk. Now we are talking about 3 hours for the same criterion, and it's still going up. This has huge implications on the way you maintain your data structures. In the old days, query optimization was looking at a query, but right now, especially when you look at these data streams, you have to appreciate that at the same time, you have to process hundreds, thousands, millions of queries, roughly on the same data set. Now, what does optimization mean? Well, it is clear what optimization means, but how do you do it, and how do you support it?. And that's what I mean by going around in circles. You start rebuilding the mechanics, the basic engines of the system, and then of course you go off to offer new services, with new constraints for the applications, and get to build applications in different manners – more real-time things, more this, and more that. And I think we have been through the cycle at least 3 times in the history as far as I can see, and right now we are in the start of a new cycle, because people are really building new engines for databases. Not just adding new object types and new blades or whatever: they are rebuilding the engine.

Because of the multicore aspect?

Mostly because of the multicore aspect, because of the huge increase in memory size, and because of the shifting balance between memory bandwidth and I/O bandwidth.

Is the community paying enough attention to those things?

Oh, yes, absolutely. That's what I really take away from this conference, that there are enough people in the technical community who are just happy to be plumbers. They don't need all of this fancy stuff, you know, analyzing trends in social communities, or discussing Hadoop versus parallel SQL. Instead, they really focus on the low level technical issues, and ask: How can we exploit the new hardware that we get, and make databases run optimally on that hardware? Think, for example, of big applications on the database at petabyte sizes, which in parallel to maintaining this huge set of data, look at a large number of streams with high bandwidth. We will talk about systems with millions of cores. These systems have to be self-optimizing, self-configuring, self-everything, because no human will be able to "understand" what they are doing. And there are no tools right now to help humans in setting up such systems. Of course, can't do it all themselves, this would be wishful thinking, but getting from systems that we know and can manage today, to those systems, will take a major leap in a couple of dimensions. People are excited about this, and at the same time, people are coming with interesting requirements from new applications that they want to do with their data. Whether the result is called a database or something else, again, what's in a name? But the technology is new. This seems to be going on with considerable vigor. So, it's good.

These [million-core] systems have to be self-optimizing, self-configuring, self-everything, because no human will be able to understand what they are doing.

Why don't database people use databases?

I am not sure if this is still true. It depends on when, in the course of a day, you look at what a database person does. When database people do their research, or do their development work, they probably use a database, because that is what they are working on, that's what they are paid for. The question came up repeatedly, in the so-called old days, when databases where, as a matter of fact, fairly hard to install, hard to manage, and expensive, compared to what they could do for people who didn't have just a large number of routine jobs, like the classical OLTP type. OLTP was characterized by a small number of repetitive applications, called transactions. But for the stuff that people normally do when they are alone in the office, doing ad hoc things, that's what databases typically weren't very good at, and other tools, like Excel where much better at. Since people normally, for their own purposes, maintain some addresses, some references, they don't need big stores, so that's not a big deal for them. They just use whatever PC tools are available. So it is a simple trade-off in terms of productivity. But this doesn't say that database people don't practice what they preach, it is just the fact that for a long time databases were built for large routine applications and individuals don't handle large routine applications. They do very many different ad hoc things, and so that was not the design space for classical databases.

What was it like when you started a new university?

This was a delusional thing. I say this in hindsight, of course. You have to be very naive and very optimistic, but naive I think is the better word, in order to do this. The reason was that I was vice president of the State University in Stuttgart for a couple of years. I had come to understand a large number of things, many more than I ever cared to learn about, of why state universities in Germany had so many difficulties, problems in running up to their real potential. It had to do with their fiscal structure, with their internal legal structure, that their administrations were owned by the state government rather than independent bodies, etc. And at that time (it was the mid-90's) there was a climate in Germany when people all over the community started to recognize this and said we need to do something, the university system has to be reformed. And I said well, if we focus on some efficient and interesting areas, why not set up something like a model shop, for certain aspects of academic teaching, and try it out, and see how it works. And of course, you couldn't do this in the public sector, because these decision processes just take forever. And there was a chance to set up a privately funded university.

Even the state government was mildly supportive of this, which is important, because they had to okay it – all the degree granting things are a monopoly of the states in the German system. The minister in charge of this at the time publicly said, "we support these private universities because we hope they will be a thorn in the flesh of the public universities" - which didn't help us a lot. It is a strange thing for the minister to say, but he did. And I really had the hope that this would create some kind of more open debate on which way to go, on evaluating different concepts, etc.

But, anyhow, we started the thing[4], which was the first German university to teach entirely in English, and the first German university to use the bachelor/master degree system. The public universities had a completely different one at that time. And, well, I think I never worked so much in my life as I did during that time. I taught classes, about four times the teaching load I had at the state university - simply because we were short of people. We really started from scratch, and initially there was almost nothing. We defined programs in IT and in the business area. And then the curriculum had to be implemented. For everything where we didn't have people, which happened from time to time, I had to jump in and do it myself, besides the other duties. We came off the ground fairly well. There were some evaluations, some rankings, which came out quite favorably.

But, what dominated the whole project throughout was the political debate. This university was always a token in political combat between different sides. You can ignore this for a while, but over time you will be confronted with so many problems that have nothing to do with what's happening in the institution, with the way the institution is set up, with the plan you have. It's just political ambushing at different levels and at different times that I felt I spent just an excessive amount of time besides the real work I did for this institution by just coping with the side effects of political intrigue. State universities are all for free, there was no tuition, and we charged, I don't know what the exact price was, $15,000 a year. And so you had to convince people: why should they pay such a high amount of money, when something like this is for free next door? You have to be much better, you have to be more efficient, etc. And you need sponsors, corporate sponsors. Those corporate sponsors are completely willing to support such projects, but once the institution gets into some kind of public conflict, or some kind of public debate with negative overtones, than corporations are very quick to withdraw. They say, "Oh, we don't want to be associated with this". "Sorry, no hard feelings, we know you do the right thing, but it's just the way it appears on the outside".

[4] In 1998 Andreas Reuter co-founded the International University in Germany.

Just preventing this from happening, and getting new sponsors, and some sponsors who withdrew, it was so tiring that, in 2004, I decided to just leave in the interest of my own health. I learned a lot from this experience. And I don't want to miss it, but I understand now that when I started this, I was just very naive. But on the other hand, there was very little experience to draw on. The good thing is that we and some other private universities that were started roughly at the same time - one to three years later - they had the net effect of changing the structure of the public universities, so they now have international programs, they're all moving, with a lot of opposition, to a professional master's system instead of the old system. So this model shop character I think was something that we really achieved, but the price was a bit high.

Is it a bad thing [to reinvent the wheel]? No! When you go from a triangular wheel to a quadratic wheel, this is progress.

Sounds like you were a thorn in the flesh at that time.

Right, and I had a lot of thorns in my own flesh.

Do you have any words of advice for fledgling or midcareer database researchers or practitioners?

I think I don't have some overarching perspective that could cover everything. From my own experience, I should say it's very important that you don't lose contact, in no phase of your career, with real problems, with real applications. Databases is not a field that is dominated by theory. You need the theory, but in the end, we need to build systems that help solve application problems. And so, in regular intervals – I don't say every day, I don't say every year – in regular intervals, you should get your fingers dirty by building something, by implementing stuff. Then see how the system behaves and derive new questions and new problems from that. It should be a continuous circle, between building things, evaluating things, and thinking about better solutions, better ways of doing stuff. I think it was Jim Gray again who said this in an unmistakable, concise way: first build the code, and then write the paper.

If you magically had enough extra time to do one additional thing at work that you are not doing now, what would it be?

Well, I think the perspective I am taking is from my current positions in the research institutes I am heading. One of our main purposes is to provide infrastructure, IT, database infrastructure, computing infrastructure, for people who are doing theoretical research in life sciences, in physics, and in other fields. In none of these fields, I have a good enough understanding of what their data management problems really are, so if I had extra time, I would pick one of those fields, let's say systems biology, go into it a little deeper than I can at the moment, and just understand what kind of data management problems, data analysis problems do they have, and how could that be supported by the technology that we have.

If you could change one thing about yourself as a computer science researcher, what would it be?

If there were an institution where I could buy or rent or whatever, more patience, I would go there and try to get some. Because, I think I am a little bit too quick in abandoning a problem.

But in computer science, patience doesn't really pay off in general, does it?

I think it does.

It changes so fast.

Well, I don't agree completely. Many of the concepts that we are talking about are fairly stable. It requires patience to really filter out the stable, the longer lasting insights and concepts from all the barrage of new words that are coming out. Of course we are changing words to denote the same thing, and again, we are not the only discipline doing this. I know others, which are about as bad. And we are changing words faster than we change concepts. Of course, one can very easily be mistaken by a new word meaning something entirely new. In many cases, it is just all the same old stuff in a new wrapping. To get to the bottom of these things, and understanding whether they are conceptual issues as opposed to whether they are just wrapping issues – that requires a certain amount of patience. I admit having been taken away by new words, saying, oh "I shouldn't miss this", and then after some time, I say "oh #$%@, this isn't so new after all", and then I jump to something else again. It is part of my personality, I'm afraid I can't change it, because you know at my age, I can't change anything at all. Had I been more patient, consistent, I might have ended elsewhere. But, that's the way it is.

Thank you very much for talking with me today.

Thank you, Marianne. It was a real pleasure.