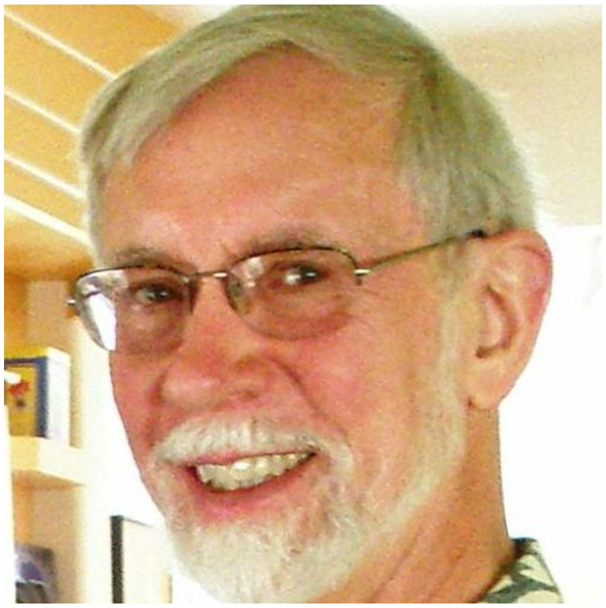
David Lomet Speaks Out

On database recovery, logs, versions and more…

by Marianne Winslett and Vanessa Braganholo



David Lomet
http://research.microsoft.com/~/lomet/

Welcome to this installment of ACM SIGMOD Record Series of Interviews with Distinguished Members of the Database Community. I am Marianne Winslett, and today we are in Providence, site of the 2009 SIGMOD and PODS conference. I have with me today David Lomet, who has been a principal researcher and the manager of the Database Group at Microsoft Research since 1995. Before joining Microsoft, Dave worked at Digital Equipment Corporation, Wang Institute, and IBM Research in Yorktown. Dave's research interests lie in access methods, concurrency control, and recovery. He has received two SIGMOD Best Paper awards, and he is an ACM Fellow and an IEEE Fellow. Dave's PhD is from the University of Pennsylvania. So Dave, welcome!

Thanks, it's nice to be here.

Dave, I read that you are one of the people who invented transactions. I have to ask – how many people invented transactions?

Well, it is really very hard to count. The first people who really used something similar to a transaction were the IMS folks -- Ron Obermark, and the IMS program isolation feature. But even before that, there was old master and new master tapes, where in fact the rollback was the old master and the committed transaction was the new master. So, there is a lot of people who played a roll. I went to Newcastle [University in Britain working on Brian Randell's Reliability Project] early in my career, and they had something called recovery blocks, which was a mechanism, but it actually implemented transactions. So I sort of provided the abstraction there for that. Jim Gray and his colleagues in San Jose provided the abstraction for the IMS program isolation facility. So, there are many inventors of transactions, but I was a contributor.

You have over 40 patents. So you must write patents at the rate that many of us write papers. Can you compare the effort involved?

It is much more tedious, actually. It is not as much of an effort as writing a paper, because, what I frequently do is I write the paper, attach the front piece patent form that goes with it, and hand that in. And that is sort of the easy part. That initiates the process. The more tedious part is having to interact with the lawyers, sometimes over an extended period of time. Some of the lawyers are marvelously good. In fact, one of them was so good that after he wrote up a description of the invention, he didn't discover a bug, but I discovered a bug having looked at his write-up. But other patent attorneys do what might be called syntactic transformation and don't always get it quite right, so it is kind of tedious sometimes. It is a mixed bag, but I think the patenting process is important if you are an industrial researcher because it is something which the company gets out of your efforts in some sort of a direct fashion, and so I believe that's a really important part of industrial research.

I think a lot of universities now would like the professors to be patenting at the same rate that you guys are.

It's expensive. It's tens of thousands of dollars. It's not a few thousands of dollars. It's tens of thousands of dollars, so it is expensive, so even Microsoft has a budget. We get budgeted for patents. So it is expensive.

Let's talk about wrong directions and missed opportunities. Where have we gone wrong in the past?

Well, I have a long career, dating back to a pre-database era in my career. One effort that I was involved with at IBM research was on a taskforce to take a look at an area called provably secure operating systems. It was an initiative by the government to make managing their multi-level secure systems somewhat easier by permitting people to run jobs at multiple security levels simultaneously on a time sharing system. But they were very concerned about the security aspects, and information leaking from the more secure side to the less secure applications. So they had a program to do provably secure operating systems, where they wanted to be able to prove that the operating system was secure. That was much simpler than trying to prove that it was correct in all its aspects, and security requirements were very well defined. But even so… So we went around, and we looked at a bunch of people who were doing program proving technology, a lot of interesting stuff. But we concluded that the prospect of any near term success in proving something as a large operating system to be secure was decidedly misguided. So we wrote a report. The end result of that report was that work in the area pretty much ceased. So that was one of my bigger negative accomplishments -- to redirect people away from that area.

All that stuff in the MOS databases must have come after that, because that was in the '80s. A lot of money went down that drain. Did people listen to your report then?

At the time, which was in the early '80s, yes. So people don't do provable secure operating systems now, surprise, I know. There is some work in multilevel secure database systems.

Not anymore, though.

Yeah. That's right, but there was some. It is a hard problem. We will probably revisit it again, and maybe at some point the state of the art of program proving will be sufficient to really do the job. My guess is that we're still not there yet.

So that would fall under the category of wrong directions rather than missed opportunities. Do you have missed opportunities also?

I've always been a big fan of something akin to capability machine hardware. It was explored in the '70s. I've also been a fan of strongly typed, low-level programming languages, in which you might be able to actually write an operating system or to write a database system in, but nonetheless, you had the type checking safety of the full mechanisms where there was no escape. We haven't pursued that. We haven't pursued that with any real vigor, and the end result is that I think we pay for it in systems which have mistakes that could have been caught had we just used the tools that we already know how to do. But we continue with these less optimal solutions, in part because we have a lot of historical code, and nobody really has the money or the time to go back and do it again, but that is a missed opportunity.

So are you a fan of these more secure versions of operating systems (like SELinux)?

I don't have very much exposure to them. I am sort of in the Microsoft community, and we have some efforts within Microsoft research exploring some of these directions, but they have yet to have real impact on the products.

You worked on product development, in architecture, programming languages, and distributed systems. Looking back, do you consider that time well spent?

I thought that all the times that I spent actually was pretty well spent. Even though actually not too much came from it, in terms of products (interesting commentary isn't it?). But, I learned a lot. I learned a lot from the developers. I met the developers, and that is really an important thing to do. Especially if you are in industrial research, because the more ties you have with the developers, the more likely you are to be able to influence what they do, and get your research into product, which is part of what we are supposed to be doing. And so I've always thought that… In fact, let me combine this with the answer to a later question which you might ask, which is my advice to younger researchers, which is: spend some time in industrial organizations, even as a visiting researcher or whatever. Spend some time with developers learning what they do, seeing what they do, understanding how they make decisions, and understanding the problems that they're facing and you'll be better for it. You'll be able to take that more into account in your research, and your research will be more well-grounded on things which might actually have impact.

For performance reasons, traditionally the DBMS code for recovery, concurrency control and access methods is tightly bound together. In your Deuteronomy project, you have been working on unbundling that functionality and pulling it apart into pieces. How can you do that now, when people couldn't before?

So it's not just for performance reasons that we do it. We really did not understand how to do it. We do it because the locking and the logging are both done when the code is in the page looking at the records. So you know exactly what page you are on, you know exactly what record you are dealing with, you can look at surrounding records, and you already have the page latched, so you know that what you do in that page is already protected by a lock. We didn't know how to split those things apart.

Don Batory had a project years ago called GENESIS in which he explored putting various architectural pieces of database systems into sort of an erector set of database pieces, which he put together. But he never succeeded in separating the concurrency control and recovery from the data management of this. "I had this blinding insight", he says, "that if you looked at a database kernel as a distributed system, and thought about as a maintaining the state on the log, and in the database, and that that state had to be reconciled, then you could apply the kind of architectural principles, recovery principles, that we had used in a previous work that we had done on distributed recovery". We could use that to in fact solve the problem of having to be able to pull apart the database kernel. It is still an open question whether or not we'll get performance which is good enough, which is one of the reasons why we have projects going on, and we'll have another one this summer on exploring this area, but we couldn't even do it before. We really didn't have the underlying technology and understanding to be able to do it before, and now we can, and we are going to see. We're going to see how it works out.

Let's talk about time travel databases: databases that keep all past versions of their tuples. Why would we want to do that, (I think many of our readers won't know why we would want to keep all that stuff) and can we afford to do that?

So let me do the "can we afford to do it" first, because that is really easy. We've got this capacity running out of our ears, right. We have disks sitting idle with unused capacity, and this is especially true for all OLTP systems, where people buy additional disks to get the access arms. So there is just this enormous storage which is going to waste. The wonderful thing about keeping history is that you can keep it, and it is relatively cold data, it is not accessed all that often. So it doesn't really interfere that much with the access patterns for the current data. So, keeping it is an easy thing, it's a cheap thing. It's frequently just unused capacity. So that's one thing. So why might we do it? Well people like to do run ups of business trends, people like to do analysis of stock trends, people like to do audit compliance (a very big aspect of this), so they have to keep historical data. I think my research has shown that you can actually do the transaction time database stuff with minimal impact on the performance of the current system. So, it's just a natural and it's a useful function that people will have, and they'll do even more with it once they have it. I'm hopeful, though nothing has been carved in stone yet.

In the olden days, how did the detailed data get from the production database to data warehouse?

People would run scripts, which would funnel the data (frequently from multiple sources), cleaning it up, bringing it into the warehouse, as a separate, sort of offline process, for the most part.

But they were lacking the level of detail that they will have now, with all the past versions?

You could always do what I've just described by keeping the log. The log has all the information on it. The question is one of convenience, and I remember Bruce Lindsay one time saying that the database is the log -- what we keep in terms of state is simply an optimization. So you can think about the transaction time stuff in exactly the same way -- it is an optimization. You can use it to do online backups, and have the additional facility of having it be a queryable backup. So there is a lot of things you can use it for. You can use it to recover from bad user transactions, and do that without the enormously expensive operation of point-in-time recovery, which requires you to install a backup, grow it forward, de-commit scores of transactions. And you can do all that, and having a transaction time database makes that easy. So there's a lot of reasons: functional reasons, infrastructure reasons, and it's cheap. The capacity is there anyway, and the cost is modest.

Ok, so, how do you keep all those old cold historical tuples from clogging up your hot pages, and messing up your performance on your hot pages?

You use the time-split B-tree, and you do page splits, and move them to historical portions of the disk. That works really easily. In fact, Mike Carey had a paper[1], probably in the early '90s, I think, on using multi-versions, and he described the number of properties of how you migrate versions out of the current page. His was a more generic statement of it, but the time-split B-tree does exactly what he did, move it in bulk. You move it periodically, and you keep the versions close for a while, but you age them out of the current database. And with the time-split B-tree you can index them as well. So it works pretty well.

Many companies desperately need dependable systems – for example, Amazon, with their website for purchases, Travelocity, and Wall Street firms. So why haven't we computer scientists been able to give them dependable systems?

Well, they have reasonably dependable systems. Most of what they have sort of grew organically. The way a lot of development works is people build a system, they try to get the functionality right, it starts operating… They then realize that they've got some sort of dependability problem, and so they start adding it. It sort of grows much like the genetic code. You know, systems fail, and they fix them, and they get more reliable over time.

Computer scientists have the luxury of stepping back and looking at the process and trying to understand what general principles work and what can guide that sort of activity. They have been doing that for a while. I mean, there were TP systems years ago that had a strategy called stateless applications, and you ran everything inside of a transaction. The line of work that we were pursuing doesn't require that you run everything inside of a transaction. In fact, it solves

[1] Paul M. Bober, Michael J. Carey: Multiversion Query Locking. VLDB 1992: 497-510.

some of the problems that running things inside of a transaction have trouble dealing with, such as, what do you do when a transaction refuses to commit? In our systems, because we keep applications persistent without using transactions, there is a place where you can write a programmable piece of code which will deal with those kinds of errors, whereas in the TP systems there tends to be no place where you can be sure that it's going to be stably available, because stability is guaranteed by the transaction mechanism itself. So we think that the Phoenix project, which is the earlier project that I did at Microsoft, is a way of dealing with some of these issues. And it's also a way of making the process a little less expensive, because, transactions are a rather expensive mechanism when what you really want is just to keep the application running across crashes.

So how do you allow the application to survive system crashes if you don't have transactions?

Well, it's like the redo part of transaction recovery. You log the nondeterministic actions that interrupt the flow of the application, and you make some assumptions, which of course you have to enforce, that applications are what are called piece-wise deterministic, which is to say they'll execute deterministically between these interactions that you've captured. Then, if the application fails, you can use the log, which has captured the nondeterministic activities, and replay those instead of the actual activities at the appropriate points when the application calls for them. And so, you can recreate the application state simply by the replay of the logged events against the re-executed application.

Is that transparent to the application developer in the way that the transaction almost are?

You can make it transparent. There are a lot of efforts that people have made, some of which are not entirely transparent, but what we did in the Phoenix project was, we intercepted messages between the distributed components of the systems. We logged those messages -- that was the source of the nondeterminism. That interception mechanism was all captured inside of the .NET remoting runtime system, and so all the user saw was the normal .NET remoting, and all our logging was done sort of transparently. So this was truly transparent to the application. The only thing they need to do was to declare that their application was to be persistent, and then put it in, if you will, the persistent directory, and then we made sure that it was.

Has that made it into any products?

It's a sad story; a sad story of almost, but the answer is no. We came very close. In fact, we demoed the system at one of Microsoft's TechFests, and a guy from the Gardner group was there and predicted that with 70% probability it would appear in a product by 2007. We didn't quite make that. But it was close.

Are database researchers paying enough attention to multicore architectures?

Oh, I think the answer is yes, they are paying plenty of attention to it. It is hard to get leverage on it, in fact it is a very hard problem, and we'll be working on it for a long time.

What do you think about cloud computing and databases?

So unlike the situation with multicore, where everybody has access to multicore machines, not everybody has access to a cloud infrastructure. So, it is much harder to do research on the cloud. In fact, in our research group, we really don't do research specifically oriented towards the cloud. My participation in cloud computing involved an effort to help a product group do something on a cloud database system, and that's become an offering for SQL data services. But it is very hard for people outside of industrial organizations to have access to the kind of infrastructure that you need to really do research on cloud databases. You can sort of play with the Amazon type of infrastructure, and things of that sort, but it is really hard to do a fully robust database system of the sort that you might want to have taken seriously as a database system. So cloud computing is much harder, cloud databases is much harder to do research on. So we probably aren't doing enough, but there are technical difficulties in being able to do it, which I don't really have any advice or suggestions as to how to deal with it.

Well maybe we should say, so what do you think are the main research issues for database computing in a cloud?

I think we've got to change the way we think about making transactions stable. I think we have to move to a replication model, not simply a commit the log and put it on disk, because the availability requirements are such that you need instant fall over.

Does that mean replicated logs, or do you mean replicating the things that are actually doing the work?

Replicating the database, so that in fact the only outage you have is the amount of time that it takes to switch you from one replica to the other.

Do we already know how to do that from Tandem?

We know a little bit about it, but that is not to say we know as much as we should. We know a little bit about quorum protocols, and things of that sort. But we're going to be living in a world where one of the features of the cloud is the intense desire to try to control the costs of the infrastructure; which means that people are going to be buying what we refer to as el-cheapo machines. El-cheapo machines have el-cheapo disks on them, and el-cheapo disks fail at somewhat higher rates than the sort of the higher powered, more expensive disks do. We need to be able to have the strategies for coping with those, and making failure operations be a normal event as opposed to an extraordinary event.

So do we also need to replicate at higher levels, or you are only worried about the storage itself? What about the servers themselves?

I anticipate that in a system where people are really concerned about having a high availability all the time, that we'll replicate the applications.

What role do you see flash memory playing in the database world over the next ten or so years?

Well, people are certainly working on it, and again, this does not pose the same problems that cloud computing does. I mean, people have easy access to flash. What they don't have access to is raw flash, but they certainly have access to flash disks. The trick will be to see how we can successfully exploit it. I've seen a number of papers in which we've started to make some inroads on the problem, but it seems pretty clear to me that there is a lot more to be done. With a little bit bolder re-architecting, I think we'll exploit it more effectively. Right now we are sort of nibbling at the edges, I think.

Well, I've heard people argue that you can't just insert into your memory hierarchy memory flash disk, that that's not going to work, because of the cost performance, and that it will always take way longer to write to flash than to read, so this asymmetry makes it a little awkward for many uses. So do you think there is a right way to use flash?

I think the simplest way, one of the ways you will see people using it initially is just as a replacement for disks, for high performance disks. Flash is much more expensive in terms of its cost per byte, but in terms of its cost per IOPS, it's in fact quite competitive, and in fact does a lot better than disks. You can in fact buy IOPS more cheaply with flash than you can with disks. So those people who were buying large disks and more disks than they needed for the access time may be tempted to use flash instead.

That makes good sense. So that, in the memory hierarchy, that's replacing the bottom rung, well, depending on what you consider to be the bottom rung, it is replacing one of the rungs by flash. Is there any obvious way to add it as an additional rung somewhere, or is it just too tricky?

One of the things that I think is a bit appealing is to implement (bringing it back to my own work) a time-split B-tree where you keep the current database in flash, but when you do a time-split, you move the time-split page, the history page, to disk. That sort of gives you plenty of capacity for the keeping of the history, but it gives you very high performance and high IOPS for the current data.

And still you are not treating it as a cache for the disk, it's like a special disk.

Yes, but I think there's a bunch of stable main memory technologies which are coming along, which might in fact have the effect of replacing main memory -- currently volatile with stable main memory. That poses another interesting set of problems. This is one of these things where the hardware base seems to go along forever with the same technology, and then all of a sudden you find that there's a phase transition and everybody's using something else. It will take a little bit of getting used to. But there is an enormous commercial pressure to be out there with a database system which exploits the new technology to get a commercial edge. This is bound to be something of great interest and researchers should be taking a look at it.

You have been involved with ACM and SIGMOD and VLDB, but even more involved with the IEEE. In particular, you have been the editor-in-chief of the Data Engineering Bulletin since 1992, you are an IEEE Golden Core Member, and you received IEEE Outstanding Contribution and Meritorious Service Awards. So I think you are a good person to ask about IEEE, which I don't think we've talked about in previous interviews.

How is the Data Engineering Bulletin different from the SIGMOD Record?

That is one of my favorite topics in fact. Rakesh Agrawal was the one who recruited me as the editor of the Bulletin. Something that he emphasized, but I think it is really true, is that every issue that we do is a special issue. It is a special issue on ongoing research. It is not the kind of work that you see in the published papers -- it is more work in progress. So in some sense, it gives you a window into what's actually going on now, so that's relatively unique. Then another factor is that I try to give it an industrial slant. I try to involve people who don't usually write papers, and let the rest of the world know what they are up to as well. It's a combination of those things, plus, the set of editors I have more or less cover the field, or at least interesting areas that are now of real interest. We try very hard to keep it current, keep it with some industrial focus, and it's a great place if you are looking or want to start looking in an area and do some research in an area -- it's a great place to start to look.

How is the IEEE Technical Committee on Data Engineering different from ACM SIGMOD?

Well, that's a long, and in some sense, a sad story. SIGMOD has a budget. They have money that they roll over from one year into the next. Technical Committee within the IEEE, at the end of the year, any surplus that they have reverts to the IEEE. So the end result is that we have much less discretion in terms of what we can do, much less discretion in terms of how we run our operations, and that's by design. I'm not a big fan of that, but that's the way it is. I happen to like the ACM model better, but when you end up being involved in an organization, you sort of end up with the rules of the organization… The Technical Committee does a pretty good job, I think, in terms of sponsoring conferences and things of that sort, and they are of course the sponsor of the Data Engineering Bulletin as well, but it's more a limited role than SIGMOD would have.

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

You mean aside from napping?

I think napping is a good answer!

It is a good answer! In the back of my mind I have the outlines of a paper on a full theory of recovery, which I need a co-author to do, because I am not really a theorist, but it requires somebody who's comfortable at theory. Mark Tuttle was great in that respect with the other work, but the idea here is to apply some of the work that Mark Tuttle and I had done but in the broader setting of covering application recovery type stuff as well as database type stuff.

Maybe you'll get a volunteer who reads this or sees this.

Maybe, that would be good!

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

I'll mention two things. I've seen people have success, if you will, seizing the moment, and being first in an area, and in some respects, I've recoiled from that. Maybe it's a little bit of timidity on my part, but I like to think things over, and make sure that I know what I'm doing before I rush out. It's not that these people don't necessarily know what they are doing, but usually there's a certain first cut aspect to that, which I tend to recoil from. I want a little bit more solid footing underneath myself. So that's one thing. Another thing is I probably made a mistake in getting a little too far away from programming. I don't do very much programming. It's probably a mistake, I probably should do more.

Among all your past research, do you have a favorite piece of work?

No single favorite piece, but I'll tell you what I like, and what I like is some combination of things which have an abstract aspect, a theoretical aspect, and a pragmatic engineering aspect, and hopefully some practical application. So things like some of the recovery work that I've done, the recovery theory that I did with Mark Tuttle, or the application recovery principles that I did with Roger Barga and Gerhard Weikum. Things of that sort have a real appeal to me, because they have both a practical flavor, but there is some abstract principles involved, and there's some hard engineering as well. So it sort of has it all, and I like the areas where you can bring these things together. I like the Deuteronomy project which I'm doing now exactly for the same reason. It's not always easy to bring everything together in one piece, but I like it when that happens.

We were talking earlier about your favorite plant in your yard, which you said was the redwood tree, and in many ways the redwood tree is that -- it has all the historical significance, it is esthetically beautiful, it smells good, it has it all!

In my yard, it also has a rather unique role because there are not that many redwood trees in Washington State, and I have 3 of them in my back yard.

Three! Well if they start reproducing, it could be the beginning of a whole new forest.

And my house will become a tree house!

Thank you very much for talking with me today!

It's been a pleasure, thanks.