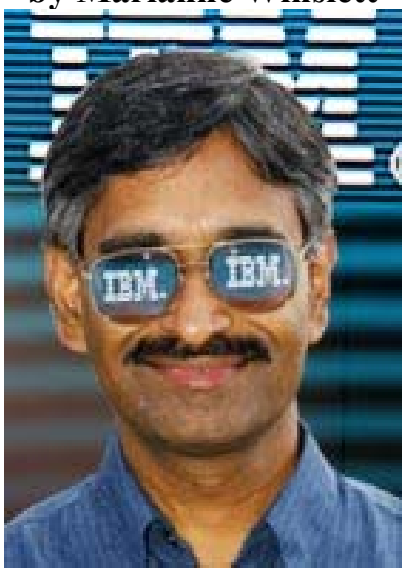
C. Mohan Speaks Out

on R*, Message Queues, Computer Science in India, How ARIES Came About, Life as an IBM Fellow, and More

by Marianne Winslett



C. Mohan
http:// www.almaden.ibm.com/u/mohan/

I would like to take a moment to thank the many people who have helped me devise interview questions for this series of columns. Often people propose questions under a promise of anonymity, so I will not name the individuals who have suggested questions---but you know who you are! Without you, these columns would not be possible. Thank you for your many excellent suggestions over the years.

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 in San Diego, site of the 2003 SIGMOD and PODS conference. I have here with me C. Mohan, who is the technical team lead for the DBCache project at IBM Almaden Research Center. Mohan is well-known for his work on transaction commitment, logging, and recovery, which has had a tremendous impact on how those functions are carried out in database products from IBM and other vendors. Mohan is the primary author of the 1992 TODS paper on ARIES that has become a staple of every database graduate student's qualifying exam preparation, as well as having the distinction of being the only paper ever accepted to TODS that exceeds TODS's 50-page length limit. Mohan is an IEEE, ACM, and IBM Fellow. He has received the SIGMOD Innovations Award and the VLDB 10-year Best Impact Paper Award. His PhD is from the University of Texas at Austin. So, Mohan, welcome!

Thank you, Marianne, for that kind introduction.

You're very welcome. Mohan, your thesis was on a theoretical topic: managing deadlocks in database locking protocols. Given this theoretical orientation, what led you to join industry after you graduated? And what led you to move from database theory to the very practical ARIES work that you did soon afterwards?

Actually, although I did my thesis in a theoretical area, it was not because I was really interested in doing theory. At the time, there was no large-scale systems project in the database area at the university where I was, so I did what I could to quickly do a thesis and finish my PhD. But at the same time I was interested in practical things. I'd even written a critique on SDD 1, the [pioneering] distributed database system, which analyzed the design in great detail. I'd sent it around to various people, and I was very eager to actually get out of school and dirty my hands by joining a place where they were doing very practical stuff. That was one of the reasons why I didn't even apply to any universities for a job. I was keen on joining a research lab, to get the ability to work with real systems, make changes to them, and thereby let products come out with some new technology.

When you arrived at IBM, you joined the R team, and you've been at IBM ever since. With perfect hindsight, what do you wish had been done differently in R*?*

I joined that project when it was midway through its life span. During those days, even in IBM Research, we were not that closely working with the product people. As a result, R* originally was focusing more on homogeneous distributed databases. We assumed that the different nodes in the distributed database network were all System R nodes, and then we worked on two-phase commit protocols, replication problems, and distributed query compilation and distributed optimization. But the problem was that since we focused only on the homogeneous aspects of distributed databases, when the time came for commercialization, it turned out that even IBM itself had many different relational DBMS products with somewhat different capabilities. Solving the heterogeneous problem in the context of the R* research project would have been a more fruitful exercise or a better starting point for that project.

Solving the heterogeneous problem in the context of the R research project would have been a more fruitful exercise*

Although the heterogeneous case is a harder problem than the homogeneous case.

That's true.

Was it too early at that point to solve the heterogeneous case?

I won't say that, because it was around that time that companies like Computer Corporation of America had people working on that topic. So our focus on the homogeneous case just might have been because the people who started the R* project were very familiar with System R, and DB2 on the mainframe was still being developed and was not released until 1984.

When I compare the work on ARIES to other work that has had a major impact on the database field, I find that the ARIES work seems to be qualitatively different, in that it's an agglomeration of a large number of details that have to be handled correctly, rather than a single big new idea. This difference explains in part why the ARIES TODS paper is so long compared to other very influential papers. Can you comment on this perceived qualitative difference, and more generally, have we reached a point in the database world where big new ideas will no longer play an important role?

I think the reason the ARIES paper is as long as it is primarily has to do with the fact that I was trying to be very comprehensive in covering related pieces of work, and also tried to justify certain features that were part of the ARIES algorithm. I looked at the past work and found that lots of important issues had not been written up in published papers (even though they might have been addressed). Being a very details-oriented person, I felt that the research community was not getting a good picture of what's involved in doing concurrency control, recovery, and storage management in an efficient way with high reliability. I'd taken

the trouble to understand a lot of these things by not only reading past papers but even going back to the System R code and understanding many features that otherwise had not been documented. I also talked to the System R people, some of whom are still in IBM in my research lab, and I even went back to [the hierarchical DBMS] IMS and looked at its code to discover certain aspects of that IBM product. Having done all that work, I felt that the rest of the research community as well as the IBM product community should benefit from all that information that had been gathered up. So I chose to describe a lot of the algorithms and options available on these three topics: recovery, concurrency, and storage management.

What did you learn from the experience of technology transfer of ARIES, and what is your relationship to the product groups at IBM today?

In fact, the ARIES algorithm as described in the paper resulted from my interactions with Don Haderle, who at that time was the chief architect of DB2 on the mainframe. It was through the interactions with him that I had realized that DB2 had done certain things very differently from the way System R people had done them, which then got commercialized as SQL/DS. System R used shadow page based recovery, and left as an open problem how to do record locking with write-ahead logging. As I tried to understand why these things were done differently, I began to get a better feel for what were the real problems and what the characteristics of the solution should be, based on Don Haderle's real-life experience with the DB2 product. So, the ARIES work resulted from the very close cooperation of a product person and a research person.

The ARIES work also led to the creation of the Database Technology Institute as a framework under which researchers and product people in IBM work together. Technology transfer became easier because we started working with product people early on, in the system design and implementation phases of our research projects. Ever since, I've been very closely associated with products. I have tried hard to make the work that I do be technologically challenging from a research community perspective, with papers coming out of such work, as well as be applicable to real-life problems and hence get incorporated into products. I've tried to balance these two aspects of doing work at IBM. I continue to get involved in product-oriented things, not only in the database area but also in areas like WebSphere and Lotus Domino/Notes. Given the fundamental nature of the ARIES work, which applies to any system which manages persistent data, the ARIES algorithms with variations have been incorporated not only in the IBM relational database products and also some other companies' products like Microsoft's SQL Server. They have also been incorporated in messaging systems like MQSeries and Lotus Domino, for log-based recovery. So ARIES has gone to quite a few places and I hope to continue to work on those sorts of aspects of IBM's products as well research.

the ARIES work resulted from the very close cooperation of a product person and a research person

Distributed commit---what is it, and why don't users like it?

This topic was my first piece of work in IBM when I joined the R* project. I was given the task of doing two-phase coordination algorithm design and implementation in R*. I took the original classical two-phase kind of protocol and worked on its variations, the so-called presumed abort and presumed commit. The two-phase commit protocol is a way of guaranteeing, in a distributed context, that transactions are atomic. When a transaction does updates to more than one recoverable storage area (database nodes or recoverable files), all updates with the transaction persist if the transaction commits. If the transaction is rolled back for any reason---user demanded rollback or the system crashes---then all the updates of the transaction are undone.

The problem with trying to apply this in the real world is that there are many environments where the owners of the data at the different nodes of the network are not necessarily willing to give up their autonomy by letting others' systems determine, possibly after a long delay, whether or not data in their own system

will get committed or rolled back. During that indeterminate period of time, access to data that had been modified but not yet committed is denied to other transactions.

So what's the solution?

The solution is something that's been worked on for a very long time, although in terms of commercialization, things haven't progressed so well. The solution is the notion of advanced transaction models: letting nodes independently commit data changes. If afterwards you need to roll back those changes, then you use the log to figure out what was changed and then apply the appropriate undo operation. For undo operations, you define the notion of compensating transactions and use them to logically undo the changes previously committed by single-site transactions. There has been a huge body of literature on this topic, but in terms of actual implementations, even in incomplete prototypes, not a whole lot has been accomplished.

And in products?

In products, even less of course. But in the context of workflow management systems, finally some of the ideas are beginning to see the light of day. Also, standardization efforts like the web services transactions (WS-TX) and web services coordination (WS-C) are trying to provide features that people could use to build these high-level transaction concepts. As these specifications become standard and the companies like IBM, BEA and Microsoft implement the standards, you will begin to see more widespread adoption of that way of doing transactions.

the messaging-based way of doing distributed transactional work has been very popular in the real world. The research community has pretty much ignored that topic

Finally.

Yep. Long time in coming.

Queuing systems: what is their place in distributed applications in the database world?

This is an area where there has been quite a bit of commercial support for a long time in the form of IMS's queued transaction processing, Digital's products, and various other companies that released transactional messaging and queuing systems. In the early 1990s, IBM introduced its MQSeries product for doing this kind of asynchronous transaction processing, which is the alternative to the synchronous RPC-style way of doing inter-application coordination or distributed activities. Because people didn't want to necessarily adopt two-phase commit protocols as a way of doing distributed computations, the messaging-based way of doing distributed transactional work has been very popular in the real world. The research community has pretty much ignored that topic, except for a few papers.

Now we find that business process integration has become very important as different companies adopt IT in a big way. Also inter-organizational work is being automated more and more. Especially with the web coming into the picture and inter-company transactions being executed using the web, the message-based way of doing distributed computations has become very significant. Some of the DBMS vendors have even introduced advanced technologies in their products to enable the implementation of database-based messaging and queuing systems.

So what are the research issues in messaging and queuing systems?

It turns out that the original transactional concurrency kind of features that were introduced in relational systems are not good enough when it comes to the increased concurrency that needs to be supported by the

messaging kinds of APIs. For example, when a user tries to get a message from a queue, even if there are some older messages which are in an uncommitted state, those messages will get skipped in order to provide the user with a later committed message that's available. This notion of being able to skip over uncommitted data in order to more quickly answer the user's request is hard to support, given the current transactional isolation features in our DBMSs. So that's one research issue.

The other research issue is that messages can have widely varying formats. If you look at the Java messaging service, it lets users define new headers on a per message basis. Different messaging vendors can add their own header fields. So the format of these messages can be very different from one message to another message. When you map these messages to relations, you get into the same kind of problems as when you try to model XML documents in relational systems. Plus, messages can be very large, so some performance implications of the kind of logging that we do also need to be addressed.

The third research issue comes from the fact that messages tend to enter the system and leave the system fairly quickly. Some messages may be persistent, other messages may be non-persistent, and the efficiency with which all these can be supported in relational systems is not that great right now.

Do these messages typically represent the equivalent of an RPC call between pieces of code?

Right.

And why would they need to be persistent?

When you map these messages to relations, you get into the same kind of problems as when you try to model XML documents in relational systems

RPCs didn't have the transaction notion to begin with. Later on there was the notion of transactional RPCs, which are done synchronously. If you want the equivalent of transactional RPC functionality with an asynchronous message-based way of doing business, then you have to guarantee "once and only once" semantics for message delivery. Once I have sent you a request to do some operation, then come rain or shine, that message has to be delivered to you. Once you produce an answer or take some action based on the message I sent you, you will then need to send some response back. We need to make sure all of that doesn't get lost, in spite of failures and so on. So that's what makes these messages persistent.

There are other situations where the information content of a message does not have to be persistent if that will improve performance, like the kind of stuff that's become very popular in the research community, namely streaming kinds of applications. Depending on what is being communicated via these messages, whether you're doing transactional work or you are just propagating some information like sensor data and such things, you may or may not care about the persistence aspects of those messages that are being handled.

Mohan, you're one of 56 IBM Fellows. What is it like to be a Fellow? How does it change what you are doing?

Since IBM has 300,000 employees, of course it feels good to be one of 56 selected people, given that an IBM Fellow is the highest technical position that one can attain in the company. But at the same time you also feel more responsibility, because we are supposed to be IBM's corporate consultants. The company expects us to not just work on a single project or a narrow topic, but be ambassadors to various parts of the company and play a leading role in enabling inter-organizational collaboration. We are also supposed to act as mentors to more junior people in the company who are especially involved in deep technical activities. In that sense, we can't be spending too much time on any one problem of our own. We need to be also willing

to travel a lot, given the fact that IBM has operations throughout the world. Even if you restrict yourself to the research domain, there are research labs scattered throughout the world.

This whole area of XML could really use the help of ***theoreticians*** *in providing a more solid foundation.*

Being an IBM Fellow is something that different people have dealt with differently. I've chosen to not only focus on the database area but also get involved in activities relating to WebSphere and the whole software group of IBM: Lotus products, Tivoli products, and so on. By focusing on the whole software group architecture of IBM, I'm able to have frequent interactions with IBMers everywhere. I've also gotten involved in activities of IBM Global Services (IGS), which does a lot of currently popular activities like system integration projects, consulting for customers, and IT operations for various customers around the world---out-sourcing kinds of work.

What topics should database theoreticians be working on now?

That's a hard one. We were at a panel [at SIGMOD 2003] just yesterday where that topic was discussed. If you look at what is currently popular in terms of products, XML is an area where a great deal of action is taking place. In terms of the formal basis for much of that, there is some work from the past that applies but there are also certain newer aspects of the problem that need to be better formalized. This whole area of XML and how it's going to be handled in the DBMS context in terms of query processing, and how concurrency control should be done for XML data when it's stored in its native form (as opposed to translated to relational records) is one big area that could really use the help of theoreticians in providing a more solid foundation.

What do you make of the fact that the other research labs that do work on the database area, like AT&T, Bell Labs, HP Labs, aren't doing so well right now?

I think the problem really is the fact that at least in the case of HP, they used to have a DBMS product but they were not really selling that product in any serious way. There was scope for some of the HP Labs work in the database area being commercialized via HP's product, but in the case of AT&T and Lucent, those were never seriously computer companies. More importantly, they were not software-product-producing companies. I've always wondered, when they were having a large number of database researchers, what the likelihood was that any of that work would see the light of day in terms of being made available to ordinary users, just because those companies themselves didn't have---

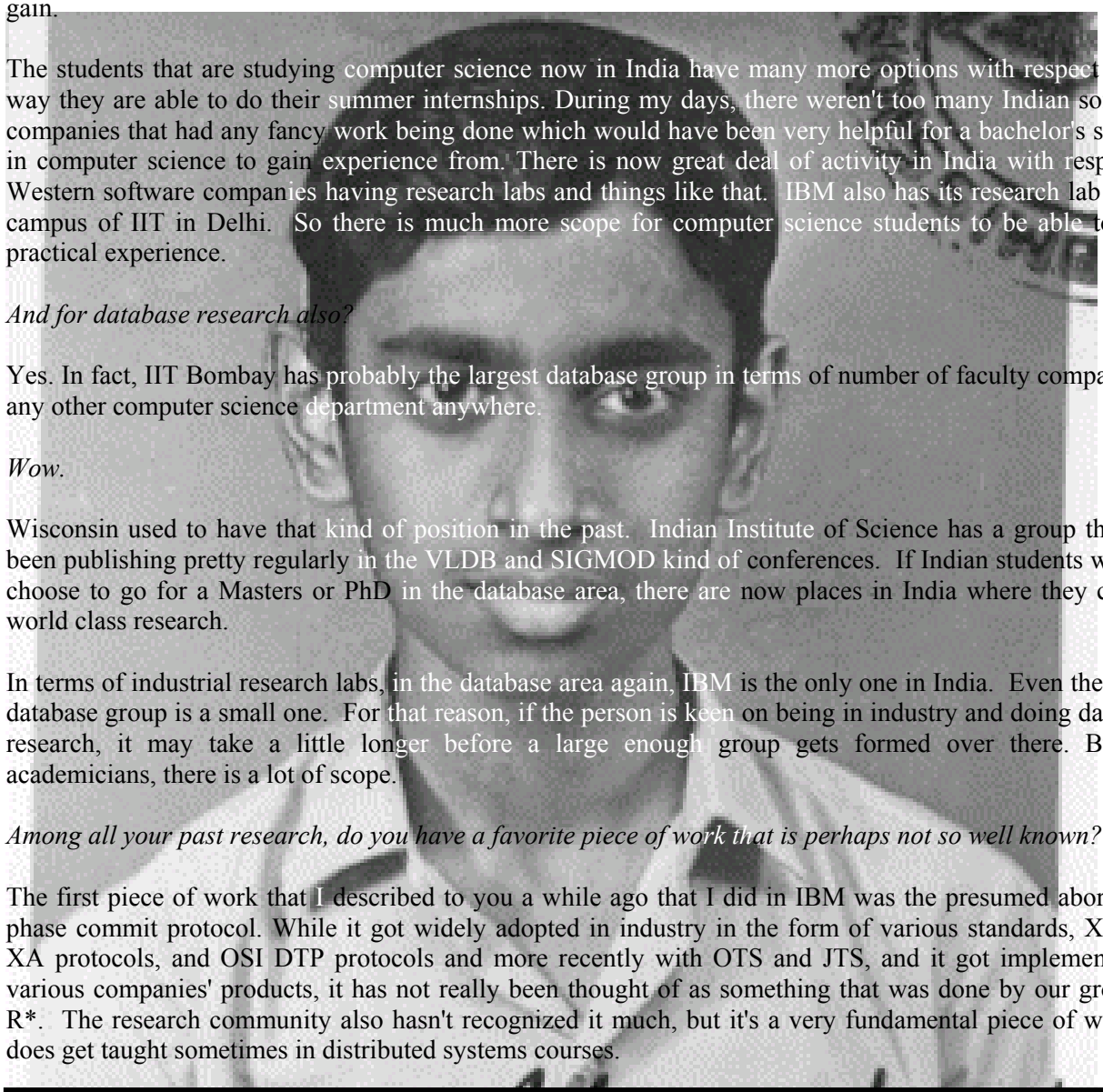
Right, good point. Well, then what about Microsoft? How would you compare?

Microsoft of course is a very different situation. Microsoft to begin with wasn't really doing any serious work in the database area, even though they were selling a database product, because they were selling the re-logoed version of Sybase. Once they decided to take that product and make it a Microsoft product, in the sense of changing its internals and so on, then they saw the need for having a research group. So the Microsoft research guys have had more successes in doing some technology transfer. But if you compare IBM Research with Microsoft Research, clearly we have been in the research game for a much longer time.

We've also had the legacy of having been the inventors of the relational model, as far as IBM Research division is concerned, and consequently we have had a longstanding relationship with our product people in the database space. In the case of Microsoft, the product existed before the research group was formed, so they will as a result have a harder time establishing their creditability with their product organization.

Mohan, your undergraduate degree is from IIT Madras. Things have changed a lot in India in the past ten years. If you were graduating from IIT today, would you do anything differently then you did at the time?

When I was studying in IIT in Madras, from 1972 to 1977, there wasn't an undergraduate program in computer science. Even though I was a chemical engineering student, I had a great deal of interest in computer science. So if I were there now, of course I'd have the option of doing a degree in computer science, and I would have been able to learn much more about computer science in a formal sense before I got to do my PhD. In my days, I had to go to the library on my own in my own spare time, and be on the mailing lists of various US universities and research labs in order to obtain the knowledge that I was able to gain.



The students that are studying computer science now in India have many more options with respect to the way they are able to do their summer internships. During my days, there weren't too many Indian software companies that had any fancy work being done which would have been very helpful for a bachelor's student in computer science to gain experience from. There is now great deal of activity in India with respect to Western software companies having research labs and things like that. IBM also has its research lab in the campus of IIT in Delhi. So there is much more scope for computer science students to be able to gain practical experience.

And for database research also?

Yes. In fact, IIT Bombay has probably the largest database group in terms of number of faculty compared to any other computer science department anywhere.

Wow.

Wisconsin used to have that kind of position in the past. Indian Institute of Science has a group that has been publishing pretty regularly in the VLDB and SIGMOD kind of conferences. If Indian students were to choose to go for a Masters or PhD in the database area, there are now places in India where they can do world class research.

In terms of industrial research labs, in the database area again, IBM is the only one in India. Even there, the database group is a small one. For that reason, if the person is keen on being in industry and doing database research, it may take a little longer before a large enough group gets formed over there. But for academicians, there is a lot of scope.

Among all your past research, do you have a favorite piece of work that is perhaps not so well known?

The first piece of work that I described to you a while ago that I did in IBM was the presumed abort two-phase commit protocol. While it got widely adopted in industry in the form of various standards, X-Open XA protocols, and OSI DTP protocols and more recently with OTS and JTS, and it got implemented in various companies' products, it has not really been thought of as something that was done by our group in R*. The research community also hasn't recognized it much, but it's a very fundamental piece of work. It does get taught sometimes in distributed systems courses.

In my days [at IIT Madras], I had to go to the library on my own in my own spare time [to learn computer science. There] are now places in India where students can do world class research

My second favorite piece of work that was in the context of ARIES was the work on index concurrency control and recovery. While the basic recovery scheme of ARIES is covered a fair bit now in textbooks and courses, the more complicated aspects of doing index concurrency control and recovery is something that is still not being taught enough in courses. That work was published in a shorter form in SIGMOD '92 and is of an extremely fundamental nature. It needs to be studied more and taught more.

My third favorite piece of work is the Commit LSN (Log Sequence Number) concept which is a simple way of recognizing that all the data on a page is committed. That was a VLDB '90 paper.

In my opinion, these are very cute, very good, and important pieces of work that need to be further adopted and taken advantage of by the research community.

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

Of late, I haven't been publishing as many papers as I used to publish at the peak of the ARIES family of algorithms work. I would like to be able to sit and write up more of the things that I've thought about or figured out. I'm just not able to find the time, or sometimes I'm not sure whether I have the desire to go through the painful process of writing it up in an understandable fashion and getting it published.

You need a student. You need a student, definitely.

I would like to work with more people and spend time, but then I travel. My involvement in various activities as an IBM Fellow also in some ways takes away the possibility of doing that.

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

Describing algorithms in an abstract form is something that I haven't really mastered

That's a hard one to answer. I guess I could be doing more of getting into many details of particular pieces of software and understanding them in greater detail as I used to do in the past. At the same time I would like to also be able to explain complicated concepts in a very easily understandable fashion. Describing algorithms in an abstract form, as some other researchers are capable of doing, is something that I haven't really mastered, and that's the reason many people find my papers too complicated to read.

Well, thank you very much.

Thank you.