Distrib. Comput. (2003) 16: 71–74
Digital Object Identifier (DOI) 10.1007/s00446-002-0072-6

DISTRIBUTED COMPUTING

Some perspectives on PODC

Nancy Lynch[1]

Massachusetts Institute of Technology, 545 Tech Square, Cambridge, MA 02139, USA (e-mail: lynch@theory.ics.mit.edu)

Hagit Attiya and Sergio Rajsbaum asked me quite some time ago to write a "personal perspective" paper for PODCs 20th anniversary. Although they gave me plenty of warning, I'm afraid I'm writing this at the last minute. So this may be a bit stream-of-consciousness.

Since I'm one of the people who started the PODC conference in the first place, I'll begin by writing down my memories of how it all started. Now, we're going back more than twenty years here, and I suspect my memory isn't completely accurate, especially about the ordering of events. But I'll do my best, and I hope you will find the story interesting in any case. Then I'll say a little about my view of how PODC has evolved. I'll describe the relationship I see between formal modeling (one of my own interests) and the algorithmic research that forms the core of PODC. Finally, I'll conclude with a few words about where I think PODC might go in the future.

1 Ancient history

My own interest in the theory of distributed systems began around 1978, while I was on the faculty of Information and Computer Science (ICS) at Georgia Tech. After having worked for many years in complexity theory for sequential computing, I was hunting around for a new research area. Distributed computing was becoming a hot new area of systems research and development, and Phil Enslow, a very energetic senior faculty member at Georgia Tech, was leading an effort to develop this area as a focus area for ICS at Georgia Tech. I read all the papers I could find that related to distributed computing and that had a theoretical slant: papers by Dijkstra on concurrent computing, papers by Lamport on time in distributed systems, and papers by many people on mutual exclusion and distributed database concurrency control. One paper that especially attracted my attention was a little note by Cremers and Hibbard [1] that proved a lower bound on the size of memory needed for 2-process mutual exclusion. This was the first example I encountered of an impossibility result for distributed computing, and I found the general concept and the specific ideas quite fascinating.

It was pretty clear from what I was hearing about practical distributed computing that the general area was going to become very important in practice. At the same time, it was clear that distributed computing was hard to think about. Many problems, algorithms, and proposed system designs were described somewhat informally, and it wasn't always clear what the algorithms and designs were supposed to do and why they were correct. It seemed that this important area was in need of a theory. So this looked to me like a great choice for a new research area.

The theory I had in mind was modeled after the already-existing theory for sequential computing, which included formal computing models like Turing machines and Random Access Machines, clever algorithms that could be described and analyzed in terms of these models, some lower bound and reducibility results, and applications of the algorithms for real computing. The work that I thought was needed for distributed computing included defining formal models for distributed systems, designing and analyzing algorithms, proving lower bounds and other impossibility results, and relating it all to actual distributed systems. Very little of this existed at the time.

I started talking with others about working on a theory for distributed computing. One of the people I talked with was Mike Fischer, with whom I had worked several years earlier at MIT. During a breakfast discussion at a STOC or FOCS conference, I discovered that Mike had also recently gotten interested in distributed computing, and wanted to work on its theory. In fact, he had similar views about what such a theory would entail: it would be a lot like the field of analysis of algorithms, but for distributed algorithms instead of sequential algorithms.

We started by working together on algorithms and lower bounds for the mutual exclusion problem. At the time, Jim Burns was a PhD student working with me at Georgia Tech and Gary Peterson was working with Mike at the University of Washington; Jim and Gary collaborated with us on some early projects involving mutual exclusion. Mike and I exchanged several visits, culminating in a very productive Winter Quarter, 1980, which Mike spent at Georgia Tech. During that quarter, we worked on many different problems in distributed computing theory, starting with mutual exclusion and other resource allocation problems.

At some point during that quarter, we had the bright idea of inviting some people that we wanted to work with to visit us for a week or so. Among these visitors was Leslie Lamport. He told us about his many ideas involving time and dis-

tributed computing, about atomic, safe, and regular registers, about strong and weak precedence relationships between register operations, about connections between relativity theory and distributed computing, about why computations in distributed systems are "really" partial orders, etc. Just before Leslie's visit, we obtained an interesting draft paper that he had written, entitled "The Albanian Generals Problem". We found this paper so fascinating that we spent the time just before Leslie's visit proving a lower bound on the number of rounds needed to solve this problem, and also spent much of Leslie's visit talking about the problem. Mike Merritt, who was then a graduate student at Georgia Tech, also participated in these discussions.

At some point around then – probably after Leslie's visit but I really can't remember – Phil Enslow organized a workshop on distributed computing at Georgia Tech. This was a serious workshop, which was supposed to actually produce something: a document delineating the important research problems in this new area. Phil divided the workshop participants into working groups, each of which was supposed to cover a sub-area. Mike Fischer, Leslie Lamport, Jim Burns, and I (and some other people) were assigned to a working group on "theory of distributed systems", and we produced some kind of document describing important research problems in this area. Unfortunately, I don't remember anything at all about what we or any of the other working groups wrote; given that we wrote it quickly and late at night, I suspect it wasn't too interesting.

After this workshop, Mike, Leslie, and I talked about the desirability of having another workshop in distributed computing theory sometime soon (but this time without requiring participants to actually produce anything). We didn't do anything about this idea for a while. The next thing I remember is that, at some theory conference, Mike Fischer and I met Robert Probert, a Canadian professor, and learned that he was interested in founding a regular conference that would have its home in Canada. He thought that distributed computing theory would provide a good topic for such a conference. Although we had so far been thinking of a one-time workshop, after Mike and I discussed the matter with Robert, we agreed that starting a regular conference was also a reasonable idea. The basic plan we agreed on was that Mike and I would organize the technical program, and Robert would handle most of the organization and logistics. I don't recall Leslie's involvement at this stage – I think he was really more interested in having a small workshop than a big, general conference.

We advertised the conference as a forum in which distributed systems researchers and distributed computing theory researchers could meet and talk to each other about research issues of common interest. In keeping with this dual orientation, we worked to obtain sponsorship from both the systems and the theory communities – thus, the conference was created under the auspices of both SIGOPS and SIGACT. The scope of the research to be covered was quite broad, including all the distributed computing topics we could think of. We considered any theory paper or systems paper that could be seen as expressing a "principle" for distributed computing as fair game for submission.

There was one more thing the conference needed: a name. The name "Principles of Distributed Computing" was pretty easy to come up with, by analogy with "Principles of Programming Languages". We used the word "Principles" instead of "Theory" on purpose, to make it clear that we weren't just looking for papers containing theorems. With this name, it was pretty clear we should use the initials PODC.

But we still had to figure out how to pronounce the initials "PODC". I first realized this when I was about to present our plans for the conference at a STOC business meeting. Reading it like an English word didn't come out right – it's basically unpronounceable (unlike POPL, which is pronounced easily as "Popple"). So, the only solution was to pronounce the final letter "C" separately. There were two choices: "Pod-cuh" or "Pod-see". The second of these won easily on esthetic grounds.

2 How PODC evolved

Behaving democratically, Mike and I elected Mike as the program chair of the first PODC, which was held in Ottawa (with Nicola Santoro doing local arrangements), and me as program chair of the second, which was held in Montreal (with Gordon Cormack doing local arrangements). As I recall, both conferences were pretty widely advertised, and got plenty of submissions and plenty of attendees.

Those first two conferences were notably broad in scope, with papers covering a wide range of systems and theory topics. A good number of systems researchers attended, along with theoretical researchers. The breadth of the conference papers, and the wide variety of interests of the participants, made these conferences a lot of fun. However, some participants thought that the scope was too broad – that the conferences didn't have enough of a focus on any particular topic. For instance, I recall that Robin Milner attended one of the conferences but found that, for his taste, there weren't enough papers involving fundamental issues of semantics of concurrency.

The character of PODC that has evolved in the ensuing years was certainly not evident at the first two meetings. In fact, those conferences had relatively few papers on distributed algorithms, which have been the main focus of more recent PODC conferences. Rather, along with some papers about algorithms, we had some on other aspects of theory, some on systems design ideas, some on applications, etc. The "Principles" represented here included more than just theorems, for example, I remember one fine paper by Paul Leach and others at Apollo Computer Co. [2], which contained nothing resembling a theorem. Instead, it contained a thoughtfully worked out discussion of the role of unique identifiers in distributed systems; it delineated carefully the situations in which UIDs should be used, and the reasons for and against their use. The program committee thought that this was a very good example of a practical "Principles" paper.

The second PODC conference, in Montreal, was the first that included a PODC invited address. And of course, the first person we invited to speak was Leslie Lamport. Leslie spoke about fundamental research issues in distributed computing, including understanding algorithms in terms of their underlying physics, connecting high-level specifications and proofs to real systems, and determining the inherent costs of synchronization and inherent limits of concurrency. He also mentioned some more specific technical problems, including specification of FIFO priority, design of self-stabilizing algorithms, and

proving bounds on the number of processes needed to solve problems in the presence of Byzantine failures.

Since then, I have attended all except one or two of the PODC conferences, so I've seen the conference evolve over time. I think it has changed quite a bit: it has become narrower and more theoretical, and within theory, more focused on distributed algorithms. The particular problems studied have varied over time: mutual exclusion and other resource allocation problems, consensus problems, clock synchronization, minimum spanning trees, implementing read/write atomic objects, wait-free computation, atomic snapshots, self-stabilizing algorithms, group communication, and others. But the general style of papers seems to have stabilized: PODC papers generally present new algorithms for theoretical problems or give results about what is or is not computable in distributed settings. Over the years, the conference has seen less and less participation from systems researchers, as the emphasis of the papers has become more solidly theoretical. For a while, the results seemed to drift away from practical relevance, but I think that in the past few years, connections between the algorithmic results and practical distributed computing have become somewhat stronger.

3 The role of formal models in PODC research

Here, I will change the tone of my "personal perspective" somewhat, switching from remembered history to opinion. The opinions I'm expressing here involve the role that I think formal models play, or should play, in PODC research.

Defining formal (mathematical) models for distributed systems has been an integral part of distributed computing theory from its very beginning. PODC papers typically devote a significant amount of space to defining the underlying computation model and the problems to be solved. This is because it is nearly impossible to demonstrate that a complicated distributed algorithm solves a problem without having clear definitions for the algorithm, for the problem, and for the properties being assumed about the computing and communication infrastructure.

Formal models are more critical for distributed algorithms than they are for sequential algorithms, because distributed algorithms are generally much harder to understand than sequential algorithms. Distributed algorithms execute nondeterministically: a single piece of distributed code is usually executed concurrently at many system nodes, possibly with different speeds at different nodes. Such nondeterminism makes it impossible to understand exactly what a distributed algorithm will do when it executes. Instead, one generally has to settle for understanding properties of executions, for example, invariants or progress properties. Defining these properties and showing that they hold require formal models.

Furthermore, anyone who has ever tried to prove lower bounds or other kinds of impossibility results – which I think includes most people in the PODC community – knows that formal models are critical for stating and proving such results. Statements of impossibility results can be very subtle, for example, they must distinguish carefully between what is under the control of a proposed algorithm and what is under the control of its environment. If one is not careful, one can easily fall into the trap of stating a problem that is unsolvable for trivial reasons (e.g., the algorithm isn't required to keep taking steps), or that is solvable by a trivial algorithm (e.g., an algorithm that refuses to accept some inputs). It is very hard to get all of this right without a precise model.

The need for good formal models became evident to me right at the start of my work on distributed algorithms: when Mike, Jim, Gary, and I wrote our first paper on mutual exclusion, we had to devote a very large amount of time and attention to getting the definitions right. And then, right after this first paper, Mike and I felt compelled to write a second paper focusing exclusively on formal models for distributed systems [3].

What kinds of formal models are needed for distributed systems? Well, it should be pretty clear by now that what is needed is some kind of interacting (reactive) state machine model, where interactions may be via shared memory or shared actions. The models should support "structured" description of algorithms using composition and levels of abstraction, and should support the usual mathematical analysis methods (like invariants and simulation relations). Also, the models should be simple enough to support nice impossibility proofs.

One can either use a general modeling framework (like my own favorite, I/O automata [4] and its extensions to incorporate timing, probabilities, etc.), or can define specially tailored models for each problem from scratch. The latter approach is taken in most PODC papers. However, I personally prefer using a general modeling framework, because it lets me use general results about the framework over and over again. In any case – whether one uses a general modeling framework or defines an individual model – it is important that the models be precisely defined. If they aren't, ambiguities in the definitions may make it impossible to understand the results precisely, in fact, they may make the results meaningless.

One of the referees (Leslie Lamport, who chose not to be anonymous) asked me to include some historical perspective about the development of the I/O automaton model. So I'll oblige:

I/O automata weren't invented on purpose. Around 1985, Mark Tuttle and I invented a simple distributed resource allocation algorithm based on tree traversal, and wanted to analyze its behavior. One of the main points we wanted to make with this work was that a good way of analyzing such algorithms is to use levels of abstraction – mapping the detailed algorithm to a higher-level view of the algorithm as a global state machine whose state contained a directed graph. (Actually, as I recall, the initial ideas for this algorithm and the general approach originated much earlier, in discussions at Georgia Tech involving Mike Fischer, Nancy Griffeth, and Arnold Schonhage, in around 1979.)

Mark and I soon found that quite a lot of work was involved in saying everything we wanted to say really carefully and clearly. The statements we wanted to make were clearly of two different kinds: some involved general properties of interacting state machines and levels of abstraction, and some were specific to the example we were studying. To separate these different kinds of concerns, we were led to use a general modeling framework. We could not use the model developed earlier by Mike Fischer and myself [3], because it is based on shared-variable communication, and we now wanted shared-action communication (to express message-passing). Since nothing

else satisfactory existed at the time, we were led to invent a new framework, and I/O automata were born. This work first appeared in PODC, in 1987.

4 The future

The field of practical distributed computing is at an exciting point right now, because of the current fast pace of technological innovation in the Internet, in the Web, in mobile computing systems, and in hybrid (continuous/discrete) control systems. This means that old distributed computing problems – for example, problems of communication, resource allocation, and data management – will require new solutions for the new settings. The new algorithms will have to tolerate more types of failures and more frequent changes to the set of participating processes, which will probably make the new algorithms more complicated. Also, completely new problems will arise, and these will lead to completely new kinds of algorithms.

These practical developments present researchers in the theory of distributed systems with new and exciting research opportunities. Again taking a historical perspective, I think that we are now in a situation very similar to where we were back in the late seventies. Again, we are faced with new kinds of distributed computing that are becoming extremely important in practice, and that do not yet have adequate theoretical foundations. Again, new theoretical work is needed to define the new settings and problems, and to develop new algorithms and impossibility results. Again, we need adequate formal models – this time, models that can handle complexities like continuous behavior and mobility. I think that PODC researchers are in a great position to play a leading role in developing all of this theory, and PODC can provide an ideal forum for the results.

References

1. A. Cremers, T. Hibbard. An algebraic approach to concurrent programming control and related complexity problems, 1975. Manuscript. Computer Science Department, USC, Los Angeles, CA
2. P. J. Leach, B. L. Stumpf, J. A. Hamilton, P. H. Levine. UIDs as internal names in a distributed file systems. In *Proceeding of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Attawa, Canada, August 1982
3. N.A. Lynch, M.J. Fischer. On describing the behavior and implementation of distributed systems. Special issue on Semantics of Concurrent Computation *Theoretical Computer Science* 13(1):17–43 (1981)
4. N.A. Lynch, M.R. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1988. Also, "Hierarchical Correctness Proofs for Distributed Algorithms," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, British Columbia, Canada, August 1987

Nancy Lynch is a Professor in the Department of Electrical Engineering and Computer science at MIT and heads the Theory of Distributed Systems research group in the MIT's Laboratory for Computer Science. She is the author of numerous research articles about distributed algorithms and impossibility results, and about formal modeling and verification of distributed systems. She has authored two books, "Atomic Transactions" (with Merritt, Weihl, and Fekete) and "Distributed Algorithms". She is an ACM Fellow and a member of the National Academy of Engineering.