CS6180 Lecture 24 – Asynchronous Distributed Computing

Robert L. Constable

Abstract

In this lecture we introduce the notion of *asynchronous distributed computing* and some of the primitives developed for reasoning about it. Distributed systems are built from distributed processes, and we will model them. This is another area of computing where the Church-Turing thesis is not fully applicable. At Cornell we developed a formal model of these processes and a logic for reasoning about them that we call a *a logic of events*.

Many ideas presented in the next two lectures have been formalized in constructive type theory and implemented in Nuprl. There is much more research to be done in this area to achieve a foundational understanding of distributed computing and integrate it into constructive type theory. Nuprl itself is now a distributed system which gives us direct experience with these concepts. Neither Agda nor Coq are yet distributed proof assistants and will probably never be. We will want to consider briefly some of the advantages of having such proof assistants that support distributed computation.

Several of our results have been published in articles cited in this lecture [4, 7, 3, 2]. A few of them are recommended reading for this lecture and the following, Lecture 25. In this lecture we will look at two simple examples of distributed computation, one is leader election and the other is called *consensus*. Consensus involves having a collection of processes agree on a value. The typical example is updating a data base which is replicated at many different sites, e.g. as with the Google database. At some point we could imagine a world wide proof assistant with replicas in several geographically distributed sites.

1 Introduction

Distributed computing is about a collection of processes computing together. Two of the classic books on this topic are *Distributed Algorithms* by Nancy Lynch [11] and *Distributed Computing: Fundamentals, Simulations, and Advanced Topics* by Hagit Attiya and Jennifer Welch [1]. They are both professors at the Technion, so we have some way to claim them as Cornell researchers as well via the NYTech ties to the Technion.

The core concept in distributed computing is the notion of *communicating processes*. These are programs that communicate with each other over a network to accomplish some task. They are not required to operate synchronously or to use clocks, although there is effort devoted to seeing how to achieve synchronous communication by simulating a shared clock or by other means of creating *virtual synchrony* [6, 5]. We will not examine this line of research directly. An image used

in discussing asynchronous computing are *message sequence diagrams*.

This topic leads directly to an interesting deep research problem. We would like the underlying type theory to have a distributed computation system since the Nuprl proof assistant has the infrastructure to support it, unlike perhaps all other proof assistants. So ideally a rich theory of distributed computation would be built into the type theory as well as the proof assistant. We might call it a *distributed computational type theory*, DCTT.

Much of this work reported in the next two lectures was joint with the Cornell distributed systems group, in particular with Ken Birman and Robbert van Renesse. They have extensive experience building real distributed systems, including the French air traffic control system. Some of the work we have done with them has led to highly reliable deployed software that are still used "in the wild" as we sometimes say.

The PRL group's own experience was very much informed by one seminar that turned into an unplanned demonstration of the power of real time collective development of proofs. When Lori Lorigo was giving a seminar about her work on shared proof development, the other PRL researchers in the seminar were changing her proof after she left a particular branch. When she looked back at a previous section of the proof, she noticed that it was not the one she had previously shown us. Someone in the audience had "improved it." Of course once this happened, it was difficult to "contain," leading to an extremely lively and memorable seminar. The impending combination of machine learning and collaborative distributed proof might be a game changing combination. Distributed proving in (very) slow motion is something all researchers understand.

2 Consensus Problems

In this section we illustrate two distributed consensus protocols. One of them is very simple, from Nancy Lynch's book. It is called *leader election in a ring*. The supplemental material includes a complete description of the protocol and a proof of its correctness.

The second example is a consensus protocol of the kind that banks use to integrate data from multiple sites to produce a single record of transactions. You can think of banking and ATM machines. These machines are available in many countries, and a family with members simultaneously located in different countries and using ATM machines or other banking services presents an interesting distributed computing problem – how to keep the account accurate and up to date as fast as possible.

In addition to examining these two protocols, we will look at the very famous Fischer/Lynch/Paterson [9] result on distributed consensus. This is also called the FLP result, referring to Michael Fischer, Nancy Lynch, and Michael Paterson. They are all friends of mine. We all started out doing computational complexity research. That was a way to "establish credibility." Then we branched out into different aspects of computing theory as can be seen from these diverse references [8, 12, 10].

2.1 Leader election in a ring

This problem arose at Xerox PARC where they had built a ring organization of their distributed computing environment. They needed a way to elect a leader in the ring.

References

[1] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley Interscience, New York, 2nd edition, 2004.

[2] M. Bickford, R. Constable, and V. Rahli. The Logic of Events, a framework to reason about distributed systems. Technical Report arXiv:1813:28695, CIS, Cornell University, 2012.

[3] Mark Bickford, Robert Constable, and David Guaspari. Generating event logics with higher-order processes as realizers. Technical Report http://hdl.handle.net/1813/23562, Cornell University, 2011.

[4] Mark Bickford and Robert L. Constable. A causal logic of events in formalized computational type theory. In *Logical Aspects of Secure Computer Systems, Proceedings of International Summer School Marktoberdorf*, 2005. To appear 2006, earlier version available as Cornell University Technical Report TR2005-2010, 2005.

[5] Kenneth P. Birman. *Building Secure and Reliable Network Applications*. Manning Publishing Company and Prentice Hall, January 1997.

[6] Kenneth P. Birman and Thomas A. Joseph. Exploring virtual synchrony in distributed systems. In *ACM Symp on Operating System Principles*, 1987.

[7] Robert L. Constable. Effectively nonblocking consensus procedures can execute forever: a constructive version of flp. Technical Report Tech Report 11512, Cornell University, 2008.

[8] M. J. Fischer and M. O. Rabin. Super-exponential complexity of Presburger arithmetic. In R. M. Karp, editor, *Complexity of Computation*, pages 27–41, Amer. Math. Soc., Providence, RI, 1974.

[9] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faculty process. *JACM*, 32:374–382, 1985.

[10] D. C. Luckham, M. R. Park, and M. S. Paterson. On formalized computer programs. *JCSS*, 4:220–249, 1970.

[11] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1996.

[12] J.L. Welch, L. Lamport, and N. Lynch. A lattice-structured proof technique applied to a minimum spanning tree algorithm. Laboratory for Computer Science MIT/LCS/TM-361, Massachusetts Institute of Technology, Cambridge, MA, June 1988.