Highly Concurrent Logically Synchronous Multicast

Kenneth J. Goldman*
M.I.T. Laboratory for Computer Science
Cambridge, MA 02139 USA

Abstract

We define the *logically synchronous multicast* problem, which imposes a natural and useful structure on message delivery order in an asynchronous system. In this problem, a computation proceeds by a sequence of *multicasts*, in which a process sends a message to some arbitrary subset of the processes, including itself. A logically synchronous multicast protocol must make it appear to every process as if each multicast occurs simultaneously at all participants of that multicast (sender plus receivers). Furthermore, if a process continually wishes to send a message, it must eventually be permitted to do so.

We present a highly concurrent solution in which each multicast requires at most $4|S|$ messages, where S is the set of participants in that multicast. The protocol's correctness is shown using a remarkably simple problem specification stated in the I/O automaton model. We also show that implementing a wait-free solution to the logically synchronous multicast problem is impossible.

The author is currently developing a simulation system for algorithms expressed as I/O automata. We conclude the paper by describing how the logically synchronous multicast protocol can be used to distribute this simulation system.

1 Introduction

We consider a set of n processes in an asynchronous system whose computation proceeds by a sequence of *multicasts* (or *partial broadcasts*). In each multicast, a process u sends a message m to an arbitrary subset S of the processes (including u). We say that a protocol solves the *logically synchronous multicast* problem if it guarantees the following conditions:

(1) Processes receive all messages in the same relative order. (Suppose messages m and m' are both sent to processes u_1 and u_2. If u_1 receives m before m', then u_2 does also, even if m and m' were sent by different processes.)

(2) If process u sends message m, it receives no messages between sending and receiving m.

(3) If process u continually wishes to send a message, then eventually u will send a message.

We may informally summarize the first two conditions by saying that it appears to all processes as if each multicast occurs simultaneously at all of its participants (sender plus receivers). Hence, the name *logically synchronous multicast.* Note that the hypothesis of the third condition does not require that u continually wish to send the *same* message, but only *some* message. This is a technical point that will be of importance later.

*This research was supported in part by the National Science Foundation under Greant CCR-86-11442, by the Office of Naval Research under Contract N00014-85-K-0168, by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125, and by an Office of Naval Research graduate fellowship.

The problem lends itself to a highly concurrent solution, since any number of multicasts with disjoint S sets should be able to proceed independently. Likewise, one would expect that the communication costs of an algorithm to solve this problem would be independent of n. We present a solution to this problem that takes advantage of the concurrency inherent in the problem and requires at most $4|S|$ messages per multicast. The strong properties of message delivery order imposed by the problem would make a fault-tolerant solution highly attractive for many applications. However, the properties of the message delivery order are strong enough to make a fault-tolerant solution impossible! By a reduction to distributed consensus, we show that there exists no wait-free solution to the logically synchronous multicast problem.

Various other approaches to ordering messages in asynchronous systems have been studied. Lamport [La] uses logical clocks to produce a total ordering on messages. Birman and Joseph [BJ] present several types of fault tolerant protocols. Their ABCAST (atomic broadcast) protocol guarantees that broadcast messages are delivered at all destinations in the same relative order, or not at all. Their CBCAST (causal broadcast) protocol provides a similar, but slightly weaker, ordering guarantee to achieve better performance. The CBCAST guarantees that if a process broadcast sends a message m based on some other message m' it had received earlier, then m will be delivered after m' at all destinations they share.

Like ours, the protocols of both [La] and [BJ] deliver messages to the destination processes according to some global ordering. However, these protocols do not solve the logically synchronous multicast problem because they allow messages to "cross" each other. That is, in their protocols a process u may send a message m and some time later receive a message ordered before m. Our problem requires that when a process u sends a message m, it must have "up to date" information, meaning that it has already received all messages destined for u that are ordered before m. (See Condition (2) above.)

Multiway handshaking protocols have been studied extensively for implementations of CSP [Ho] and ADA [DoD] (for example, see [Ba1] and [Ba2]). These protocols enforce a very strict ordering on system events, and therefore achieve less concurrency (than ours and the others mentioned above). This is necessary because the models of CSP and ADA permit processes to block inputs. Since a decision about whether or not to accept an input may depend (in general) on earlier events, each process can only schedule one event (input or output) at a time. Our problem permits processes to schedule multiple input events at a time.

One interesting feature of our problem is that it lies in between the two general approaches described above. It permits more concurrency than the multiway handshaking protocols, yet imposes a strong, useful structure on the message delivery order. Other related work includes papers by Awerbuch [Aw] and Misra [Mi], which study different problems in the area of simulating synchronous systems on asynchronous ones. In both cases, the computational models being simulated are very different from ours, but it is interesting to note that some of Misra's techniques, particularly those for breaking deadlock, can be applied to our problem.

The remainder of the paper is organized as follows. Section 2 provides a brief introduction to the I/O automaton model. In Section 3, we present the architecture of the problem and a statement of correctness in terms of the model. In Section 4, we formally present the algorithm using the I/O automaton model. In Sections 5 and 6, we sketch a formal correctness proof and present the message and time complexities. We prove in Section 7 that there exists no wait-free solution to the logically synchronous multicast problem.

The author is currently developing a simulation system for algorithms expressed as systems of I/O automata. The logically synchronous multicast problem was motivated by a desire to distribute the simulation system on multiple processors using asynchronous communication. We conclude the paper by describing how the logically synchronous multicast protocol can be used to achieve such a distributed simulation.