A Class of Termination Detection Algorithms
For Distributed Computations

Devendra Kumar
Department of Computer Sciences, University of Texas at Austin

Abstract

We present a class of efficient algorithms for termination detection in a distributed system. These algorithms do not require the FIFO property for the communication channels. Assumptions regarding the connectivity of the processes are simple. Messages for termination detection are processed and sent out from a process only when it is idle. Thus it is expected that these messages would not interfere much with the underlying computation, i.e., the computation not related to termination detection. The messages have a fixed, short length. After termination has occurred, it is detected within a small number of message communications.

The algorithms use markers for termination detection. By varying assumptions regarding connectivity of the processes, and the number of markers used, a spectrum of algorithms can be derived, changing their character from a distributed one to a centralized one. The number of message communications required to detect termination after its occurrence depends on the particular algorithm — under reasonable connectivity assumptions it varies from order $N$ (where $N$ is the number of processes) to a constant.

This paper introduces message counting as a novel and effective technique in designing termination detection algorithms. The algorithms are incrementally derived, i.e., a succession of algorithms are presented leading to the final algorithms. Proofs of correctness are presented. We compare our algorithms with other work on termination detection.

1. Introduction

We develop a class of efficient algorithms for termination detection in a distributed system. We do not require the FIFO property for the communication channels, which is usually assumed in other works. (The FIFO property for a communication channel means that messages in the channel are received in the same order as they were sent.) Our assumptions regarding connectivity of processes are simple. We have categorized our algorithms in three classes. Algorithms in classes 1 and 2 assume that there exists a cycle involving all processes in the network. This cycle need not be an elementary cycle, i.e., a process may be arrived at several times in a traversal of the cycle. Moreover, the edges of the

---

1This work was supported by Air Force Grant AFOSR 81-0205.
cycle need not be primary edges, i.e., the edges involved in the underlying computation; secondary edges may be introduced in the network to facilitate termination detection. (We use the terms edges, lines, and channels interchangeably.) Normally the length of this cycle would affect performance of the algorithms; by using secondary edges, if necessary, the length of this cycle may be kept to a minimum. Algorithms in class 3 assume the existence of cycles in several parts of the networks.

In these algorithms, messages for termination detection are processed and sent out from a process only when it is idle. Thus it is expected that these messages would not interfere much with the underlying network computation, i.e., the computation whose termination is to be detected.

Except for algorithms in class 1, the messages for termination detection in these algorithms have a fixed, short length (a pair of integers). In all algorithms presented, termination is detected within a small number of message communications after its occurrence.

In devising an algorithm for detecting termination, deadlock, or some other stable property [Chandy 85a, Chandy 85b], one important issue is how to determine if there are no primary messages in transit (primary messages are those transmitted in the underlying computation; secondary messages are those related to termination detection). Several approaches have been developed to handle this issue — acknowledgement messages [Chandy 85a], using a marker to "flush out" any messages in transit (with the assumption of FIFO property) [Misra 83, Chandy 85a], etc. One contribution of this paper is to suggest a new approach — counting the number of primary messages sent and received. As shown in this paper, this approach has several desirable features — it results in simple and flexible connectivity requirements, it does not require the FIFO property for the communication channels, and it does not generate too much overhead in terms of the number of secondary messages after the occurrence of termination. Moreover, we show that it is not necessary to count and transmit information regarding number of primary messages on individual lines — it is sufficient to count and transmit information about the total number of primary messages received and the total number of primary messages sent by individual processes.

Classification of Our Algorithms

Algorithms in class 1 are based on counting primary messages on every line. Each process keeps a count of the number of primary messages it has received or sent on each adjacent line (i.e., input line or output line respectively). As mentioned above, algorithms in class 1 assume that there exists a cycle C including every process of the network at least once. A marker traverses the cycle, and uses these counts in detecting termination. After termination has occurred, it will be detected within \( |C| - 1 \) communications of the marker. (\(|C|\) refers to the length of the cycle C, i.e., the number of edge traversals required to complete the cycle.) The problem with this algorithm is that each message is long — it consists of \( E \) number of integers where \( E \) is the total number of primary lines in the network.

Algorithms in class 2 reduce the message length. In these algorithms, each process counts the total number of primary messages received by it, and the total number of primary messages sent by it. Here counts are not being kept for individual adjacent lines. A marker traverses the cycle C, and collects this information to detect termination. In this case the message length is short (two integers). After the occurrence of termination, it will be detected within \( 2|C| - 2 \) message communications. Note that if C is an elementary cycle then