A Distributed Mutual Exclusion Algorithm

ICHIRO SUZUKI and TADAO KASAMI
Osaka University, Japan

A distributed algorithm is presented that realizes mutual exclusion among N nodes in a computer network. The algorithm requires at most N message exchanges for one mutual exclusion invocation. Accordingly, the delay to invoke mutual exclusion is smaller than in an algorithm of Ricart and Agrawala, which requires $2*(N-1)$ message exchanges per invocation. A drawback of the algorithm is that the sequence numbers contained in the messages are unbounded. It is shown that this problem can be overcome by slightly increasing the number of message exchanges.

Categories and Subject Descriptors: C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*distributed systems*; D.4.1 [**Operating Systems**]: Process Management—*mutual exclusion*

General Terms: Algorithms

Additional Key Words and Phrases: critical section, message exchange, delay, optimality

1. INTRODUCTION

Consider a computer network consisting of N nodes. The nodes have no memory in common, and can communicate only by exchanging messages. The communication delay is totally unpredictable, and messages are not guaranteed to be delivered in the same order in which they are sent. The problem is to design a distributed algorithm that realizes a mutual exclusion requirement that, at any moment, at most one node may stay in its critical section.

Ricart and Agrawala [1] presented a distributed algorithm that works as follows. A node i attempting to enter the critical section sends a REQUEST message to all other nodes. The message contains a sequence number and the node identifier i, which are used to define a priority order among requests. A node j, receiving a REQUEST message of node i, sends a REPLY message to node i immediately, if either j is not requesting or the request of i has priority

This paper was originally submitted to CACM in November 1981 and was later transferred to TOCS as the relevant CACM department was terminated. Publication of the paper was delayed due to editorial error. Part of the results presented in this paper are similar to those reported in a CACM Technical Correspondence by Ricart and Agrawala in February 1983. The work reported in this paper was completed independently of that reported in the Technical Correspondence.
Authors' addresses: Ichiro Suzuki, Department of Electrical Engineering/Computer Science, Texas Tech University, Lubbock, TX 79409; Tadao Kasami, Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, Toyonaka 560, Japan.

over j's. Otherwise, node j defers the reply until after j's request is granted. Node i enters its critical section when it receives a REPLY message from all other nodes.

The algorithm requires $2^*(N-1)$ message exchanges for each mutual exclusion invocation. It is claimed that: (1) this number is the optimal if "nodes act independently and concurrently" in a "symmetrical, fully distributed" algorithm; and (2) the algorithm achieves a minimal possible delay, assuming first that "no information is available bounding transmission time delays or giving actual transit times", and second that "no node possesses the critical section resource when it has not been requested".

2. A NEW ALGORITHM

The basic idea in our algorithm is to transfer the privilege for entering the critical sections by using a single PRIVILEGE message. Initially node 1 has the privilege. A node requesting the privilege sends a REQUEST message to all other nodes. A node receiving a PRIVILEGE message is allowed to enter its critical section repeatedly, until the node sends PRIVILEGE to some other node.

A request message of node j has the form REQUEST(j, n) where j is the node identifier and n ($n = 1, 2 \ldots$) is a sequence number which indicates that node j is now requesting its $(n+1)$st critical section invocation. Each node has an array RN of size N for recording the largest sequence number ever received from each one of the other nodes. If REQUEST(j, n) is received by node i, then node i updates RN by RN[j] := max(RN[j], n). (Node i uses RN[i] to generate its own sequence numbers.)

The PRIVILEGE message has the form PRIVILEGE(Q, LN) where Q is a queue of requesting nodes and LN is an array of size N such that, LN[j] is the sequence number of the request of node j granted most recently. When node i finishes executing its critical section, the array LN, contained in the last PRIVILEGE message received by node i, is updated by LN[i] := RN[i], to indicate that the current request of node i has been granted. Next, all node identifiers j, such that RN[j] = LN[j] + 1 (i.e., node j is requesting), is appended to Q provided that j is not already in Q. When these updates are complete, if Q is not empty then PRIVILEGE(tail(Q), LN) is sent to the node found at the front of Q. If Q is empty then node i retains the privilege until a requesting node is found by arrivals of REQUEST messages.

The algorithm, Algorithm A, is shown in Figure 1. Each node executes procedures $P1$ and $P2$. $P1$ is called when a node attempts to enter the critical section. $P2$ is executed indivisibly whenever a REQUEST message arrives. Initially, the Boolean variable HavePrivilege is true only in node 1. "in(Q, j)" is a predicate for testing if a node identifier j is in a queue Q. "append(Q, j)" inserts j at the rear of Q. "head(Q)" is the element at the front of Q. "tail(Q)" is the new value of Q after head(Q) is removed.

Algorithm A is deadlock free and starvation free. Critical sections are granted in a first-come-first-served manner to some extent. It requires, at most, N message exchanges per one mutual exclusion invocation; $(N-1)$ REQUEST messages and one PRIVILEGE message, or no message at all if the node having the privilege happens to be the only requesting node.

```
const I: Integer;  (* the identifier of this node *)
    var HavePrivilege, Requesting:bool;
    j, n: integer;
    Q: queue of integer;
    RN, LN: array[1 . . N] of integer;
    (* The initial values of the variables are:
      HavePrivilege = true in node 1, false in all other nodes;
      Requesting = false;
      Q = empty;
      RN[j] = −1, j = 1, 2, . . . , N;
      LN[j] = −1, j = 1, 2, . . . , N; *)
  procedure P1;
  begin
    Requesting := true;
    if not HavePrivilege then
      begin
        RN[I] := RN[I] + 1;
        for all j in {1, 2, . . . , N} − {I} do
          Send REQUEST(I, RN[I]) to node j;
        Wait until PRIVILEGE(Q, LN) is received;
        HavePrivilege := true
      end;
    Critical Section;
    LN[I] := RN[I];
    for all j in {1, 2, . . . , N} − {I} do
      if not in(Q, j) and (RN[j] = LN[j] + 1) then Q := append(Q, j);
    if Q ≠ empty then
      begin
        HavePrivilege := false;
        Send PRIVILEGE(tail(Q), LN) to node head(Q)
      end;
    Requesting := false
  end;

  procedure P2;  (* REQUEST(j, n) is received; P2 is indivisible *)
  begin
    RN[j] := max(RN[j], n);
    if HavePrivilege and not Requesting and (RN[j] = LN[j] + 1) then
      begin
        HavePrivilege := false;
        Send PRIVILEGE(Q, LN) to node j
      end
  end;
```

Fig. 1. Algorithm A.

Assuming that no information is available on transmission delays, a time bound inherent in any distributed mutual exclusion algorithms is:

The privilege to enter the critical section cannot be transferred in less than a one-way trip communication time.

As is discussed in [2], attaining this bound requires that a requesting node sends at least one message (e.g., *request message*) containing the information that the

node is requesting. A time bound associated with such *notification of request* is:

A request of a node cannot be recognized by other nodes in less than a one-way trip communication time.

Algorithm A attains the first bound since the PRIVILEGE message is sent directly to the next node. The second bound is attained since a requesting node sends a REQUEST message to all other nodes simultaneously. In this sense Algorithm A achieves a minimal possible delay. Apparently Algorithm A performs better than Ricart and Agrawala's algorithm in which a requesting node must complete a round-trip communication with all other nodes.

3. A MODIFICATION

Algorithm A can be modified so that the sequence numbers will be bounded. If each node generates the sequence numbers by RN[i] := RN[i] + 1 mod L, where $L \geq 2$ is a fixed integer, then a nonrequesting node may erroneously be regarded as requesting. This is because the communication delay is totally unpredictable, and therefore, old requests and more recent requests of a node having close sequence numbers may exist in the system simultaneously. The following method can be used to avoid this problem.

When node i receives L messages REQUEST(j, k), $0 \leq k \leq L - 1$, issued successively by node j, node i sends a REPLY message to node j. Node j, after finishing its Lth, $2*L$th . . . critical section invocation, can send the PRIVILEGE message only when it receives a REPLY message from all other nodes. This ensures that all REQUEST messages of node j in the "previous round" have been received by all other nodes, and hence these old messages will not be mixed up with message that node j issues in the "new round".

The modified algorithm, Algorithm B, is shown in Figure 2. RequestCount[j] records the number of REQUEST messages received from node j. When RequestCount [j] = L, a REPLY message is sent to node j. ReplyCount is used to count the number of REPLY messages received. Procedure $P3$ is invoked when REPLY arrives.

Algorithm B requires at most $L*N + (N - 1)$ message exchanges for L mutual exclusion invocations by a single node. The average number of message exchanges per invocation, $N + (N - 1)/L$, approaches N (the number of message exchanges in Algorithm A) as L becomes large.

Algorithm B is faster and uses fewer messages than Ricart and Agrawala's algorithm. The total number of information bits to be exchanged, however, is larger in Algorithm B.

4. DISCUSSION

If Algorithm A can be called symmetrical, then it is a counterexample to the claimed optimality of Ricart and Agrawala's algorithm. Algorithm A satisfies all the requirements of Ricart and Agrawala's algorithm, except the condition that "no node possesses the critical section resource when it has not been requested" (in Algorithm A a node retains the privilege even if no request exists). Whether such an algorithm can be called *symmetrical* is a problem of definition regarding the term. In any case, node identifiers will have to be used as a tiebreaker. To

```
const I: integer;   (* the identifier of this node *)
      L: integer;
var   HavePrivilege, Requesting:bool;
      ReplyCount, j, n: integer;
      Q: queue of integer;
      RN, LN, RequestCount: array[1 .. N] of integer;
      (* The initial values of the variables are:
        HavePrivilege = true in node 1, false in all other nodes;
        Requesting = false;
        ReplyCount = 0;
        Q = empty;
        RN[j] = −1, j = 1, 2, . . . , N;
        LN[j] = −1, j = 1, 2, . . . , N;
        RequestCount[j] = 0, j = 1, 2, . . . , N; *)

procedure P1;
begin
  Requesting := true;
  if not HavePrivilege then
    begin
      RN[I] := RN[I] + 1 mod L
      for all j in {1, 2, . . . , N} − {I} do
        Send REQUEST(I, RN[I]) to node j;
      Wait until PRIVILEGE(Q, LN) is received;
      HavePrivilege := true
    end;
  Critical Section;
  LN[I] := RN[I];
  if RN[I] = L − 1 then   (* end of a round of node I *)
    begin
      Wait until ReplyCount = N − 1;
      ReplyCount := 0
    end;
  for all j in {1, 2, . . . , N} − {I} do
    if not in(Q, j) and (RN[j] = LN[j] + 1 mod L) then Q := append(Q, j);
  if Q ≠ empty then
    begin
      HavePrivilege := false;
      Send PRIVILEGE(tail(Q), LN) to node head(Q)
    end;
  Requesting := false
end;

procedure P2;   (* REQUEST(j, n) is received; P2 is indivisible *)
begin
  RequestCount[j] := RequestCount[j] + 1;
  if Request Count[j] = L then
    begin
      Send REPLY to node j;
      RequestCount[j] := 0
    end;
  if RequestCount[j] = 1 then RN[j] := n   (* beginning of a new round of node j *)
                         else RN[j] := max(RN[j], n);
  if HavePrivilege and not Requesting and (RN[j] = LN[j] + 1 mod L) then
    begin
      HavePrivilege := false;
      Send PRIVILEGE(Q, LN) to node j
    end
end;

procedure P3;   (* REPLY is received *)
begin
  ReplyCount := ReplyCount + 1
end;
```

Fig. 2. Algorithm B.

summarize, symmetrical is not a well-understood or well-defined concept. A more fruitful approach, it seems, would be to evaluate algorithms in terms of the number of message exchanges, number of information bits exchanged, delay, and robustness to node or link failures.

REFERENCES

1. RICART, G., AND AGRAWALA, A. K. An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM 24*, 1 (Jan. 1981), 9–17.
2. SUZUKI, I., AND KASAMI, T. An optimality theory for mutual exclusion algorithms in computer networks. In *Proceedings of the 3rd International Conference on Distributed Computing Systems* (Oct. 18–22, Fort Lauderdale, Fla.), IEEE, N.Y., 365–370.

(Reference [2] was added at the time of revision.)

Received November 1981; revised May 1985; accepted September 1985