Technical Correspondence

ON MUTUAL EXCLUSION IN COMPUTER NETWORKS

Ricart and Agrawala have proposed a distributed mutual exclusion algorithm that requires for each critical section invocation 2*($N - 1$) messages to be sent systematically between the nodes of a computer network with N nodes [1]. This number of messages is claimed to be optimal "in the sense that a symmetrical, distributed algorithm cannot use fewer messages if requests are processed by each node concurrently."

This last assertion does not seem to be totally true, since it does not include the assumption that the requests are managed in a first-come-first-served discipline based on the use of time stamps [2]. Even if this discipline is actually implemented in the algorithm, there is no link in the paper between this option and the optimality criterion.

If we relax this scheduling option, it is possible to find another solution [3], which needs a number of messages varying between 0 and 2*($N - 1$). This algorithm fulfills exactly the same requirements as Ricart and Agrawala's solution:it is also symmetrical, distributed, ensures the mutual exclusion, and is free of deadlocks and starvation.

The basic idea leading to this algorithm is the following:once a node i has received a REPLY message sent by a node j, the authorization implicit in this message remains valid until i has decided to send a REPLY message to j. This decision may happen only after the reception of a REQUEST message coming from j; during this inverval, node i will be able—as far as j is concerned—to enter its critical section more than one time without consulting node j.

In order to implement this solution, a Boolean "authorization" vector $A[1{:}N]$ is maintained in each node's local memory area. We also use the time stamps to solve conflicts between requesting nodes, but the first-come-first-served discipline is not necessarily respected; that is the case when a node reenters its critical section while a lower time-stamped REQUEST message has been sent to it by a nonconsulted node, and is not yet received.

Algorithm

Each node executes an identical algorithm and uses two processes to implement it.

Shared Database

```
CONSTANT
  ME, N;
INTEGER
    Our_Sequence_Number initial (0),
    Hgst_Sequence_Number initial (0);
BOOLEAN
    A[1:N] initial (false),
    Using initial (false),
    Waiting initial (false),
    Reply_Deferred [1:N] initial (false);
```

Procedures

```
procedure REQUEST-RESOURCE
begin integer j;
  Waiting := true;
  Our_Sequence_Number :=
                          Hgst_Sequence_Number + 1
  for j := 1 step 1 until N do
    if j ≠ ME and (not A [j])
    then
        send (REQUEST (Our_Sequence_Number, ME), j)
    fi
  od;
    waitfor (A[j] = true for all j ≠ ME);
    Waiting := false; Using := true
end REQUEST-RESOURCE.

procedure RELEASE-RESOURCE
begin integer j;
  Using := false;
  for j := 1 step 1 until N do
    if Reply_Deferred [j]
    then A[j] := Reply_Deferred [j] := false;
        Send (REPLY (ME), j)
    fi
  od
end RELEASE-RESOURCE.

procedure TREAT-REPLY-MESSAGE(j)
begin
  A[j] := true
end TREAT-REPLY-MESSAGE.

procedure TREAT-REQUEST-MESSAGE (Their_Sequence_Number, j)
begin BOOLEAN Our_Priority;
  Hgst_Sequence_Number := max (Hgst_Sequence_
                     Number, Their_Sequence_Number);
  Our Priority := Their_Sequence_Number > Our_
                                  Sequence_Number
    or ((Their_Sequence_Number = Our_Sequence_
                             Number) and j > ME);
  if Using or (Waiting and Our_Priority)
  then Reply_Deferred [j] := true
  fi;
  if not (Using or Waiting) or
    (Waiting and (not A [j]) and (not Our_Priority))
                                  then A[j] := false;
      send (REPLY(ME), j)
  fi;
  if Waiting and A[j] and (not Our_Priority)
  then A[j] := false;
      send (REPLY(ME), j);
       send (REQUEST (Our_Sequence_Number, ME), j)
  fi
end TREAT-REQUEST-MESSAGE.
```

1. One of the processes actually uses the critical resource: it calls cyclically the REQUEST-RESOURCE and the RELEASE-RESOURCE procedures to do that.
2. The other one treats the received messages: according to their type, it calls the procedure TREAT-REQUEST-MESSAGE or TREAT-RELEASE-MESSAGE.

The algorithm is presented here in the form of an abstract data structure, with a common database shared by the four procedures above. Except for the *waitfor* statement, the procedures are indivisible.

O.S.F. Carvalho
G. Roucairol
Université Pierre et Marie Curie
Institut de Programmation
75230 Paris Cedex 05, France

REFERENCES

1. Ricart G., and Agrawala A. An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM 24*, 1 (Jan. 1981) 9–17.
2. Lamport L. Time, clocks and the ordering of events in a distributed system. *Comm. ACM 21*, 7 (July 1978) 558–565
3. Carvalho O.S.F., and Roucairol G. Une amélioration de l'algorithme d'exclusion mutuelle de Ricart et Agrawala. Laboratoire Informatique Théorique et Programmation. Internal Report No. 81-58, Nov. 1981.

AUTHORS' RESPONSE

The Carvalho and Roucairol modification of our algorithm should be particularly interesting in practical applications where only a few nodes are the frequent participants in mutual exclusion; it reduces the number of messages substantially in this case.

Our algorithm does admit nodes to the critical section in a first-come-first-served (FCFS) order when the action of one node happens before the action of another [2]. The algorithm immediately updates sequence numbers at all other nodes whenever a sequence number is advanced. Hence, the sequence numbers or time stamps are synchronized as closely as possible without using clocks based on physical time. The sequence number choice leads to FCFS admission to the critical section. Our algorithm as modified by Carvalho and Roucairol also leads to FCFS entry to the critical section as nearly as the nodes can determine from the limited number of messages sent. A node whose actual REQUEST messages "happened before" another node's REQUEST message (in Lamport's sense) will enter the critical section first.

Perhaps the most salient difference between the original algorithm and the Carvalho and Roucairol modification is the fact that nodes give an indefinite permission through the REPLY message and not a specific response. In this way, one node (the one last entering its critical section) is always designated as a "master node" which retains a special right—the right to enter its critical section without coordination. We had felt that such a distinction violated the spirit of a symmetric algorithm. In fact, Assumption 2 of Sec. 5.2 of [1] specifically disallowed such preconditioning.

No algorithm can be truly symmetric and guarantee that ties will be resolved in finite time; both of the mutual exclusion algorithms and many other network algorithms use an ordering of the nodes (such as their numbers) as a tiebreaker. Whether a "symmetric" algorithm can use node ordering in a more obvious manner is the question. The first "round" of the original algorithm and the Carvalho and Roucairol modification are identical. After the first round of the modified version, the winner is awarded the right to re-enter its critical section without consultation. Assume that the right to enter the critical section is conferred by a "token." Then in the modification, the token rests with the last node to enter its critical section. By making the token explicit instead of implicit, other interesting mutual exclusion algorithms may be constructed. For example, the algorithm in the Appendix compares favorably as shown in Table I.

The explicit token algorithm of the Appendix suggests yet a further departure: the standard technique of circulating a token among nodes of a communications network. The token gives the bearer exclusive rights to insert a message on the common communications medium. This method uses sequential, node-by-node processing of the token. A node-by-node version of the original algorithm (which does not leave any distinguished nodes when the network is quiescent) was described in Sec. 6.3 of our paper.

In what might be viewed as a tradeoff for using fewer messages, performance of the Carvalho and Roucairol modification will suffer occasionally when, toward the end of a REQUEST/REPLY cycle, it is discovered that the cycle must be restarted with a node k for which $A[k]$ was previously TRUE. In the worst case, no critical sections will be granted for a time proportional to the number of nodes. Also, implementers should note that the procedures must be executed indivisibly; the algorithm will not work if the two processes are allowed to run independently.

In short, Carvalho and Roucairol have raised an interesting question about what constitutes a "symmetric" network algorithm. Our standards admit ours and that of Lamport. If the line is drawn in other ways, the door is open to several different algorithms.

Table I.

Algorithm	Number of messages	Delays to invoke exclusion			
		Note 1	Note 2	Note 3	Note 4
Ricart and Agrawala	2*(N −1)	0.5	1.0	1.0	1.5
Carvalho and Roucairol	0..2*(N − 1)	≧0.5	0.0	1.0	(N − 1)/2
Appendix	0 or N	0.5	0.0	1.0	1.0

N is the number of nodes participating in the mutual exclusion. Assume all nodes are fully interconnected and may communicate in parallel.

Note 1: Average number of roundtrip message times to grant critical section after previous node releases critical section assuming full overlap of initial REQUEST messages with previous node's acquisition and critical section time.
Note 2: Minimum number of roundtrip message times to grant critical section from network quiescent state (after initialization).
Note 3: Typical number of roundtrip message times to grant critical section from network quiescent state (after initialization).
Note 4: Maximum number of roundtrip message times to grant ANY critical section from network quiescent state (after initialization).

Appendix

The algorithm below uses a new message called "TOKEN" to pass the privilege of entering the critical section among a group of cooperating nodes. In the code below, the TOKEN is initialized at process number 1, but it could also have been generated by one pass of the mutual exclusion algorithm of [1]. By retaining the TOKEN even during quiescent periods it is possible to grant mutual exclusion more efficiently—the singular existence of the TOKEN is directly translated into a single node allowed to enter its critical section.

The TOKEN is moved to a node which needs it by the use of REQUEST messages. A requesting node generates a new sequence number in its own series and sends it via a REQUEST message to all other nodes. When a node which has the token but does not need it notices that the request sequence number for some other node is higher than than the TOKEN remembers from its last visit to that node, it passes the TOKEN to the requesting node.

The algorithm satisfies the requirements for mutual exclusion with no deadlock and no starvation.

The procedures should be executed indivisibly; local critical section operations to protect local variables have not been included. The WAITFOR procedure is intended to wait for the arrival of a TOKEN message and copy its data into the "token-data" array.

```
CONSTANT ME,          ! Number of this node;
          N:          ! Total number of nodes;
INTEGER ARRAY token_data, request_data [1:N];
INTEGER our_sequence_number INITIAL (0);
BOOLEAN have_token INITIAL (ME = 1);
BOOLEAN in_critical section INITIAL (FALSE);

PROCEDURE request_resource;
BEGIN
  IF NOT have_token THEN
  BEGIN
    our_sequence_number :=
                          our_sequence_number + 1;
    BROADCAST (REQUEST (our_sequence_number,
                                              ME));
    WAITFOR (TOKEN (token_data)):
  END;
  in_critical_section := have_token := TRUE;
END;

PROCEDURE release_resource;
BEGIN
  INTEGER j;
  token_data[ME] := our_sequence_number;
  in_critical_section := FALSE;
  FOR j := ME + 1 STEP 1 UNTIL N, 1 STEP 1 UNTIL
                                          ME - 1 DO
    IF request_ data[j] > token_data[j] AND have_token
    THEN
    BEGIN
      SEND (TOKEN (token_data), j);
      have_token := FALSE;
    END;
END;

PROCEDURE treat_request_message
                        (their_sequence_number, j);
BEGIN
  request_data[j] :=
        max(request_data[j], their_sequence_number);
  IF have_token AND NOT in_critical_section THEN
                                  release_resource;
END;
```

Glen Ricart
Ashok Agrawala
University of Maryland
College Park, MD 20741

ON FILM COMPRESSION TECHNIQUES

Michael Pechura's article "File Archival Techniques Using Data Compression" [1] on file compression techniques using a Huffman code assignment was instructive and thoughtful. The mechanism provided allows compression of files containing binary and source, a distinct advantage over source only compression algorithms.

However, it being easier to critique than create, I would like to inject an observation about calculating the performance and about the decompress algorithm (EXPAND).

At the end of step 4, the substitution alphabet for each "character" of the input file is available. As a precondition to step 1, the frequency of occurrence for each character and the total number of characters in the input file is known. Therefore, it is possible to calculate the expected improvement.

$$\text{improvement} = \Sigma \text{ frequency*bit\_size} - \text{original size (bits)} \over \text{\# characters}$$

The figure for improvement can then be used to determine the efficacy of using the Data Compress.

As a note, it is possible to reduce some of the overhead in EXPAND by removing the explicit links given for the tree constructed during the COMPRESS phase by observing that a linked complete, static binary tree can be transformed into an unlinked position dependent vector using the following procedure:

(1) The index of the first element in the unlinked tree is 1.
(2) The position of the left link in the table is given by 2*index—1.
(3) The position of the right link in the table is given by 2*index.

Each entry in the resultant table must contain some representation for tuple ⟨terminal mark, substitution⟩ where the terminal mark indicates that the search stops and that the substitution character must be applied (as was done in the original algorithm).

Arthur I. Schwarz
Ultrasystems, Inc.
Irvine, CA 92715

Reference
1. Pechura, M. "File archival techniques using data compression. *Comm ACM 25*, 9 (Sept. 1982), 605–609.

AUTHOR'S RESPONSE:

The formula given by Arthur Schwarz for expected file compression is hard to follow due to a lack of definition for the symbols used. The following may be a more understandable form of what he is trying to express.

Assume that the original file length is *M* bytes, each 8 bits. Then the original file contains 8*M* bits. The compressed file length consists of three parts: the representation of the length of the original file (A bits), the decoding table (B bits) and the actual data of the file in compressed form (C bits). Thus the total length of the compressed file is A+B+B bits. Values for A, B, and C are given by

A = 32 (the length is stored as a 32-bit number);
B = 16*(2N − 1), where N is the number of distinct characters in the original file;
C = $\sum_{i=1}^{N} L_i*F_i$, where L_i is the length in bits of the code assigned to the ith distinct character and F_i is the number of occurrences of the ith distinct character in the original file.

Ignoring possible effects of allocating file space in increments larger than one byte, the percent improvement to be expected would be

$$100\,\frac{8M - (A+B+C)}{8M}.$$

As for the suggestion that an unlinked position dependent vector be used to reduce storage requirements for the decoding table, the following observations may be made:

(1) The decoding tree is in general not complete, thus prohibiting the use of the vector approach. The decoding tree will be complete when the frequency occurrence of all characters in the original file are nearly all equal. In those cases where Huffman coding gives the greatest improvement, those in which some of the characters have considerably higher frequency and are thus assigned short codes, the decoding tree will be far from complete.

(2) The decoding tree used had two bytes at each node, their content being either pointers to the branches off the node or the recognized character. If a vector could be used, each entry would also be of two bytes, one byte required for the recognized character if the entry is a terminal node plus a byte to indicate that the node is terminal. Thus no storage would be saved since there would be no reduction in the number of nodes.

Michael Pechura
Cleveland State University
Cleveland, OH 44115

Abstracts from Other ACM Publications

In the ACM Transactions on Computer Systems February Issue

Computational Algorithms for State-Dependent Queueing Networks
Charles H. Sauer

Queueing networks are important as performance models of computer and communication systems. Exact numerical solution of a queueing network is usually only feasible if the network has a *product form* solution in the sense of Jackson. Product form networks allow a rich variety of forms of state-dependent behavior. However, efficient computational algorithms have not been developed for several of the allowed forms of state-dependent behavior. This paper develops the two most important computational algorithms, Convolution and Mean Value Analysis, to apply to forms of state-dependent behavior allowed in product form networks. It is demonstrated that these two algorithms are of equal generality, contrary to common belief.

For Correspondence: C. H. Sauer, IBM, Dept. 450, Bldg. 984, 11400 Burnett Rd., Austin, TX 78758.

On The Generation of Cryptographically Strong Pseudorandom Sequences
Adi Shamir

This paper shows how to generate from a short random seed a long sequence of pseudorandom numbers which is cryptographically strong in the sense that knowing some sequence elements cannot possibly help the cryptanalyst to determine other sequence elements. The method is based on the RSA cryptosystem, and it is the first published example of a pseudorandom sequence generator for which such a property has been formally proved.

For Correspondence: A. Shamir, Dept. of Applied Mathematics, The Weizmann Inst. of Science, Rehovot, Israel.

Interprocess Communication and Processor Dispatching on the Intel 432
George W. Cox, William M. Corwin, Konrad K. Lai, and Fred J. Pollack

A united facility for interprocess communication and processor dispatching on the Intel 432 is described. The facility is based on a queueing and binding mechanism called a port. The goals and motivations for ports, both abstract an implementation views of them, and their absolute and comparative performance are described.

For Correspondence: G. W. Cox, Intel Corp., 5200 N. E. Elam Young Pkwy., Hillsboro, OR 97123

Cache Performance in the VAX-11/780
Douglas W. Clark

The performance of memory caches is usually studied through trace-driven simulation. This approach has several drawbacks. Notably, it excludes realistic multiprogramming, operating system, and I/O activity. In this paper, cache performance is studied by direct measurement of the hardware. A hardware monitor was attached to a VAX-11/780 computer, whose cache was then measured during normal use. A reproducible synthetic timesharing workload was also run. This paper reports measurements including the hit ratios of data and instruction references, the rate of cache invalidations by I/O, and the amount of waiting time due to cache misses. Additional measurements were made with half the cache disabled, and with the entire cache disabled.

For Correspondence: D. W. Clark, Digital Equipment Corp., 295 Foster St., Littleton, MA 01460.

Implementing Atomic Actions on Decentralized Data
David P. Reed

Synchronization of accesses to shared data and recovering the state of such in the case of failures are really two aspects of the same problem—implementing atomic actions on a related set of data items. In this paper a mechanism that solves both problems simultaneously in a way that is compatible with requirements of decentralized systems is described. In particular, the correct construction and execution of a new atomic action can be accomplished without knowledge of all other atomic actions in the system that might execute concurrently. Further, the mechanisms degrade gracefully if parts of the system fail: only those atomic actions that require resources in failed parts of the system are prevented from executing, and there is no single coordinator that can fail and bring down the whole system.

For Correspondence: D. P. Reed, Laboratory for Computer Science, MIT, Cambridge, MA 02139.