A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems

MAMORU MAEKAWA
University of Tokyo

An algorithm is presented that uses only $c\sqrt{N}$ messages to create mutual exclusion in a computer network, where N is the number of nodes and c a constant between 3 and 5. The algorithm is symmetric and allows fully parallel operation.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*mutual exclusion*; C.2.1 [**Computer Systems Organization**]: Network Architecture and Design—*network communications* C.2.4 [**Computer Systems Organization**]: Distributed Systems—*network operating systems*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Computer networks, decentralized systems

1. INTRODUCTION

Proposed is an algorithm that uses only $c\sqrt{N}$ messages to create mutual exclusion in a computer network, where N is the number of nodes and c a constant between 3 and 5. It is assumed that the nodes communicate only by messages and do not share memory. An error-free underlying communications network supports message transfers in which transit times may vary but messages between two nodes are delivered in the order sent.

The creation of mutual exclusion in a computer network under distributed control is not trivial. Ricart and Agrawala [8] proposed an algorithm that uses $2(N - 1)$ messages: $(N - 1)$ messages to convey a request to all other nodes and $(N - 1)$ messages to obtain permissions from them. It is thus based on a unanimous consensus rule. The algorithm requires that each node requesting mutual exclusion communicate to all other nodes. It is a distributed algorithm, in the sense that each node always bears an equal amount of responsibility to control mutual exclusion and that each node is required to perform an equal amount of work to obtain mutual exclusion, such as the number of request messages. The voting technique used in Thomas [11] is based on a majority consensus rule and requires that a node requesting mutual exclusion obtain a permission vote from only a majority of the nodes. Thus, in the best case, the

Author's address: Department of Information Science, Faculty of Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku Tokyo, 113 Japan.

number of permission messages required to obtain mutual exclusion is reduced to a half, $N/2$. It is also a distributed algorithm in the above sense. The approach was extended by Gifford [5] and Skeen [10] to allow nodes to cast more than one vote. In these weighted voting schemes, it was sufficient to obtain a majority of the votes to obtain mutual exclusion, not necessarily from a majority of the nodes. Garcia-Molina and Barbara then analyzed the relationship between weighted voting and sets of nodes with pairwise nonnull intersections [4]. These weighted voting schemes enjoy the same advantage as the protocol proposed in this paper, in that communication with all nodes in the system is not required. They are not distributed algorithms, however, because nodes with higher weights bear more responsibility to control mutual exclusion than others. In fact, if a particular node has a full weight and all others have no weight, the algorithm is reduced to a centralized control.

The algorithm presented in this paper is a distributed algorithm and requires only $3\sqrt{N}$ messages per mutual exclusion: $\sqrt{N}$ messages to convey a request, $\sqrt{N}$ messages to obtain permissions, and $\sqrt{N}$ messages to release mutual exclusion. It can be proven that this number is optimal for distributed algorithms. The approach taken parallels the voting technique used in Thomas. It also uses *deferral*, the technique used in Ricart and Agrawala. An additional technique, *relinquishment*, is used, however, to avoid deadlocks.

2. REQUEST RESOLUTION

In distributed systems, each network node issues a mutual exclusion request at an arbitrary time. In order to arbitrate these requests, any pair of two requests must be known to one of the arbitrators. Since nodes themselves must serve as arbitrators, any pair of two requests must reach to a certain common node. If we assume that node i obtains a permission from each member of a subset S_i of the nodes of the network to obtain mutual exclusion then there must exist at least one common node between a pair of S_i and S_j for any i and j so that the common node can serve as an arbitrator. Therefore, the S_i's must satisfy the pairwise nonnull intersection property. Assuming that the network consists of N nodes numbered from 1 to N, this nonnull intersection property is stated as follows:

(a) For any combination of i and j, $1 \leq i, j \leq N$, $S_i \cap S_j \neq \emptyset$.

The request resolution rule then requires that when node i attempts to invoke mutual exclusion, it send a REQUEST message to every member of S_i and obtain a permission from all of them. Since each member of S_i serves as an arbitrator, the requesting node knows that it is the only node that has been granted mutual exclusion, when every member of S_i returns a permission message. Node S_i then proceeds to its critical section. This nonnull intersection property is a necessary condition for the S_i's so that mutual exclusion requests can be resolved. In addition, the following properties are required or desirable for truly distributed algorithms:

(b) S_i, $1 \leq i \leq N$, always contains i.
(c) The size of S_i, $|S_i|$, is K for any i. That is,

$$|S_1| = |S_2| = |S_3| = \cdots = |S_N| = K.$$

(d) Any j, $1 \le j \le N$, is contained in the D S_i's, $1 \le i \le N$.

Property (b) is included simply to reduce the number of messages to be sent and received by a node, respectively, because, if a requesting node i is itself a member of its own node subset S_i, a permission from itself is obtained without a message transmission. Properties (c) and (d) are included to have a truly distributed algorithm. Property (c) implies that each node needs to send and receive the same number of messages to obtain mutual exclusion. Property (d), on the other hand, implies that each node serves as an arbitrator for the same number of nodes. That is, each node bears an equal amount of responsibility for mutual exclusion control.

A centralized algorithm assigns a single node as a controller (arbitrator) for mutual exclusion management. It satisfies properties (a) and (c), where $K = 1$, but violates property (d). Ricart and Agrawala's algorithm satisfies all of the above properties, where $K = N$ and $D = N$. Thomas's majority consensus algorithm can also satisfy all the above properties, where $K = N$ and $D = N$. Weighted voting schemes satisfy property (a) but usually violate properties (c) and (d).

3. THE CHOICE OF S_i's

The selection of S_i's is not unique. There exists a number of ways to select a set of S_i's that satisfies the above properties. From properties (b) and (d), each member of S_i can be contained in $(D - 1)$ other subsets. Therefore, the maximum number of subsets that satisfy property (a) is given by

$$(D - 1)K + 1.$$

Since N is desired to be set to this maximum number so that K is minimized for a given N, we have

$$N = (D - 1)K + 1.$$

Furthermore, $K =$ D must always hold, because N is the number of distinct members, which is given by KN/D, the total number of members divided by the number of duplications of each member. N is thus related to K by

$$N = K(K - 1) + 1.$$

The problem of finding a set of S_i's that satisfies these conditions is equivalent to finding a finite projective plane of N points. It is known that there exists a finite projective plane of order k if k is a power p^m, of a prime p [1]. This finite projective plane has $k(k + 1) + 1$ points. Hence, in our terms, a set of S_i's exists if $(K - 1)$ is a power of a prime. For other values of k, we can create a set of S_i's by relaxing conditions (c) and (d) to some extent. For values of N, which cannot be expressed as $K(K - 1) + 1$, we can also apply the same method to create a degenerated set of S_i's. The creation of S_i's is discussed in detail in Section 7. Here, we only show examples for $K = 2$, 3, 4 and 5 (Figure 1).

From the above discussion, it is clear that K gives the optimal value for a given N when all the properties (a)–(d) are required. With a fractional error, we see that $K = \sqrt{N}$.

$$\begin{aligned}
S_1 &= \{1,\ 2\}\\
S_3 &= \{1,\ 3\}\\
S_2 &= \{2,\ 3\}
\end{aligned}$$

(a)

$$\begin{aligned}
S_1 &= \{1,\ 2,\ 3\}\\
S_4 &= \{1,\ 4,\ 5\}\\
S_6 &= \{1,\ 6,\ 7\}\\
S_2 &= \{2,\ 4,\ 6\}\\
S_5 &= \{2,\ 5,\ 7\}\\
S_7 &= \{3,\ 4,\ 7\}\\
S_3 &= \{3,\ 5,\ 6\}
\end{aligned}$$

(b)

$$\begin{aligned}
S_1 &= \{1,\ 2,\ 3,\ 4\}\\
S_5 &= \{1,\ 5,\ 6,\ 7\}\\
S_8 &= \{1,\ 8,\ 9,\ 10\}\\
S_{11} &= \{1,\ 11,\ 12,\ 13\}\\
S_2 &= \{2,\ 5,\ 8,\ 11\}\\
S_6 &= \{2,\ 6,\ 9,\ 12\}\\
S_7 &= \{2,\ 7,\ 10,\ 13\}\\
S_{10} &= \{3,\ 5,\ 10,\ 12\}\\
S_3 &= \{3,\ 6,\ 8,\ 13\}\\
S_9 &= \{3,\ 7,\ 9,\ 11\}\\
S_{13} &= \{4,\ 5,\ 9,\ 13\}\\
S_4 &= \{4,\ 6,\ 10,\ 11\}\\
S_{12} &= \{4,\ 7,\ 8,\ 12\}
\end{aligned}$$

(c)

$$\begin{aligned}
S_1 &= \{1,\ 2,\ 3,\ 4,\ 5\}\\
S_6 &= \{1,\ 6,\ 7,\ 8,\ 9\}\\
S_{10} &= \{1,\ 10,\ 11,\ 12,\ 13\}\\
S_{14} &= \{1,\ 14,\ 15,\ 16,\ 17\}\\
S_{18} &= \{1,\ 18,\ 19,\ 20,\ 21\}\\
S_2 &= \{2,\ 6,\ 10,\ 14,\ 18\}\\
S_7 &= \{2,\ 7,\ 11,\ 15,\ 19\}\\
S_8 &= \{2,\ 8,\ 12,\ 16,\ 20\}\\
S_9 &= \{2,\ 9,\ 13,\ 17,\ 21\}\\
S_{11} &= \{3,\ 6,\ 11,\ 17,\ 20\}\\
S_3 &= \{3,\ 7,\ 10,\ 16,\ 21\}\\
S_{13} &= \{3,\ 8,\ 13,\ 15,\ 18\}\\
S_{12} &= \{3,\ 9,\ 12,\ 14,\ 19\}\\
S_{15} &= \{4,\ 6,\ 12,\ 15,\ 21\}\\
S_4 &= \{4,\ 7,\ 13,\ 14,\ 20\}\\
S_{17} &= \{4,\ 8,\ 10,\ 17,\ 19\}\\
S_{16} &= \{4,\ 9,\ 11,\ 16,\ 18\}\\
S_{19} &= \{5,\ 6,\ 13,\ 16,\ 19\}\\
S_5 &= \{5,\ 7,\ 12,\ 17,\ 18\}\\
S_{21} &= \{5,\ 8,\ 11,\ 14,\ 21\}\\
S_{20} &= \{5,\ 9,\ 10,\ 15,\ 20\}
\end{aligned}$$

(d)

Fig. 1. Subsets of integers with pairwise nonnull intersection property. (a) $K = 2$; (b) $K = 3$; (c) $K = 4$; (d) $K = 5$

4. ALGORITHM

Each node executes an identical algorithm. The algorithm is based on the fact that, if node i locks all members of S_i, no other node can capture all its members because of property (a). Therefore, when it invokes mutual exclusion, node i tries to lock all members of S_i. If it succeeds, it can enter its critical section. If it fails, it waits for all its member nodes to be freed, at which point it captures and locks them. It then enters its critical section. Since there is a danger of deadlock when more than one node simultaneously requests mutual exclusion, a node will yield to others if the priority of its request is lower than that of any other conflicting request. The request's priority is determined by the sequence number (timestamp) of the request's corresponding REQUEST message. A REQUEST with a smaller sequence number is given higher priority and is said to *precede* other REQUESTs with larger sequence numbers. If a newly arrived REQUEST at a member node precedes the current locking REQUEST, the node sends an INQUIRE message to the node originating the current locking REQUEST to inquire whether the originating node will really succeed in capturing all its members. The originating node will return a RELINQUISH message when it becomes apparent that the node will not be able to capture all its members. On the other hand, if the originating node has succeeded in capturing all its members, it will return a RELEASE message only after it has completed its critical section operation.

The algorithm is now described below:

(1) When node i invokes mutual exclusion, it sends a REQUEST message to every member of S_i. Node i pretends to have received a REQUEST. The REQUEST message is given a sequence number greater than any REQUEST message sent, received, or observed at this node.

(2) Upon receiving a REQUEST, a member node of S_i marks itself *locked* for the REQUEST if it is not currently locked for another REQUEST, and then returns a LOCKED message to the requesting node i. If the node is locked for a REQUEST from another node, the REQUEST from node i is placed in the WAITING QUEUE of the node. (These REQUEST messages placed in the WAITING QUEUE are called *outstanding REQUESTs.*) It is then tested to determine whether the current locking REQUEST or any other outstanding REQUEST at the node precedes the received REQUEST. (See below for the definition of the *locking REQUEST.*) If so, a FAILED message is returned to node i. Otherwise, an INQUIRE message is sent to the node originating the current locking REQUEST to inquire whether this originating node has succeeded in locking all its members. If an INQUIRE has already been sent for a previous REQUEST and its reply message (either RELINQUISH or RELEASE) has not yet been received, it is not necessary to send in INQUIRE. REQUEST A is said to precede REQUEST B if (the sequence of number $A <$ the sequence number of B) or ((the sequence of number $A =$ the sequence number of B) and (the node number of $A <$ the node number of B)). Each node can be locked by only one REQUEST at a time, and this REQUEST is called the locking REQUEST. Any subsequent REQUESTs arrived at the node are placed in the WAITING QUEUE of the node in decreasing order of the precedence defined above.

(3) When a node receives an INQUIRE message, it returns a RELINQUISH message if it knows that it will not succeed in locking all its members; that is, it has received a FAILED message from some of its members. By so doing, the node relinquishes its member node to a more preceding REQUEST. This breaks a circular locking, which is necessary to avoid deadlocks. The node cancels the LOCKED message previously received from the member node. When the node has succeeded in locking all its members and is in its critical section, it returns a RELEASE message, but only after

it has completed its critical section. If an INQUIRE message has arrived before it is known whether the node will succeed or fail to lock all its members, a reply is deferred until this becomes known. If an INQUIRE message has arrived after the node has sent a RELEASE message, it is simply ignored.

(4) When a node receives a RELINQUISH message, it relieves itself of the current locking REQUEST and then locks itself for the most preceding REQUEST in the WAITING QUEUE. Thus, regardless of which REQUEST had caused the sending of an INQUIRE, the node is locked for the REQUEST that happens to be most preceding when a RELINQUISH message is received. The current locking REQUEST is placed in the WAITING QUEUE, whereas the most preceding REQUEST is removed from it. A LOCKED message is then returned to the node originating the new locking REQUEST.

(5) If all members of S_i have returned a LOCKED message, node i enters its critical section.

(6) Upon completing the critical section, node i sends a RELEASE message to each member of S_i.

(7) When a node receives a RELEASE message, it relieves itself from the current locking REQUEST. It deletes this locking REQUEST and then relocks itself for the most preceding REQUEST in the WAITING QUEUE if the queue is not empty. A LOCKED message is returned to the node originating the new locking REQUEST. If the WAITING QUEUE is empty, the node marks itself *unlocked*.

(8) The above steps (1)–(7) are repeated for each mutual exclusion request.

5. AN EXAMPLE

Imagine a 13-node network using this algorithm. Initially, the sequence number at each node is zero.

Figure 2a shows a sequence of mutual exclusion invocations in which nodes 7, 8, and 11 invoke mutual exclusion in the order below. They all send a REQUEST message with a sequence number 1 to their respective members.

(1) Node 11 is the first to attempt mutual exclusion. Its REQUESTs have arrived at nodes 12 and 13 and have locked them, but its REQUEST to node 1 is still on its way.

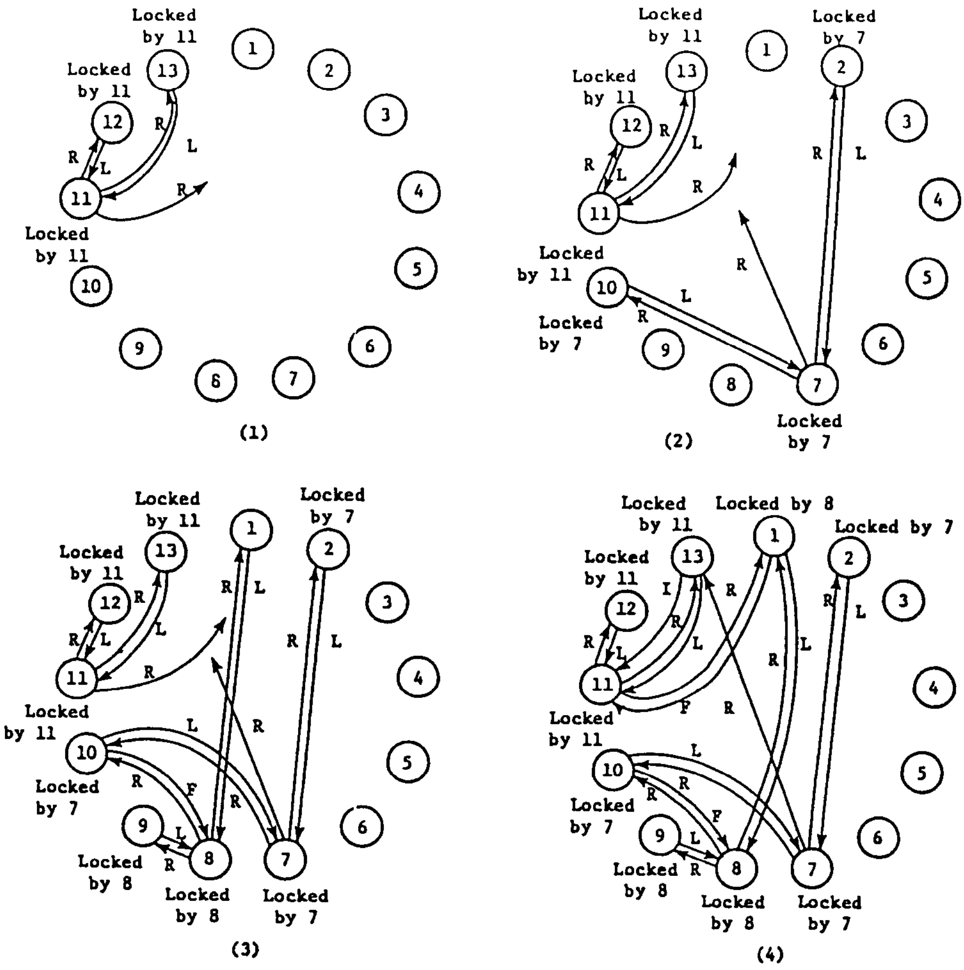
(2) Node 7 then invokes mutual exclusion. Its REQUESTs have arrived at nodes 2 and 10 and have locked them but its REQUEST to node 13 is still on its way.

(3) Node 8 then invokes mutual exclusion. It locks itself and sends a REQUEST to nodes 1, 9, and 10 but fails to lock node 10 because node 10 has already been locked by a preceding REQUEST from node 7.

(4) The REQUEST message originating at node 11 has finally arrived at node 1, while the REQUEST message from node 7 arrives at node 13. Node 1 then returns a FAILED, whereas node 13 sends an INQUIRE message to node 11.

This sequence creates a situation where nodes 7, 8, and 11 circularly lock each other. Node 8 receives a FAILED message and cannot enter its critical section. Likewise, node 11 cannot enter its critical section because it receives a FAILED message from node 1. Node 7 still waits because it has not received a LOCKED from all its member nodes.

When an INQUIRE message has been received at node 11, node 11 knows that it cannot enter its critical section and thus returns a RELINQUISH message to node 13. This will cause node 13 to be released for the most preceding REQUEST in its waiting queue, which is the REQUEST from node 7. This REQUEST then locks node 13 and returns a LOCKED to node 7. Node 7 then can enter its critical section (Figure 2b).



S_1	=	{1,	2,	3,	4}
S_2	=	{2,	5,	8,	11}
S_3	=	{3,	6,	8,	13}
S_4	=	{4,	6,	10,	11}
S_5	=	{1,	5,	6,	7}
S_6	=	{2,	6,	9,	12}
S_7	=	{2	7,	10,	13}
S_8	=	{1,	8,	9,	10}
S_9	=	{3,	7,	9,	11}
S_{10}	=	{3,	5,	10,	12}
S_{11}	=	{1,	11,	12,	13}
S_{12}	=	{4,	7,	8,	12}
S_{13}	=	{4,	5,	9,	13}

Fig. 2a. Circular locking. R = Request; L = Locked; F = Failed; I = Inquire; Q = Relinquish.

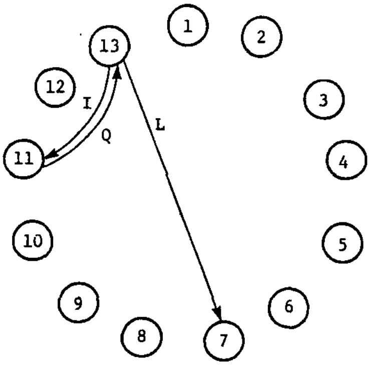


Fig. 2b. Relinquishing.

Upon completing the critical section, node 7 deletes its REQUEST from its member nodes by sending a RELEASE message. This will cause node 8 to successfully lock all its members and enter its critical section. Finally node 11 completes its critical section.

It is possible that a new REQUEST is initiated during the above process. Suppose that a REQUEST from node 3 has arrived at node 13 after the INQUIRE message was sent but before the RELINQUISH message from node 11 arrives at node 13. Since this REQUEST precedes any REQUEST at node 13, and since it is known that an INQUIRE was sent, the REQUEST waits for a RELINQUISH. When the RELINQUISH message is received at node 13, the REQUEST from node 3 locks node 13 instead of the REQUEST from node 7. Node 3 will then succeed in locking all its members when node 8 relinquishes itself to node 3.

6. PROOF

6.1 Mutual Exclusion

Assume the contrary, that more than one node are simultaneously in the critical section. The following arguments show that this is not possible:

(1) All the nodes in the critical section must have received a LOCKED message from all their respective member nodes (step 5).

(2) Since a node in a critical section never releases its member nodes until it completes its critical section (step (6)), and since each member node returns a LOCKED message only when it locks itself for the corresponding REQUEST (steps (2) and (4)), there must be a node that is simultaneously locked for more than one REQUEST owing to property (a).

(3) However, this contradicts the specification of the algorithm that allows only one REQUEST to lock a node at any instance (steps (2) and (4)).

(4) Therefore, more than one node cannot simultaneously be in the critical section.

6.2 Deadlock

Assume that deadlock is possible. Then there must exist a circular waiting among the nodes requesting mutual exclusion. This is not possible, however, because

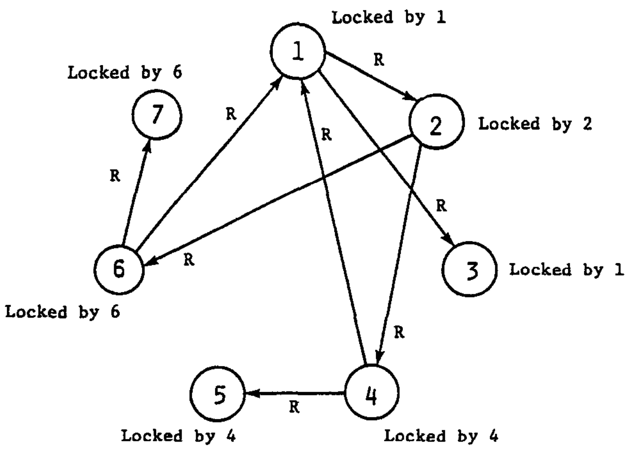


Fig. 3. A circular waiting. 1 waits for 2; 2 waits for 4 and 6; 4 waits for 1; 6 waits for 1.

(1) Any sequence number/node number pair (timestamp) of a REQUEST is unique because the sequence number of each node is incremented for a new REQUEST and each node number is unique (step 1). Any set of sequence number/node number pairs can be uniquely ordered with the largest and the smallest.

(2) Then, in this circular waiting, there must exist a node whose REQUEST's timestamp is preceded by those of both of its adjacent nodes in the circular waiting. The removal of this preceded node leads to a break of the cycle.

(3) Two adjacent competing nodes in the circular waiting have at least one node common as a member node due to property (a). Then at this common node, two REQUESTs can be ordered in terms of their timestamps (step 2).

(4) If the preceding REQUEST cannot lock the node because the preceded node is currently locking, it causes an INQUIRE message to be sent to the node originating the preceded REQUEST (step 2). (Note that if a preceding REQUEST can always lock a node, there will not be any circular waiting.)

(5) The node originating the preceded REQUEST will relinquish the completed member node by returning a RELINQUISH message if it knows that it will not succeed in locking all its members (step 3). By observation (2) above, there exists a node whose REQUEST's timestamp is preceded by those of both of its adjacent nodes in the circular waiting. Since this node is in a circular waiting, one of its REQUESTs must have arrived at one of its member nodes later than a REQUEST from one of its adjacent nodes in the circular waiting. Therefore, the node must receive a FAILED message (step 2). The node then returns a RELINQUISH message. This breaks the circular waiting and the node that has received the RELINQUISH will succeed in locking all its member nodes (step 4).

Example. Let us assume that all REQUESTs in Figure 3 have the same sequence number. Then the REQUEST from node 1 is most preceding. The circular waiting is broken because the REQUESTs from node 4 and 6 are preceded by the REQUESTs from node 1 and thus their members are relinquished. This will allow node 2 to succeed in locking all its members and allow it to enter its critical section.

6.3 Starvation

The starvation of node i occurs when other preceding REQUESTs are continuously locking or waiting at a member of S_i. In this case, a FAILED is returned to node i, which will wait for a LOCKED message. A LOCKED message will eventually be returned to node i from the member node when the REQUEST originating at node i becomes the most preceding outstanding REQUEST at this member node. This will occur after, at most, $(K - 1)$ REQUESTs have been processed at the member node, because any subsequent REQUEST that arrives at the member node will have a sequence number larger than the current REQUEST originating at node i. Therefore, in a finite time, node i will succeed in locking this member node. Since this is true for every member node, node i will succeed in locking all its member nodes in a finite time.

7. THE CREATION OF S_i's

The choice of S_i's affects the number of messages required to create mutual exclusion. It is desirable to have S_i's that are symmetric and of which the size of each subset is minimum. Symmetry is required to have a truly distributed system, as discussed in Section 2. These two conditions are both satisfied when there exists a finite projective plane of N points. Although it is known that a finite projective plane of order k exists if k is a power of a prime, very little is known about general finite projective planes for other values of k. The Bruck–Ryser theorem [1] is the only result in this direction, and states that there exists no finite projective plane of order k if either $k - 1$ or $k - 2$ is divisible by 4 and if k cannot be expressed as the sum of two integral squares ($k \neq a^2 + b^2$ for a and b nonnegative integers). If a corresponding finite projective plane does not exist or if N is not expressed as $K(K - 1) + 1$, one or both of the above two conditions must be sacrificed. We show two methods that create a near-optimal set of S_i's.

Method 1. Suppose that $(K - 1)$ is not a power of a prime number. Then there may not exist a corresponding finite projective plane. However, we can create a degenerated set of S_i's for this value of $K = L$ by the following method:

(a) We first create a symmetric set of S_i's for M where $(M - 1)$ is a power of a prime number and M is the smallest integer larger than L. In this set of S_i's for $K = M$, each component is contained in M subsets.

(b) We then replace each component greater than $N = L(L - 1) + 1$ in this set of S_i's for $K = M$ by a number smaller than or equal to $N = L(L - 1) + 1$. Then each component will be contained in L subsets in the resulting set of S_i's. We assume that this replacement is made by a different number each time. The resulting set of S_i's is not symmetric in the sense that the size of S_i is not always L.

The mutual exclusion algorithm using these S_i's produces a somewhat unbalanced performance for nodes because some node may have to send an extra message. But, on the average, the load of each node is balanced. Therefore, the number of messages per mutual exclusion remains the same as that calculated in Section 3. The largest gap between L and M is only 2 for $K \le 20$ ($L = 15$ and $M = 17$) and only 4 for $K \le 50$ ($L = 34$ and $M = 38$).

When N cannot be expressed as $K(K - 1) + 1$, we can create a degenerate set of S_i's in a similar way. In order to obtain S_i's for $N = 5$, for instance, we first create a set of S_i's for $M = 7$ using the method described in 2.2 and then replace 7 and 6 with 5 and 4, respectively, and remove S_7 and S_6, which produces the following set of S_i's:

$$\begin{aligned} S_1 &= \{1, 2, 3\}, \\ S_2 &= \{2, 4\}, \\ S_3 &= \{3, 5, 4\}, \\ S_4 &= \{1, 4, 5\}, \\ S_5 &= \{2, 5\}. \end{aligned}$$

Method 2. Consider a grid of $L \times L$, and number the L^2 grid points from 1 to L^2. A subset S_i is defined to be the set of grid points on the row or the column passing through point i. Then it is clear that $S_i \cap S_j \neq \emptyset$ for any i and j, $1 \le i, j \le L^2$. The set of S_i's is symmetric in the sense that $| S_i | = 2L - 1$ for any i and that any i is contained in $(2L - 1)$ subsets. In this construction,

$$| S_i | = 2\sqrt{N} - 1 \qquad \text{for any} \quad i.$$

Therefore, the number of messages per mutual exclusion is about twice that calculated in Section 3.

If N is not a square of an integer, we can create a degenerate grid whose outermost row (and column, if necessary) is reduced in size. Any fractional row or column is completed by complementing its missing part from another row or column when S_i's are determined.

8. MESSAGE TRAFFIC

We discuss two cases separately.

8.1 Under Light Demand

When the demand is light and contention rarely occurs, one instance of mutual exclusion requires $(K - 1)$ REQUEST messages and $(K - 1)$ LOCKED messages to ensure that all members of S_i have been locked, and $(K - 1)$ RELEASE messages to clear the REQUESTs. A total of three $(K - 1)$ messages are required. Table I shows the comparison with Ricart and Agrawala's algorithm. It is seen that under light demand the proposed algorithm almost always requires fewer messages than Ricart and Agrawala's algorithm.

The above examples are for those values of K for which a finite projective plane exists. When a finite projective plane does not exist, some redundancy exists among S_i's and the number of messages required to create mutual exclusion increases accordingly. Such cases are shown in Table II, where the values are computed assuming that a mutual exclusion request is made uniformly from each node. The advantage of the proposed algorithm is apparent, even in these degenerate cases.

Table I

N	Proposed algorithm	Ricart and Agrawala's algorithm
3 ($K = 2$)	3	4
7 ($K = 3$)	6	12
13 ($K = 4$)	9	24
21 ($K = 5$)	12	40
133 ($K = 12$)	33	264
381 ($K = 20$)	57	760

Table II

N	Proposed algorithm	Ricart and Agrawala's algorithm
5	4.8	8
6	5.5	10
10	8.1	18
18	11.7	34

8.2 Under Heavy Demand

Under heavy demand, a new REQUEST will most likely fail to lock its destination node. Thus, we expect to have $(K - 1)$ REQUEST messages, $(K - 1)$ FAILED messages, $(K - 1)$ LOCKED messages to obtain mutual exclusion, and $(K - 1)$ RELEASE messages to release the REQUEST. A total of four $(K - 1)$ messages are required per mutual exclusion.

If a new REQUEST is initiated from a node that has neither requested mutual exclusion nor participated in the algorithm as a member node for a certain period, it will most likely precede other REQUESTs. It then causes an INQUIRE message to be sent, for which a RELINQUISH is returned. In this case, $(K - 1)$ REQUEST messages, $(K - 1)$ INQUIRE messages, $(K - 1)$ RELINQUISH messages, and $(K - 1)$ LOCKED messages are required to obtain mutual exclusion. Hence, five $(K - 1)$ messages are altogether required per mutual exclusion. This is the worst case because a RELINQUISH message is not needed when the node is already in a critical section or is winning to obtain mutual exclusion. Furthermore, it is expected that under heavy demand almost all nodes participate in the algorithm as a requestor or a member node.

9. NODE FAILURE

It is important to consider node failures in distributed systems. Although nodes can fail in many ways [7–9], only those failure nodes that stop functioning and cannot return messages are considered here. In such failures, all information kept in the failed nodes is lost. We assume that a node failure can be detected by another node and a failed node is removed from the system. A simple approach for node removal is to have another node to take over the role of the failed node. This corresponds to the degeneration described above and will cause the overtaking node to play a somewhat greater role.

Table III. A Comparison with Other Algorithms

	Fully centralized algorithm (one control node)	Proposed algorithm	Ricart and Agrawala's algorithm
The number of nodes to which a REQUEST must be sent	1	$\sqrt{N} - 1$	$N - 1$
The number of nodes about which each (control) node keeps dynamic information	N	$\sqrt{N}$	N
The number of nodes, dynamic information about which is lost by a (control) node failure	N	$\sqrt{N}$	1
The number of nodes about which each (control) node keeps static information	0	$\sqrt{N}$	N
Removal of a (control) node	Needs a back-up control node	Dynamically possible	Dynamically possible
Overtaking by another node	Needs a back-up control node	Dynamically possible	Not necessary

In considering a node removal, we must consider the amount of information kept in and transferred between nodes that is necessary to execute a mutual exclusion algorithm.

Table III summarizes for three algorithms, including a fully centralized algorithm. Thomas's majority consensus algorithm [11] is basically the same as Ricart and Agrawala's algorithm. The fully centralized algorithm is executed by one control node that manages mutual exclusion. It requires only one REQUEST to be sent to the control node per mutual exclusion, whereas the other algorithms require ($\sqrt{N} - 1$) and ($N - 1$) messages, respectively. This is a penalty that has to be paid to have a distributed algorithm.

In order for any mutual exclusion algorithm to operate, each (control) node must have operational information, by dynamic and static. The dynamic information is information about messages and the status of the related nodes, whereas the static information is information that will never be changed once initialized, such as the total number of nodes and the node numbers. A node removal affects both the dynamic and static information. In case of the dynamic information, a removal of a (control) node does not cause a loss of the dynamic information if the dynamic information is duplicated in other nodes. A failed node can simply be removed. However, the static information in each (control) node must be modified. In Ricart and Agrawala's algorithm, a node removal causes no loss of dynamic information but requires a modification of the static information in each node. This requires $O(N)$ messages. On the other hand, in the fully centralized algorithm, all dynamic information is lost by the removal of the control node, whereas no static information is lost. A backup controller is required to take over the failed controller and $O(N)$ messages are required to regain dynamic information. In the algorithm proposed in this paper, dynamic information about $(\sqrt{N} - 1)$ nodes is lost by a removal of the failed node. This is the reason why another node should logically take over the role of the failed node. Any other node can logically take that role over because each node executes an identical algorithm. The lost dynamic information can be regained by $O(\sqrt{N})$, instead of $O(N)$, messages. The static information also needs to be modified in only $(\sqrt{N} - 1)$ nodes. Therefore, it is generally expected that the proposed algorithm requires fewer messages than the other two algorithms to remove a node despite the fact that some dynamic information is lost. This is summarized in Table III.

10. VARIATIONS

The algorithm presented in Section 4 simultaneously sends a REQUEST to each member node and allows fully parallel operation. Thus, the delay incurred in running the algorithm is the sum of the time it takes to send a REQUEST and receive a LOCKED message. This is the minimum delay required to run any mutual exclusion algorithm. Ricart and Agrawala's and Thomas's algorithms also basically have the same delay. If a greater delay is tolerated, REQUEST messages can be sent in a systemwise prespecified order, one by one, only after a LOCKED message is returned for the previous REQUEST. This will simplify the algorithm and requires only two $(K - 1)$ messages to create an instance of mutual exclusion. This can further be reduced to K message passes by cyclically passing a REQUEST among the member nodes. In either method, additional $(K - 1)$ messages or message passes are required to clear the REQUESTs.

11. CONCLUSIONS

A distributed algorithm that creates mutual exclusion using $c\sqrt{N}$ messages, where c is a constant between 3 and 5, has been presented. The algorithm is symmetric and allows fully parallel operation. It also allows a node removal. The proposed algorithm is optimal in terms of the number of messages used to create mutual exclusion among fully distributed algorithms, where the term *distributed* is used here to mean that each node serves as an arbitrator for the same number of nodes.

Several mutual exclusion algorithms for distributed systems are available [3-6, 8-11], as well as a number of their variations. These algorithms vary in many respects, including the degree of distribution of control, the degree of parallel operation, traffic intensity, the delay incurred, applicable network topologies, and reliability. In applying these algorithms to a real system, a suitable algorithm will be selected depending on such factors as network topology, network size, and performance, reliability, and extensibility requirements.

ACKNOWLEDGMENTS

Kiyoshi Ishihata read the draft and suggested many improvements. The comments of the referees were most helpful in clarifying ambiguous points and improving the presentation.

REFERENCES

1. ALBERT, A. A., AND SANDLER, R. *An Introduction to Finite Projective Planes.* Holt, Rinehart, and Winston, New York, 1968.
2. CARVALHO, O. S. F., AND ROUCAIROL, G. On mutual exclusion in computer networks. *Commun. ACM 26,* 2 (Feb. 1983), 146–147.
3. CHANG, E. J. H., AND ROBERTS, R. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Commun. ACM 22,* 5 (May 1979), 281–283.
4. GARCIA-MOLINA, H., AND BARBARA, D. How to assign votes in a distributed system. Tech. Rep. 311, Dept. of Electrical Engineering and Computer Science, Princeton Univ., Princeton, N.J., 1983.
5. GIFFORD, D. K. Weighted voting for replicated data. In *Proceedings of the 7th Symposium on Operating System Principles* (Pacific Grove, Calif., Dec. 10–12), ACM, New York, 1979, pp. 150–162.
6. HIRSCHBERG, D. S., AND SINCLAIR, J. B. Decentralized extrema-finding in ciruclar configurations of processors. *Commun. ACM 23,* 11 (Nov. 1980), 627–628.
7. LAMPORT, L. The implementation of reliable distributed multiprocess systems. *Comput. Networks* 2 (1978), 95–114.
8. RICART, G., AND AGRAWALA, A. K. An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM 24,* 1 (Jan. 1981), 9–17.
9. SCHNEIDER, F. B. Synchronization in distributed programs. *ACM Trans. Program. Lang. Syst. 4,* 2 (Apr. 1982), 125–148.
10. SKEEN, D. A quorum-based commit protocol. In *Proceedings of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks* (Feb. 1982), pp. 69–80.
11. THOMAS, R. H. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst. 4,* 2 (June 1979), 180–209.

Received June 1983; revised March 1984; accepted July 1984