

## A DISTRIBUTED ALGORITHM FOR MULTIPLE ENTRIES TO A CRITICAL SECTION

Kerry RAYMOND

*Department of Computer Science, University of Queensland, St Lucia, Queensland 4067, Australia*

Communicated by P.C. Poole

Received 11 March 1988

Revised 30 August 1988

Ricart and Agrawala's algorithm for distributed mutual exclusion is extended to enable up to  $K$  nodes to be within the critical section simultaneously. Our algorithm requires at most  $2*(N-1)$  messages per entry to the critical section, and occasionally fewer.

*Keywords:* Distributed programming, synchronization, mutual exclusion, critical section

### 1. Introduction

In keeping with earlier work [1–4], we assume a computer network of  $N$  nodes, communicating solely via messages. Message delivery is assumed to be guaranteed by underlying communications software, but the transit time for messages and their order of delivery are unpredictable.

Ricart and Agrawala [3] have presented an algorithm for mutual exclusion (allowing at most 1 node to execute in a critical section). Their algorithm requires  $2*(N-1)$  messages to be exchanged for each entry to the critical section. We propose an extension to this algorithm to allow up to  $K$  nodes to execute within the critical section. Surprisingly, our algorithm usually requires the same number of messages as the original algorithm, and occasionally fewer.

### 2. Ricart and Agrawala's algorithm

A node  $X$  may enter its critical section if it is aware that all of the other  $N-1$  nodes are not in the critical section. To determine this, node  $X$  sends a REQUEST message to the other  $N-1$  nodes.

Upon receiving a REQUEST message, a node which is not within a critical section immediately sends a REPLY message to  $X$ . If a node  $Y$  is within the critical section, it defers sending the REPLY message to  $X$  until node  $Y$  has exited from the critical section.

If node  $Y$  is also wishing to enter the critical section, then the sequence number of  $X$ 's request is compared with the sequence number sent in node  $Y$ 's REQUEST messages (with node identity used as a tie-breaker). If  $X$ 's sequence number is smaller, node  $Y$  sends its REPLY to  $X$ . Otherwise node  $Y$  defers its REPLY to  $X$  until node  $Y$  has entered and then exited from the critical section.

When node  $X$  receives all  $N-1$  REPLY messages, it may enter the critical section. This requires  $N-1$  REQUEST messages and  $N-1$  REPLY messages, a total of  $2*(N-1)$  messages per entry to the critical section.

### 3. Our extended algorithm

The aim of our algorithm is to restrict the number of nodes executing within the critical section to a maximum of  $K$  nodes. Therefore our algorithm permits a node to enter the critical

section if no more than  $K - 1$  of the other  $N - 1$  nodes are currently executing within the critical section, i.e., at least  $(N - 1) - (K - 1)$  of the other  $N - 1$  nodes are NOT within the critical section. A node wishing to enter the critical section sends REQUEST messages to each of the other  $N - 1$  nodes, and may enter the critical section as soon as  $N - K$  REPLY messages are received.

Like Ricart and Agrawala's algorithm, the sending of a REPLY message from node  $Y$  to node  $X$  is deferred if node  $Y$  is executing within the critical section or if  $Y$  is wishing to enter the critical section and has the lower sequence number.

In Ricart and Agrawala's algorithm, REPLY messages arrive at node  $X$  only while node  $X$  is waiting to enter the critical section. In our algorithm, the node  $X$  will receive the first  $N - K$  messages while waiting to enter the critical section, but the remaining  $K - 1$  REPLY messages will arrive while  $X$  is executing in the critical section, or after  $X$  has left the critical section, or during the wait for another entry to the critical section and so on. REPLY messages pertaining to an earlier entry to the critical section must not be mistaken for REPLY messages pertaining to the current attempt.

A node may defer many REPLYs to another node. For example, consider a network of three nodes  $X$ ,  $Y$  and  $Z$  with at most 2 nodes allowed in the critical section. Initially  $X$  obtains the right to enter the critical section, and executes within the critical section for a long period of time. During this time, node  $Y$  can enter and exit the critical section many times. Each time node  $Z$  will send a REPLY (sufficient to allow  $Y$  to enter), and node  $X$  will defer the REPLY. In this way, node  $X$  may be deferring many REPLYs to node  $Y$ .

#### 4. Information held by each node

Each node participating in the algorithm holds the the following data:

Requesting\_CS, Executing\_CS: boolean := false, false;

```
/* true when a node is requesting/executing the
critical section */
Max_Seq: integer := 0;
/* the largest sequence number seen in any RE-
QUEST message */
Our_Seq: integer := 0;
/* the sequence number of the current attempt to
enter the mutual exclusion */
```

The sequence number used in REQUEST messages (Our\_Seq) is always chosen to be larger than any sequence number previously received in REQUEST messages from another node (recorded in Max\_Seq).

```
Reply_Count: array [1..N] of integer := 0;
/* the number of REPLY messages still to be
received from each node */
```

Each node maintains a count of the number of outstanding REPLY messages still to come from each of the other nodes. If Reply\_Count [ $Y$ ] = 0, then all REQUEST messages sent to  $Y$  have been REPLYed to, and hence node  $Y$  is not executing within its critical section. Thus a REPLY to an earlier REQUEST is not regarded as a REPLY to any current attempt to enter the critical section.

Therefore this node may enter the critical section when

$$|\text{Not\_In\_CS}| \geq N - K.$$

where

$$\begin{aligned} \text{Not\_In\_CS} \\ = \{ Y \mid Y \neq \text{me and Reply\_Count}[Y] = 0 \} \end{aligned}$$

```
Defer_Count: array [1..N] of integer := 0;
/* Defer_Count [Y] is the number of REPLYs
to node Y that have been deferred until this node
exits the critical section */
```

When a node  $X$  exits its critical section, it must send all of the deferred REPLYs to node  $Y$ . Although at most one of these deferred REPLYs can pertain to any current attempt by node  $Y$  to enter the critical section, node  $Y$  must receive all of these REPLYs to ensure that the array

Reply\_Count is correct. As it would be wasteful to send each REPLY as a separate message, a REPLY message contains a field indicating the number of REPLY messages that are represented by this message. Thus one physical message is sent instead of several.

## 5. Detailed presentation of our extended algorithm

Each node knows the number of nodes in the network  $N$  and its own identity  $me$ , a number in the range 1 to  $N$ . The code executed for each event is given in the following subsections.

### 5.1. Node $X$ wishes to enter the critical section

```

Requesting_CS := true;
Our_Seq := Max_Seq + 1;
for Z := 1 to N do
  if Z ≠ me then
    begin
      send REQUEST (Our_Seq) to Z;
      Reply_Count [Z] := Reply_Count [Z] + 1
    end
end

```

### 5.2. Node $X$ receives REQUEST ( $S_Y$ ) from $Y$

```

Max_Seq := max (Max_Seq, S_Y);
if Executing_CS or (Requesting_CS and (Our_Seq, me) < (S_Y, Y))
then Defer_Count [Y] := Defer_Count [Y] + 1
else send REPLY (1) to Y

```

### 5.3. Node $X$ receives REPLY (Count) from $Y$ :

```

Reply_Count [Y] := Reply_Count [Y] - Count;
if Requesting_CS and Not_In_CS ( ) >= N - K
then
  begin
    Requesting_CS := false;
    Executing_CS := true;
    enter Critical Section
  end
end

```

where Not\_In\_CS ( ) is defined as:

```

Cnt := 0;
for Z := 1 to N do

```

```

  if Z < me and Reply_Count [Z] = 0
  then Cnt := Cnt + 1;
Return (Cnt)

```

### 5.4. Node $X$ exits critical section

```

Executing_CS := false;
for Z := 1 to N do
  if Defer_Count [Z] ≠ 0 then
    begin
      send REPLY (Defer_Count [Z]) to Z;
      Defer_Count [Z] := 0
    end
end

```

Each piece of code must be executed in local mutual exclusion provided by a semaphore or other mechanism. There are no delays in these pieces of code, and so there is no need for more complex synchronization.

Note that it is possible for a node  $X$  executing the critical section to receive a "lower" REQUEST from node  $Y$ . This is not a cause for concern, as there will always be sufficiently many other nodes not executing the critical section to allow node  $Y$  to enter the critical section.

## 6. Proof outlines

### 6.1. At most $K$ nodes executing the critical section

Let  $E = \{\text{identities of nodes executing the critical section}\}$ .

**Theorem.**  $|E| \leq K$ .

**Proof.** Suppose not. Assume that at time  $T$ ,  $|E| = M > K$ . Consider the (sequence number, node identity) pairs used by the nodes of  $E$  to gain entry to the critical section. These pairs form a total order. Hence the nodes in  $E$  can be labelled  $E_1, E_2, \dots, E_K, E_{K+1}, \dots, E_M$ , so that

$$(S_{E_1}, E_1) < (S_{E_2}, E_2) < \dots < (S_{E_K}, E_K) \\ < (S_{E_{K+1}}, E_{K+1}) < \dots < (S_{E_M}, E_M).$$

Consider the node  $E_{K+1}$ . To enter the critical section, this node must have received REPLYs

from  $N - K$  of the other  $N - 1$  nodes. Or rather, at most  $(N - 1) - (N - K) = K - 1$  nodes did not send a REPLY. Thus of the  $K$  nodes  $E_1, E_2, \dots, E_K$ , one of them  $E_{X(\leq K)}$  sent a REPLY to  $E_{K+1}$ .

Consider the situation when node  $E_X$  received the REQUEST  $(S_{E_{K+1}}, E_{K+1})$ . There are 3 cases.

*Case 1.*  $E_X$  was executing the critical section or attempting to enter the critical section with sequence number  $S_X$ . In this case,  $E_X$  would defer replying to  $E_{K+1}$ .

*Case 2.*  $E_X$  was neither executing nor wishing to execute the critical section. In which case  $\text{Max\_Seq}_{E_X}$  would become  $\geq S_{E_{K+1}}$ , and hence  $E_X$  could not be executing the critical section at time  $T$  with  $(S_{E_X}, E_X) < (S_{E_{K+1}}, E_{K+1})$ .

*Case 3.*  $E_X$  was executing or attempting to execute the critical section on a previous occasion with sequence number  $R$  such that

$$(R, E_X) \leq (S_{E_X}, E_X) < (S_{E_{K+1}}, E_{K+1}).$$

Hence  $\text{Max\_Seq}_{E_X}$  would become  $\geq S_{E_{K+1}}$  and hence  $E_X$  could not be executing the critical section at time  $T$  with  $(S_{E_X}, E_X) < (S_{E_{K+1}}, E_{K+1})$ .

Thus it is impossible for any node  $E_X$ ,  $X \leq K$  to REPLY to the REQUEST of node  $E_{K+1}$ .  $\square$

### 6.2. Deadlock does not occur

Deadlock occurs when fewer than  $K$  nodes are executing the critical section and there exist other nodes wishing to enter the critical section who cannot do so. We claim that deadlock never occurs with our algorithm.

Suppose deadlock could occur, and that at time  $T$ , the system is deadlocked. Using the (sequence number, node identity) pairs of the nodes executing the critical section, label these nodes  $E_1, E_2, \dots, E_{\alpha(\leq K)}$  as in section 6.1. In a similar manner, label the nodes attempting to enter the critical section  $W_1, W_2, \dots, W_w$  by ranking the (sequence number, node identity) pairs.

Consider the node  $W_1$ . This node has sent REQUEST messages to all other  $N - 1$  nodes, and requires  $N - K$  replies, i.e. no more than  $K - 1$  nodes can defer their reply. No code segment associated with any event of the algorithm can delay. Therefore the REQUEST message sent to

node  $X$  will not produce a REPLY only when the node  $X$  chooses to defer the REPLY. Therefore all the nodes not executing and not wishing to execute the critical section will REPLY. All other nodes wishing to enter the critical section must REPLY as  $(S_{W_1}, W_1) < (S_{W_i}, W_i)$  for all  $i$  in  $2..w$ . The only nodes therefore that may defer the REPLY to node  $W_1$  are the nodes executing the critical section, and there are less than  $K$  of them. Hence at least  $N - K$  of the other nodes must eventually REPLY to  $W_1$ . Therefore the system is not in deadlock: contradiction.

### 6.3. Starvation does not occur

If a node waits indefinitely to enter the critical section while other nodes are entering and exiting the critical section, then the node is said to be starved. Using similar reasoning to Ricart and Agrawala, we claim that starvation cannot occur.

Consider node  $X$  wishing to enter the critical section with sequence pair  $(S_X, X)$ . Node  $X$  will send REQUEST messages to all other nodes. When such a REQUEST message arrives at another node  $Y$ , it may be immediately replied to, or deferred if node  $Y$  is executing in the critical section, or attempting to do so with a "lower" sequence pair. Even if the REPLY is deferred, the value of  $\text{Max\_Seq}$  will become  $\geq S_X$ , and hence any subsequent attempt to enter the critical section by node  $Y$  will use a "higher" sequence pair.

Therefore, once  $X$ 's REQUEST message has arrived at node  $Y$ , node  $Y$  may enter the critical section at most once before  $X$  will be allowed to enter. Thus eventually  $X$ 's REQUEST will be sufficiently "small" to obtain entry to the critical section.

## 7. Performance

For a node to enter its critical section, it requires  $N - 1$  REQUEST messages to be sent, and  $N - K$  REPLY messages to be received. Thus  $2N - K - 1$  messages per critical section is a lower bound for this algorithm.

However each REQUEST message must eventually generate a REPLY message, and thus  $2 * (N$

$- 1$ ) messages per critical section represents an upper bound for this algorithm. This is the number of messages required by Ricart and Agrawala's algorithm, and is thus the expected upper bound for our algorithm as no new messages have been added to the original algorithm.

The difference between the number of messages required by the original algorithm and the extended algorithm is a result of sending a number of deferred REPLYs together as a single message. When a node is executing the critical section for so long that another node may enter the critical section more than once in that time, such multiple REPLY messages will be generated. Thus nodes which remain in the critical sections for (relatively) long periods will send fewer physical REPLY messages than those nodes which execute their critical sections more rapidly. These "faster" nodes will send at most one REPLY per physical message, and thus the lower bound of  $2N - K - 1$  is not actually achievable.

Simulation results indicate the savings resulting from sending several REPLY messages in one physical message are usually quite small, and that the average behaviour of this algorithm is  $2 * (N - 1)$  messages per critical section. Even the most contrived examples rarely reduce the number of messages per critical section to less than  $2N - 1 - (K/2)$ .

## 8. Conclusion

We have presented a starvation- and deadlock-free algorithm to allow up to  $K$  simultaneous executions of a critical section based on the technique of Ricart and Agrawala. Although seemingly a more complex problem, our algorithm requires no more messages than that of Ricart and Agrawala (i.e.  $2 * (N - 1)$  messages per critical section), and occasionally fewer.

## Acknowledgements

The author wishes to thank Andrew Lister, David Horton and the anonymous referees for reading and critically commenting on this work.

## References

- [1] M. Maekawa, A  $\sqrt{N}$  algorithm for mutual exclusion in decentralized systems, *ACM Trans. Comput. Syst.* 3 (2) (1985) 145-159.
- [2] K. Raymond, A tree-based algorithm for distributed mutual exclusion, *ACM Trans. Comput. Syst.*, to appear.
- [3] G. Ricart and A.K. Agrawala, An optimal algorithm for mutual exclusion in computer networks, *Comm. ACM* 24 (1) (1981) 9-17.
- [4] I. Suzuki and T. Kasami, A distributed mutual exclusion algorithm, *ACM Trans. Comput. Syst.* 3 (4) (1985) 344-349.