A Synchronization Algorithm for Distributed Systems

Tai-Kuo Woo
Department of Computer Science
Jacksonville University
Jacksonville, FL 32211

Kenneth Block
Department of Computer and Information Science
University of Florida
Gainesville, FL 32611

Abstract

Synchronization is an important aspect of computing. System performance can be greatly reduced if an inefficient synchronization algorithm is used. Here we propose an algorithm for achieving mutual exclusion, a major task of synchronization, for distributed systems using a tournament approach. In the algorithm, a request message is passed from a leaf node to the root node, and then back to the leaf node again, signaling that a process is permitted to enter the critical section. A Huffman coding technique is used to minimize the number of messages required and to reduce the bound of delay. High fault tolerance is achieved through backtracking the messages that have been acknowledged by the faulty node.

1 Introduction

A problem that often arises in distributed systems is synchronizing concurrent processes, particularly guaranteeing mutually exclusive access to shared resources. An efficient mutual exclusion algorithm can maximize the parallelism among concurrent processes. For distributed systems, the protocol of a mutual exclusion algorithm cannot be made to rely on access to shared memory, but must communicate through message passing. At present there are two models for achieving mutual exclusion in distributed systems. The first model takes a probabilistic approach [1, 3, 4, 5]. Each process generates a random number and then compares its number with others. The winner gets into the critical section. The second model uses a deterministic approach. Processes reach agreement by comparing local counters in the nodes, such as time stamps, sequence numbers, etc. [8, 10, 2]. Even though the results may depend on the arrival sequence of the messages, the decision of who gets into the critical section is based on the state of the system. Most mutual exclusion algorithms require that a process wishing to enter the critical section send messages to every other process. The number of messages required for each entry to the critical section is $O(n)$ where n is the number of nodes in a distributed system.

In this paper, we use a tournament approach for achieving mutual exclusion. The idea is to

let the processes compete with one another as in a tournament. If a process wins, it advances to the next level and competes with another process, the winner of the level below. This competition continues until a process reaches the top and then enters the critical section. While the idea of using tournament is not new [9, 6], what distinguishes this paper from others is the way messages are passed and the arrangement of the nodes, which is based on Huffman coding theory. Also, most algorithms, whether probabilistic or not, have weak reliability, since reaching agreement requires a process to send messages directly to every other process. The approach presented in this paper lessens this problem; a faulty node only affects its descendents, and other processes can still enter critical section. Once a faulty node is detected, it is replaced dynamically through a backtracking technique.

This paper is organized as follows. In Section 2, we delineate the synchronization algorithm. In Section 3, we describe how the Huffman coding technique can be used to reduce the message traffic. In Section 4, we discuss the strategy for increasing fault-tolerance. In Section 5, we look at network management. A conclusion is given in Section 6.

2 Description of the Algorithm

The protocol for a node wishing to enter the critical section is as follows. A node competes with its neighbor at a designated node by sending a request message containing information such as the ID of the request node and its immediate designated nodes. A designated node is the place where two nodes with processes interested in the critical section compete against each other. For instance, we can choose node 1 as the designated node for nodes 2 and 3. Whenever these two nodes have a request, the request messages are forwarded to node 1. An alternative is to let the message frame contain the node number of the request and the level number. The level number is initialized to 1. Each designated node calculates the node number of the designated node at the next level by using the formula provided in the procedure below. The designated node

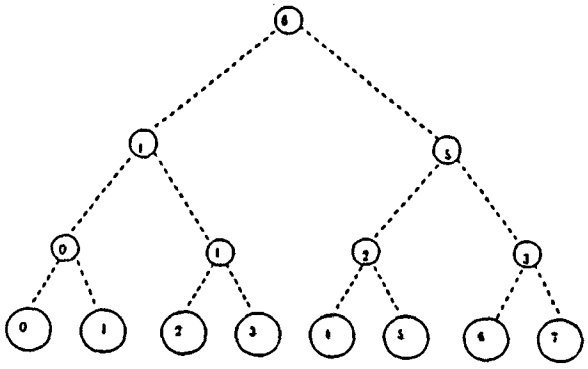

Figure 1: A Tournament Among Nodes

that determines who goes to the next level immediately returns an acknowledgment to both of its children and forwards the message to another designated node at a higher level. Figure 1 shows the tournament among nodes. The solid white circles represent nodes, and the grey circles denote designated nodes. Each designated node holds a token, all of which are initialized to be true. A request message from node 1 would include designated nodes 0, 4, and 6. In the figure, nodes 0 and 1 compete at designated node 0; nodes 2 and 3 compete at designated node 1; and so on. The designated node determines the winner based on which request message arrives first. The request message arriving first finds the token is true and is forwarded to the designated node at higher level. A request message that gets to the designated node late is withheld until the node whose message arrives first exits the critical section. Since a node is only assigned as a designated node once, it can receive at most two request messages. When a node exits the critical section, it issues messages to unblock the request messages at the designated nodes. For example, suppose both nodes 0 and 1 want to enter the critical section and the request message from node 1 arrives at node 0 first (i.e., the message arrives at node 0 before node 0 generates a process for entering the critical section.)

The request message from node 1 is forwarded to node 4, and the request message from node 0 is blocked. For ease of detection of faulty nodes, an ACK_withheld and an ACK_forward are returned to nodes 0 and 1, respectively. Node 4 forwards the request message from designated node 0 to designated node 6 if no request message is received from node 1. When the request message returns to node 1, it enters the critical section. When node 1 finishes the critical section, it sends messages_release to nodes 6, 4, and 0 to unblock the request messages.

The following procedures implement the protocol.

```
Procedure Enter_Critical_Section;
begin
      {Send a message to its immediate
       designated node.}
      Designated node ID:= Node ID/2.
      message(request_critical_section,
      node ID, level, designated node ID) ⟶
      Designated node ID.
      Go to sleep
      {It wakes up and enters the critical
       section when it receives a message of
      request_critical_section with its node ID
      and level equals log2n) }
end ;

Procedure Exit_Critical_Section;
begin
     Designated node (k):= n−2;
     {Starting with the designated node at the
      highest level }
     {Starting with the designated node at the
highest level }
     for k:= log2 n to 1 do;
       begin
         Message(release_critical_section, node
        ID) ⟶ Designated_node(k);
         Designated node (k − 1):=
          2 * Designate node(k) − n
         {Compute the designated node at one
         level below }
       end
end;
```

```
Procedure Receiving_Message;
begin
     case message of
       release_critical_section:
          if There is a request message being
            withheld, then
            if level = log2 n then
             Forward the withheld request
             message to node ID.
            else
             Designated node ID :=
             Designated node ID/2 + n/2;
             Forward the withheld request
             mesage to designated node ID
             at a higher level.
          else
            Set the token on;
       request_critical_section:
          level :=level + 1;
          if token is off then
              Withhold the message;
              Return ACK_withheld
          else
              if level = log2 n then
                 Designated node ID :=
                  Node ID;
                 {The request message has
                  reached the top level and
                    is going to be returned to
                    the node which issues
                    the request }
              else
                 Designated node ID :=
                 Designated ID /2 + n/2;
                 {Compute the designated
                 node at the next level }
              Set the token off and
              forward the request message to
              designated ID at a higher level.
              Return ACK_forward;
     end case
end;
```

2.1 Proof of Correctness

Lemma 2.1 *Mutual exclusion is guaranteed. Mutual exclusion ensures that only one process is in the critical section at a time.*

Proof of Lemma 2.1 The claim is true because each node has two immediate children and each child can only send one request message. At the root, only one request message can go through. When releasing the critical section, each withheld request message advances one level and the request message withheld at the root gets into the critical section. As a result, at any time, only one process stays in the critical section.

Lemma 2.2 *Deadlock is impossible. Deadlock is the situation where no process can ever enter the critical section.*

Proof of Lemma 2.2 The claim is true. If only one process is at the highest level, it proceeds to the critical section. If more than one process is at the highest level, one of the processes advances to the next level and enters the critical section eventually.

Lemma 2.3 *Starvation is not possible. Starvation is the situation where a process is prevented indefinitely from entering the critical section.*

Proof of Lemma 2.3 A process which is withheld at a level can never be passed by any of its descendents. When a process exits the critical section, the processes above the withheld process move to a higher level and allow the withheld process to advance one level. Eventually, the withheld process enters the critical section.

3 A Technique for Reducing Message Traffic

In distributed systems, nodes have different probabilities of entering the critical section; i.e., some of them enter the critical section more frequently than the others. The number of messages and the delay required for each entry to the critical section is reduced if we place the nodes with high probability of requesting the critical section near the root. For example, in a system of four nodes, 0..3, with probabilities 0.01, 0.1, 0.1, and 0.79, respectively, it takes two messages for each entry to the critical section if the nodes are placed at leaves equidistant from the root. However, if we rearrange the nodes, placing the high probability nodes at the leaves closer to the root, as shown in Figure 2, the expected number of messages required for each entry to the critical section is reduced to 1.22.

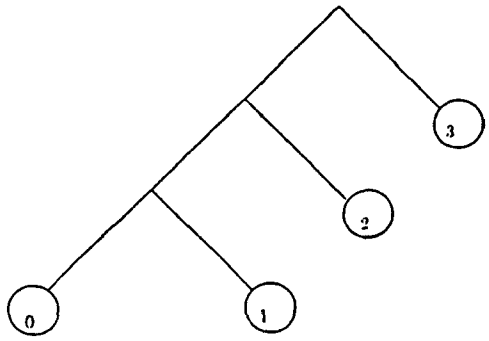

Figure 2: A New Arrangement of Nodes

3.1 Description of the Technique

The technique for finding the optimal arrangement of nodes can be found in coding theory. In a Huffman coding scheme [7], a set of items is to be coded in binary for transmission. Each item is associated with a probability of occurrence. To minimize the average-length binary code, a short binary code is used to represent an item with a high probability of occurrence. If we treat each binary digit as a message, the problem of minimizing the average length of binary code can be transformed into the problem of minimizing the average number of messages required for each entry to the critical section. The Huffman coding algorithm works as follows. First, it orders the items in descending order according to their probabilities of occurrence. Second, the code of the item with the lowest probability is concatenated with a "0" at the front, and the code of the item with the second lowest probability is concatenated with a "1." Third, the probabilities of these two items are summed up to form a new item, and all the items are reordered. Again, the last two items are concatenated with a "0" and a "1", respectively. This process is repeated until

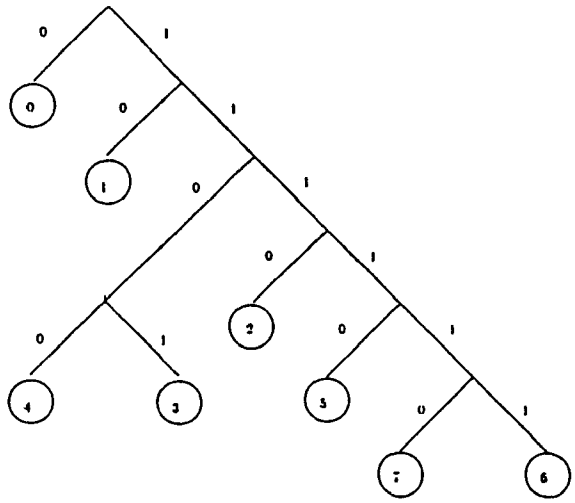

Figure 3: A Configuration of Nodes Using Huffman Coding Theory

only two items are left, and each is assigned a binary digit. For example, items 0 through 7 have probabilities 1/2, 1/4, 1/16, 1/16, 1/16, 1/32, 1/64, and 1/64. Applying the Huffman coding algorithm would generate the binary codes shown in Table 1.

Item	Probability	Binary Code
0	1/2	0
1	1/4	10
2	1/16	1110
3	1/16	1101
4	1/16	1100
5	1/32	11110
6	1/64	111111
7	1/64	111110

Table 1: Binary Codes of Items Using Huffman Coding Scheme

To transform the binary codes into the tournament tree, we start with the root, assigning 1 to the right link of the root and 0 to the left link. If any of the paths from the root down matches the binary code of a node, it terminates. The paths which do not match any binary code branch to the next level. Eventually, all the paths find a match in the tree. Figure 3 shows the new configuration of nodes using Huffman coding theory.

In the Huffman coding scheme, the probabilities of items are known. The probability that a node will issue a request to enter the critical section can be estimated statistically from its past history.

There is no doubt that the percentage of reduction of the number of messages required depends on the variance of the probabilities that nodes may issue a request for entry to the critical section; the higher the variance the greater the percentage of reduction of messages.

3.2 Performance Evaluation

Here, we perform a simulation to determine the effectiveness of using Huffman coding to rearrange the nodes. In the simulation, we calculate the percentage of reduction of messages (the reduction coefficient, RC) required for each entry to the critical section by using the Huffman coding technique for n = 4, 8, and 16, where n is the number of nodes in the system. The reduction coefficient is calculated by the formula below:

$$RC = 1 - \frac{NMH}{log_2 n}$$

where NMH is the expected number of messages required using Huffman coding theory and $log_2 n$ is the number of messages required without using Huffman coding. The NMH is obtained by taking the average of five simulation runs. On each simulation run, a set of probabilities is generated with the specified standard deviation. Then, by associating each node with a probability and appling the Huffman coding algorithm, an optimal arrangement of nodes is determined. The NMH is calculated by the formula below.

$$NMH = \sum_{i=1}^{n} p_i l_i$$

where p_i is the probability that node i may request to enter the critical section and l_i is the number of links between the node and the root in the tree. As shown in Tables 2, 3, and 4, the reduction coefficient increases as the standard deviation of the probabilities (σ) that the nodes may issue a request to enter the critical section increases.

σ	NMH	RC
0.16	1.76	0.12
0.20	1.64	0.18
0.24	1.50	0.25
0.28	1.38	0.31
0.32	1.36	0.32
0.36	1.20	0.40

Table 2: Reduction Coefficient of Using Huffman Coding (n=4)

σ	NMH	RC
0.09	2.61	0.13
0.11	2.40	0.20
0.13	2.28	0.24
0.15	2.16	0.28
0.17	2.13	0.29
0.19	2.04	0.32

Table 3: Reduction Coefficient of Using Huffman Coding (n=8)

σ	NMH	RC
0.05	3.48	0.13
0.06	3.40	0.15
0.07	3.20	0.20
0.08	2.88	0.28
0.09	2.8	0.30
0.10	2.36	0.41

Table 4: Reduction Coefficient of Using Huffman Coding (n=16)

4 Network Reliability

There are two major issues in network reliability: fault detection and fault tolerance. The distributed algorithm proposed in this paper provides high reliability.

First, fault detection is very easy to implement. A node is monitored by its immediate children. When a node receives a request message from it children, an ACK is returned. If a child node does not receive an ACK before a timeout period expires, the child node determines that its parent node is faulty. To achieve high fault tolerance, we use a backtracking technique. When a node forwards a request message to its parent and an ACK is not returned before a timeout period expires, it forwards the request message to the last node whose request message has been acknowledged. For instance, node 5 forwards a request message from node 1 to node 7 and an ACK is returned to both node 5 and node 6. Note that an ACK should include the ID number of the node that originates the request to enter the critical section. Later on if either node 5 or node 6 detects that node 7 is faulty, the new request message, which could come from any of the descendents of node 5 or node 6, is forwarded to node 1, because node 1 was the last node to successfully function on the chain leading to node 7.

To achieve multiple fault tolerance, a queue of ACKs can be established at each node and the replacing node for the faulty node can be designated according to the ordering of the nodes in the queue.

5 Network Management

Since in the distributed algorithm each node receives request messages from its immediate descendents, the addition of a node is very straightforward. First, the position of the new node on the tree is calculated using the Huffman coding scheme. Second, the old parent notifies its immediate child that a new parent has been established. For example, if a new node 4 is to be inserted into the link between nodes 1 and 2, node 2 notifies node 1 that all request messages should be sent to node 4.

Similarly, a node being deleted from the network notifies its immediate children that all request messages should be sent to its neighbor.

6 Conclusion

In this paper, we have described a mutual exclusion algorithm for distributed systems using a tournament approach. A process wishing to enter the critical section must issue a request message, which is then passed through the nodes on the path from the leaves to the root. When the request message is returned to the requesting node, the process enters the critical section. When exiting the critical section, the process sends a message to each node on the path the message has passed through. As a result, the number of messages required for each entry to the critical section is $2log_2n$, where n is the number of nodes. To further reduce the number of messages, the Huffman coding technique is used to determine the position of each node in the tournament based on the probability that a node may issue a request for entry to the critical section. Performance evaluation shows that this algorithm produces a significant reduction in messages, which also implies a reduction in the average bound of delay. The reliability of the protocol is enhanced through a backtracking technique. When a node receives a request message from one of its immediate children, it returns an ACK to both of its children immediately. When the children detect a faulty parent (i.e., no ACK is received), the request message is forwarded to the node whose request message passes through the faulty node. Multiple fault tolerance can be achieved by using a queue to store the nodes that have passed through a faulty node. Also, since a node needs to know only its parent and immediate children, adding and deleting nodes can be done on the fly. Moreover, the approach of using Huffman coding technique can be applied to other application fields, such as file structures and database retrieval, without adding overhead. For instance, in a hierarchical file structure, frequently retrieved files are placed near the root to reduce the amount of searching time. In dynamic hashing and indexing, the items recalled most often should have the least rehashing and reindexing.

References

[1] Burns, J. E. Symmetry in Systems of Asynchronous Processes. Proceedings of 22nd Annual ACM Symposium on Foundations of Computer Science. 1981, pp. 169-174.

[2] Carvalho, O. and Roucairol, G. On Mutual Exclusion in Computer Networks. Communications of the ACM, 26(2), Feb. 1983, pp. 146-147.

[3] Chang, C. K. Bidding Against Competitor. IEEE Transactions on Software Engineering, 16(1), Jan. 1990, pp. 100-104.

[4] Cohen, S, Lehmann, D, and Pnueli. Symmetric and Economical Solution to the Mutual Exclusion Problem in Distributed Systems. Theoretical Computer Science. Vol. 34, 1984, pp. 215-226.

[5] Francez, N. and Rodeh, M. A Distributed Abstract Data Type Implemented by a Probabilistic Communication Scheme. Proceedings of the 21^{th} Annual ACM Symposium on Foundations of Computer Science, 1980, pp. 373-379.

[6] Graunke, G. and Thakkar, S. Synchronization Algorithms for Shared-Memory Multiprocessors. IEEE Computer, June 1990, pp. 60-69.

[7] Huffman, D. A Method for the Construction of Minimum Redundancy Code. Proceedings of IRE, 40, 1952.

[8] Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 21(7), July 1978, pp. 558-565.

[9] Peterson, G. and M. Fischer. Economic Solutions for the Critical Section Problem in a Distributed System. Proceedings of the Ninth ACM Symposium on Theory of Computing, 1977, pp. 91-97.

[10] Ricart, G. and Agrawala, A. K. An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, 24(1), Jan. 1981, pp. 9-17.