Simulation of a Distributed Mutual Exclusion Algorithm Using Multicast Communication

Jonathan Pearlin and Robert Signorile*
Boston College
Fulton Hall 460
Computer Science Department
Chestnut Hill, MA 02056
*Email: signoril@bc.edu
*Phone: 617-552-3936

KEYWORDS

Distributed Coordination, Distributed Mutual Exclusion

ABSTRACT

The development of a distributed mutual exclusion algorithm operating via multicast communication is considered necessary to ensure the proper performance of distributed systems. We developed an algorithm that takes advantage of the characteristics of a multicast network and various synchronization mechanisms, such as a timestamp and the election of a central arbitrator from the multicast group. These synchronization safeguards not only enforce mutual exclusion, but also aid in the enforcement of reliable communication between nodes in a distributed system. The implementation of a dynamic rotating server selected from among the members of the multicast group, in combination with a timestamp applied to all messages sent between nodes in the system, forces the ordering of messages sent within the system. This adds a level of predictability and stability to a distributed system and its communication mechanism.

INTRODUCTION

Distributed systems utilize a network of computers in order to share the workload of an application evenly amongst the members of the network. Such a system must not only coordinate between processes in the system, but must also provide the same functionality and assurances that one finds in a non-distributed program. These assurances include the mutual exclusion of access to shared resources among processes vying for entry into a critical section. Mutual exclusion consists of the prevention of deadlock between processes and the prevention of the starvation of a process attempting to acquire entry to the critical section. Without these assurances, one cannot rely on a system where processes communicate via messages in order to gain entrance to a critical section to be safe. Mutual exclusion in a distributed system is therefore required to not only ensure that the work is evenly distributed between processes, but that resources available to the system are also shared evenly and fairly.

PREVIOUS RESEARCH

Providing mutual exclusion to a distributed system requires the consideration of several important factors, including the way in which processes communicate with each other, the type of network on which the system is based and the coordination of processes in the system. Distributed mutual exclusion algorithms achieve this coordination by using techniques such as message passing or token passing to communicate between nodes (Ricart et al. 1981, Suzuki et al. 1985). Furthermore, these algorithms are often based on point-to-point communication in which each node must communicate directly and separately with any other node in the network it wishes to address. Because of this behavior, the focus of such algorithms has been on the reduction of the number of messages that are required to maintain mutual exclusion while preserving synchronization between the processes in the system.

The problem of synchronization in a distributed system is often addressed by utilizing a logical clock shared between all members of the system. The use of a logical clock in the form of a sequence number or timestamp, as proposed by Lamport, imposes ordering on the events that occur in the system (Lamport 1978).

While coordination of the processes in a distributed system is an important consideration in the design of a distributed mutual exclusion algorithm, the method of communication between processes is crucial to the effectiveness of the algorithm. Much of the work in the field of developing distributed mutual exclusion algorithms have been based around the problem of reducing the number of messages necessary to ensure a safe entry into the critical section. Ricart and Agrawala proposed a “message passing” algorithm that requires 2

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

(N – 1) messages in order to achieve distributed mutual exclusion in a point-to-point communication-based network (Ricart et al. 1981). In their algorithm, "sequence numbers" are used to create a total ordering of events in the system. While assuring mutual exclusion, the algorithm has a large overhead in terms of the messages needed to achieve mutual exclusion.

Our algorithm promises to obtain mutual exclusion in a constant number (one or two) of messages per request for entrance into the critical section: one message to contact the rotating "server" with a request and one message back to entire system informing the group of the server's decision. It can be shown that this algorithm is distributed and that it provides mutual exclusion to the system (the term "distributed" is used to describe the equal number of turns that each node takes as the central arbitrator in the system).

THE PROPOSED ALGORITHM

The proposed algorithm is based on a combination of message passing and election based distributed mutual exclusion algorithms. At all times, the algorithm has a node that has been "elected" to serve as the central arbitrator for resource requests for a fixed term. When a node receives an incoming message, its first course of action (whether it is currently the server or not) is to determine if it is its turn to be the server by examining the timestamp value contained in the message. Server status rotates from one node to the next, giving all nodes in the system a turn to act as the server and handle an equally distributed amount of work. Any node wishing to request a resource or enter the group does so by issuing its request to the entire multicast group. The server (who is just another node in the group) listens for such requests and processes and responds to them appropriately. Once the server node has made a decision regarding the resource request, it broadcasts its response to the entire group. The requesting node, upon receipt of this response, enters the critical section if it has been determined by the server that it is safe to do so. All nodes (other than the node that has been granted access to the resource) simply mark the resource as in use and wait for the node with access to the critical section to broadcast a release command before attempting to request the resource again.

The distributed mutual exclusion algorithms described earlier in this paper all use some form of synchronization to ensure that events are processed in the correct order of occurrence. However, these algorithms deal with fixed networks of computers using point-to-point communication. The use of multicast communication adds an extra level of complexity to the notion of synchronizing events in a distributed system. Here, nodes do not communicate directly with one another and cannot be sure that their message has reached every other member in the group (this is the nature of multicast communication and must be assumed for the purposes of creating synchronization). This means that all nodes in the group cannot be allowed to control a logical clock apparatus. Instead, there must be a central source that regulates the clock in order to force synchronization upon all of the nodes in the system. Naturally, the server node was chosen to be in charge of synchronizing the logical clock created for the system (specifically, the algorithm uses a simple timestamp that acts like a real clock – when it equals a prescribed limit, the clock is reset to zero). By designating the server as the only node with authority to change the value of the clock, the algorithm ensures that all clocks in the system will be synchronized and that the workload of the system stays distributed equally.

The management of the synchronization mechanism is crucial to the performance of the proposed algorithm. The current server node increments the timestamp when it receives a message that pertains to a resource or membership request. The following diagram displays this process:

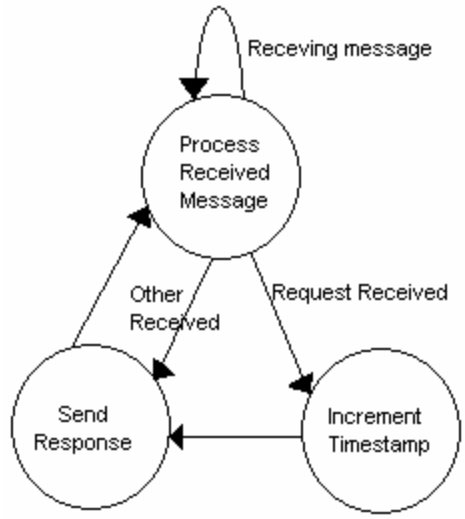


Figure E: State diagram showing when the server node increments the timestamp value.

As we can see from the diagram, the timestamp is increased after this message is received and right before a response is sent out to ensure that subsequent incoming messages are not discarded. The server has the sole responsibility and authority to change the value of the timestamp, which derives from its role as the central arbitrator in the system. This is the reason that synchronization can be achieved even with the use of multicast communication. By subjugating all other

nodes under the authority of the server node, the system is assured that the synchronization mechanism will remain stable.

While the inclusion of a timestamp is necessary to ensure that ordering of events in the system is possible, an issue does arise regarding the rotating server status. The server designation (as discussed earlier) is based on a dynamic calculation, which takes into consideration the number of members currently in the group, the current value of the timestamp, a node's relative position in the group based on its logical identification number and a constant which represents the number of turns each node will serve as the server. Therefore, because a node's status as server is not fixed and changes dynamically based on its relative position in the group (numerical ordering based on identification number), the server node must be consciously aware of the current timestamp and when its turn comes to an end. A potential problem can arise when the server is on its last turn and receives a request which increments the timestamp. In order to deal with this potential synchronization problem, a "SYNC" message was introduced to the algorithm to force synchronization on a server's last turn. The following diagram represents this addition to Figure E:

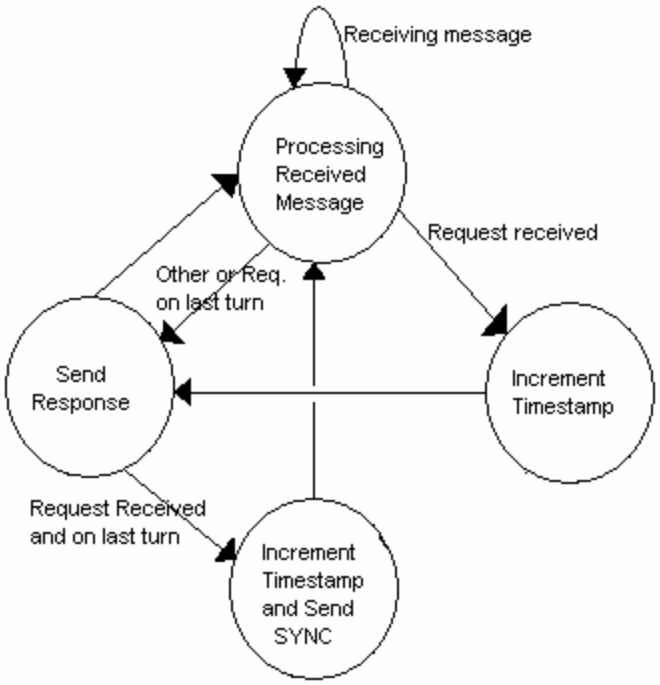


Figure F: State diagram showing the addition of the SYNC message to server module

This gives the server the power to respond to the current resource request and then increment the timestamp value without having to worry about creating the situation described in detail above. If the server is currently on its last turn, it sends its response, then updates the timestamp value (effectively ending its turn as server), and sends out the "SYNC" message with the new timestamp value. All nodes accept the timestamp value contained in a "SYNC" message, regardless of its value. This is because the value is guaranteed to be correct, as the only node that can issue such a message is the acting server. This necessary safeguard is the result of the server apparatus and its implementation. By developing a system based on the dynamic designation of a node to act as the server (a modification on election-based mutual exclusion algorithms), the addition of a special type of message to ensure synchronization and to prevent a node from taking on the role of the server prematurely is essential. This added safety feature protects the synchronization of the system's logical clock and as a result, protects the distributed nature of the algorithm.
In order for a distributed system to achieve mutual exclusion, no two nodes should have access to the same shared resource at the same time. This means that a node must exit the critical section before any other node can gain access to the same critical section.

Assertion: The proposed algorithm achieves mutual exclusion

Proof: In order to prove that the proposed algorithm achieves mutual exclusion, we must assume the contrary condition: two nodes can both have access to the same shared resource at the same time. In order for two nodes to have access to the same critical section at the same time, one of two cases must be possible:

1. Node A must have already been in the critical section at the time in which Node B entered the same critical section.
2. Node A and B simultaneously entered the critical section.

In the first case, node B is granted access to the critical section that is already occupied by node A. This means that the server node has made two grant responses for the same critical section without receiving a release from node A. This is impossible with the proposed algorithm. Upon receiving a request for the critical section from node A, the server node would see that the critical section is still available. It would lock the critical section (mark it as in use by node A) and issue a broadcast message to the group stating that node A now has permission to use the critical section. While this is occurring, assume that node B also issues a request for the same resource. While waiting for its response, node B receives the message from the server indicating that node A has been given the critical section. Node B would therefore mark the critical section as being in use by node A. This also means that node B would not be

able to enter the critical section until node A broadcasts its release message to the entire group (this is because the server node cannot issue permission to the same resource until it is released). Therefore, the first case cannot occur and mutual exclusion is preserved.

In the second presented case, the server node receives two different requests for the same resource simultaneously. The server node cannot issue two different responses for the same resource. Instead, the server will issue one response indicating which node has been granted access to the critical section. Because of the "passive" nature of the non-server nodes, the node that is not granted access will accept the decision and mark the resource as in use. When the node that is granted access is done with the resource and broadcasts its release message to the group, the node that was denied in its request will again be able to issue a request for the resource. The use of the central arbitrator to determine access to shared resources ensures that no two nodes can enter the critical section by way of simultaneous resource requests. Thus, mutual exclusion is preserved in the proposed algorithm.

Deadlock occurs in a distributed system when no node can enter the critical section despite requests being issued for entrance. This is usually caused by outstanding requests for the critical section or a failure in the permission granting authority.

Assertion: Deadlock cannot occur in the proposed algorithm.

Proof: In order to prove that the proposed algorithm avoids deadlock, we must assume the contrary condition: deadlock can occur among nodes wishing to access a critical section. This would mean that all of the nodes that have made a request for the critical section are waiting indefinitely for a response. In other words, the central arbitrator in the system has failed to respond to a request for a resource. However, this cannot be the case. A node cannot issue a request for a resource that is already in use, as there is no mechanism to block on a resource. Instead, a node checks its list of system resources to see if a particular resource is available. If it is available, then the node is free to issue a request for the resource. If it is not available, the node periodically checks to see if the resource's user has freed the resource before requesting that resource again. Either way, a node receives a broadcasted response from the server to the entire group informing all nodes as to which node has secured the right to use the resource. Therefore, the node cannot be in a state of constantly waiting for a response on a resource (this could occur if the server node only communicated directly with the node that is being granted access to the critical section and left other requesting nodes in the dark about its decision). This also means that a node receives information about the availability of a resource from the server when any node in the system requests a resource. Because of this, a node will not issue a request for a resource it already views as being in use.

On the surface, this "passive" nature, combined with the periodic checking of a resource's availability, could appear to cause a race condition, in which nodes are constantly contacting the server asynchronously to find out about the status of a resource. This is not what is meant by "periodically" checking to see if the resource is free. Instead, the node checks its own list of the system resources to see if a resource is available. As soon as the node views the resource as free, it may then reapply for the resource. Because all nodes receive a broadcasted response from the node that possesses the resource when the resource is released, the process is synchronized in terms of when all of the nodes view the resource as available for use. This synchronization therefore avoids a race condition. Thus, deadlock is avoided by using the broadcast nature of multicast communication in conjunction with the "passive" state of a node requesting a resource.

It is also possible for deadlock to occur if no node has taken on the role of the system's central arbitrator. This situation could arise if the node that is to become the server never receives the "SYNC" message sent out by the current server during its last turn. Because the current server sends out the "SYNC" message and then relinquishes its turn as server, neither it nor the next server believe that they are the active server if the message is lost (which is possible with multicast communication). The developed algorithm has been constructed to avoid permanent deadlock caused by such a scenario. Every node checks the timestamp of each incoming message and determines if it is its turn to be the server before processing the message. This means that even though the correct timestamp value never reached the node that is to be the next server, the correct value is present in all of the other nodes that received the message. Since the timestamp value is included in every packet sent out, those who send out packets with an invalid timestamp (this would be any node that did not receive the "SYNC" message) will have their messages discarded by those nodes that received the "SYNC" message and have the correct timestamp value. This preserves the correct state of the system. Furthermore, the next time any of the nodes (including the former server) that have the correct timestamp value send out a packet, all of the nodes with the wrong timestamp value (including the node that is suppose to be the server) will finally receive the correct timestamp value. This, in effect, would cause the node that is supposed to be the

server to notice that it is its turn to be the server. During this period without a server, the system would remain stable because only a server can make system-changing decisions (such as granting access to a critical section or adding members to the group) and increment the timestamp. This ensures that the system stays in the same state while it is attempting to rectify the lost communication. Such synchronization gives the system some margin of error in terms of the rotating server apparatus. While it may be true that a message or two directed to the server may be lost if this scenario occurs, it does not result in permanent deadlock of the system. This is because of the "passive" nature of requesting nodes, which do not block while waiting for a response. Multicast communication is unreliable and therefore extra safeguards must be put into place in the system in order to allow it to recover from lost packets.

Finally, deadlock could result in the system if a node that has been granted the critical section stops responding to the system. This would mean that a resource could be indefinitely held by one of the nodes and therefore would prevent other nodes from every acquiring that resource (this could also cause starvation, as a node would not be able to get the resource that it needs). However, the node acting as the server in the developed algorithm has the means to deal with this situation. In order to determine if a node is still responding to the group, the server uses a "ping" apparatus. If, at the end of its turn, the server determines that a node that is currently holding a resource is not responding to the group, the server can reclaim the resource for the group by broadcasting the resource's availability to the entire group. This frees the resource and allows any node that wishes to use the resource next to issue a request for that resource. Therefore, the "ping" apparatus serves the dual purpose of preventing the server status from falling to a non-responding node, as well as reclaiming resources from failing nodes. Thus, deadlock (and starvation, as discussed above) is avoided.

Starvation occurs in a distributed system when a node is continuously prevented from accessing the critical section while other nodes are allowed to access the critical section. This can occur in systems where unfair weighting is used to process requests for the critical section.

Assertion: Starvation cannot occur in the proposed algorithm.

Proof: In order to prove that the proposed algorithm avoids starvation, we must assume the contrary condition: starvation can occur, preventing a node wishing to access a critical section from ever being granted access. This would mean that the node continuously has its request denied by the central arbitrator, possibly as a side effect of network traffic (the node's messages are constantly arriving too late to be granted access). However, as mentioned earlier in the discussion of the algorithm, the server node has priority to enter the critical section because it is the node that grants access to the critical section. Therefore, a node may not have a chance to enter the critical section while it is not the server, but upon becoming the server, its requests supersede any requests received at the same time (the server node acts just like a regular node when requesting resources – it does not block on an outstanding resource request). This means that any starving node will not starve forever – it will eventually become the access granting authority (due to the rotating server apparatus) with the ability to grant its own access to the critical section, if it is available. Furthermore, as mentioned previously, the server node has the ability to determine if a node that currently holds a resource has stopped responding to messages from the group. If it finds this to be the case, it has the authorization to reclaim the resource for the group. Thus, starvation is prevented in the proposed algorithm.

SIMULATIONS

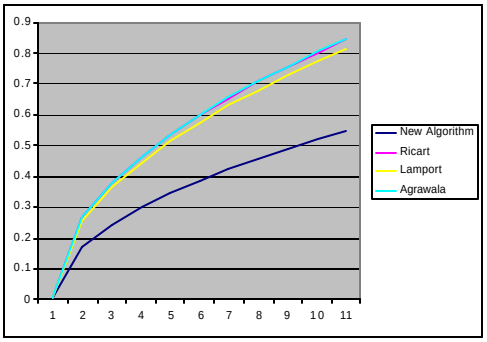
We simulated this algorithm as part of a distributed system represented N processes running on M machines. The number of processes and machines were dynamic in that they could be altered during the simulation (i.e. destroyed or created). The resources were fixed in size; the critical section was entered/exited many times per process. We also implemented Lamperts and Ricart and Agrawala algorithms as comparisons.

Algorithms that provide mutual exclusion to a distributed system are evaluated primarily on the bandwidth utilized to communicate with the other computers in the system, the effect the algorithm has on the throughput of the system, and the effect of synchronizing the computers in the system. The current body of work in this field is primarily focused on enforcing mutual exclusion in a distributed system based on using a point-to-point protocol for means of inter-node communication. Algorithms that meet this condition include message-passing algorithms, election algorithms and token-based algorithms.

The goal for all of these algorithms is ultimately the same: the reduction of the number of messages that is required to be exchanged between processes in order to enforce mutual exclusion. By reducing the number of messages needed to ensure mutual exclusion, these algorithms attempt to improve the optimality of the overall system in terms of bandwidth and throughput. (Lamport 1978, Suzuki et al. 1982, Garcia-Molina 1982)

The proposed algorithm achieves mutual exclusion in a constant number (one or two) of message exchanges. This is the result of the combination of four characteristics of the developed distributed system: the rotating server apparatus, the logical identification numbering of nodes, the "passive" disposition of non-server nodes and the broadcast nature of multicast communication. The deployment of a node to act as the central arbitrator in the system means that all resource requests must be directed to the server node. Therefore, any node wishing to request a resource must only issue one message, asking the server node (whomever it is – the requesting node does not need to have direct knowledge of the server to have its request heard) for entrance into the critical section. For the server's part, it needs to only send one response to the entire group (as a result of the "passive" non-server nodes and the broadcast protocol), instead of contacting each node individually to notify them of its decision. Each node accepts the incoming response regardless of whether or not it has requested a resource. This whole transaction requires an exchange of two messages to obtain mutual exclusion. If the server wishes to enter the critical section and it is open, then the number of messages required to secure the critical section is reduced to one (the broadcasted message to the entire group informing the group that the resource has been secured). This constant number of messages exchanged is the minimum number of messages required to ensure mutual exclusion in regards to the proposed algorithm. Here, we can fully see the benefit of using multicast communication in order to reduce the number of messages required to provide mutual exclusion to a distributed system. The proposed algorithm takes full advantage of the nature of multicast communication in order to provide mutual exclusion through the exchange of a constant number of messages.

In our simulations, we compared the time to converge (i.e. the time to notify all processes) that resources were needed and released. The below table is normalized to the new algorithm



CONCLUSIONS

This paper presents an algorithm that provides mutual exclusion using multicast communication to a distributed system. The proposed algorithm combines the effectiveness of message passing and election based distributed mutual exclusion algorithms in order to facilitate mutual exclusion. We have also implemented this algorithm as part of a distributed 3D game, with great success.

REFERENCES

Lamport, Leslie. "Time, Clocks, and the Ordering of Events in a Distributed System." Communications of the ACM. Volume 21, Number 7. July 1978. 558-565.

Ricart, Glenn and Agrawala, Ashok K. "An Optimal Algorithm for Mutual Exclusion in Computer Networks." Communications of the ACM. Volume 24, Number 1. January 1981. 9-17.

Suzuki, Ichiro and Kasami, Tadao. "An Optimality Theory for Mutual Exclusion Algorithms in Computer Networks." Proceedings of The 3rd International Conference on Distributed Computing Systems. October 18-22, 1982. 365-370.

Maekawa, Mamoru. "A vN Algorithm for Mutual Exclusion in Decentralized Systems." ACM Transactions on Computer Systems. Volume 3, Number 2. May 1985. 145-159.

Suzuki, Ichiro and Kasami, Tadao. "A Distributed Mutual Exclusion Algorithm." ACM Transactions on Computer Systems. Volume 3, Number 4. November 1985. 344-349.

Garcia-Molina, Hector. "Elections in Distributed Computer Systems." IEEE Transactions on Computers. Volume C-31, Number 1. 1982. 48-59.

BIOGRAPHY

Robert Signorile is an Associate Professor in the Computer Science Department of Boston College. His research interests include multimodal simulation, simulation in business, networks and distributed computing. He has published regularly in applied simulation, simulation methodology, distributed systems and networks.

Jonathan Pearlin is a recent graduate of the Boston College Computer Science department. His work revolves around operating systems support for distributed gaming.