Master Token Resource Management Algorithm for Distributed System

Priority Queue Strategy

ABHIJEET UDAY GOLE
VIDYALANKAR SCHOOL OF INFORMATION AND TECHNOLOGY, MUMBAI
abhijeet_gole@yahoo.com

ABSTRACT

This paper gives a comprehensive and efficient technique to implement mutual exclusion in stations competing for resources. The paper implements a single centralized manager for managing these resources over distributing system. The token manager is passed over these systems, using Priority Queuing Strategy it tries to give the critical resource to the neediest and prioritized station.

Keywords: Distributed Systems, Resource Management, Ring System, Token, Node, Mutual Exclusion, Critical Section, Master Token, Priority Queue.

1. INTRODUCTION

A distributed system (DS) is a collection of independent computers that appears to its users as a single coherent system [1]. In a distributed computing system any given "node" has only a partial or incomplete view of the total system and a system-wide common clock does not exist. Processes may share common hardware or software resources, cooperating in such a way that they can work in parallel and independently of each other. The access to a shared resource must be synchronized to ensure that only one process is making use of the resource at a given time. The problem of coordinating the execution of critical sections (CS) by each process is solved by providing mutually exclusive access in time to the CS. Each process must request permission to enter its CS and must release it after it has completed its execution. A mutual exclusion algorithm must satisfy the following requirements [2, 3]: At most one process can execute its CS at a given time.

i. If no process is in its CS, any process requesting to enter its CS must be allowed to do so at finite time.

ii. When competing processes concurrently request to enter their respective CSs, the selection cannot be postponed indefinitely. A requesting process cannot be prevented by another one to enter its CS within a finite delay.

To simplify, an algorithm must provide mutually exclusive access to the source, ensure deadlock freedom, ensure starvation freedom, and must provide some fairness in the order that requests are granted. Priority Queue System may be preemptive or not. A major problem with this is indefinite blocking or starvation or a low priority process. Aging is a technique to prevent indefinite blocking by gradually increasing the priority of the process as it remains in the system unattended [13]

2. PREVIOUS WORK

Two approaches have been used to implement a mutual exclusion mechanism in a distributed computing system. In a centralized approach, one of the nodes functions as a central coordinator. The central coordinator is fully responsible for having all the information of the system and for granting permission to make use of a shared resource. In a distributed approach, the decision-making is distributed across the entire system. Distributed mutual exclusion algorithms are designed based on two basic principles: the existence of token in the system or the collection of permission from nodes in the system.

3. PERMISSION-BASED ALGORITHM

All the permission-based algorithms are introduced so far basically work in the same way. The node that wants to enter the CS sends messages to other processors. Also, associated with each request there is a timestamp. When there are competitions for the CS, the one with the lowest timestamp should enter first. In Lamport’s event ordering mutual exclusion algorithm [4], a node that wants to enter the CS, broadcasts a message to all nodes in the system. The node that made the request enters the CS if it received responses from all other nodes. After the node finished with the CS, it again broadcasts a message to all other nodes. For a N node system a total of 3(N-1) messages are required to handle one request. Ricart-Agarwala's algorithm [5] extends
Lamport’s algorithm. The extension is that in Ricart-Agrawala’s algorithm, the response message is deferred. Similar to Lamport’s algorithm, Ricart-Agrawala's algorithm requires totally ordered events and all nodes being alive. About the number of messages, it does not need the release message, so it requires 2(N-1) messages for handling one request. Maekawa's algorithm [6] associates a set of nodes with each node, and this set has a nonempty intersection with every set associated with each other node. A node i must obtain permission from all other nodes in its home set Si before it can enter its CS. The number of messages required to handle a request is 3 times the size of the request set. For a system with N nodes, the size of each request set is roughly square root of N. So total 3√N messages are required to handle a request. The simplest of token-based algorithm is the Token Ring algorithm [7]: In this algorithm, the nodes in the system form a logical ring. A token is passed around the ring. A node can enter the CS if it holds the token. In average N/2 messages are required to handle one request in a N node system. In Suzuki-Kasami’s broadcast algorithm [8]: When a node wants to enter the CS; it broadcasts a message to all other nodes. Whoever holds the token sends the token directly to the node that wants the token. The algorithm requires N messages for handling each request. In raymond’s tree-based algorithm [9]: the token is always kept at the root node. When a node wants to enter the CS, it sends a request to its parent. The parent sends a request to its parent, recursively, this request will reach the root node. The root node, upon receiving the request sends the token down to the child that requested the token and is on top of the request queue. Once the node gets the token, it can enter the CS. In this algorithm, it requires an average of 2logN messages for handling each request.In Fatim’s Algorithm [10]: All nodes in the system are assigned unique identification numbers from 1 to N. There is only one requesting process executing at each node. Mutual exclusion is implemented at the node level. Processes are competing for a single resource. At any time, each process initiates at most one outstanding request for mutual exclusion. All the nodes in the system are fully connected. IN 1964, Knuth considered an abstract data type (ADT), called priority queue [14]. It consists of a set of items H. Each item is assigned a priority value chosen from a totally ordered domain. A lower value means a higher priority and vice-versa. [15] Priority queues are common in parallel and distributed applications. They represent the optimal data structure for many problems:

- Parallel branch and bound algorithms for decision-making in finance, or games are implemented using priority queues.
- In numerical iterative schemes, such as graph algorithms, priority queues are used to compute the “next” item to access; e.g., both Krusal’s algorithm for the minimal spanning tree problem (arising, for example, in network design) and Dijkstra's algorithm for the shortest path problem are based on DeleteMin operations.
- Heuristics for NP-complete problems, e.g., bin packing and traveling salesman, require backtracking with next visited elements chosen according to size or distance metrics.
- Pattern matching or data compression techniques for quality assurance and pattern recognition for identification purposes can use priority queues for processing signs ordered by their relative significance.
- In fuzzy search on large data sets, priority queues are used for providing statistical information, e.g., hit lists from internet search engines and reliability ratings for internet sites.

4. A Formal Background

A Token Ring System is a very powerful tool for Distributed Computing Resource management. At the physical level the system behaves like any other normal DS, but at the application layer level the algorithm implements the token ring resource management to apply mutual exclusion among all the nodes competing for resources. The algorithm (Master Token Ring Resource Management Algorithm for DSs) is meant for systems where mutual exclusion is a necessity and at any given time for a resource only one node can access it. The algorithm also takes into consideration the allocation of multiple resources to a particular node still maintaining mutual exclusion for concurrent access of the resources. It is very important to understand the application area and rigorous requirement analysis should take place before implementation of this algorithm in any DS environment. The different strategies designed are to increase the flexibility of application of this algorithm (Master Token Ring Resource Management Algorithm for DSs).
Algorithm for DSs) in the DS environment according to the requirements of the system.

Uniform allocation algorithms difficult to formulate in distributed ownership

The problem of making such efficient assignments is referred to as the resource allocation or scheduling problem, and it is commonly formulated in the context of a scheduling model that includes a system model, which is an abstraction of the underlying resources. The system model provides information to the allocator regarding the availability and properties of resources at any point in time. The allocator uses this information to allocate resources to tasks so as to optimize a stated performance metric. This paradigm is useful for high performance applications, which have tight constraints. Efficient scheduling of resources is critical in meeting these constraints. However, an increasing number of organizations now have environments that are not amenable to this resource management paradigm; their customers are interested in throughput and their computing resources are distributively owned [11]. In DSs it is very difficult to formulate the fair resource management algorithm for the nodes which are competing for it. Since the network management in such systems itself is difficult task in itself in the DS. In such cases the allocation algorithm has to take into consideration a few very important aspects like

a) Identify node, making the first request.
b) Identify node, currently using the resource.
c) Identify node, is using the resource being used or keeping it idle.
d) Identify the node is alive or dead.
e) Identify deadlock if any.
f) Identify priority over resources.
g) Identify intention of node requesting for resource (Update or Read-only)
h) Verify authentication of the Node.
i) Identify monopoly of Node over a one or more resources
j) Identify time amount of time a Node is holding a resource.

5. Master Token Ring Resource Management Algorithm

![Fig1: Conceptual diagram for Master Token Ring resource management algorithm](image)

The algorithm demands to maintain only one token in the Network with respect to the resource management system. Whichever site wants an access to the resource will grab the token and then the token will provide necessary access rights; if the resource is free. The token will create an entry of the process in its table notifying that process ‘x’ is using resource ‘R’. Thus, when another site will need the same resource it will be notified that Resource is busy. Thus the site will just put a claim stamp in the resource in the token. Similarly, whichever process needs the resource ‘R’ will create its entry in the claim queue, thus setting a priority for allotting the resource when it is free. In case of multiple resources to be used concurrently a site may request for more than one resource without interrupting other sites; by just creating a data log of number of resources the site is using in the token as in case of one resource being used. Whenever a site goes down (this will be taken care by the fault detection algorithm running in the background) another token will be generated. The fresh token will get all the data that was in the previous token by recording the status from the site that generates it. (Whenever a token is passed from one site to another the record details of the token is copied before passing it to another and an approval packet is sent to the preceding site notifying that the site has received the token). The site keeps on requesting for the resource without knowing that if it has got the token or not. These requests are buffered in the local memory and whenever the token is received the token checks upon these requests and performs necessary allocations and then moves on. Thus the system maintains the transparency with respect to the underlying implementation strategy used. The management of eligibility of a node to access the resource it has claimed for will depends on two things one the resource is in the available list of the token and the node is the first one request for the resource with respect to the claim list.

6. FLOW OF THE ALGORITHM

**Master Token Ring Resource Management Algorithm (Priority Queue Strategy)**

No. of Resources: \( R_n \) \( \{ n=0 \text{ to}(\max \text{ no. of Resources}) \} \)

\( K^{th} \) Resource: \( R_k \) \( \{ k=0 \text{ to} n \} \)

No. of Nodes: \( N_m \) \( \{ 1= m = \max \text{ no. of Nodes} \} \)

\( i^{th} \) Node: \( N_i \) \( \{ i = 1 \text{ to} m \} \)
Master Token Ring Resource Management Main Algorithm

Step 1 : Start
Step 2 : initialize all Ni; For all Rn:=0; Assign Priority to all N = PN
   // Make all the nodes up which are configured for the ring
   Let N1 listen to PP and SP;

Step 3 : N1 listens to SP< T;
   // Node(node 1) listens to the successor
   //node(last node) for token
   If RN = Null then
      Create_T(AL, RAL, CL);
      // Function to Create a new Token.
   End if
   If (N1 → Ri) exists in CL then
      If CL[Ri] // When n1 requests for Ri
         Allocate_T(N1→Ri);
         // Timer Function to Monitor the Allocation
         Start_Timer(N1,Ri);
         // Function to allocate Resource to the node
         Next_T(Ni+1);
         // Writes the Token to the Next Succeeding Node
      End if
   End if
   If (N1 → Ri) exists in CL then
      If CL[Ri] // When n1 requests for Ri
         Allocate_T(N1→Ri);
         // Timer Function to Monitor the Allocation
         Start_Timer(N1,Ri);
         // Function to allocate Resource to the node
         Next_T(Ni+1);
         // Writes the Token to the Next Succeeding Node
   End if

Step 4 : while Nn >= 2
   do
      If T! = Null then
         // Token Captured by the Node
      For i from 0 to NRA
         If Ri=AL[i] and Ri=CL[Ri] then
            If NRAN<= CLRA then
               Ni=ChkPriority(Ni,Ri,CL);
               Allocate_T(Ni→Ri);
               Ni<PN+1; // Increment Priority by 1
            // Timer Function to Monitor the Allocation
            Start_Timer(Ni,Ri);
            End if
            Else
               Claim_T(Ni→Ri);
               // Function to Put the Resource in Claim List
               Next_T(Ni+1);
            End if
            Step 5 : End
               // Check Priority of the Node
               Step 2.1 ChkPriority(N,R,Claim_List[i])
               Step 2.2 if (Claim_List[i].length>1 & Claim_List[i]==R)
                  while !=Claim_List.length // If more requests for
                  Do
                     Compete[j]=NC&PN;
                     // Competing Node and Priority Number
                     i=i+1;
                     End do
               Step 2.3 Sort(Compete); // sort in Ascending Order
               Step 2.4 return Compete[0]; // Node with the Lowest Number has Highest Priority
            End if
         End for
   End while
   Step 5 : End
       // Check Priority of the Node
       Step 2.1 ChkPriority(N,R,Claim_List[i])
       Step 2.2 if (Claim_List[i].length>1 & Claim_List[i]==R)
          while !=Claim_List.length // If more requests for
          Do
             Compete[j]=NC&PN;
             // Competing Node and Priority Number
             i=i+1;
             End do
       Step 2.3 Sort(Compete); // sort in Ascending Order
       Step 2.4 return Compete[0]; // Node with the Lowest Number has Highest Priority

7. FAULT DETECTION

The hard problems in distributed computing are not the problems of how to get things on and off the wire. The hard problems in distributed computing concern dealing with partial failure and the lack of a central resource manager. The hard problems have to do with differences in memory access paradigms between local and distributed entities. People attempting to write distributed applications quickly discover that they are spending all of their efforts in these areas and not on the communications protocol programming interface. [12]. The Master Token Ring Resource
Management algorithm does not take into consideration any type of network topology, but the fault detection is critical no matter what the topology. Thus, for this purpose whenever a token allocates a resource to any node. The node tries to pass the token in the to the next succeeding node if there is no response after predefined number of tries, then the node is considered to be dead and will be removed from the network by removing all the resource allocated to it and updating the token log entry, this will be the mainly done by the node which is preceding the dead node.

8. ADVANTAGES & DISADVANTAGES

Advantages
- Resource management in case of Multiple resources is much simpler and effective and more Robust.
- Fairness is provided to all sites.
- The Critical Resources are given for fixed amount of time to any node.
- Token is never engaged with one site for a fixed amount of time.
- Token behaves like a virtual coordinator between the sites competing for resources so no one node needs to be the coordinator.
- System Fault detection is easier.
- Many resources can be made available to a site without blocking the others from accessing the other resources on the network.
- The user never knows about the token which is being circulated in the background thus maintaining the transparency of the underlying implementation of the DS architecture and respecting the paradigm.
- By giving appropriate priority to the competing nodes, we establish concurrent priority system for various resources(Resource Based Priority). Thus providing fairness in the resource access.

Disadvantages
- The token passing always goes on, even if one is demanding for any resource or even if no has resource allocated to it.
- The critical resource will be forcefully removed from the Node if the time expires.
- In non-preemptive approach starvation of resource is possible.

Note: To avoid monopoly by one of the site the number of resources to be allocated to one of the given site will be restricted to a critical value. Also the time for which the resource allocated to the user is fixed by the Network Administrator.

9. EFFICIENCY COMPARED TO OTHER ALGORITHMS

The best feature of the algorithm is it never fails until there is only one node alive. The token message is passed infinite number of time consistently amongst the node as a constant speed depending on the amount of resources published for sharing. But for requesting for a resource and allocation of the same will require only one message. The system never crashes even if the nodes are going down in any possible manner or if the token gets lost during the transmission. The consistence of the resource is assured.

Our proposed method handles the loss of token in an efficient manner and as each node is responsible for its token so the responsibility of resending token completely lies on each node.

Disadvantages
- The token passing always goes on, even if one is demanding for any resource or even if no has resource allocated to it.
- The critical resource will be forcefully removed from the Node if the time expires.
- In non-preemptive approach starvation of resource is possible.

Note: To avoid monopoly by one of the site the number of resources to be allocated to one of the given site will be restricted to a critical value. Also the time for which the resource allocated to the user is fixed by the Network Administrator.

Number of Node = N
Average = Avg
Fig 2: Comparison Table

10. STATISTICS FOR TOKEN CIRCULATING TIME

Time for Token to arrive

This graph shows the average time taken by token to arrive at a given client for different number of machines in the ring and for different number of resources. Using this graph it is possible to calculate the average time that will pass before any end user will be capable of using the resource for given number of clients and number of resources.

The above graph also gives us a clear picture of the consistent behavior shown by the system, the nodes and resources are scaled up and so is the time required for the token increases. The increase is time is directly proportional to the increase resources and nodes. The algorithm was implemented for 10 nodes physically remote from each other and the data collected from them was scaled up to 100,200,300,400 and 500 nodes respectively as shown in the table and graph above. The resources allocated where spread across these nodes and the maximum number for resource allocated to one node was two.
The above Standard Deviation of the results show a linear rise in the time as the number of nodes are increase. Thus, the overall consistency of time required for the token is maintained. The token never remains with node for a fixed amount of time depending on the number of nodes in the network.

Even if the resources are assigned randomly to various nodes the system behaves in the same consistent manner; this makes the system behavior more predictive even in heavy congestion on the network. These results are yielded out of two rounds of random requests by various nodes over varying quantity of resources.

12. CONCLUSION

The main focus of this paper was the presentation of a single Token manager system for the resource management in DS using the Priority Queue strategy, the advantage of Token manager over servers in current distributed operating systems is the knowledge about the running application. Master Token has all the information about the shared system resources including information about other applications that it needs for the best possible management of resources. There’s a considerable reduction in the resource management overhead, due to a single token manager distributed among the multiple nodes competing for resources across the network system. The Algorithm helps to provide fairness to the most prioritized nodes, this help the overall system working a much efficient manner. To avoid monopoly of a particular node, over a certain resource, we increase the gradually Priority Number resultanty decreasing its priority. The whole resource management consists of a set of cooperating nodes and token resource manager to coordinate the resource management. The working of the system is in two dimensional approaches to give fairness amongst competing nodes and to help the system recover from dead lock if any.

REFERENCES


[12] JIM WALDO, GEOFF WYANT,ANN WOLLRATH,SAM KENDALL SMLI “A Note on Distributed Computing” TR-94-29 November 1994

