Computer Engineering and Applications Vol. 3, No. 3, September 2014

An Improved Multi-Cycle Deadlock Detection and Resolution Algorithm for Distributed Systems

Mohsen Askari[1], Rozita Jamili Oskouei[2]

Department of Computer[1], Department of Computer Science and Information Technologyy[2] Islamic Azad University, Ramsar Branch[1], Institute for Advanced Studies in Basic Sciences (IASBS) [2]

Ramsar[1], Zanjan[2],Iran[1, 2] mohsen.askari65@gmail.com [1] , r_jamili@iasbs.ac.ir [2]

ABSTRACT

Distributed systems exhibit a high degree of resource and data sharing creating a state inwhich deadlocks might make their appearance. Since deadlock detection and resolution is one of the important concerns in distributed systems which lead to minimizing available resources, therefore instigating the system through put decrease. Our proposed algorithm detects and resolves the multi-cycle deadlocks, whether the initiator is involved in the deadlock cycle directly or indirectly. Also the chance of phantom deadlock detection is minimized. This algorithm not only can manage the simultaneous execution of it but also detects the multi-cycle deadlocks in the distributed systems. Our algorithm introduces a modified probe and victim message structure. Moreover, no extra storage required to store probe message in each node which is known as memory overhead in the distributed systems.

Keywords: Distributed Systems, Deadlock Detection, Deadlock resolution, Distributed Algorithm, Wait-For-Graph, Probe Message.

1. INTRODUCTION

When a process in a distributed system needs a resource which is located in another site, it sends a message to the site through a network connection in order to access the required resource. If the required resource is available, it will be allocated to the process unless it is being used by another process where the requesting process will be blocked until the regarded resource is released and ready to obtain. Deadlock occurs when a set of processes wait for each other indefinitely of time to gain their intended resources. A deadlock state persists in the system until it finds its resolution by the deadlock handler. The presence of a deadlock in the system creates at least two major deficiencies. First, all the resources held by deadlock processes will not be available to other processes. Second, deadlock persistence time is added to the response time of each process involved in the deadlock cycle. This situation is not acceptable for this type of systems users. Dependency relationship between processes in distributed systems is shown by a directed graph called Wait-For-Graph (WFG) [1].

In this graph each node corresponds to a process and an edge directed from one node to another indicates the first process is waiting for the resource that the other process is on hold.

A cycle in the WFG represents a deadlock. Four categories have been proposed for classifying distributed deadlock detection algorithms [2] consist of: path pushing, diffusing computation, global state detection, and edge chasing algorithm.

Among various deadlock detection algorithms, edge- chasing has been considered as the most widely used techniques. In edge-chasing algorithm, a special message called probe is generated by an initiator process and propagated along the WFG edges. Deadlock is detected when this probe message gets back to the initiator. The key limitation of edge-chasing algorithm is that, whenever the initiator doesn't belong to deadlock cycle, this algorithm can't detect deadlock [1, 3, 4, and 6]. Several edge-chasing algorithms have been introduced, but each has its own disadvantages.

In this paper, we introduce a new algorithm based on edge chasing algorithm which improves the MC2DR algorithm proposed by Abdurrazzaq [5]. In our algorithm, the probe and victim message structures as well as the choice method of victim node are changed. Each node doesn't need extra storage to store the received probe message. Besides the pervious algorithms could not detect deadlocks in some of multiple deadlock cycles with presence of deadlock in its considered system. The resolution method in the MC2DR algorithm [5] is not correct. In some states, the present approaches cannot solve the detected deadlock. In proposed algorithm, we try to resolve all problems and present one spacious algorithm in respect of previous algorithms. Our algorithm resolves the deadlock as soon as it is detected resulting to deadlock average persistence time decrease in comparison with MC2DR algorithm [5]. In new algorithm, the numbers of probe and victim message fields are reduced; also, after resolving the detected deadlock, it needs no message to clear the ProbeStorage in each node. As was mentioned, before changing the probe and victim message fields, they were as overhead for deadlock detection algorithm whereas increasing the time of detection and removal of deadlocks in the system.

2. RELATED WORKS

The main idea of using probes was first introduced by Chandy-Misra and Hass [4]. In this algorithm, first the initiator sends a message called probe, this message moves alongside outgoing edges to other process or (processes) which are waiting for them. Each process by receiving this message sends it to other process for which it is waiting. Then, if the initiator of algorithm receives the probe message, created and propagated by initiator, it will evaluate the message as a proof of the existence of the cycle and will detect the deadlock. Sinha and Natarajan [6] proposed using the priorities to reduce the number of probe message in deadlock detection. Choudhary [1] found some weaknesses of this algorithm.

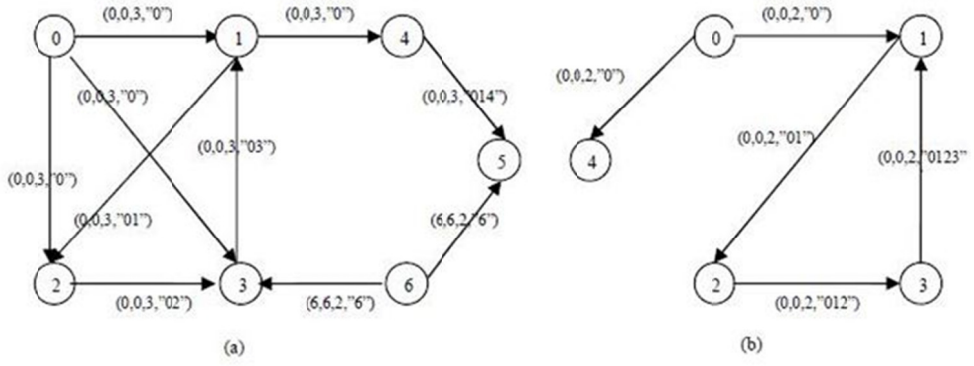

FIGURE 1. Examples of WFG with Edges Labelled by Probe Message

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

Kim Y.M [7] proposed the idea of barriers to allow the deadlock to be resolved without waiting for the token return. None of the algorithms are able to identify deadlocks in which the initiator is not directly involved in the cycle, though Lee proposed an algorithm in which deadlocks can be detected even when the initiator does not belong to any deadlock [11, 12]. The proposed algorithm with N. Farajzadeh [8] uses the priority for reducing the number of probe message in deadlock detection phase.

However exploiting such priorities in some cases would not contribute to deadlock detection while benefiting from regarded algorithm in distributed systems. In the MC2DR algorithm [5] by taking memory overhead cost into account in each node (ProbeStorage), this algorithm cannot detect all the deadlocks in the distributed systems.

In Figure.1(a), node ‘0’ has started execution of deadlock detection algorithm and has sent probe message(0,0,3,”0”) to its all successor nodes. Node ‘2’ has updated the Route-String field of probe message and has stored the modified probe (0,0,3,”02”) in its probeStorage, and then forwarded this message to node ‘3’. Node ‘1’ sends probe message (0,0,3,”01”) to the node ‘2’ at the same time. After receiving this message, node '2' first attempts to check if ProbeStorage is vacant. While ProbeStorage is full, the node tries to find out whether saved Route-String is a prefix of the received probe's Route-String . But the stored Route-String in the ProbeStorage equals to “02”, and “02” is not prefix of the “01”. In accordance with the MC2DR algorithm [5], the received message from node ‘1’ would be eliminated by node ‘2’ and the deadlock cycle {1, 2, 3, 1} cannot be detected.

In Figure.1 (b) node '0' is supposed to initiate algorithm and send probe message (0,0,2,"0") to its all successor nodes. As node 1’s ProbeStorage is empty, it has updates the probe message, stores the changed probe to (0, 0, 2,”01”) in ProbeStorage and forward this message to node ‘2’. Nodes 2 and 3 have update only the Route-String field of the probe message and send it to their successor. Node ‘1’ eventually gets its forwarded probe message back and detect the deadlock cycle {1, 2, 3, 1}. In accordance with MC2DR algorithm for resolution of the detected deadlock, node ‘0’ would be victimized because it has the highest DepCnt value among the route-string fields. But with victimizing node ‘0’, this deadlock cycle still exists in the system.

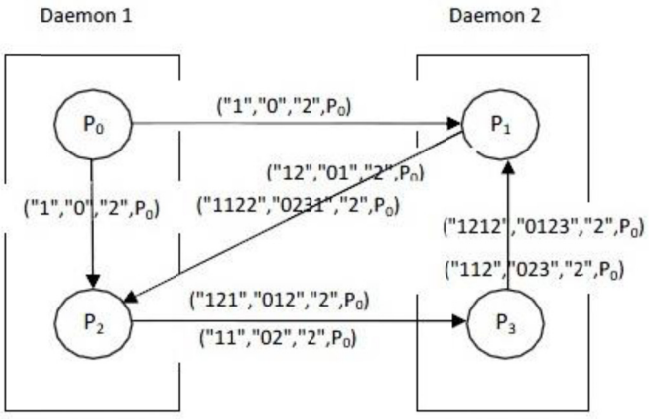

FIGURE 2. A Distribution System with 2 Machine and Processes

The proposed algorithm by Rahim Alipour in [10] considered a Daemon concept in the distributed system. Daemon is a process that runs in the background and is in sleep mode under normal condition. Each daemon has a database that saves the information of probes in the daemon. Their algorithm is so weak that it cannot detect many deadlocks besides in some states their method for resolution of deadlock doesn't work correctly and leads to deleting a process that does not affect deadlock resolution procedure.

In Figure 2, when the probe ("1122","0231","2", P0) gets back to P2, since the ID of this process is available in process string field of the received probe message the daemon detects a cycle. Therefore, in the daemon initiator the name of this probe is not available in the field of the corresponding array of probes of the initiator process, and the algorithm supposes the detected deadlock cycle has already become discrete for any reason. This similar scenario occurred in node P1 with probe ("1212","0123","2", P0). Thus this algorithm will not be able to detect deadlock cycle {1, 2, 3, 1}, although it does exist in the system.

Therefore, we displayed with simple examples that the previous algorithms are not able to identify deadlocks in some states, moreover the resolution method which they exploit, is not able to detect and resolve all of deadlocks. In this paper we will try to solve these problems and present an efficient algorithm for detection of multi-cycle deadlocks and resolution of each in the distributed systems. Our algorithm can detect all deadlocks, reachable from the initiator of the algorithm, even though the initiator does not belong to any deadlock cycle, whereas the chance of detecting phantom deadlocks is minimized.

3. SYSTEM ASSUMPTION

Each data object in our distributed system has a unique lock that can't be shared by more than one node. Each process has unique individual ID in the system. A node can make request for locks residing either at local or remote sites. There is no shared memory in the system and nodes communicate with each other by message passing. Message in the network act as FIFO (i.e.) messages are arrived at the destination nodes in the same order which they were sent from the source nodes. Such communication channels are reliable, messages neither are lost, nor replicated and transferred in an error-free manner. The reliability and ordering of the communication can be easily implemented by using combinations of retransmissions, acknowledgements and sequence numbers.

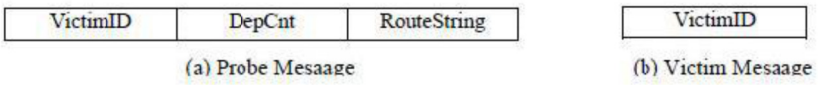

FIGURE 3. Structures of Probe and Victim Message

The probe message, used in this algorithm consists of three fields which are shown in Figure 3(a). The first field, InitID, contains the algorithm initiator identification. DepCnt (Dependency Count) of a node represents the numbers of successors for which the node is waiting for resources. Route-String field, contains the node IDs visited by probe message. In this algorithm the probe and victim message structures are changed and their numbers of fields are three and one, respectively. Therefore, propagation of these two messages among Wait-For-Graph edges requires less cost; moreover proceeding of them in each node needs less time.

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

At each node there will be a flag structure whose name is Initflg. In each node that initiates a probe message and forwards at least one message, this flag has the true value.

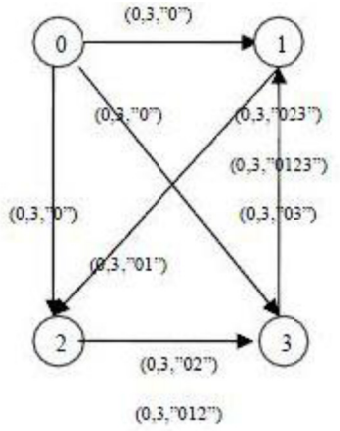

FIGURE 4. Example of Wait-For-Graph

4. PROPOSED ALGORITHM

Strategies for Algorithm Initiation. A threshold is assumed in the distributed system. If the waiting time in each node for acquiring a particular resource (or resources) exceeds the threshold and this node's Initflg is false, node initiates the deadlock detection algorithm by creating a probe message and sends it to all successor (or successors) nodes. At the beginning of the algorithm, the victimID, DepCnt and values of probe message are equal to initiator ID and the number of successors of this node respectively and the Initflg is false.

Probe Message Forwarding Policy. By receiving the probe message, a node first compares its own DepCnt value with probe's value. If the node's DepCnt value is not higher, then the probe's VictimID and DepCnt values will be kept intact. Otherwise the probe's VictimID and DepCnt values will be updated based on this node's ID and DepCnt values respectively. Before forwarding the probe message to all successor nodes, probe's Route-String field were updated by appending the node's ID at the end of existing Route-String and the Initflg were also changed to true value. For example in Figure 4, node '0' has initiated execution of the algorithm and has sent probe message (0, 3,"0") to successor nodes 1,2 and 3 while departure node's flag value is changed to true. Node's 1, 2 and 3 have updated only the Route-String field of received probe message and flag value, and then they sent this message to their successor.

Deadlock Detection. With the reception of a probe message, the node first checks whether its ID is in a received probe's Route-String or not. If it is, deadlock will be detected and then the probe message will be discarded as well, then node sends the victim message to all successors to restore all flags to false value. Otherwise this node appends its ID value at the end of Route-String field of probe message. A probe message is also discarded by a node in three states in this algorithm. First, the node detects the deadlock. Second, the node has no successor (or successors) node, and third is that the node Initflg value is equal to true. Therefore, our proposed algorithm can detect the deadlock cycles by Route-String field in each node. For example in Fig. 3, node '0' initiated execution of algorithm and sent probe message (0, 3,"0") to all successors. In node '1', deadlock was detected by reception probe

message (0, 3,"0123") from node '3', because this node ID '1' exists in the Route-String field of received probe message "0123".

Deadlock Resolution Method. In this algorithm, a deadlock is resolved by aborting one node that exists in multiple deadlock cycles. In order to carry out this idea successfully, a victim node with highest DepCnt value among the processes of the deadlock cycle is selected. Victimization of this node, can lead to break the cycle more likely because with high probability this node exists in a more deadlock cycles and the other process will have more chance to become complete. This idea is used in MC2DR algorithm, but as explained in previous section, does not work correctly. We try to use this idea correctly in here. If the detector node has the highest DepCnt value, therefore this node kills itself immediately. But, if the detector node has not the highest DepCnt value besides if in the probe message's victimID field another node ID is recorded, it performs as the following.

If the ID of the victimID field precedes the detector node ID value in the Route-String of probe message, the detector node victimizes itself because the victimID node is not a member of the detected deadlock cycle. This problem exists in the MC2DR algorithm. We solved this concern by putting aforementioned condition in our new algorithm. For example in Fig. 3, node '1' has detected the deadlock cycle {1,2,3,1} by receiving probe message (0,3,"0123"). The detector node first checks whether its ID has been in a victimID field or not. But in this example the ID of node '0' exists in the victimID field. Now, the detector node checks whether the ID of victim node has preceded the detector node ID value in the Route-String of probe message. In this example, '0' in "0123" precedes the '1' ID in the Route-String field. Therefore, the detector node kills itself because node '0' is not a member of the deadlock cycle.

If the ID of victim node tags along with the detector node's ID in the Route-String field, the detector node will send victim message to all successors. After receiving this message, the victim node first forwards it to all of its successors and then releases all locks held by; then kills itself. Each node, by receiving this message, restores the Initflg to false value and sends it to its successors (if exists). This procedure for any receiver node will be terminated when the Initflg value is false. In this case, the receiving node will not send the Victim message to its successors.

Deadlock Detection and Resolution Algorithm. For particular node i, pseudo code for our algorithm, presented at following.

ALGORITHM 1. Deadlock Detection and Resolution

```
1.  Algorithm_Initiation(){
2.    Int w; //the waiting time for a particular resource
3.    Probe p; //allocate memory for p
4.
5.    If (w>threshold && Initflg==false) {
6.      P=Create_probe(i);
7.      Send_probe(i,p);
8.    }
9.    Create_Probe( node i) {
10.     p.victimID = i.ID;
11.     p.DepCnt = i.DepCnt;
12.     p.RouteString = i.ID;
13.   }
14.   Send_Probe( node i, probe p) {
15.     Int k = i.DepCnt;
16.     While (k) {
17.       Send (k,p);
```


ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

```
        K--;
      }
      Initflg=True;
    }
    Receive_Probe( probe p) {
      If (p.DepCnt < i.DepCnt){
        p.VictimID = i.ID;
        p.DepCnt = i.DepCnt;
      }
      If (i.ID is in a p.RouteString) {
        Deadlock is detected;
        Discard (p);
        If (p.VictimID = = i.ID) {
          Send_victim(p.victimID);
          Release (all locks held by node i);
          Node i kills itself;
        }
        Else {//P.victimID is not equal to i.ID
          If (p.victimID in the p.RouteString is before of i.ID) {
            Send_victim(p.victimID);
            Release (all locks held by node i);
            Node i kills itself;
          }
          If (p.victimID in the p.routestring is after of i.ID) {
            Send_Victim(p.victimID);
          }
          If (p.victimID in the p.routestring is after of i.ID) && (p.Depcnt==i.Depcnt){
            Send_victim(p.victimID);
            Detector node kills itself;
          }
      }
    }
    Else {// Deadlock is not detected
      p.RouteString = p.RouteString + i.ID;
      Initflg=True;
      Send_Probe(i,p);
    }
  }
  Recieve_Victim( int VictimID) { // forward Victim message to all successor;
    If (i.Initflg=false) {
      Discard this message;
    }
    Else {
      Send_Victim (victimID);
      Initflg=False;
      If (victimID = = i.ID) {// this is a victim node.
        Release (all locks held by this node);
        Kill (this node);
      }
    }
  }
```

5. SIMULATION AND PERFORMANCE COMPARISON

As it's described in section2, on one hand provided algorithm by Rahim Alipour wasn't able to identify deadlocks in systems while Abdurrazzaq algorithm could

detect deadlocks better than other available algorithms. And on the other hand, Abdurrazzaq's algorithm processes an ability of detecting more deadlocks among other ones. Taking these facts into account, performance of our proposed algorithm has been compared with that of Abdurrazzaq algorithm.

We have run the simulation program using fixed site (20) connected with underling network speed of 100Mbps, but with varying multiprogramming level (MPL), ranges from 4 to 9.

The numbers of aborted nodes are shown in Figure 5. This value in our method will be less than the recorded amount in Razzaq algorithm. This situation does occur, since a deadlock is resolved by abortion of one node existing in multiple deadlock cycles.

FIGURE 5. The Number of Aborted Node

Figure 6 shows the numbers of detected deadlocks in both algorithms. The proposed algorithm can detect all deadlocks in the system that cannot be identified in some multi-cycle states of MC2DR algorithm.

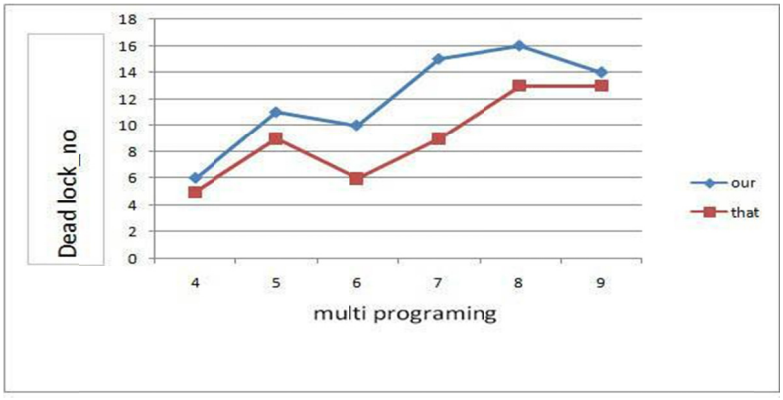

FIGURE 6. The Number of Detected Deadlock

This issue finally leads to increase the persistence duration of deadlock in Razzaq algorithm. Average deadlock detection time is shown in Fig. 8. This parameter is obtained from the average blocked process waiting time. In our algorithm this

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)

parameter is almost half of the similar amount in MC2DR algorithm. This is because Razzaq algorithm cannot be able to detect all of the deadlocks in the system.

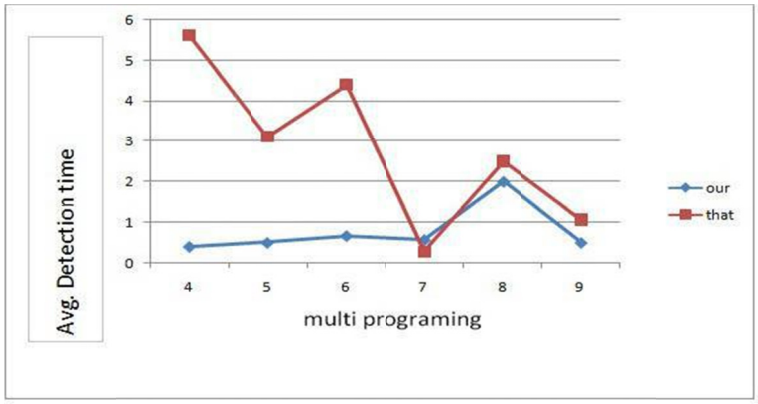

FIGURE 8. The Avarage Time Detection Deadlock

6. CONCLUSION

Recommended algorithm is considered to be both reliable and liable due to its high-detection potential in all deadlocks including multi-cycles ones which begin from the starting available node. This ability is believed to be unique in comparison with former algorithms which couldn't cover all deadlocks in systems. Moreover, new algorithm deadlock detection mechanism in the beginning of its process- thanks to its logical flag structure, besides its victim node selection mechanism are believed to be extremely efficient (i.e.) regarded algorithm victimizes a node which is known to participate in several deadlock cycles.

REFERENCES

[1]. Choudhary, A.N., "A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution.", (1989), IEEE Trans Software. Vol. 15, pp.10-17

[2]. Knapp, E., "Deadlock Detection in Distributed Database.", (1988), ACM Computing Surveys,Vol.3, pp.303-328

[3]. Chandy, K.M, Misra, J., "A Distributed Algorithm for Detecting Resource Deadlock in Distributed Systems.", (1982), In: Proceeding of the ACM Symposium on Principles of Distributed Computing. New York, pp.157-164

[4]. Chandy, K.M, Misra, J.Hass L.M., "Distributed Deadlock Detection.", (1983), ACM Transaction on Computer Systems, Vol. 1. pp. 144-156

[5]. Abdur Razzaque, M.D., Mamun-Or-Rashid, M.D. and Hong, C.S., "MC2DR:Multi-Cycle Deadlock Detection and Recovery Algorithm for Distributed Systems.", (2007), Lecture Notes in Computer Science, Vol. 4782, pp.554-565.Springer-Verlag, Berlin Heidelberg

[6]. Sinha, M.K., Natarajan, N., "A Priority Based Distributed Deadlock Detection Algorithm." (1985), IEEE Trans. Software Engineering, Vol. SE-11, No.1, pp.67-80

[7]. Kim, Y.M, Lai, T.W., Soundarajan, N., "Efficient Distributed Deadlock detection and Resolution Using Probes, Token and Barriers.", (1997), Proceeding on Parallel and Distributed Systems, pp. 584-591

[8]. Farajzadeh, N., Hashemzadeh, M., Mousakhani, M., Haghighat, A., "An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems.", (2005), Proc.5th IEEE Int'l Conf. Computer and Information Technology (CIT'05)

[9]. Farajzadeh, N., Hashemzadeh, M., Haghighat, A.,, "Optimal Detection and Resolution of Distributed Deadlocks in the Generalized Model.", (2006), proc.14th IEEE Int'l Conf. Parallel, Distributed, and Network-Based Processing (PDP'06)

[10]. Rahim Alipour, Z., Haghighat, A., "Daemon- Based Distributed Deadlock Detection and Resolution.", (2010), Word Academy of Science, Engineering and Technology

[11]. Lee, S., "Centralized Detection and Resolution of Distributed Deadlocks in Generalized Model.", (2004), IEEE Transactions on Software Engineering, Vol. 30, Issue 9, pp. 561-573

[12]. Lee, S., Kim, J.L., "An Efficient Distributed Deadlock Detection Algorithm.", (1995), proceeding of 15th IEEE International Conference on Distributed Computing Systems, pp.169-178

ISSN: 2252-4274 (Print)
ISSN: 2252-5459 (Online)