Daemon- Based Distributed Deadlock Detection and Resolution

Z. RahimAlipour, A. T. Haghighat

Abstract**—**detecting the deadlock is one of the important problems in distributed systems and different solutions have been proposed for it. Among the many deadlock detection algorithms, Edge-chasing has been the most widely used. In Edge-chasing algorithm, a special message called probe is made and sent along dependency edges. When the initiator of a probe receives the probe back the existence of a deadlock is revealed. But these algorithms are not problem-free. One of the problems associated with them is that they cannot detect some deadlocks and they even identify false deadlocks. A key point not mentioned in the literature is that when the process is waiting to obtain the required resources and its execution has been blocked, how it can actually respond to probe messages in the system. Also the question of 'which process should be victimized in order to achieve a better performance when multiple cycles exist within one single process in the system' has received little attention. In this paper, one of the basic concepts of the operating system - daemon - will be used to solve the problems mentioned. The proposed Algorithm becomes engaged in sending probe messages to the mandatory daemons and collects enough information to effectively identify and resolve multi-cycle deadlocks in distributed systems.

Keywords**—**Distributed system, distributed deadlock detection and resolution, daemon, false deadlock.

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf

I. INTRODUCTION

IF a process in a distributed system needs a resource, which is located in another machine, it sends a message to that machine through a network connection to access the required resource. If the required resource is available, it will be allocated to the process and if it is being used by other processes, the requesting process will be blocked until the resource is released and obtained. Deadlock occurs when a set of processes wait for each other for an indefinite period of time to obtain their intended resources. 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 Therefore, the problem of prompt and efficient detection and resolution of deadlocks is an important issue in a distributed system [3]

Z. R. Author is with the Department of Electrical, Computer & IT, Islamic Azad University, Qazvin Branch, Qazvin, Iran (corresponding author to provide e-mail: z.rahimalipour@gmail.com).

A. T. H. Author is with the Department of Electrical, Computer & IT, Islamic Azad University, Qazvin Branch, Qazvin, Iran (corresponding author to provide e-mail: Haghighat@qazviniau.ac.ir).

Dependence relationship between processes in distribution systems is shown by a directed graph called Waite-For Graph [4]. In this graph, each node corresponds to a process and an edge directed from one node to another indicates that the first process is waiting for the resource the other process is holding. A cycle in this graph indicates the presence of a deadlock in the system. There are several resource request models defined for the process operations in Distribution systems [6]. The simplest one is single-resource model in which a process is only able to request at most one resource at a time. In the AND model, a process will be able to request a set of resources and wait until all requested resources are provided. In OR Model, a process that needs some resources will not be active unless at least one of its required resources has been provided. AND-OR model is a combination of the two models. In this model, any combination of resources is possible. This model is the more general form of AND-OR model in which a process simultaneously makes a request for q resource and remains blocked until it is granted out of q resources. Another model is called the unrestricted model. In this model, there is no particular structure for resource request.

Four categories have been proposed for classifying distributed deadlock detection algorithms [6]: path pushing, edge chasing, diffusing computing and global state detection algorithms. Edge chasing algorithms are regarded as one of the most important deadlock detection algorithms due to their high application and feasibility. In this method, a special message called probe is generated by an initiator process and propagated along the edge of WFG. Deadlock is detected when this probe message gets back to the initiator, forming a dependency cycle.

Edge-chasing algorithms have been mentioned a lot in the literature [2], [3], [4], [8], and [12]. The key limitation in these algorithms is that they are unable to detect deadlocks whenever the initiator does not belong to the deadlock cycle; that is when the detector process is the same as the initiator process. Although this problem has been solved in some algorithms, they still detect false deadlocks [9], [10].

In addition, these algorithms cannot detect deadlocks when a single node becomes involved in several deadlock cycles. Another question often ignored in previous studies is how a process can answer deadlock detection messages received when it is stuck in a deadlock cycle and is therefore on sleep mode?

In this paper we will try to solve these problems using the concept of daemon in the operating system and with

introducing an applicable structure for a probe message and providing an efficient algorithm in order for a correct detection of deadlocks and therefore, minimizing the possibility of false deadlock detection in distributed systems. This study is mainly concerned with the detection of multi-cycle deadlocks. There has been an attempt to find an effective method for resolving such deadlocks.

The rest of the paper is organized as follows; a thorough study of state-of-the-art probe based algorithms and the criticisms against them are presented in section 2.section 3 include describe of the proposed algorithm with some executions. Correctness proof of algorithm is given in section 4. A brief Performance analysis is presented in section 5 and performance comparison with simulation presented in section 6 . Section 7 is the conclusion of this paper.

II. Related works

The main idea of using probes was first introduced by Chandy-Misra and Haas [3] .The key concept in CMH algorithm is that the initiator propagates probe message in the WFG and declares a deadlock upon receiving its own probe back. Probe message in this algorithm has three parameters (i,j,k), which respectively include: the blocked process ID, the sending process ID and the ID of the process that should receive the message. Deadlocks occur when we have a message in the form of (i,j,i) that is when the process that has initiated the probe operations receives the same probe message . Therefore, a cycle is identified in the system and a deadlock is detected.

Another algorithm was presented by Mitchell and Merritt which is similar to Chandy-Misra and Haas algorithm except that each process has two different labels; 'public' and 'private' [11]. The two labels have equal values in the beginning. This algorithm is able to detect the deadlock by propagating public labels in the backward direction in WFG. When a transaction gets blocked, the public and private labels of its node in WFG increase in value and undergo greater changes than the public labels of the blocked transaction. A deadlock is detected when a transaction receives its own public label; this method ensures that there is only one detector in the system.

Sinha and Natarajan presented a bipartite algorithm that includes detecting and resolving deadlocks [12]. During the detection step, a probe message is used and processes should save some of the probe messages. In the deadlock resolving step, priority is used to reduce the number of probe messages and the process with lowest priority in a cycle is chosen as the victim accordingly. Also unnecessary probe messages that are stored in the system by other transactions are deleted through the victim process.

Chadhary et.al presented a modified algorithm that somewhat fixed the problems in Sinha and Natarajan's algorithm [4]; problems such as deadlock detection failure and false deadlock detection. However, this algorithm was later reviewed by Shemkalyani and Singhal and modified again and its correctness was substantiated [8]. 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 [9.10]. In this algorithm a tree is generated through propagating probes in the system and deadlocks are detected based on the information obtained from data dependency between the tree nodes. However, this algorithm cannot identify all the deadlocks reachable from the initiator and may detect false deadlocks during concurrent executions.

In the algorithm proposed by Faraj Zadeh et .al, a probe with two parameters was introduced: initiator ID and an integer string called Route- String which includes the IDs of the passing edges from any of the Graph nodes [5]. In this paper, the storage was considered for graph nodes in which probe messages passing in any of the nodes are saved. If the corresponding storage is empty in the passing probe of a node, the probe is stored and forwarded to the next nodes, otherwise, the message ID in storage and the received message ID will be checked for correspondence. Finally, if the path-string of the message in storage be a prefix of the received message path-string, deadlocks are detected. However, in the multi-cycle deadlock detection issue was ignored this algorithm.

The Algorithm proposed by the Abdorrazzaq et .al addresses the multi-cycle deadlock detection issue and the algorithm is able to identify and resolve these deadlocks through providing structures for probe and victim messages [1]. Although this algorithm is good at solving many of the problems in this field, it does this at the cost of a memory overhead for each process to achieve this goal.

III. The proposed algorithm

In Distribution systems, presenting a comprehensive algorithm that can detect a deadlock with certainty and resolve it in an efficient manner is almost impossible [14]. The algorithm presented in this paper is an optimal algorithm for detecting and resolving deadlocks especially during concurrent executions in the system.

A. *System assumptions*

A distributed system consists of a set of processes connected by a network Communication delay is limited but unpredictable. A distributed program is a set of n-asynchronous processes (p1, p2, ... , pn) in which communication is made through message passing. Each Process has a unique individual ID in the system and there is no shared memory. It is FIFO assumed that messages in the network act as FIFO and that they are reliable i .e, messages do not get lost or are not replicated and therefore they are transferred in an error-free manner.

B. *Daemon application*

According to OS definitions, Daemon is a process that runs in the background and is in sleep mode under normal

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf

conditions [14]. When an event takes place in the system, it wakes up and logs it. Each machine can host several daemons in a distributed system. Here daemon is considered as one of the core components of the operating system.

In this paper, a daemon is considered for each machine having a database with the following components:

Process ID	Process Requirements	Port	Array of Probes

By utilizing the above definition in the database of every daemon, probes will be easily able to engage in message exchange between the daemons.

In this database, the name of the process includes the process IDs for any daemon. Communication between processes is specified by the process requirements field; so if a Process is waiting to get more resource(s) held by one or more other processes, the process IDs are stored in this field. Corresponding with any requirement, if this requirement is inside the daemon, the port field value is NULL; otherwise the related port number of the process daemon is respectively stored. All the probes passing through each process will be stored in the array of probes so that the algorithm can optimally track the daemons. In a Given distributed system, a probe message is produced and propagated in each daemon and the name of each probe is shown with its associated daemon ID. For example, a probe generated in daemon No.1 and propagated in the system is named pb1.

C. Algorithm description

A process can be in two states: ruining and blocked. In the running state (active) processes obtain all the requested resources and are running or are ready for run. A process is blocked when waiting to obtain some resources. Deadlock occurs when a set of processes are waiting for each other to obtain unspecified resources. The proposed algorithm is a probe-based algorithm and its probe message has 4 fields: victim ID, Maximum Requirements, Processes string and daemon sting.

Victim ID	Maximum Requirement	Processes String	Daemon String

Victim ID is a process ID that must be killed to resolve a deadlock and its value at the beginning of the algorithm is equal to process ID from which the probe has initiated. In the path of a probe, maximum resource requirements of a process held by other processes are stored in Maximum Requirements field and concurrently victim ID is updated. Maximum Requirements value in the beginning of the algorithm shows the number of requested resources in the first process of a daemon. Process strings and daemon strings store the probe routes sequentially.

When process strings reach a process that is available in the prefix of array of probes, deadlock has occurred. This process is called deadlock detector. After deadlock detection, a message called "victim message" is used to resolve the deadlock. This message leads to the removal of a process ID which has been stored in the victim ID field. For this purpose, the daemon in which the victim ID is available is identified by means of daemon string, and a victim message is sent for deleting it. If the field of victim ID contains a process that is not in the identified process cycle, the execution of algorithms leads to deleting a process that does not affect deadlock resolution procedure. Therefore, victim ID field will be updated by the detector's ID.

D. Algorithm execution

A threshold is assumed for processes in the system. If the waiting time for acquired resources exceeds the threshold, the daemon initiates the deadlock detection algorithm by generating a probe message. Through requirements field, the daemon can find out to what process to send a probe if the specified process is outside the current daemon, the address of the destination daemon will be available in the port field. When a probe passes through a daemon, the information about it will be registered in the database of the daemon.

Suppose we have a distribution system with 3 machines and 6 processes and requirements in accordance with Figure1.

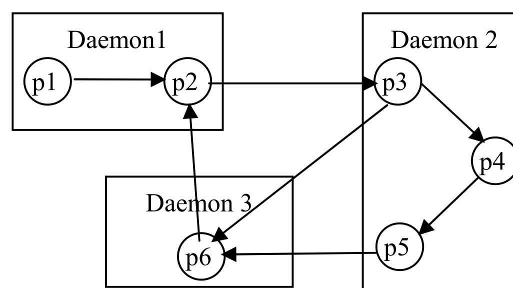

Fig. 1 A distribution system with 3 machines and 6 processes

In Given Distribution System, a daemon has been assumed for each machine. Address of the daemons is presumably considered the same as their number. As it was mentioned, one database is assumed for each daemon in which process requirements for the resources available to other processes can be found. (See figure 2)

The algorithm starts when process p1 in daemon D1 is waiting for more than expected threshold time. In this case, D1 will be the deadlock detection initiator. Daemon D1 creates a probe message and propagates it in the system. The probe message created will be like ("1","1", 1, p1). This message may be sent from the information in the database of daemon D1 to the processes involved in the daemon. Figure 3 shows how this message is propagated in the system. Finally when the probe gets back to p2, as the ID of this process is available in probe processes string field, the daemon detects a cycle and proclaims p2 as the deadlock detector process. Now we will review cycle creation and correct deadlock detection procedure discussed up to now. Therefore, in the daemon initiator we will try to see if the name of the probe is available

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf

in the field of the corresponding array of probes or not.

DB_D1

Process name	Requirements	Port	Array probe
p1	p2	-	
p2	p3	2	

DB_D2

Process name	Requirements	Port	Array probe
p3	p6, p4	3,-	
p4	p5	-	
p5	p6	3	

DB_D3

Process name	Requirements	Port	Array probe
p6	p2	1	

Fig. 2 Database per Daemon

If the name of the probe is available, cycle is proved to exist and a victim message is sent from daemon D1 to daemon D2 in which the victim process p3 is available. Otherwise, the probe is discarded because the cycle has already become discrete for any reason.

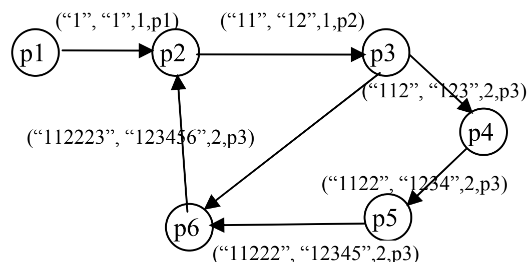

Fig. 3 Propagation of probe message

Then, all the daemons available in the daemons string will be announced to delete the probe message. All the probes in the probe array field of p3 process are also discarded (concurrent execution problem) and the initiator probe is announced to remove the probes from its array of probes.

E. *Concurrent execution problem*

In a distributed system, concurrent execution of an algorithm on a few machines is inevitable. As long as a daemon in a machine sends probe messages for detecting deadlocks, there may be other machines simultaneously propagating other probes messages leading to false deadlock detection, especially when we face a multi-cycle system.

We will take Figure 4 as a multi-cycle system. Deadlock detection algorithm will simultaneously be initiated by any of the 3 daemons of p1, p2 and p5 processes. Having finished pb1, pb2 and pb3 probes will respectively contain process strings of "1231", "24512", and "5645". Suppose that pb3 probe has completed its work earlier. Then daemon D3 should victimize the p5 process. In probe array field of p5 process in daemon D3 database, pb2 may also be available in addition to this probe, and with the end of pb2 route, false deadlock will be found. So a message is sent to the daemon of the second probe i.e. D2, to remove the name of this probe from the field array of the daemon probes.

After detection cycle "24512" is identified by Pb2, this probe will be discarded because p5 has been already destroyed and the initiator daemon probes array is without pb2.

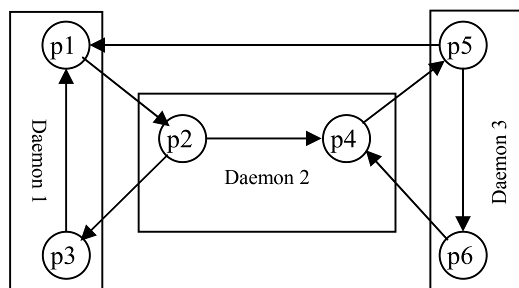

Fig. 4 Distributed multi- cycle

After the end of pb1, p2 process is victimized for resolving the deadlock caused by the "1231" cycle and the updates are done according to the above procedure

F. *Deadlock Detection and Resolution Algorithm*

For a particular node *i*, pseudo code for deadlock detection and recovery algorithm is presented in Appendix.

IV. Correctness Proof

A deadlock detection algorithm is correct if it detects every deadlock that occurs and does not detect any false deadlock. In this section, we will show that the proposed algorithm is correct under the following assumptions:

Theorem 1. If a deadlock is detected, the corresponding nodes are really in deadlocked state. Phantom deadlocks are not detected.

Proof. Let's prove it using proof by contradiction. A set of nodes could be detected as in a phantom deadlock, when the detection algorithm misinterprets the existence of a deadlock cycle. This type of misinterpretation can be taken place in this algorithm only and if only at least any two nodes have the same ID. In the case, a false deadlock is detected even though the traveling path of probe message does not make a cycle. But this contradicts with our network model, described in section 3.1.

Theorem 2. If a deadlock occurs, in system, (safety) it will be detected (progress).

Proof. Using concept of daemons, all of the process requirements in the system, are specified. Because the probe propagation in system done by means of daemons, it can

published to all of possible route in communication network. So, if a cycle exist in the system, prefix relationship in the probe's process string field, eventually will be detect deadlocks.

V. PERFORMANCE ANALYSIS

Message traffic and the amount of time required for detecting a cycle are the factors often used for analyzing the performance of a distributed deadlock detection algorithm [7]. In most of the papers, the number of messages in the system is considered to be a coefficient of the number of processes in the system, but in the proposed algorithm, the number of messages will be a coefficient of the number of daemons in the system. If p is assumed to be the number of the daemons in the system, message traffic will be less or equal to p. This number is much less than the numbers mentioned in other algorithms. If the required time for following an edge in the cycle process is assumed to be one unit and 't' shows the communicating time between daemon processes and 'm' represents the number of deadlocked processes, then required time for detecting the cycle equals (m+t). In other words, the order of time complexity in the algorithm is o (n).

TABLE I
COMPARISON OF ALGORITHMS PERFORMANCE

ALGORITHM	COMPLEXITY OF MESSAGE	DELAY
Chandy ,Mysra, Haas [3]	<m.n	2d + 2
Sinha, Natarajan [12]	Best:2 (n-1) Worst: m (n-1)	O (n)
Mitchell ,Merritt [11]	M (n-1)	O (n)
kShemkalyani, Singhal [8]	4e	O (n)
Lee, Kim [10]	2e	2m
Lee [9]	<2e	d + 2
Faraj Zadeh et .al [5]	≤e	O (n)
Proposed Algorithm	≤p	O (n)

In Table 1, a comparison between the performance of the proposed algorithm and existing algorithms is presented. It should be mentioned that n, d, and e respectively represent the number of processes in the graph, graph diameter and the number of graph edges in the system.

VI. SIMULATION AND PERFORMANCE COMPARISON

We have run the simulation program using fixed sites (20) connected with underlying network speed of 100 Mbps, but with varying multiprogramming level (MPL), ranges from 4 to 14. Also range of process size in experiments is varying between of 5 to 10. Performance of our proposed algorithm has been compared with that of Razzaq algorithm [1].

As shown in Fig. 5, with increasing MPL or process size value, message overhead are also increased. Increasing the number of probe messages to detect deadlocks, lead to increasing systems overhead communication in both algorithms. Probes in proposed algorithm are produced by Daemons, but each process in Razzagh algorithm is able to probe production separately. So the number of messages produced in our method will be less than the Razzagh algorithm. Although messages in both algorithms (with the use of strings) have variable size but Lee showed with simulation tests that the average path-strings length field is small enough that even under the heavy load can be ignored it.

Average deadlock detection duration is shown in figure 6. This parameter is obtained from the average blocked process waiting time and as can be seen in the figure, with using Daemon in proposed algorithm, deadlock persistence time in system is almost half of similar amount in Razzaq algorithm.

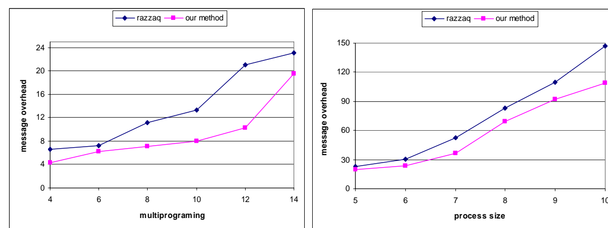

Fig. 5 Systems message overheads

Same result when variable size of process has been used is obtained. Reduce the deadlock persistence time reason is faster resolving deadlocks in our method. Deadlock resolution with victimize of process that has highest demand of the other process can lead to break more likely cycles. Thus average blocked process Waite time on the system will be decreased and the other process will have more chance to completion.

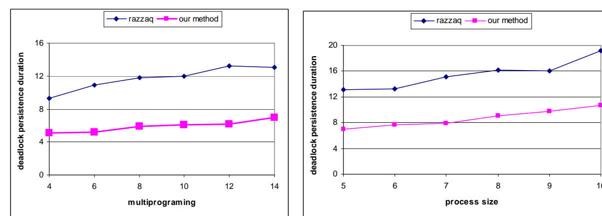

Fig. 6 Average deadlock detection duration

VII. CONCLUSION

In this paper, a distributed algorithm was proposed for detecting and resolving deadlocks. This algorithm is able to discover the deadlocks in operating systems correctly using the concept of daemon and minimizes the possibility of false deadlock detection by presenting a suitable structure for probe messages.

APPENDIX

Pseudo code for deadlock detection and recovery algorithm:

```
Struct Probe {//structure of probe
    String Daemons
    String Processes
    Int DepCount
    Index Victim
}
Algorithm Initiation () {
    Int W; //pre defined time out
}
```

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf

```
Probe Probe_Initiate(Process p)
{
Probe myProbe = new probe ();
    myProbe.Daemons.ADD(p_Daemon);
    myProbe.Processes.ddADD(p);
    myProbe.DepCount = p_Requirements;
    myProbe.Victim = X;
Return myProbe;
}
Is_Exist_Cycle (Probe pb_X)
{
    Daemon d= pb_X.Daemons.firstmember();
    Process p= pb_X.Processes.firstmember();
    If ((pb_X) exist in (Array Probe) of (p) at (d))
        Return true;
            Return false;
}
Discard (Probe pb_X)
{
    Remove pb_X from all daemons' Database;
    Free (pb_X);
}
Kill (Probe pb_X)
{
    Daemon d = pb_X.Victim_Daemon
    For each (Probe pb in d.Array_Of_Probes(pb_X.Victim))
        Call Pb.Daemon.Initiator to remove pb_X From Daemon DB
            For each (Probe pb in pb_X)
              Call pb_X.Victim_Daemon to KILL x.Victim;
}
Send Probe (Probe pb_X)
{
READ DATA FROM THIS DAEMON;
SENDING PROBE TO ALL PORTS OF THIS PROCESS;
}
Receive Probe (Probe pb_X)
{
    THIS.DAEMON.Array_Probe.Add(pb_X);
    If (process.ID isn't prefix of pb_X.Processes)
{
    If (process_Requirements > pb_X.DepCount)
{
    pb_X.DepCount = Process_Requirements;
    pb_X.victim = Process_ID;
}
pb_X.Daemons.ADD(Process_Daemon);
pb_X.Processes.ADD(Process_ID);
Send Probe (pb_X);
}
Else
{
If (Is_Exist_Cycle ())
        //Deadlock Detected;
{
            Kill (pb_X);
            Update_Probes(pb_X);
}
Discard (pb_X);
Exit;
}

}
Algorithm Main () {
//waiting_for_Requirements (p)
    While (waiting_for_Requirements (p)>W)
    {
    Probe pb_X = Probe_Initiate(p);
    Send Probe (pb_X);
     }
       }
```

REFERENCES

[1] Abdur Razzaque. Md., Mamun-Or-Rashid. Md., Ch.Hong,"MC2DR:Multi-cycle Deadlock Detection and Recovery Algorithm for Distributed Systems", LNCS 4782(HPCC2007), Sep 26-28 2007, pp. 554-565

[2] Chandy, KM, Misra, J,"A distributed algorithm for detecting resource deadlocks in distributed systems". In Proc. ACM SIGA CT-SIGOPS Syrup, 1982, pp. 157-164

[3] Chandy KM, Misra .J, Haas LM, "Distributed Deadlock Detection", ACM Transactions on Computer Systems, May 1983,Vol 1,No. 2.PP 144-156

[4] Choudhary et al," A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution", IEEE Trans Software, January 1989, vol.15, No.1, pp .10-17

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

[6] Knapp, E, "Deadlock Detection in Distributed Databases". ACM Computing Surveys, Dec.1988, vol.3, no. 4, pp.303-328.

[7] Kshemkalyani AD, Singhal M ,"Distributed detection of generalized deadlocks". In: Proceedings of the 17th International Conference on Distributed Computing System, IEEE Computer Society Press, 1997, pp 553–560

[8] Kshemkalyani, A. D, Singhal, M, "Invariant based verification of a distributed deadlock detection algorithm," IEEE Trans. Software Eng, Aug. 1991, vol 17, pp. 789-799.

[9] Lee.S,"Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model", IEEE Transaction on Software Engineering, September 2004, Vol. 30 , No.9 ,pp. 561-573

[10] Lee, S., Kim, JL,"An Efficient Distributed Deadlock Detection Algorithm". In: Proc. 15th IEEE Int'l Conf. Distributed Computing Systems, pp. 169–178 (1995)

[11] DP Mitchell and MJ Merritt,"A Distributed Algorithm for Deadlock Detection and Resolution", Proc. Third ACM Symp. Principles of Distributed Computing, pp. 282-284, Vancouver, Canada, Aug. 1984.

[12] MK Sinha and N. Natarjan, "A priority-based distributed deadlock detection algorithm", IEEE Trans. Software Eng., Vol. SE-11, No. 1, Jan. 1985, 67-80.

[13] Singhal, M, "Deadlock Detection in Distributed Systems", IEEE Computer, Nov.1989, No 22, pp. 37-48.

[14] Tanenbaum ,A."Modern Operation Systems", 3 e, (c) Prentice-Hall, Inc. 2008

Open Science Index, Electrical and Computer Engineering Vol:4, No:3, 2010 publications.waset.org/10923/pdf