An Efficient Distributed Deadlock Detection and Prevention Algorithm by Daemons

Alireza Soleimany[2], Zahra Giahi[1]
[1]Computer Engineering Department, Islamic Azad University, Lahijan Branch, Lahijan, Iran

[2]Computer Engineering Department, Islamic Azad University, MeshkinShahr Branch, Meshkin Shahr, Iran

Abstract:

The deadlock is one of the important problems in distributed systems and different solutions have been proposed for solving 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. 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. Besides, before allocating one resources to a process waiting for it, a reasonable action is to 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, prevention

1. Introduction

Recently, many researches were done in the field of distributed systems like H.Zheng et.al(2008) and Kofahi(2005) . One of the most important fields is process management and resource allocation.
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 that proposed by Chandy et.al(1983).Dependence relationship between processes in distribution systems is shown by a directed graph called Waite-For Graph that proposed by Choudhary (1989). 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 like that proposed by Knapp(1982). 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: 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

Manuscript received April 5, 2012
Manuscript revised April 20, 2012

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 Chandy and Misra (1982,1983), Choudhary(1989), Farajzadeh et.al(2005), shemkalyani et.al(1991), Sinha and Natarjan (1985).

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 like as Lee et.al (1995, 2004).

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. In section 3 we describe the proposed algorithm with sample executions. Section 4 consists of correctness proof of proposed algorithm. Performance comparisons are presented in section 5 and finally we conclude the paper in section 6.

2. Related works

The main idea of using probes was first introduced by Chandy-Misra and Haas(1982) .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(1982,1983) algorithm except that each process has two different labels; ‘public’ and ‘private’ . 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 (1985) presented a bipartite algorithm that includes detecting and resolving deadlocks. 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(1982,1983) presented a modified algorithm that somewhat fixed the problems in Sinha and Natarajan’s algorithm; problems such as deadlock detection failure and false deadlock detection. However, this algorithm was later reviewed by Shemkalyani and Singhal (1991) and modified again and its correctness was substantiated . None of the algorithms are able to identify deadlocks in which the initiator is not directly involved in the cycle, though Lee et.al(1995,2004) proposed an algorithm in which deadlocks can be detected even when the initiator does not belong to any deadlock. 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(2005), 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. 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(2007) 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 . 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.

3. 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 . The algorithm presented in this paper is an optimal algorithm for detecting, preventing and resolving deadlocks especially during concurrent executions in the system. The proposed method is an enhance to [17].

3.1. 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.

3.2. Daemon application

According to OS definitions by Tanenbaum (2008), Daemon is a process that runs in the background and is in sleep mode under normal conditions. 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.

3.3. 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.

3.4. 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 Figure 1.

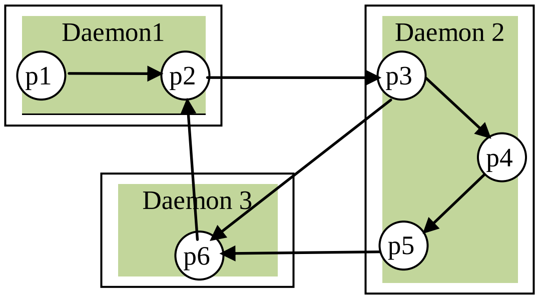

Figure 1. A distribution system with 3 machines and 6 process

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)

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	

Figure 2. Database per daemon

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

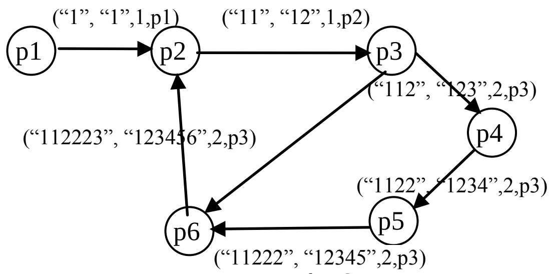

Figure 3. Propagation of probe message

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.

3.5. 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".

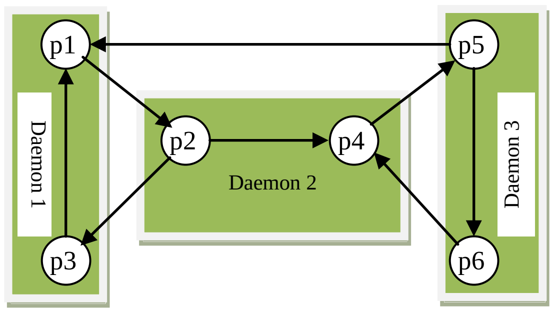

Figure 4. Distributed multi- cycle

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

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.

3.6. Prevention before deadlock occurred

When a resource is free and some processes are in wait queue to use it, we likely are able to detect and prevent from those allocations tend to deadlock. Resource allocation is done by each daemon. Figure5 shows pseudo code of resource allocation.

```
AllocFunc (resource M [n1], Request R [n2])
{
 For (each free resource in M) DO
       Find a requester in R that is in the  last phase of allocation or
              needs only one  resource
      else until there are reliable proceses ,
             allocate resource to them
      else if there is not reliable process , restart this function
             after T seconds
}
```

Figure5. Pseudo code of allocation function

Each daemon has a list of waiting process and a list of available resources. Daemon finds those processes are in the last phase of allocation or those only need one resource to complete their execution. It’s completely reasonable action beacuase resources of these processes are returned to system; so resources of system increase. If there are not a process, which needs one resource or in the last phase, the proposed allocation algorithms search for reliable process. The reliable process is a process that gets a resource but does not cause any deadlock. If the proposed allocation algorithm does not find any reliable process while there are some free resources, it restarts allocation function after T seconds. T depends on number of available resources and grade of multiprogramming.

4. Performance evaluation

In this paper, the proposed method with and without prevention function is simulated. Result of simulation is compared with MC2DR that proposed by Abdur Razzaque et.al(2007). In the simulation, there are 10 computers with five resources in each one. The needed resources of each process are a poisson distribution with average of 5. Hold time of each resource is produced by a poisson distribution with average 6 seconds. Simulation time is 300 seconds.

Table1 presents the number of dead locks and wait time in different methods. The proposed method is more efficient than MC2DR.

Process producing rate per second	The proposed method without prevention		The proposed method with prevention		MC2DR	
	#deadlocks	Wait time(sec)	#deadlocks	Wait time(sec)	#deadlocks	Wait time(sec)
2	56	1.01	9	2.09	87	1.23
4	92	1.89	21	2.31	132	2.32
6	221	2.43	48	2.47	389	3.71
8	297	2.88	63	2.53	561	4.59
10	452	3.12	82	2.49	967	6.75

In according to results of table1, allocation function has impressive role in performance of the proposed method.

5. 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. Moreover, an allocation method was proposed to find reliable process for allocating. This function has dramatic improvement in results.

In the future works, the proposed methods are implemented in real environments.

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 ," 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,"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

[15] H.Zheng, Y.Yue Du and Sh. Xia Yu,”Modeling Non-Repudiation in Distributed Systems”, Information Technology Journal, 228-230,2008.

[16] N.A.Kofahi, S.Al-Bokhitan and A.Al-Nazar,”On Disk-based and Diskless Checkpointing for Parallel and Distributed Systems: An Emprical Analysis”, Information Technology Journal,367-376,2005.