Journal of
Information Technology & Software Engineering

Shivendra et al., J Inform Tech Softw Eng 2015, 5:2
DOI: 10.4172/2165-7866.1000148

Research Article Open Access

Distributed Deadlock Detection Technique with the Finite Automata

Shivendra Kumar P*, Hari Krishna T and R. K. Kapoor

Department of Computer Application and Research, National Institute of Technical Teachers Training and Research, Bhopal, India.

Abstract

In the distributed, system deadlocks is a fundamental problem. A process may request resources in any order, which may not be known in advance and a process can request resource while holding others. Deadlocks can occur if the sequence of the allocations of resources to the processes is not controlled. Fast and efficient deadlock detection is very challenging and difficult task in distributed systems. In this paper distributed deadlock is detected by the distributed control manager. We proposed a distributed deadlock detection algorithm based on the finite automata to detect deadlock in a distributed environment. In this proposed solution we draw the wait for graph for the distributed transaction with the help of finite automata. Our proposed algorithm avoids the transmitting massage to other nodes; it is based on the expansion of an unvisited node in the wait for graph with the help of finite automata. This finite automaton based deadlock detection technique works fast and takes less number of comparisons to detect the deadlock in the wait for graph.

Keywords: Distributed systems; Deadlock detection algorithm; Wait for graph; Finite automata.

Introduction

In a distributed system, deadlock is a situation which occurs when a process enters a waiting state because a resource requested is being held by another waiting process, which in turn is waiting for another resource [1]. If the process is unable to get full filled their request forever because the resource is requested by it is being held by another process, then the system is said to be in a deadlock [2]. Two common places where deadlocks may occur are with processes in an operating system (distributed or centralized) and with transactions in a database [3]. Deadlock is a common problem multiprocessing systems, parallel computing and distributed systems, where software resources and hardware resources are used to perform the task [4]. There are two major types of model available for the deadlock that is the AND model (also called the multiple-resource model), a process is allowed to make several resource requests, and it is blocked until all of the requests are granted. Processes in this model can be involved in several deadlock cycles at once [5,6]. In the OR model (also called the communication model), a process makes several requests and is blocked until any one of them is granted [6]. The AND-OR model allows a combination of request types, such as a request for resource X and either Y or Z [5]. Our proposed model is based totally on the AND model.

Necessary Conditions for Deadlock

Mutual exclusion

There must be some non-sharable resource. That only one process can access at a time [7].

Hold and wait

A process is holding a resource and requesting an additional resource which already held by other processes [5].

Ti-h (RI and Ti-w Rj) Where

Ti: Transaction i

Ri, Rj: Resources respectively i^{th} and j^{th}

h: resource in hold.

w: resource in wait.

No preemption

A resource can be only when the process has completed (voluntarily) its task.

Circular wait

A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource [8].

Ti→Tj→Tk→Ti

Under deadlock detection technique, the manager allows for the system to occur deadlock. Then apply the detection algorithm to detect that a deadlock has occurred or not, occurred and subsequently it is recovered if there is a deadlock in the system [8].

Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system. After a deadlock is detected, it can be recovered by using any of the two methods:

Process termination

When one or more processes involved in the deadlock maybe aborted or process has been killed. We can choose to abort all processes involved in the deadlock on the basis of utilization of CPU, resources held by the process and etc.

Resource preemption

This is the way to remove the deadlock from the system. So in this way Resources allocated to different processes is preempted and allocated to other processes until the deadlock is broken [5]. There are several variations to these algorithms that seek to optimize different parameters like, number of messages, length of messages, and frequency of detection.

In the proposed algorithm we have taken a number of probe

***Corresponding author:** Shivendra Kumar P, Department of Computer Application and Research, National Institute of Technical Teachers Training and Research, Bhopal, India, Tel: 0755 266 1600; Email: shivendrapandey786@gmail.com

Received March 27, 2015; **Accepted** May 05, 2015; **Published** May 20, 2015

Citation: Shivendra Kumar P, Hari Krishna T, Kapoor RK (2015) Distributed Deadlock Detection Technique with the Finite Automata. J Inform Tech Softw Eng 5: 148. doi:10.4172/2165-7866.1000148

massage comparisons and size of the message as a parameter to optimize the deadlock detection technique. In the paper, we have used the state transitions to find out the deadlock in the distributed system. We also have avoided sending a triple probe massage [4]. The number of massage comparisons with probe message is more and it is difficult to manage the probe massage [6]. In this paper we have shown that with the use of finite automata concept we need not to send the probe message in the outgoing link. In the Chandy Misra Haas model the concept of sending the number of probe massage has been used to find out deadlock and so the number of probe massage and their comparison is high in the AND model. In our proposed algorithm, there is no overhead for sending and waiting for the probe massage for the initiator and there is a very less number of transition it takes to detect the deadlock in the system because with the use of finite automata in the proposed algorithm, this decrease not only the comparisons of massages but also remove the concept of probe massage overhead.

Literature Survey

In the distributed deadlock detection techniques, there are many algorithms in practice. Some widely used algorithms are Obermarck's Path Pushing Algorithm, Chandy-Misra-Haas Edge-Chasing Algorithm

Obermarck's Path-Pushing Algorithm

In the Path Pushing algorithm the information about the global wait for graph is distributed in the form of paths from one site to another site [9]. In this algorithm, there is number of Processes (T1, T2, T3..... TN) and There is a special virtual node Ex. The transaction is totally ordered. The following steps show the process of Obermarck's path-pushing algorithm:

For each site S,

- Construct a wait-for-graph using the transaction-to-transaction wait-for relationships.
- Receive any strings of nodes transmitted from other sites and add them into the wait-for-graph.

a. For each transaction identified in the string, create a node of the TWFG if none exists at this site.

b. For each transaction identified in the string, starting with the first, create an edge to the node representing the next transaction in the string.

- Create wait-for edges from EXTERNAL to each node representing a transaction's agent that is expected to receive on a communication link.
- Create wait-for edges to EXTERNAL from each node representing a transaction's agent that is expected to send on a communication link.
- Analyze the resulting graph, listing all elementary cycles.
- Select a victim to break each cycle that does not contain the node external. As each victim is chosen for a given cycle, remove all cycles that include the victim.

a. Site must remember the transaction identifier of the victim such that it can discard strings received involves the victim.

b. If the victim transaction has an agent at this site, then the fact that the transaction was chosen as a victim must be transmitted to each site known to contain an agent of the victim transaction. Otherwise, the site has to transmit the fact to each site that sends a string containing the victim's identifier to S.

- Examine each remaining cycle that contains the node External. If the transaction identifier of the node External is waiting for is greater than the node that waits for external, then

a. Transform the cycle into a string, which starts with "EX" and terminates with a node identifier that identify the node waiting for External on the site.

b. Send the string to each site which the terminating node in the string is waiting for.

- O (n (n-1)/2) messages
- O (n) message size
- O (n) detect deadlock

Chandy-Misra-Haas Edge-Chasing Algorithm

The scheme proposed by Chandy, Misra and Haas uses local WFGs to detect local deadlocks and probes to determine the existence of global deadlocks [4]. Chandy-Misra-Haas edge-chasing algorithm uses a probe message (i,j,k) deadlock detection initiated for process 'Pi'and send to and is sent by the site of 'Pj' to the site of Pk [4].

Deadlock initiated at pi: If

Pi is locally dependent on itself, then declares a deadlock else

send probe (i, j, k) to home site of Pk for each j, k such that all of the following hold

Pi is locally dependent on PjPj is waiting on Pk

Pj and Pk are on different sites

Receipt of probe (I, j, k) by node of Pk: check the following conditions:

Pk is blocked dependentk(i)=false

(Pk does not yet know that Pi is dependent on Pk)\Pk has not replied to all requests of Pj

if these are all true, do the following

setdependentk(i)=true(Pk now knows that Pi is dependent on Pk) if k=i declare that Pi is deadlocked

else

send probe (i,m,n) to the home site of Pn for every m and n such that the following all hold

Pk is locally dependent on Pm

Pm is waiting on Pn

Pm and Pn are on different sites

Performance of Chandy-Misra-Haas edge-chasing algorithm

- m(n-1)/2 messages for m processes at n sites
- 3-word message length
- O(n) delay to detect

The Proposed Algorithm

In our proposed algorithm, we have devised an approach to detect the deadlock with the help of finite automata. In this approach the process id is taken as an input (sigma) and the process is as a state. With

the help of transition function, we draw the transition table; this helps us to detect the deadlock in the distributed system [10-13]. In the process of transition the unvisited or unexpended vertex/transition is selected to visit and we fully expand (visits its neighbors through directededge) the selected node/transition. And again one of the unvisited nodes from thetransition table is selected. If any node/transition has been visited once, we need not to explore it further. The process of expansions of vertices/transition is repeated until either we get the deadlock in the transition table or all the nodes have been expanded. A flag value has been given to the starting node of the transition so whenever we have fully expanded a node (visited all its neighbours through direct edge), we check the flag value to decide whether there is a deadlock in the system or not. If we do not get the deadlock for the first time, then we take the second node and repeat whole process for this node also. Continue the same process for all available nodes in the distributed system.

This is illustrated with the help of following example: Suppose we have 10 resources (A,B,C,D,E,F,G,H,I,J) and

4 Processes (P0, P1, P2, P3)

Here h denotes resource in hold and w denote wait for the resource. With reference to figure 1 below:

P0—h→A, P0—w→C;

P1—w→A, P1—h→B, P1—w→D;

P2—h→C, P2—w→H, P2—h→J;

P3—w→B, P3—w→F, P3—h→H, P3-h→G;

P4—w→E, P4—f→F;

P5—h→D, P5—h→E, P5—w→I;

P6—h→I, P6—w→G, P6—w→J;

In this proposed algorithm the wait for graph is based on transition input and current state.

Here we have taken the transition input from the resource allocated process, for which the current process is in waiting state (Figure1).

Distributed Deadlock Detection

In this proposed algorithm we have used distributed control system to detect the deadlock where the wait for graph (WFG) is spread over different sites. Any site can initiate the deadlock detection process by constructing a global wait-for graph from local wait-for graphs at a deadlock detector or by a distributed algorithm like edge chasing [14,15]. The control manager will decide when to take the decision to go for detecting the deadlock. It might be based on some threshold value for the CPU utilization, throughput, and/or some other parameter, or when the system is irresponsive. A distributed system is an environment where a number of heterogeneous machines are running parallel and performing the task, so when request are made in the distributed environment, these machines uses mutually exclusive resource so sometimes for a resource it is possible that one process using it and another process for the same resource. So in the distributed system, it is very difficult for the distributed system manager to manage the resource. So while keeping one resource and waiting for the other resource will lead to the wastage of resource and processor utilization too. Some it is also possible where there are more than one process holding a resource and waiting for another which is held by another process and the other process is also waiting for the resource that is held by the first process; under such situation deadlock situation occurs. This type of situation is very common in the distributed environment. We have taken an example of distributed systems; in the example there are four machines that are running in the distributed environment, fashion and processes are requesting the resources from the distributed manager and distributed manager grant them the required resources (Figure 2). In our algorithm we have setup a flag value with the initiator of the transition table. So every time when we fully expand (visit all the outgoing edges) the process we check the flag value .And according to the flag value we decide the deadlock. If there is change in the flag value then it indicate the deadlock and if there is no change in the flag value then check with the others.

Distributed deadlock detection algorithm:

M (Q, δ, Σ)

Q=Set Of Processes;

Q=P1, P2…….Pn;

Σ=Set Of Input Symbol;

Σ=1, 2,…n;

δ=Transition Function

δ: Q × Σ→Q

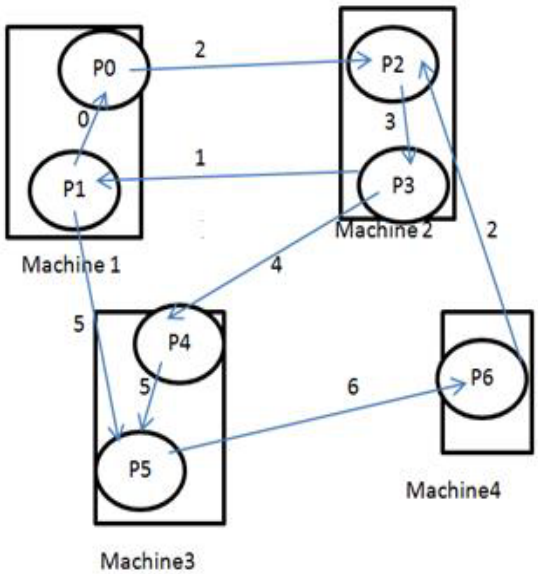

Figure 1: Wait for graph for distributed systems.

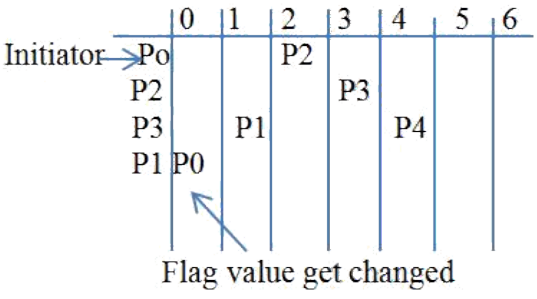

Figure 2: Transition table for process P0.

Step1: Select a process (Qi);

Step2: Initially create a matrix of size 1 × Σ;

Qi × Σ;

Where: Qi is the selected state for deadlock detection;

Initialize: Flag=0: for process Qi;

Step3: Make transitions for Qi,

We have transition function

δ: Q × Σ→Q

δ: Qi × Σ→Q

Repeat step 3 for All Σ for state Qi;

Step 4: Select one of the unvisited process Qk to visit, from the Qi;

Select a state from

δ: Qi × Σ→Qk;

We have transition function:-

δ: Q × Σ→Q

δ: Qk × Σ→Q

For all Σ state Qk;

If (Qk × Σ→Qi)

{Set flag=1; Go to step 6;}

Else if (there is a unvisited process) Go to step 4;

Else go to step 7;

Step 6: report there is deadlock,

Go to step 8;

Step 7: report there is no deadlock,

Go to step 8;

Step 8: exit;

Performance of Proposed Algorithm

The time complexity of our proposed algorithm is based on number of process 'n' in the distributed system. Our proposed algorithm takes less number of comparisons in average case and in best case it takes very less comparisons [13-16].

Analysis of Comparisons to Find Out the Deadlock in the System

For the initiator's transmission there is zero comparison. Otherwise when we select one of the processes to visit and after transmitting all the transitions, we check two things,

1. Is there is change in flag value; if yes then report deadlock
2. Else (go for another comparison) If the flag value is not changed but there is some process to visit then go and select another process and visit it. Else report deadlock

Here for the 1st process no comparison is made and for remaining (n-1) processes it will take 2 comparisons each time,

Thus total number of comparisons is, 0+2(n-1) i.e. 2(n-1)

Analysis of Number of Transmissions to Find Out the Deadlock in the System

First process can take transmission up to (n-1) nodes, and second can take all the transmission (n-2) and third can take all the transmission (n-3) and the n-1st node will take 1 transmission so it will take [13].

(n-1)+ (n-2) + (n-3) +……+ 1=n(n-1)/2 Therefore, total number of transmission are n(n-1)/2

For summary of performance of proposed algorithm, refer table at appendix 1. Since we have not used any message thus message size is zero.

Conclusion

In this paper we have investigated Obermarck's path pushing algorithm, Chandy-Misra-Haas edge-chasing algorithm for distributed deadlock detection techniques [15-16]. We have proposed a deadlock detection algorithm in distributed environment. Our deadlock detection algorithm is totally based on finite automata concept so execution of algorithm is based on the transition function of detection algorithm. By this algorithm, we have avoided the phantom deadlock problem in the distributed environment and so our deadlock detection approach detects only the real deadlocks and reports this. This proposed algorithm is more efficient due to reduced overheads.

References

1. Farajzadeha N, Hashemzadeha M, Mousakhania M, Haghighata AT (2005) An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems. The Fifth International Conference on Computer and Information Technology. Shanghai.
2. Yeung CF, Hung SH , Lam KY, Law CK (1994) A New Distributed Deadlock Detection Algorithm ForDistributed Database Systems. Browse Conference Publications 1: 506-510.
3. Brzezirislti J, Helary JM, Raynal M (1995) Deadlocks in Distributed Systems: Request Models andDefinitions. IEEE 186-193 Cheju Island.
4. Chendy M, Misra J, Haas LM (1983) Distributed Deadlock Detection. ACMTmns. Comput Syst 1: 143-156
5. Holliday JL, Abbadi EA (2000) Distributed Deadlock Detection. University of California at Santa Barbara.
6. Lee S, Lee Y A (1999) Distributed Algorithm for Deadlock Detection under OR-request Model. Conference: Reliable Distributed Systems, IEEE.
7. Obermarck, R (1982) Distributed Deadlock Detection Algorithm ACM Trans.on Database Systems
8. Chen S, Deng Y, Attie P, Sun W (1996) Optimal deadlock detection in distributed systems based on locally constructed wait-for graphs. Proc. Int'l Conf. Distributed Computing Systems. IEEE, pp. 613-619.
9. Knapp E (1987) Deadlock Detestion in Distributed Database. ACM Comp.Sur 19: 302- 328
10. Ajay DK, Singhal M (1997) Distributed Detection of Generalized Deadlocks. IEEE 553-560 Baltimore.
11. Kawazu S, Minami S, Itoh K, Teranaka K (1979) Two-Phase Deadlock Detection Algorithm in Distributed Databases. IEEE 5: 360-367.
12. Kshemkalyani A, Singhal M (1989) Deadlock detection in distributed systems. IEEE Computer 22: 37-48.
13. Lee S (2002) Fast Detection and Resolution of Generalized Distributed Deadlocks. IEEE, pp. 429-436.
14. Lee S (2004) Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model IEEE Trans. On Software Engineering 30: 561-573.
15. Shyam B, Dhamdhere DM (1990) A New PriorityBased Probe Algorithm for Distributed DeadlockDetection," Indian Inst. of Technology, Bombay, Technical Report.
16. Roesler M, Burkhard WA (1989) Resolution of Deadlocks in Object-Oriented Distributed Systems. IEEE Trans. Computers 38: 1212-1224.