International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 3, Issue 2 (Mar-Apr 2015), PP. 76-80

AN EFFECTIVE METHODOLOGY FOR DEADLOCK DETECTION

Mamta Mishra
Azad Institute of Engineering and Technology
mamtamishrait@rediffmail.com

Abstract- In distributed algorithms, the pieces of the algorithm are run concurrently and independently in different processes over the distributed systems. Deadlocks are very thorny to detect in distributed systems. This is so because no site has precise knowledge about the system, and each inter-site communication involves a finite and unpredictable delay. Furthermore in distributed systems, depending on the applications, processes make requests according to different resource model such as Single Resource Model, AND Model, OR Model, AND-OR Model, P out-of Q Model (generalized model) and Unrestricted Model.

I. INTRODUCTION

The available generalized deadlock detection algorithms are grouped into two categories namely Centralized Algorithms and Distributed Algorithms. In Centralized algorithms, the global state of the system is preserved at the single site, whereas in the Distributed algorithms, the information needs to determine a deadlock is maintained across multiple sites. But, both kinds of algorithms have a few precincts. Distributed algorithms have required additional round of messages for resolving deadlocks, whereas Centralized algorithms have suffered with single point of failure, large communicational overhead, and congestion of communication links near the control site and local computational complexity. To overcome those problems, the requirement for novel and better generalized deadlock detection algorithm is apparent Hence, all processes have equal amount of information, and bear equal responsibility to take the final decision. However, a single site needs to have enough memory space and processing power in the centralized algorithms. So, the centralized algorithms are resource intensive. Moreover, the distributed algorithms are more reliable than the centralized algorithms due to the absence of single point of failure. Also, they are easily scalable.

II. SYSTEM MODEL AND PROBLEM CHARACTERIZATION

The system consists of n processes, where each has unique identity. The processes are communicating through a logical communication channel by message passing. There is no shared memory in the system. The messages are delivered at the destination in the same order as sent by the sender, with arbitrary but finite delay. The messages are neither lost nor duplicated and the entire system is fault-free. The events in the system are classified into internal and external events, and they are time stamped using logical clock. They are further classified into computation events and control events. The computation event triggers the computational messages such as DEMAND, RESPONSE, CANCEL and ACKN due to the execution of applications. Whereas, the control event generates the control messages including INVOKE and STATUS as a result of the execution of deadlock detection algorithm.

III. DESCRIPTION OF THE ALGORITHM

Whenever a process i blocks on a pi out-of qi demands, it initiates the deadlock detection algorithm. A process i, called originator, records the consistent snapshot of distributed wait for graph by propagating the INVOKE messages along the outgoing edges in the wait for graph. When the replies are propagated backwards to the originator, the algorithm reduces the snapshot to determine a deadlock. The proposed algorithm follows the method of the algorithm proposed by Kshemkalyani and Singhal (1994) for handling concurrent executions. According to the method, the algorithm assigns a unique priority to each instance based on the originator's identifier, and the time at which it was blocked. It supports the execution of higher priority instance and suspends the execution of lower priority instances in the conflicting processes. Hence, each originator maintains its own snapshot to detect a deadlock. For simplicity, this section describes the single instance execution of proposed algorithm.

A. Explanation of the Algorithm

When process i wants to find out whether it is deadlocked, it sends a INVOKE(i,i) message to all its successors (outi). The first parameter of the INVOKE message is id of the process that propagates the message and the second parameter is the id of the originator. When process j receives the INVOKE message from process i, it performs one of the following actions.

i. If it is the first INVOKE message and process j is blocked, it sets its fatherj to i and sends the INVOKE(j,originator) message to all the processes in outj.
ii. If it has already received a INVOKE message (i.e, fatherj ≠ udef), it includes the id of i in the set inj. It also reduces mj by one (mj = 0 implies that the process j receives the INVOKE message from all its successors).
iii. If process j is active, it sends STATUS(originator, j, true, φ, φ) to process i. The first parameter of the STATUS message is the id of the originator. The second and third parameter represents the id and the state of the process that sends the message respectively. The fourth parameter is the id of the process that would be a victim in case of deadlock and fifth parameter is the number of predecessors of a process victim. Since process j is active, it can not be a victim. Therefore, the fourth and fifth parameter value is set as φ in the message.
iv. If process j receives the INVOKE message through a phantom edge (i.e, i∉INj), it sends a STATUS(i, j, true, φ, φ) message immediately to process i.

Whenever process i receives the STATUS message, it reduces ni (Initally, ni=|OUTi|) by one. Once it receives the STATUS message from all its successors (i.e, ni=0), it evaluates its unblocking condition (fi). If fi is evaluated as true, it sends the STATUS message to its predecessors without changing the victim and |invictim| in the message.

Otherwise, it updates the fourth and fifth parameter of the message by comparing number of its predecessors (|ini|) with |invictim|. If |ini| ≥ |invictim|, it sets process i as victim and sends STATUS(originator, i, false, i, |ini|) to its predecessors. Else, it sends STATUS (originator, i, false, victim, |invictim|) to its predecessors.

In some cases, process i is waiting to receive the STATUS message from its own predecessor j in response to its INVOKE message (i.e, when j∈ini ∧ j∈outi) for determining its state. In such cases, process i can not send the STATUS message to its predecessors including the process j. This problem is resolved as follows. When process i receives the STATUS message, it reduces ni by one. In addition, it counts the number of processes that act as both predecessor and successor (loop). Therefore, it attempts to simplify its unblocking condition (fi) at the time it has received ni–loop STATUS messages. It then sends STATUS message to its predecessors. This will ensure that any process that is reachable from the originator does not wait indefinitely to determine its state.

After receiving the STATUS message from all its immediate successors, the originator evaluates its unblocking condition. If the unblocking condition of the originator is not evaluated as true, the algorithm declares deadlock. In that case, the originator sends ABANDON message to a process victim directly to resolve it.

B. Example Execution

This section illustrates the working principle of proposed algorithm with the help of an example shown in Figure 1. Let us consider the distributed wait for graph that spans six processes labeled P1 to P6. Assume that, process P1 initiates the deadlock detection algorithm and the messages are propagated in such a way to induce a Breadth First Search (BFS) spanning Tree. All the processes except P6 are blocked. The unblocking conditions of all blocked processes are given as follows: F1 = (P2 ∧ P3), F2 = (P4 ∧ P5)∨ P6, F3 = P5, F4 = P5∨ P6 and F5 = P3∧ P6.

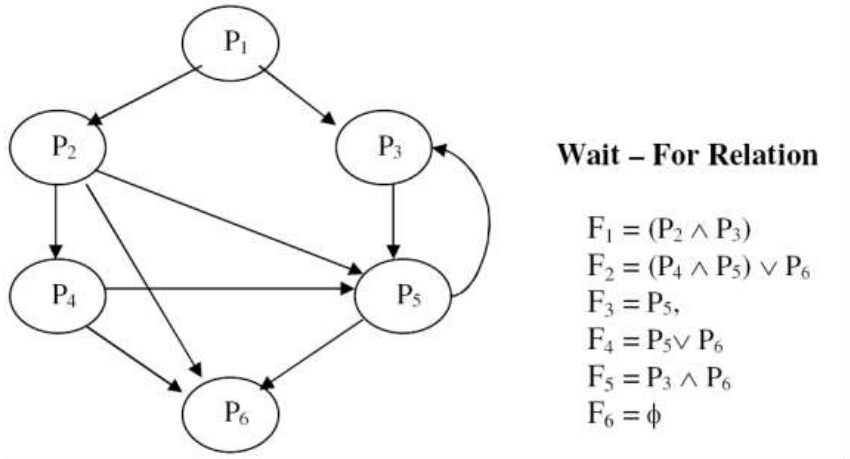

Figure 1 The Distributed wait for graph

Figure 2 shows the Directed Spanning Tree induced by the execution of the proposed algorithm where solid lines represent the tree edges and dotted lines represent non-tree edges.

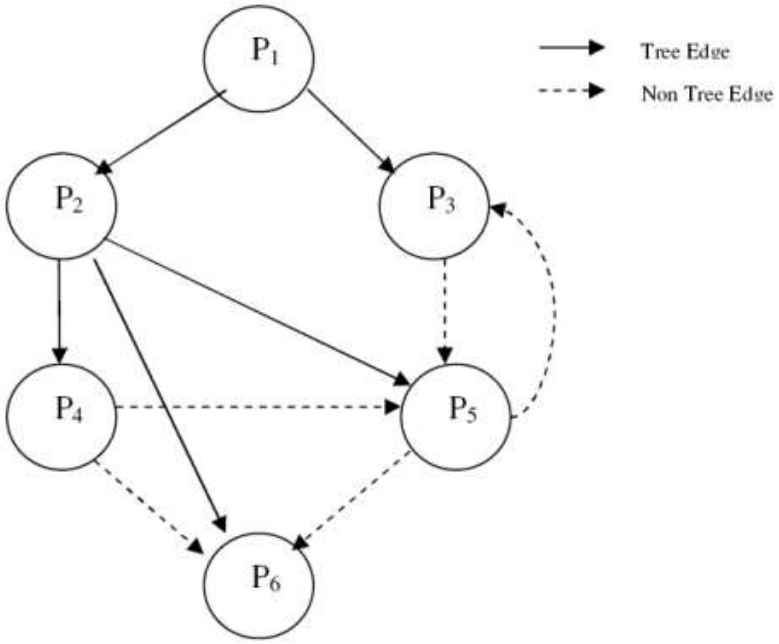

Figure 2 The Directed Spanning Tree induced by SDRA

When the originator invokes the algorithm, it diffuses the INVOKE messages across the wait for graph. Figure 3 shows the propagation of INVOKE message in the distributed wait for graph by the algorithm

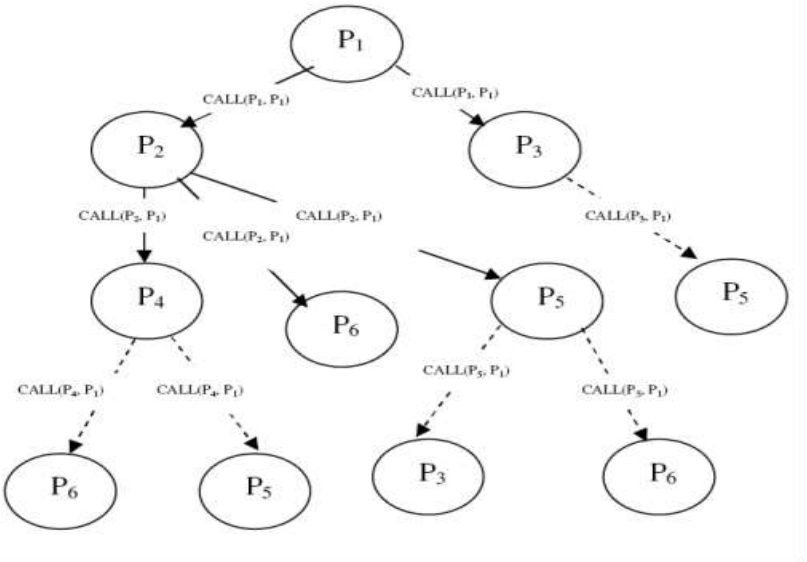

Figure 3 Propagation of INVOKE messages in SDRA

The stepwise execution of SDRA algorithm is given below. Assume that
Process P1 initiates the execution.

1. Process P1 sends the INVOKE(P1, P1) message to processes P2 and P3 respectively.
2. When process P2 receives the INVOKE message from a process P1, it propagates the INVOKE(P2, P1) message to processes P4, P5 and P6 respectively.
3. When process P3 receives the INVOKE message from a process P1, it sends INVOKE(P3, P1) message to P5.
4. When process P4 receives the INVOKE message from a process P2, it sends INVOKE(P4, P1) message to its successors P5 and P6 respectively.
5. When process P5 receives the INVOKE message from process P2, it sends INVOKE(P5, P1) message to its own successors P3 and P6 respectively.
6. When an active process P6 receives the INVOKE message from a process P2, it sends STATUS(P1, P6,true, φ, φ) to process P2.
7. When process P5 receives the INVOKE message from a process P3 through a non-tree edge, it does not propagate the INVOKE message.
8. When an active process P6 receives the INVOKE message from a process P4, it sends STATUS(P1, P6,true, φ, φ) to process P4.
9. When process P3 receives the INVOKE message from a process P5 through a non-tree edge, it does not respond to the message.
10. When an active process P6 receives the INVOKE message from a process P5, it sends STATUS(P1, P6,true, φ, φ) to process P5.
11. When process P5 receives the INVOKE message from process P6, it sends STATUS(P1, P5, false, P5, 2) to processes P2, P3 and P4 respectively.
12. When process P2 receives the STATUS message from a process P5, it waits for the arrival of STATUS message from its successor P4.
13. When process P3 receives the STATUS message from a process P5, it sends STATUS(P1, P3, false, P3, 2) to process P1.
14. When process P2 receives the STATUS message from a process P4, it sends STATUS(P1, P2, true, P5, 2) to process P1.

IV. PERFORMANCE ANALYSIS SDRA

This section compares the performance of proposed algorithm in Chapter 3, with that of Bracha and Toueg's algorithm and Kshemkalyani and Singhal's algorithm (1999). The reason behind the selection of those two particular algorithms is that the process's data structure of those two algorithms is same as in SDRA. Also, all the algorithms have detected the generalized deadlock in a distributed manner.

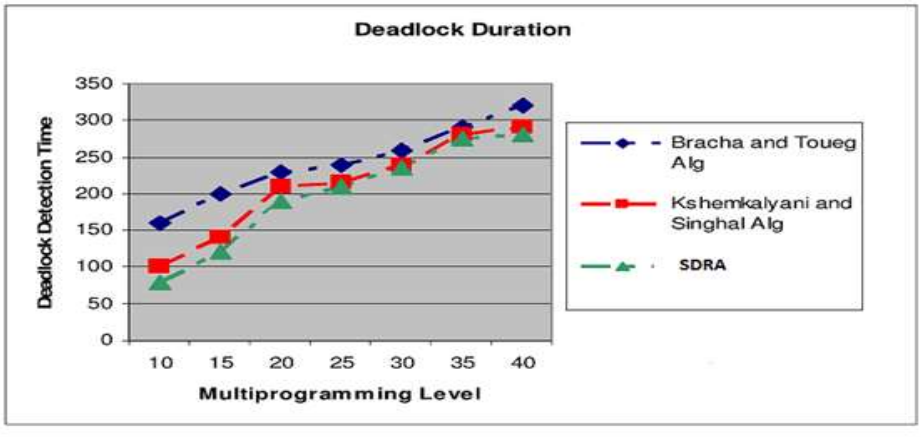

Figure 4 Comparison of Deadlock Duration of three Decentralized algorithms

Figure 4.1 shows the deadlock duration plotted as a function of the MPL of the system. As shown in the Figure, mean deadlock detection duration resulting from SDRA is less than that from Bracha and Toueg's algorithm. It is observed that deadlock duration of Kshemkalyani and Singhal's algorithm and SDRA is almost same for higher MPL values, which is consistent with complexity comparison presented in Table 1. It is also observed that the deadlock detection duration increases with MPL until the number of processes reaches 30, and then tapers to flat. The reason behind this is due to the increase of simply blocked nodes with MPL.

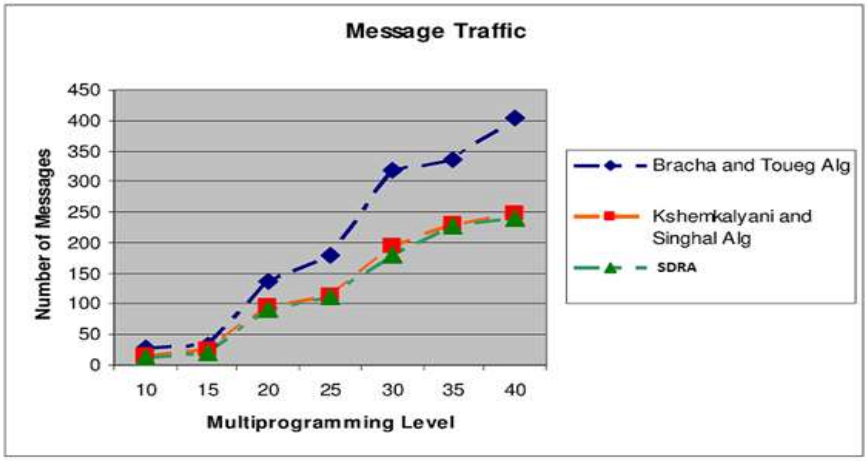

Figure 5 Comparison of Message traffic of three Decentralized algorithms

Figure 5 shows the mean number of deadlock detection messages generated per algorithm execution with varying multiprogramming levels. As shown in the Figure, Bracha and Toueg's algorithm passes 1.5 times more messages than SDRA for higher MPL values. It is observed that SDRA and Kshemkalyani and Singhal's algorithm have required almost same number of messages to detect deadlocks according to the congruence with the theoretical expectation as in Table 1

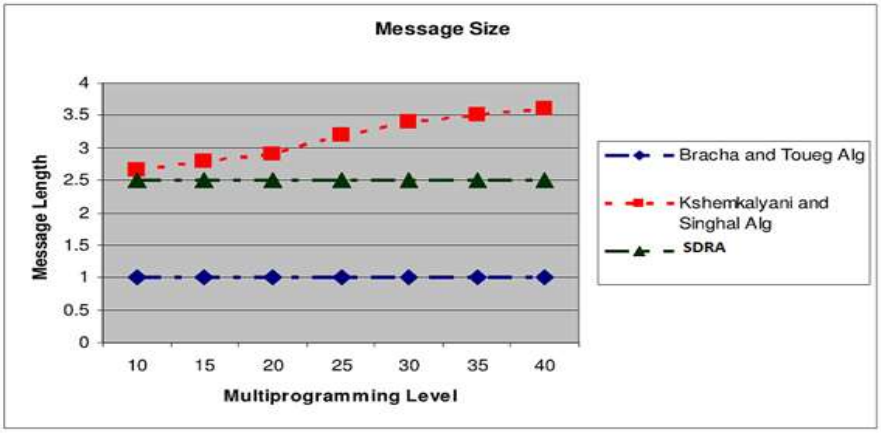

Figure 6 Comparison of Message Size of three Decentralized algorithms

Figure 6 shows the mean length of deadlock detection messages in terms of number of node identifiers for each algorithm. It is observed that, as the MPL is increased in the system, the message length of Kshemkalyani and Singhal's algorithm is also increased. However, the message length of proposed and Bracha and Toueg's algorithm is a constant. If the system is in deadlock, the Bracha and Toueg's algorithm aborts the initiator to resolve a deadlock. However, it might not resolve all deadlocks reachable from the initiator. On the contrary, the Kshemkalyani and Singhal's algorithm selects a victim by invoking additional procedure as the centralized algorithms. Since the initiator of proposed algorithm identifies an appropriate victim without invoking any additional procedure, the deadlock resolution time is very less in proposed algorithm as compared to the Kshemkalyani and Singhal's algorithm. It is observed that a deadlocked process having highest predecessor is aborted and it is more likely that abortion of such a process might resolve a deadlock.

Algorithms Comparison Factor	**Wang.et al's algorithm (1990)**	**Kshemkalyani and Singhal's algorithm (1994)**	**Kshemkalyani and Singhal's algorithm (1999)**	**SDRA**
Deadlock Duration	$3d+1$	$2d$	$2d+2$	$2d$
Message Complexity	$6e$	$4e-2n+2l$	$2e$	$2e$
Message Size	O(1)	O(1)	O(e)	O(1)
Deadlock Resolution	No Scheme	e Messages	No Scheme	1 Message

Table 1 Performance Comparison of Distributed algorithms for Detecting Generalized Deadlocks

V. CONCLUSION

A new distributed deadlock detection algorithm namely SDRA is presented. In this algorithm, the probes are propagated along through the edges of wait for graph and the replies are sent backwards towards the originator. The reducibility of a blocked process is decided once it has received the STATUS messages from all its descendants. If the originator is not reduced at the end of termination, the algorithm declares a deadlock. The correctness of proposed SDRA is formally proved. It is shown that the message complexity of 2e and time complexity of 2d is better or equal to the existing algorithms. The notable improvement of this algorithm is that it significantly reduces the message length without using any explicit techniques. Also, it victimizes a single process to resolve deadlock and eliminates the message overhead associated with deadlock resolution as compared to the existing distributed algorithms. However, the proposed SDRA requires 2d time units to detect deadlocks like the existing algorithms as opposed to centralized generalized deadlock detection algorithms.

REFERENCES

[1] Ashfield, B., Deugo, D., Oppacher, F. and White, T. "Distributed Deadlock Detection in Mobile Agent Systems", Proceedings of the International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 146-156, 2002.

[2] Bhalla, S. and Hasegawa, M. "Automatic Detection of Multi-Level Deadlocks in Distributed Transaction Management Systems", Proceedings of the International Conference on Parallel Processing Workshops, pp. 297-304, 2003.

[3] Bukhres, O., "Performance Comparison of Distributed Deadlock Detection Algorithms", Proceedings of IEEE Eighth International Conference on Data Engineering, pp. 210-217, 1992.

[4] Cao, J., Zhou, J., Zhu, W., Chen, D. And Lu, J."A Mobile Agent Enabled Approach for Distributed Deadlock Detection", Proceedings of the Grid and Cooperative Computing, pp. 535-542, 2004.

[5] Cheng, X., Yang, X. and Jin, F. "An Agent-Based Deadlock Detection/Resolution Algorithm for the AND Model", Proceedings og the Sixth Conference on Parallel and Distributed Computing Applications and Technologies, pp. 759-761, 2005.

[6] Geetha, V. and Sreenath, N. "Fault Informant Distributed Deadlock Detection Using Colored Probes', Proceedings of the Computer Networks and Information Technologies, Communications in Computer and Information Science, Vol. 142, No. 1, pp.128-132, 2011.

[7] Hashemzadeh, M., Farajzadeh, N. and Haghighat, A.T. "Optimal detection and resolution of distributed deadlocks in the generalized model", Proceedings of the Fourteen Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 133-136, 2006.