**IOSR Journal of Engineering (IOSRJEN)** **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 06, Issue 04 (April. 2016), ||V2|| PP 01-05

# A Modified Algorithm for Distributed Deadlock Detection in Generalized Model

Sakshi Surve[1], Sharad Bhatt[2]
[1](University of Mumbai) geetams24@rediffmail.com)
[2](University of Mumbai) Bhatt.sh2@gmail.com)

***Abstract: -*** In a distributed system, a deadlock may occur when set of processes wait for resources from each other. A process involved in a deadlock waits indefinitely unless a special action is taken. Deadlock leaves the system into a blocking state with no process getting complete and also it reduces the throughput. In this paper a technique is presented that will improve the performance of Srinivasan distributed deadlock algorithm for multiple executions. In Srinivasan algorithm whenever multiple initiators invoke the algorithm one after the other same deadlock cycle and similar message transfers may be reported in more than one execution. So, in order to handle multiple executions an algorithm has been proposed which will not only reduce the number of message transfers but will also reduce the deadlock detection time. The proposed algorithm gives priority to different algorithm execution depending on their process id's along with this it allows lower priority executions to continue so that a deadlock that are not directly reachable from the higher priority execution could be detected.

***Keywords: -*** *Distributed Deadlock, Generalized Model, Wait-for Graph*

## I. INTRODUCTION

In a distributed computing environment remote resources are needed by processes for their computation. Processes sends request message to remote sites, when remote resources are needed. Depending on the availability or unavailability of the resources, the processes are either given the requesting resources or are made to wait indefinitely. This leads to deadlock in distributed systems. A deadlock is defined as a time-dependent state in which a set of processes are waiting for resources from other processes in the same set indefinitely. As deadlock leaves the system into a blocking state with no transaction getting complete; therefore, they must be detected.

In distributed systems many resource request model exists and depending on these models deadlock detection techniques also varies. In AND [2] model, a process remains blocked until all requested resources are granted. The existence of cycle in the wait-for graph (WFG) [2, 3] is a necessary and sufficient condition to detect deadlock. In OR model [2], a process remain blocked until it is not able to acquire any of the requested resource. The existence of knot in the WFG is a necessary and sufficient condition to determine a deadlock. In P out-of Q model [2], also known as generalized model, a process makes Q resource requests, and gets unblocked only when any P resources are granted. Neither cycle nor knot is a necessary and sufficient condition to detect a deadlock. So, the generalized deadlock can be detected by examining the presence of some complex topology in the WFG.

This paper proposes a method that will improve the performance of Srinivasan [1] distributed deadlock algorithm. In Srinivasan algorithm whenever multiple initiators invoke the algorithm one after the other same deadlock cycle and similar message transfers may be reported in more than one execution. So, in order to handle multiple executions an algorithm has been proposed which will not only reduce the number of message transfers but will also reduce the deadlock detection time. The proposed algorithm gives priority to different algorithm execution depending on their process id's along with this it allows lower priority executions to continue so that a deadlock that are not directly reachable from the higher priority execution could be detected. The proposed algorithm will be evaluated and compared with Srinivasan algorithm on parameters like deadlock detection time and number of message transfer for both single and multiple executions.

## II. RELATED WORK

Existing algorithms for the distributed deadlock detection in generalized model can be classified into two kinds:

- Distributed
- Centralized.

In G. Bracha and S.Toueg [8] probes are forwarded along the edges of the WFG by the initiator, and the replies which represent granting of the requests are propagated backward to determine deadlocks. It uses 4*e* messages and 4*d* time units to detect deadlocks, where *e* and *d* refers to the number of edges and the diameter of

the WFG respectively. The algorithm in [9] uses a token to detect deadlock with $n^2/2$ messages in 4*n* time units. In Kshemkalyani's algorithm [6], each node maintains the information required for deadlock detection. The information at each node is updated, whenever reply to a probe is propagated back to the initiator. This algorithm uses 2*e* messages and has a time complexity of 2*d* + 2 time units in worst case. The algorithm in [4] constructs a distributed spanning tree (DST) through propagating probe messages along the edges of the WFG. It then reduces the DST when reply to each probe message is received by the initiator. It uses less than 2e messages has a time complexity of 2d time units in worst case.

For a centralized algorithm, only the initiator collects and reduces the wait-for information instead of distributed reduction. In Chen's algorithm [10], the initiator constructs an "image" of the WFG incrementally to determine a deadlock. It only spends 2*n* messages and 2*d* time units. Instead of having the initiator control the algorithm as in [10], the algorithm in [7] makes each node maintain the information required for deadlock detection. As the reply to a probe is propagated backward to the initiator, the information is updated at each node and carried by the reply. The algorithm spends 2e messages in 2d time units with O(e) message size. Different from Chen's algorithm, the initiator in Lee's algorithm [7] receives replies from leaf nodes only, and it needs less than 2*e* messages and only *d* + 2 time units. In the algorithm [1], the initiator performs reduction once it receives a reply and it terminates the execution when it detects and resolves a deadlock. This algorithm uses messages of length e+2n has a time complexity of *d* + 2 time units in worst case.

## III. PROPOSED METHOD

As mentioned earlier, the proposed work is to improve the performance of Srinivasan algorithm, by adding the functionality of handling more than one instance of algorithm. For single execution the proposed method works similar to that of Srinivasan but for handling multiple execution it uses the concept of priority, as in [5], [6].

### 3.1 Description for Single Execution

This section describes the Srinivasan algorithm.

The initiator of this algorithm, say process i, constructs a directed spanning tree by propagating CAL messages to each of its successor which includes its unblocking function ($R_i$). If this is the first CAL message that a process j receives, it becomes a child of sender and sends a REPORT message that carries its unblocking function ($R_j$) directly to the initiator. Along with this process j sends CAL message to its own successor. While if this is not the first CAL message that a process received, a RESPONSE message is sent to the initiator by the process j only after it receives CAL message from all its predecessors. The unblocking condition ($R_i$) of process i on resource is expressed as a predicate using AND and OR operator. For example $R_i = X \wedge (Y \vee Z)$ denotes that resource required by process 'i' is either (X and Y) or (X and Z). Whenever a blocked process 'i' sends a REPORT message to the initiator, it gets included in the initiator's UNBLOCKING SET. In the other case if an active process 'i' sends a REPORT message, it gets included in initiator's ACTIVE SET and all unblocking function in its UNBLOCKING SET are evaluated in following manner: Select an unblocking condition from the UNBLOCKING SET and checks whether process in ACTIVE SET are sufficient to make Fi as true. Transfer that process from UNBLOCKING SET to ACTIVE SET. Repeat that for all unblocking condition. Finally, all processes remaining in the UNBLOCKING SET are declared as deadlock process.

### 3.2 Algorithm Specification

A formal description of algorithm [1] executed at process 'i' is given below.

**Data structure of a process 'i'**: (Initial values are inside the parenthesis).

- $p_i$: the process from which first CAL has been received (NULL).
- $W_i$: process 'i' weight value (0).
- $in_i$: the set of processors which are predecessor of 'i'($IN_i$).
- $out_i$: the set of processors which are successor of 'i'($OUT_i$).
- Ri: the condition for process 'i' to be active (Ri).
- $n\_pd_i$: the number of predecessor of 'i' ($|IN_i|$).
- $n\_sc_i$: the number of successor of 'i' ($|OUT_i|$).

Additional datastructure at an initiator

- $UF_{init}$: a set of unblocking functions of the form $<i, R_i, n\_pd_i>$ ($\phi$).
- $AC_{init}$: the set of active processes ($\phi$).
- $W_{init}$: the accumulated weight value (0).

**Message Formats:** (w indicates the weight value)

- CAL(init, j, w): A call sent by j. init represents the initiator of the algorithm.
- REPORT(j, $R_j$, $n\_pd_j$): sent by process j as a reply to first CALL message.
- RESPONSE(j, $w_j$): sent by process 'j' .

I. When process 'i' is the initiator of the algorithm
$init_i$ =i;
$p_i$ =i;
$UF_{init} = UF_{init} \cup \{(i, Ri, n\_pd_i)\}$;
send CAL(init, i, $w_i/n\_sc_i$) to each k ε $out_i$

II. When process k receives CAL(init, i, w/ $n\_sc_i$) from process i:
$n\_pd_k$--;
**if** ($parent_k$= NULL and k ε $in_i$) **then**
$p_i$=k;
$init_i$=i;
send REPORT(i, Ri, $n\_pd_i$) to $init_i$

II.1 **if**($n\_sc_i$ > 0) then
send CAL(init, i, $w_k$/ $n\_sc_i$) ∀k ε $out_i$
**else**

II.1.2 send RESPONSE(i, $w_k$) to $init_i$

II.2 **else if** (p≠NULL AND kε$in_i$) **then**

II.2.1 **if**(i= initiator) **then**
$w_{init}=w_{init}+w_k$;

II.2.2 **else if**($n\_pd_i$=0) **then**
Send RESPONSE(i, $w_i$) to $init_i$
**else**
$W_i=w_i+w_k$;

III. When process k receives REPORT(i, $R_i$, $n\_pd_i$) from process i:
**if**(Ri=ϕ) **then**
$AC_{init}=AC_{init}\cup\{i\}$;
computation();
**else**
$UF_{init}=UF_{init} \cup \{(i, R_i, n\_pd_i)\}$;

IV. When process k receives RESPONSE(i, $w_i$) from process i:
$W_{init}=w_{init} + w_i$;

V. procedure computation()
**for each** i ε UFinit **do**
**begin**
**if**(compute(i, $R_i$) = true) **then**
$AC_{init} = AC_{init} \cup \{i\}$;
$UF_{init} = UF_{init} - \{i, R_i, n\_pd_i\}$;
**end for**

VI. procedure resolve()
**if** ($UF_{init}$ = ϕ) **then**
**No Deadlock ; exit;**
**else**
**for each** i ε UFinit **do**
**begin**
Print Deadlocked Nodes;
**end for**
**end else**

### 3.3 Example Execution

An algorithm execution is illustrated in Fig. 1. The blocking conditions are $F_1= (2\vee3)\wedge4$, $F_2=1$, $F_4=1$. In DST shown in Fig. 1(b), tree and non-tree edges are pictured with solid and dashed arrows respectively. We assume the following scenario of message propagation:

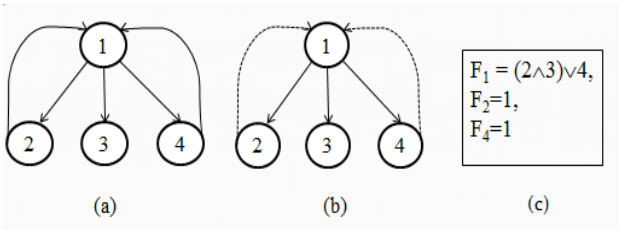


Figure 1: An illustration of the algorithm execution. (a) The Wait for Graph. (b) The DST built by the algorithm. (c) Requesting condition of each process.

1. Process 1 initiates the algorithm and sends CAL(1,1,1/3) to process 2,3,4.
2. When process 2 receives CAL from process 1, it sends REPORT(2,$F_2$,0) to 1 and CAL(1,2,1/3) to 1.
3. When process 3 receives CAL from process 1, it sends REPORT(3,$F_3$,0) and RESPONSE(1,0) to 1.
4. When Process 4 receives CAL from process 1, it sends REPORT(4,$F_4$,0) and CAL(4,2,1/3) to 1.

The unblocking condition collected at the initiator is shown in Fig. 1(c). When the initiator receives re REPORT from process 3, the unblocking functions in the set UFinit are reduced by it and process 1, 2, 4 are declared deadlocked.

### 3.4 Description for Multiple Executions

In this section, the proposed strategy for handling more than one executions of the Srinivasan algorithm is presented. In srinivasan algorithm, whenever multiple initiators invoke the algorithm one after the other same deadlock cycle and similar message transfers may be reported in more than one execution. In order to overcome

this drawback, an algorithm has been proposed that will reduce the deadlock detection time and number of message transfers for multiple executions.

Although the proposed algorithm has some similarity with [5, 6, and 7] but it differs from them in following ways:

- Every algorithm execution is given a priority based on initiator id, whereas in [5, 6 and 7] priority is assigned based on sequence number, local time and initiator id.
- In the proposed algorithm processes involved in higher priority executions do not abort the executions of lower priority initiators, whereas in [5, 6] higher priority executions abort lower priority.

### 3.4 Algorithm

The strategy given in section 3.3 is formally described in the following. For simplicity only activities additional to section 3.2 are presented. Each message used in the proposed algorithm will have an additional parameter for priority denoted as msg_pr.

Additional data structure at Process i:

- $curr\_pr_i$: the priority of the algorithm execution in which process i is involved. The process which initiates the algorithm will set this priority depending on its process id and forwards it through CALL message.

(P.1) when process i receives CAL(msg_pr, init, j,w):
(P.1.1) **if** msg_pr < $curr\_pr_i$ **then**
send REPORT(msg_pr, j, $R_j$, $n\_pd_j$) to init.
(P.1.2) **else if** msg_pr > $curr\_pr_i$ or $curr\_pr_i$=0 **then**
**begin**
$curr\_pr_i$=msg_pr;
Execute Step II in 3.2;
**end else**
(P.2) When initiator receives REPORT(msg_pr, j, $R_j$, $n\_pd_j$):
(P.2.1) **if** msg_pr < $curr\_pr_i$ **then**
Discard the message;
(P.2.2) **else if** msg_pr > $curr\_pr_i$ **then** ;
(P.2.3) **else** Execute step III in 3.2;
(P.3) When initiator receives RESPONSE(msg_pr, j, $w_j$):
(P.3.1) **if** msg_pr < $curr\_pr_i$ **then**
Discard the message;
(P.3.2) **else if** msg_pr > $curr\_pr_i$ **then** ;
(P.3.3) **else** Execute step IV in 3.2;

## IV. CONCLUSION

In this paper we presented a technique that will improve the performance of srinivasan algorithm for handling more than one algorithm execution. The proposed technique uses the concept of priority to handle more than one algorithm execution. Proposed method will perform better in deadlock detection time and number of message transfers than srinivasan algorithm. However, for single execution it performs similar to srinivasan algorithm. The proposed algorithm can be used to detect deadlocks in different areas like management of resource in operating system, network communication and transactions management in distributed database.

## REFERENCES

[1] S. Srinivasan and R. Rajaram, "An improved, centralized algorithm for detection and resolution of distributed deadlock in the generalized model", *International Journal of Parallel, Emergent and Distributed Systems, vol. 27, no. 3, pp. 205-224, 2012*

[2] E. Knapp, "Deadlock detection in distributed database systems," *ACM Comput, Surv, 19(4), (1987), pp. 303-327.*

[3] M. Singhal, Deadlock detection in distributed systems, *IEEE Comput,22(1989), pp. 37-48.*

[4] S. Srinivasan and R. Rajaram, "A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems," *J. Distributed Parallel Databases 29(4) (2011), pp. 261-276.*

[5] A. D. Kshemkalyani and M. Singhal, "Efficient Detection and Resolution of Generalized Distributed Deadlocks," *IEEE Trans. Software Eng., vol. 20, no. 1, pp. 43-54, Jan. 1994.*

[6] A. D. Kshemkalyani and M. Singhal, "A One-Phase Agorithm to Detect Distributed Deadlocks in Replicated Databases," *IEEE Trans. Knowledge and Data Eng., vol. 11, no. 6, pp. 880-895, Nov. 1999.*

[7] S. Lee, "Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model," *IEEE Trans. Software Eng., vol. 30, no. 8, pp.561-573, Sept. 2004.*

[8] G. Bracha and S. Toueg, "A Distributed algorithm for generalized deadlock detection," *Distributed Computing, 2 (1987), pp. 127-138.*

[9] J. Wang, S. Huang and N. Chen, "*Distributed algorithm for detecting generalized deadlocks,"* Tech. Report, Department of Computer Science, National Tsing-Hua University (1990).

[10] S. Lee, "Efficient generalized deadlock detection and resolution in distributed systems," *Proceedings of the 21st International Conference on Distributed Computing Systems (2001), pp. 47-54.*