VGS Algorithm - an Efficient Deadlock Resolution Method

Kunwar Singh Vaisla
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), India

Menka Goswami
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), India

Ajit Singh
Deptt. of Comp. Sc. & Engg.
BT KIT, Dwarahat-263656
Dist – Almora (UK), India

ABSTRACT
The occurrence of deadlocks should be controlled effectively by their detection and resolution, but may sometimes lead to a serious system failure. After implying an efficient detection algorithm the deadlock is resolved by a deadlock resolution algorithm whose primary step is to either select the victim then to abort the victim transaction or cause it to rollback. This step resolves deadlock but is not efficient one. This paper proposes a new deadlock resolution algorithm which doesn't cause any aborts /roll backs in fact it is based on the mutual cooperation of transactions and a random number representing time duration for which the process holding the resource will be suspended.

Keywords
Deadlock, WFG, Transactions, Resources

1. INTRODUCTION
A deadlock occurs when there is a set of processes waiting for resource held by other processes in the same set.The processes in deadlock wait indefinitely for the resources and never terminates their executions and the resources they hold are not available to any other process [21]. A deadlock lowers the system utilization and hinders the progress of processes. Also the presence of deadlocks affects the throughput of the system. The dependency relationship among processes with respect to resources in a distributed system is often represented by a directed graph, known as the Wait for Graph (WFG) [13]. In the WFG each node represents a process and an arc is originated from a process waiting for a resource to a process holding the resource.

In a distributed system, a deadlock occurs when there is a set of processes and each process in the set waits indefinitely for the resources from each other. Therefore it is quite essential that a fast deadlock detection and resolution mechanism is applied otherwise the processes involved in the deadlock will wait indefinitely and will lower the system utilization and hinders the progress of processes. [6].

A deadlock needs to be resolved timely because if not resolved, the deadlock size will increase with the deadlock persistence time as more processes will be trapped in the deadlock where a deadlock size is defined as the total number of blocked processes (BP) involved in deadlock, where BP is the process that waits indefinitely on other processes. [1]

Because of deadlock none of the any processes involved can make any progress without obtaining the resources for which they are waiting.

A deadlock has an adverse performance effect that offsets the advantage of resource sharing and processing concurrency.

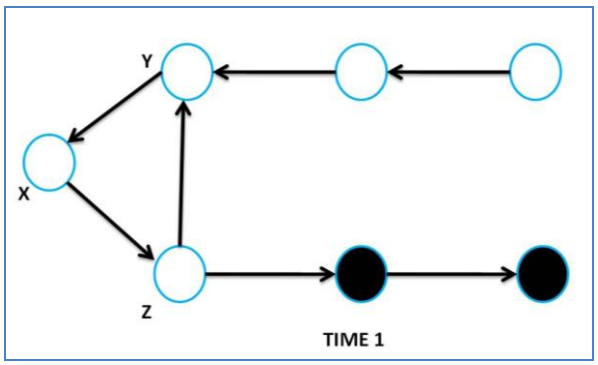

Figure 1: A few processes in deadlock, referred from [1].

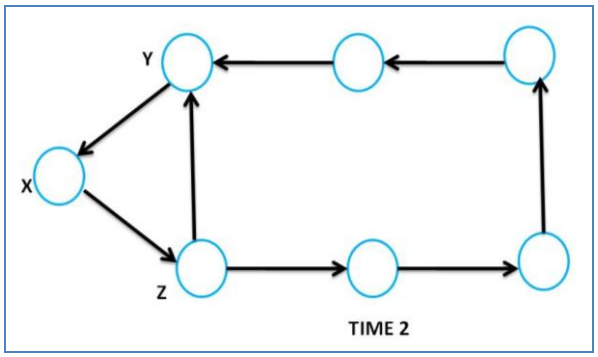

Figure 2: Increasing deadlock size as more processes trapped in deadlock, referred from [1].

Because distributed systems are vulnerable to deadlocks, the problems of deadlock detection and resolution have long been considered important problem in such systems. Several models have been proposed for the processes operating in distributed system. As per the AND model, a process sits idle until all of the requested resources are acquired. In the OR model, a process resumes execution if any of the requested resources is granted. In the P-out-of-Q model also known as the generalized model, a process makes Q resource requests and remains blocked until it obtains any P resources. A generalized model is found in many domains such as resource allocation in distributed operating systems [2] and communicating processes [3].

A deadlock is defined differently depending on the underlying model. Since a process becomes blocked if any of its resource requests is not granted, a deadlock in the AND model corresponds to a cycle in the WFG. In the OR model, the presence of a knot in the graph implies a deadlock [4].In the generalized model a deadlock involves a more complex topology in the WFG. A cycle is a necessary but not sufficient condition for deadlock in this model [5].

2. RELATED WORK:

The deadlock detection and resolution algorithm always require that transactions should be aborted .For this reason several issues must be carefully considered.

1) Aborts are more expensive than waits.
2) Unnecessary aborts result in wasted system resources.
3) Optimal concurrency requires that the number of aborted transactions be minimized.[6]

These factors must be considered so that the transaction being aborted will have the least impact on system performance and throughput. Deadlock prevention, deadlock avoidance, deadlock detection and finally resolution are the common strategies for handling deadlocks. It has been noted that both the deadlock prevention and deadlock avoidance strategies are conservative and less feasible in handling the deadlock problem in general, whereas the deadlock detection/resolution strategy is widely accepted as an optimistic and feasible solution to the deadlock problem because of its exclusion of the unrealistic assumption about resource allocation requirements of the process [7].

Basically the deadlocks present in a system are detected by a periodic initiation of an effective deadlock detection algorithm and then resolved by a deadlock resolution algorithm and it is always tried that the resolution algorithm used does not cause any unnecessary aborts / roll backs.

The appropriate scheme for handling deadlocks in distributed systems is detection and resolution. A typical method to resolve deadlock is to select a proper victim. The victim is to abort itself for deadlock resolution.

The primary issue of deadlock resolution [8], [9], [10] is to selectively abort a subset of processes involved in the deadlock so as to minimize the overall abortion cost [11], [12]. This is often referred to as the minimal abort set problem. The victim (aborted) processes need to cancel all pending requests and releases all acquired resources so that false deadlocks detection and resolution could be avoided. [12], [13].

To further reduce the abortion cost, check pointing is sometimes introduced to prevent the victim processes from being rolled back from scratch [14].

Usually, the deadlocks are resolved by aborting deadlocked processes. Therefore, two facts have to be considered when analyzing the cost associated to deadlock resolution algorithms: the cost of detecting a deadlock and the time that the aborted processes have wasted [15, 16]. Deadlock situations when detected should be resolved as soon as possible but ensuring a minimum number of abortions and only those processes should be aborted which has been selected as victim. Thus, algorithms (*safe-resolution algorithms*) verifying the safety correctness criterion of resolving only true deadlocks should be designed [17].

Whenever multiple transactions are in a waiting state the probability of deadlock occurrence increases. For this reason, restart methods of concurrency control appear more attractive; however, restarting global transactions is more expensive than waiting [6].

Chen et al's algorithm [18], Mendivil's et al algorithm [19] all believe in aborting transactions. In Chen et al's algorithm an optimal set of victim processes is identified for abort with the properly selected abortion cost to avoid starvation and live lock problem. In Mendivil's algorithm a process with the lowest priority is aborted.

In [17] deadlock resolution has been considered for OR request models. In this action *abort i* is executed when candidate node *i* has received n informed probe from each node it had sent a notify probe. In such a moment, no other node of the system has information about i, so its abortion will not cause a posterior false deadlock resolution. Basically, a node decides abort itself based on local information.

In [20] a history based deadlock detection and resolution (DD&R algorithm) for the SR model is proposed. In this algorithm victim is not defined a 'priori' when a cycle is formed i.e. the lowest priority process instead the victim is dynamically calculated. It resolves deadlock by aborting node (only processes).

In [21] a deadlock detection and resolution algorithm has been proposed. According to thus algorithm if a deadlock exists then an algorithm is applied which reduces the connect edges from the system. At the end of algorithm no connect edges are there in the system and therefore no deadlock in the system.

[22] Resolves deadlock with the help of use of random number. It also helps in minimizing the chance of detecting phantom deadlocks

Study of several authors [17, 5, 20] reveals that the primary step of each deadlock resolution algorithm is to select a victim and then to abort it. Although abortion will resolve deadlock but it will cause the transaction to start from beginning and again struggle for all the resources which it require therefore abortion or rollback is not a good choice.

If we go by the literature review then it can be observed that most of the algorithms reviewed above are safe deadlock resolution algorithms and all of them chose to abort or rollback the victim node.

A deadlock resolution algorithm in distributed systems is correct if it satisfies the following two criteria:
1. Liveness: If a deadlock is present in the system, it should be resolved by the algorithm in finite time.
2. Safety: If the algorithm detects and resolves a deadlock, the deadlock is present in the system and there is no other algorithm instance that resolves the same deadlock. [24]

3. PROBLEM SPECIFICATION:

In fact most of the deadlock detection algorithms in literature are safe detection algorithms and they are considered correct because they detect in finite time, all deadlock of the system and do not detect false deadlock. Generally the algorithms which are under detection criteria don't take into account how a detected deadlock is resolved. It is only assumed that it is properly resolved. The algorithms do not explicitly model the resolution of detected deadlocks. Neither the system nor the code of the algorithm includes the effect of resolutions [17]. Most of the reviewed algorithms imply rollback/abort as the solution to deadlocks. The only ways in which they differ is how they select the victim. Most of the strategies of victim selection have been reviewed in the literature, the only drawback of such strategies is that it leads to abort of the victim, or they restart the victim which leads to wastage of resources, wastage of the work done by the aborted process, low throughput of system and it makes execution time of processes unpredictable. May be sometimes the aborted process have to be restarted in order to complete their work. And as it has been discussed that restarting a transaction is

more expensive than waiting, therefore aborting a transaction needs to be avoided.

Therefore in this paper an algorithm has been proposed which do not cause any aborts or rollbacks instead it resolves the deadlock with the mutual cooperation of the transactions.

4. VGS ALGORITHM FOR DEADLOCK RESOLUTION

This section describes the solution to deadlocks in distributed systems i.e. VGS Algorithm an efficient deadlock resolution algorithm. In a distributed system if deadlock is detected at a site, then the site coordinator can apply VGS algorithm to resolve the deadlock. This algorithm is based on the mutual cooperation of the transactions and is described as follows:

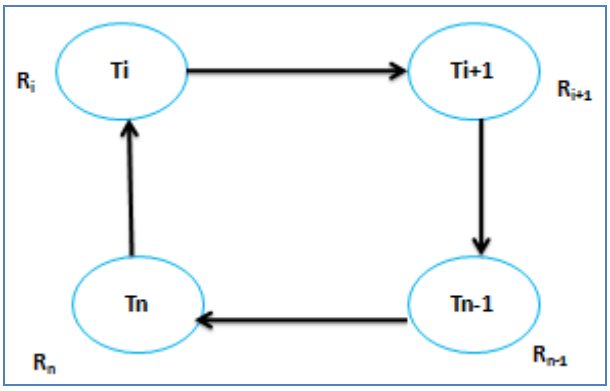

Figure 3: A deadlock cycle

TI REQUESTS Ri+1

Ti+1 REQUESTS Ri+2

.

.

Tn-1 REQUESTS Rn

TN REQUESTS Ri

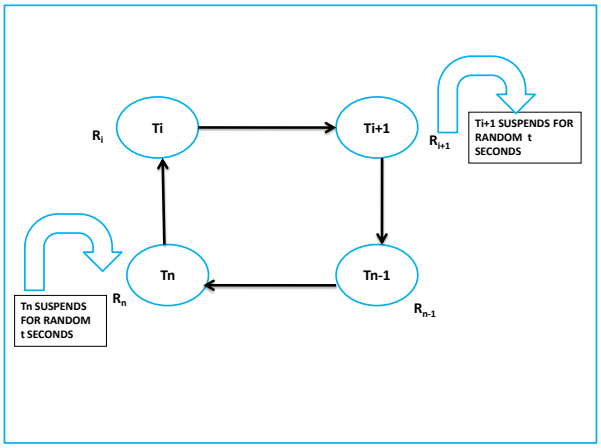

Figure 4: Transaction Ti+1, Tn suspended and release resources

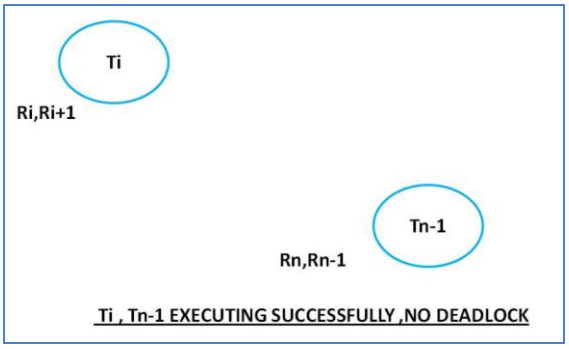

Figure 5: Ti, Tn-1 successfully executing.

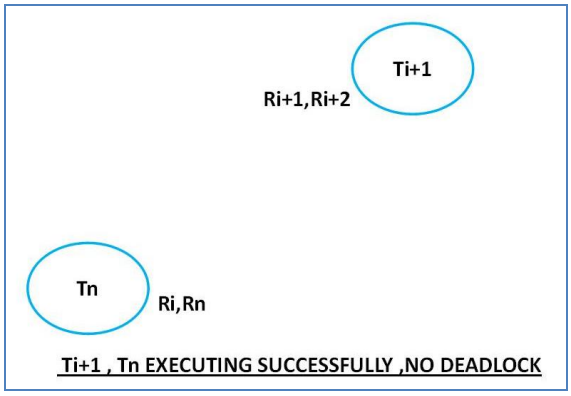

Figure 6: Tn, Ti+1 successfully executing

Suppose Ti, Ti+1, Ti+2……Tn are the transactions involved in a deadlock. They form a deadlock cycle such that Ti holds resource Ri, Ti+1 holds resource Ri+1, Ti+2 holds resource Ri+2……..Tn holds Rn and Ti is requesting for resource Ri+1 , Ti+1 is requesting for resource Ri+2 ……,Tn is requesting for Ri. Since each transaction is holding a resource and waiting indefinitely for other resource held by the other transaction, they form a deadlock cycle and none of them is being able to proceed ahead.

In the proposed deadlock resolution algorithm transaction, coordinator observes the scenario and it suspends Ti+1 for some random t seconds and it releases resource Ri+1 which is acquired by the requesting transaction Ti. It has been allotted the resource for the t seconds which is the time for which Ti+1 has been suspended. Ti is supposed to utilize Ri+1 and execute successfully in t seconds.

If Ti successfully executes before t seconds it sends a message to coordinator that it has successfully executed and to resume transaction Ti+1 and gives its resource Ri+1 back to Ti+1. If Ti is not able to complete its execution within t second coordinator preempts resource Ri+1 from Ti and provides it back to Ti+1. The value Ri+1 is the value partially updated by Ti. Now Ti+1 will check whether Ti is still requesting for Ri+1. If it is requesting ,Ti+1 informs coordinator and is suspended again for some random t seconds and resource Ri+1 is again allotted to Ti, Ti acquires it and resumes its execution and when completed before t seconds Ti informs coordinator to resume Ti+1 and gives back resource Ri+1 to Ti+1.

Similarly coordinator blocks Tn for some random t seconds and it releases resource Rn which is acquired by the requesting transaction Tn-1. It has been allotted the resource for the t seconds which is the time for which Tn has been suspended. Tn-1 is supposed to utilize Rn and execute successfully in t seconds.

If Tn-1 successfully executes before t seconds it sends a message to coordinator that it has successfully executed and to resume transaction Tn and gives its resource Rn back to Tn. If Tn-1 is not able to complete its execution within t seconds coordinator preempts resource Rn from Tn-1 and provides it back to Tn. The value of Rn is the value partially updated by Tn-1. Now Tn checks whether Tn-1 is still requesting for Rn. If it is requesting Tn informs coordinator and is suspended again for some random t seconds and resource Rn is again allotted to Tn-1, Tn-1 acquires it and resumes its execution and when completed before t seconds Tn-1 informs coordinator to resume Tn and gives back resource Rn to Tn.

Like this the deadlock is successfully resolved without causing any aborts/roll backs. The transaction execute successfully with mutual cooperation of each other. The algorithm for deadlock resolution is as follows:

TRANSACTION (Ti, Ti+1….Tn),

RESOURCE (Ri, Ri+1……Rn)

START:

// suppose on using an efficient deadlock detection mechanism a deadlock is detected in the system.

Suppose Ti……..Tn be the transactions involved in a deadlock and form a cycle.

BEGIN,

Ti holds resource Ri

Ti+1 hold resource Ri+1

...

...

Tn holds resource Rn

and

Ti requests resource Ri+1

Ti+1 requests resource Ri+2

.

.

Tn requests resource Ri

Each transaction is in a circular wait and hold condition

DO,

Coordinator suspends transaction Ti+1 and Tn for random t seconds and releases resource Ri+1 and Rn respectively.

{

Ri+1 is now taken by transaction Ti and it executes.

{

IF Ti executes successfully before t seconds

{

{

Ti informs coordinator to resume Ti+1 .Ti+1 resumes and takes the resource Ri+1 back.

}

Now Ti+1 will wait for resource Ri+2 and will proceed successfully as there is no deadlock now

}

ELSE

{

Ti+1 preempts the resource from Ti and value of Ri+1 will be the value partially updated by Ti

}

Ti+1 CHECKS

IF

Ti is still requesting for resource Ri+1

{

{

Coordinator again suspends Ti+1 for random t seconds and gives the resource Ri+1 to Ti

Ti will acquire the resource Ri+1 and will lock it.

After Ti executes successfully it releases Ri+1.

Ti informs coordinator to resume Ti+1 and gives its resource Ri+1 back

}

Now Ti+1 will wait for resource Ri+2 and will proceed successfully as there is no deadlock now

}

ELSE Ti+1 will wait for resource Ri+2 and will proceed successfully as there is no deadlock now

}

Rn is locked by Tn-1

// as coordinator had suspended Tn and released the resource Rn for Tn-1

IF Tn-1 executes successfully before t seconds

{

{

Tn-1 informs coordinator to resume Tn. Tn resumes and takes the resource Rn back

}

Now Tn will wait for resource Ri and will proceed successfully as there is no deadlock now

}

ELSE

Tn preempts resource from Tn-1 and the value of Rn will be the partially updated by Tn-1

Tn CHECKS

IF Tn-1 is still requesting for resource Rn

{

{

Coordinator again suspends Tn for random t seconds and releases the resource Rn

Tn-1 will acquire the resource and will lock the resource Rn

After Tn-1 executes successfully it releases Rn resource

Tn-1 informs coordinator to resume Tn and gives its resource Rn back

}

Now Tn will wait for resource Ri and will proceed successfully as there is no deadlock now

}

ELSE

Tn will wait for resource Ri and will proceed successfully as there is no deadlock now

}

//the coding for both Ti+1, Tn will run parallel**.**

The algorithm is based on the fact that when transaction Ti+1 will suspend and release its resources for transaction Ti then Ti will execute and be in the way of being committed. Now as we know transaction Tn is waiting for Ti to release resource Ri so that it can proceed and execute. But Ti is executing with resources Ri and Ri+1 so it is much better and efficient if we suspend transaction Tn. It is more appreciable because of following features:

1.Transaction Tn needs resource Ri , which is held by Ti therefore it is trapped and cannot proceed therefore instead of waiting for resource Ri, Tn should also suspend and release its resource and let other transaction Tn-1 proceed.

By the time Ti executes, suspending Tn will make resource Rn available to Tn-1 and Tn-1 will execute i.e the waiting time of transaction Tn for Ri will not be wasted instead it will be utilized.

2. Deadlock will be resolved speedily.

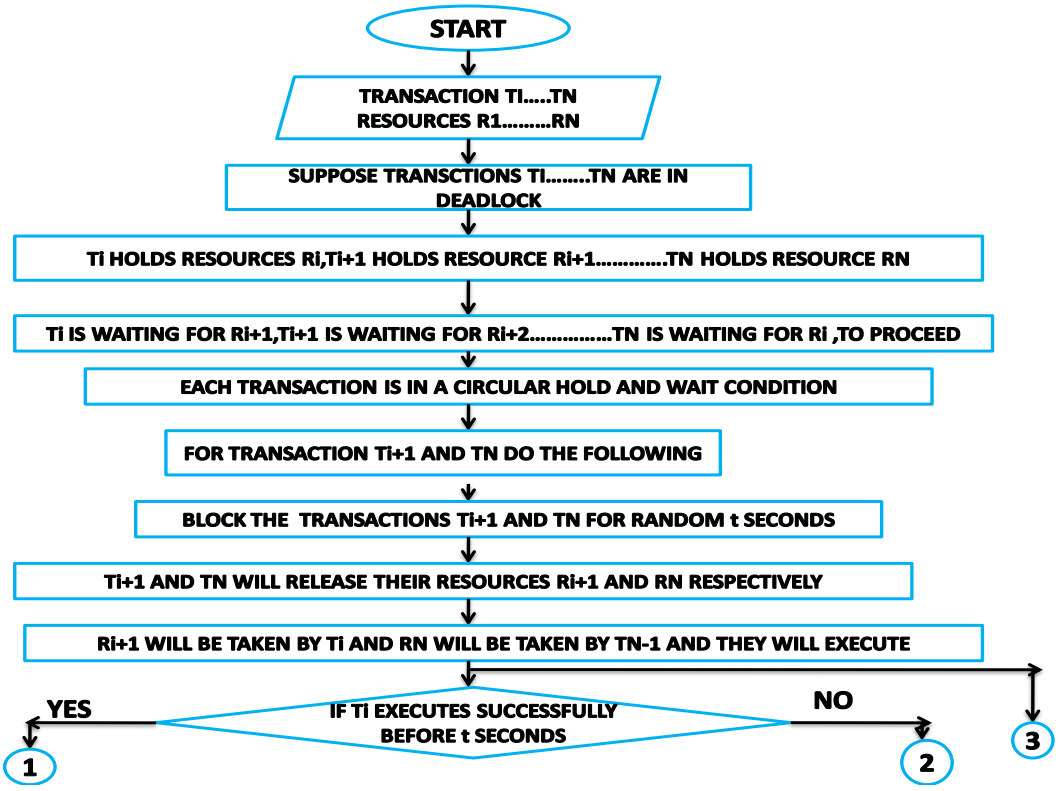

Figure 7(i): Flowchart of the working of the VGS deadlock resolution algorithm.

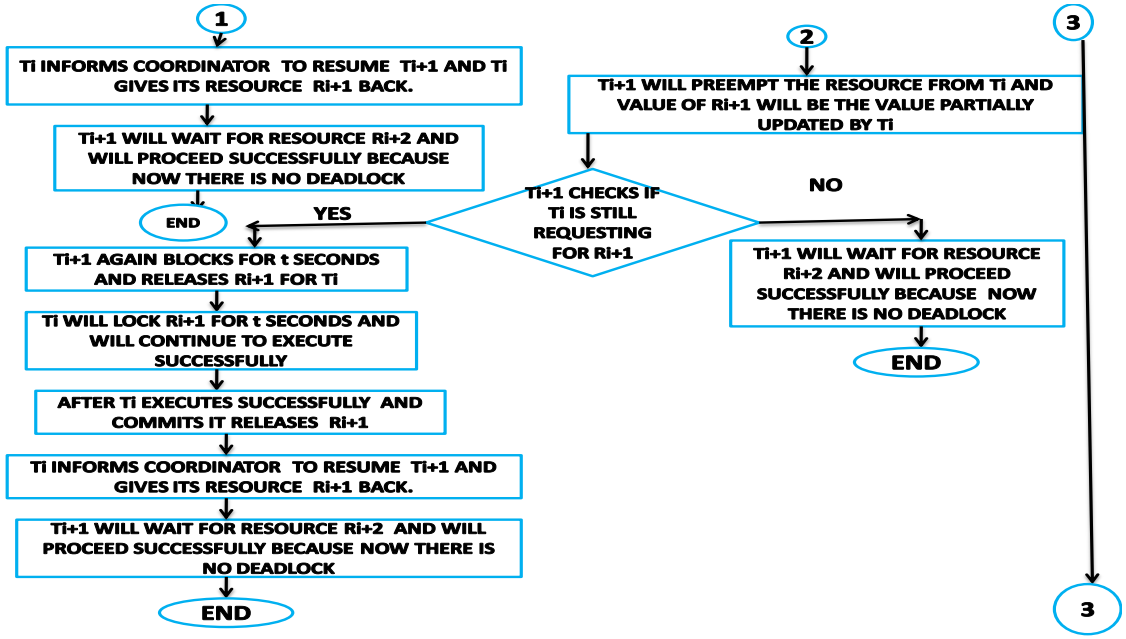

Figure 7(ii): Flowchart of the working of the VGS deadlock resolution algorithm

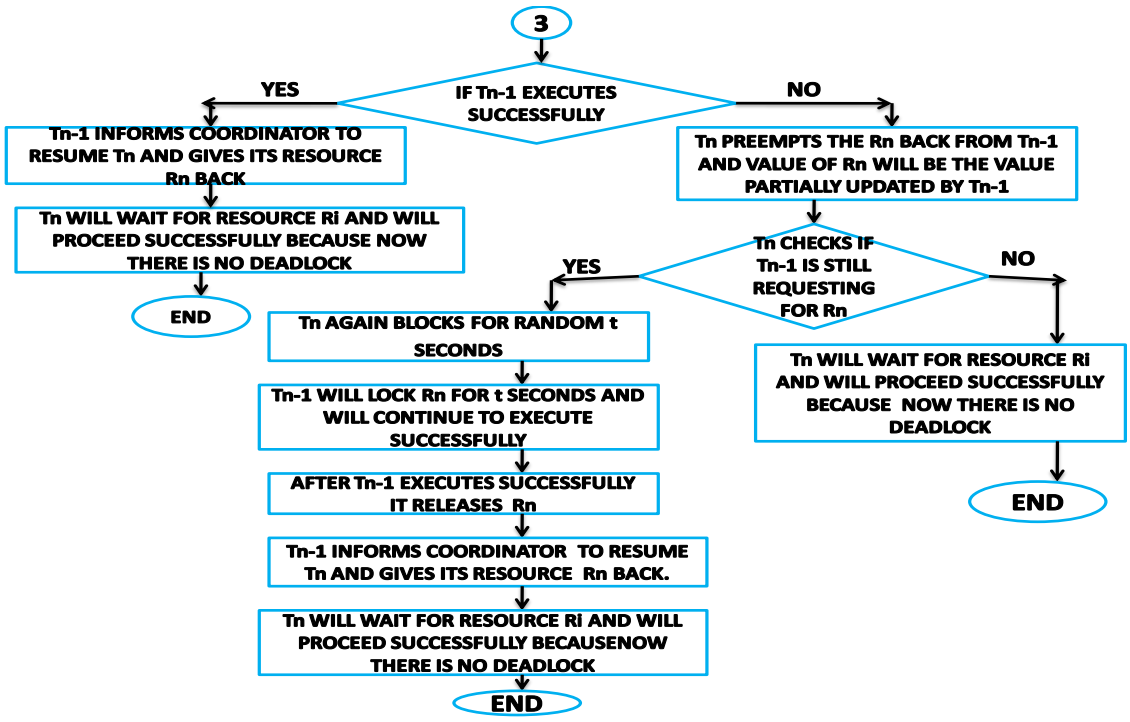

Figure 7(iii): Flowchart of the working of the VGS deadlock resolution algorithm.

VGS algorithm is based on a very simple basis. In the above figure 3 a deadlock cycle is there, Ti needs resource Ri+1 which is possessed by Ti+1in a hold condition. Here, to proceed Ti needs only Ri+1 and it doesn't depends on any other transaction for resources , so if Ti+1 is suspended(i.e. the transaction has been ceased for some time) for a random duration of t seconds Ti can proceed and successfully commit, also Tn needs resource Ri. Since Ri is being processed by Ti so it cannot be made available therefore Tn can also suspend itself for random duration t seconds and Tn's resource Rn can be made available to Tn-1 transaction to proceed.

The proposed VGS algorithm does not resolves deadlock by aborting any process. It considers the fact that when a process aborts it cancels all of its pending requests and it has to release the resources that it holds. Moreover the work done by the aborted process gets wasted. May be sometimes the aborted process have to be restarted in order to complete their work. Obviously the abortion increases the response time of the process because it has to perform the work previously wasted again. [23]

5. CONCLUSION

In this paper we presented deadlock resolution algorithm which resolves deadlocks effectively. As the paper describes in this algorithm the transactions resolve deadlock with the mutual cooperation of each other. Transaction Ti+1 and Tn suspend themselves and let other transactions proceed successfully and continuously co-operate them till they are not able to commit successfully. As compared to other resolution algorithms which cause abort or rollback it does not cause any such aborts or rollbacks, which proves its effectiveness. In the proposed algorithm the distributed system's site coordinator manages its own transactions and resolves any deadlock when detected.

6. REFERENCES

[1] Akikazu IZUMI, Tadashi DOHI and Naoto KAIO, Deadlock Detection Scheduling for Distributed Processes in the Presence of System Failures, 2010 Pacific Rim International Symposium on Dependable Computing, DOI 10.1109/PRDC.2010.49, 2010 IEEE.

[2] G. Bracha and S. Toueg. A distributed algorithm for generalized deadlock detection. Distributed Computing, 2:127–138, 1987.

[3] A. D. Kshemkalyani and M. Singhal. Distributed detection of generalized deadlocks. Proc. 17th Int'l Conf. Distributed Computing Systems, pages 553–560, May 1997.

[4] A. Boukerche and C. Tropper. A distributed graph algorithm for the detection of local cycles and knots. IEEE Trans. Parallel and Distributed Systems, 9(8):748–757, Aug. 1998.

[5] Soojung Lee. Fast Detection and Resolution of Generalized Distributed Deadlocks, Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP.02) 1066-6192/02, IEEE computer society 2002.

[6] Omran Bukhres, Jeanne Alm and Noureddine Boudriga, A Priority-Based PCG Algorithm for Global Deadlock Detection and Resolution in Multidatabase Systems, IEEE 0-8186-3710-2/9, 1993.

[7] Yibei Ling, Shigang Chen, Cho-Yu Jason Chiang, On Optimal Deadlock Detection Scheduling, IEEE TRANSACTIONS ON COMPUTERS, VOL. 55, NO. 9, SEPTEMBER 2006.

[8] 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.

[9] S. Lee and J.L. Kim, "Performance Analysis of Distributed Deadlock Dectection Algorithms," IEEE Trans. Knowledge and Data Eng., vol. 13, no. 3, pp. 623-636, May/June 2001.

[10] X. Lin and J. Chen, "An Optimal Deadlock Resolution Algorithm in Multidatabase Systems," Proc. 1996 Int'l Conf. Parallel and Distributed Systems, pp. 516-521, 1996.

[11] P.P. Macri, "Deadlock Detection and Resolution in a CODASYL Based Data Management System," Proc. 1976 ACM SIGMOD Int'l Conf. Management of Data, pp. 45-49, 1976.

[12] M. Singhal, "Deadlock Detection in Distributed Systems,"Computer, vol. 40, no. 8, pp. 37-48, Nov. 1989

[13] 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.

[14] Y. Ling, J. Mi, and X. Lin, "A Variational Calculus Approach to Optimal Checkpoint Placement," IEEE Trans. Computers, vol. 50,no. 7, pp. 699-708, July 2001

[15] K. Min. Performance Study of Distributed Deadlock Detection Algorithms for Distributed Database Systems. PhD thesis, Univ. of Illinois at Urbana-Champaign Nov. 1989.

[16] S. Warnakulasuriya and T. M. Pinkston. A formal model of message blocking and deadlock resolution in interconnection networks. IEEE Transactions on Parallel and Distributed Systems, 11(3):221–229, March 2000.

[17] J. Villadangos, F. Fari˜na, A. C´ordoba, J.R. Gonz´alez de Mend´ıvil and J.R. Garitagoitia, Knot Resolution Algorithm and its Performance Evaluation, Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03) 0-7695-1875-3/03,IEEE computer society,2003.

[18] S. Chen, Y. Deng, and W. Sun, "Optimal Deadlock Detection in Distributed Systems Based on Locally Constructed Wait-For Graph," Proc. 16th Int'l Conf. Distributed Computing Systems, pp. 613-619, 1996.

[19] J.R. Gonzales de Mendivil, J.R. Garitagoitia, C.F. Alastruey, and J.M. Bernabeu-Auban, "A Distributed Deadlock Resolution Algorithm for the AND Model," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 5, pp. 433-447, May 1999.

[20] A. Cordoba, F. Fari˜na, J.R. Garitagoitia, J.R. Gonz´alez de Mend´ıvil, J. Villadangos A Low Communication Cost Algorithm for Distributed Deadlock Detection and Resolution, Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03),IEEE computer society,0-7695-1875-3/03,2003.

[21] H. A. Ali**,** T. EL-DNAF, and MSALAH, A proposed algorithm for solving deadlock detection in distributed database systems 0-7803-8575-6/04, 2004, IEEE.

[22] Mehdi Hashemzadeh, Nacer Farajzadeh, Abolfazl T. Haghighat, Optimal Detection and Resolution of Distributed Deadlocks in the Generalized Model, Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06),IEEE computer society, 1066-6192/06, 2006.

[23] Manuel Prieto, Jesús Villadangos, Federico Fariña, Alberto Córdoba, An *O*(*n*) distributed deadlock resolution algorithm, Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06) IEEE computer society, 1066-6192/06, 2006 .

[24] Soojung Lee, Fast, Centralized Detection and Resolution of Distributed Deadlocks in the Generalized Model, published by IEEE computer society, 0098-5589/04, 2004.