Proceedings of the Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV 2021).
DVD Part Number: CFP21ONG-DVD, 978-0-7381-1182-7

Survey on Deadlocks in Distributed Database Systems

Dr. Y. Bhavani
Associate professor, Dept. of IT
Kakatiya Institute of Technology & Science
Warangal, India
yerram.bh@gmail.com
ORCID: 0000-0002-8578-7126

Dr. K. Praveen Kumar
Assistant Professor, Dept. of IT,
Kakatiya Institute of Technology & Science,
Warangal, India
pravee20@gmail.com

K. Dharmateja
Bachelor of Technology III year, Dept. of IT
Kakatiya Institute of Technology & Science
Warangal, India
kondadharmateja1998@gmail.com

P. Pranathi
Bachelor of Technology III year, Dept. of IT
Kakatiya Institute of Technology & Science
Warangal, India
pranathi.pash123@gmail.com

R. Sowmya
Bachelor of Technology III year, Dept. of IT
Kakatiya Institute of Technology & Science
Warangal, India
ragillasowmya25@gmail.com

***Abstract*—In Distributed Database System (DBS) and multitasking system, occurrence of deadlocks is one of the most serious problems. If a site request a resource that is already hold by another site and which is waiting for another resource then the scenario is called as distributed deadlock. Different distributed environments require a suitable deadlock detection algorithm to detect deadlocks. Different distributed environments needs to maintain their platforms by avoiding deadlocks. To achieve this the environment should be fed with optimized deadlock detection and avoidance algorithms. In this paper different deadlock detection algorithms that uses Wait For Graph and resolution algorithms to trace out deadlocks are discussed. We have also elucidated optimization techniques, for resolving deadlock in an efficient manner. We made a comparison between different deadlock detection algorithms based on different parameters like, delay time, message size, number of messages and whether the algorithm detects false deadlocks or not. Based on the comparisons we have suggested few deadlock detection algorithms for the distributed environment.**

Keywords— Distributed systems, Deadlock, Deadlock detection algorithms, DBS, TWFG, Wait-For-Graph.

I. INTRODUCTION

In Distributed Database (DDB) at least two systems are associated with many sites and each has its own database. The software that handles DDB is called "Distributed Database Management System (DDBMS)". "The combination of both DDB and DDBMS is known as Distributed Database System (DBS)". [14]

Deadlocks can be handled in three ways:

1) Prevention: prevention requires the configuration of system, so that no deadlock will occur.
2) Avoidance: allocation of resources to be done at runtime or real-time, to ensure that no deadlock will form.
3) Detection: first deadlocks are permitted to occur and some protocols will detect them later.

In general, detection is more preferable way to resolve the deadlock issues than prevention and avoidance. Because they are more complex to achieve in real time and they restrict the convenience of developing/choosing the configuration of system.

Different Deadlock Detection Techniques are as follows:

i. Linear transaction structure (LTS): When different transactions of a unique location are part of Wait For Graph then local deadlock cycle occurs.

ii. Distributed Transaction Structure (DTS): When various transactions of a multiple location are part of Wait For Graph then global deadlock cycle occurs.

A. Generalized model

Generalized model can be expressed in terms of two algorithms.

a. *Centralized algorithm:* In Centralized algorithm single node works to detect deadlock. Chen's et al. algorithm and Lee's algorithm uses centralized algorithm. Chen's algorithm uses 2n messages and 2d time units to define deadlocks, where as Lee's algorithm requires less than 2e messages and only d+2 time units.

b. *Distributed algorithm*: Unlike centralized algorithm in Distributed algorithm [10] different nodes work together to detect deadlocks. To define deadlocks, G. Bracha and S. Toueg uses 4e messages and 4d time units, where e is a number of edges and d is diameter of Wait-for-graph. It also uses tokens with $n^2/2$ messages and 4n unit times to detect deadlock.

In distributed approach there is no single point failure, because if one node is failed other node can work and make the system reliable. In this approach, speed of deadlock detection also increases. Whereas in Centralized approach if the single node fails whole system will be crashed out.

II. RELATED WORK

In this paper, deadlock detection and resolution algorithms in distributed environment is discussed, which are shown below.

A. Kawazu's Algorithm

Kawazu's algorithm [1**]** consists of LOCAL and GLOBAL sub-algorithms. Local deadlocks are detected by Algorithm LOCAL and it also confirms whether it require global detection process are not. Only the information provided by Sub-WaitGraph is sufficient for Algorithm LOCAL. GLOBAL Algorithm will detect global deadlocks only when algorithm LOCAL confirms there is a need of global deadlock detection.

A global deadlock will occur when transactions from multiple sites involves in a deadlock. In order to detect global deadlocks, information from Sub-Wait-Graphs should be gathered. When collecting information, it is not appropriate to lock any of these Sub-Wait-Graphs. Not all Sub-Wait-Graphs are gained all at once. A graph composed of collected Sub-Wait-Graphs that are distinct from Wait-Graph G and so called as Pseudo-Wait-Graph.

When compared to centralized systems this algorithm is advantages because of its distributed nature and overall network throughput.

B. Obermack's Algorithm

Obermacks algorithm *[2]* to detect deadlock uses Transaction-Wait-For Graph (TWFG) instead of global TWFG. In this algorithm it makes use of significant hub at every site, this hub is utilized to represent part of TWFG, which is not known to the site. The disadvantage of this algorithm is it distinguishes false deadlocks due to not utilizing the global TWFG.

C. HO's Algorithm

Hos algorithm [*4]* is based on centralized technique.

- It uses only two levels i.e., Master control nodes and Cluster control nodes.
- Detection of inter-cluster deadlocks is the responsibility of the master control node. Cluster control nodes are used to detect deadlock among their members and report dependencies to the master control node outside their cluster.

It contains two types:

1) HO Ramamurthy (One-Phase Algorithm)
2) HO Ramamurthy (Two-Phase Algorithm)

In One-Phase algorithm the control or central site will maintain both resource status table (local resources) and process table (local processes). When a cycle is noticed in resource table and process table then system is said to be in a deadlock. In this approach the time consumption is less but space complexity will increase.

- Control site will form WFG and checks for deadlocks by periodically requesting status from all sites.
- Only if a process table info agrees with a resource table info, then we will perform transaction.

In Two-phase algorithm central or control site will maintain a resource status table. When a cycle is identified for the first time the system will not be declared as deadlock, this cycle will be checked again for the vacant resources. If same cycle identified after checking, then the system is said to be in a deadlock. In this technique the chance of occurring false deadlock will be reduced but time consumption is more.

In Two-Phase algorithm

- Each site maintains local processes status table.
- Control site will be built WFG and checks for deadlocks by periodically requesting status from all sites.
- When a deadlock is detected, the control site will repeat status request but abort the transactions that have updated.

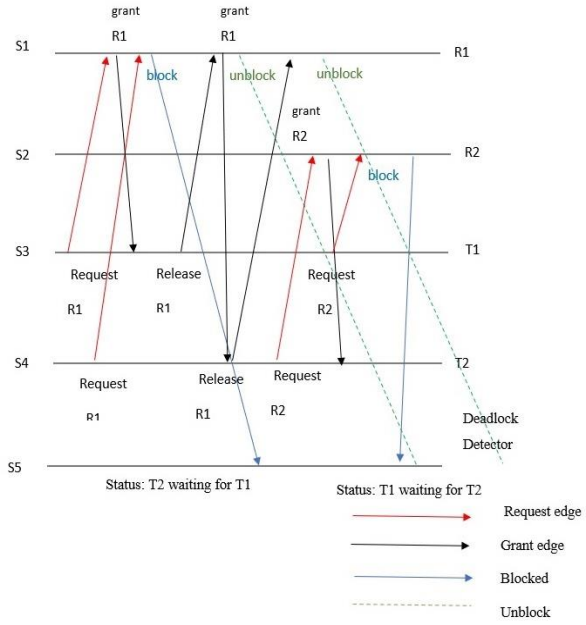

Fig 1.0 Event trace diagram.

From the above fig. 1.0 when the transaction t2 is requested the resources R1, it was already granted to T1. Now T2 will wait for T1 to release the resource R1. As soon as T1 release R1, it will be granted to transaction T2. Similarly, when T1 is requested resource R2 it was already granted to T2, so now T1 will wait for T2 to release the resource R2.

The above process is explained using the fig 1. 1.

T_1: lock R_1	T_2: lock R_1 T_1: unlock R_1	T_2: unlock R_1 T_1: lock R_2	T_2: lock R_2 T_1: unlock R_2	T_2: unlock R_2

Fig 1.1 Two transactions running concurrently.

D. Sinha's Scheme

Sinha's scheme *[6]* is an augmentation to Chandy's algorithm.[3] It depends on needs of transactions, by this we can decrease the quantity of messages needed for Detecting algorithm. Sinha's algorithm doesn't develop the TWF diagram, however with the assistance of test messages it looks for cycles. A test message is a 2-tuple <i, j>, i being the transaction that confronted and initiated deadlock detection and j is the transaction of the least need among all the transactions. This model comprises of transactions and information directors that are answerable for conceding and delivering locks. A transaction requests for a lock on a data item which is sent to the information directors. If a request is not permitted, then the information directors initiates detecting deadlock by delivering a probe to the transaction which holds a lock on the data item. If a lower priority transaction sends a probe to a waiting transaction, the probe is removed. The probes are propagated based on the priorities of transaction. When a transaction starts waiting for a lock, all the probes from its queue are propagated. When an information director holds back to the probe which is initiated, deadlock is detected. The youngest transaction in the cycle is aborted if probe holds the priorities.

Disadvantages of this algorithm is detecting false deadlocks and unable to find complete deadlocks. Since the transaction continually waiting for a deadlock is ignored and probes of aborted transactions are not removed properly.

E. Knot Detection Algorithm

Chang et al. *[7]* suggested a deadlock detection algorithm called Knot algorithm in this a cycle is identified as a knot. In a directed graph a node is called as a part of knot, only when the node is attained from all the nodes which are associated to it.

In fig.1 the subgraph with the vertices {A, B, C, D, E} is a knot because all the nodes in this subgraph are attainable from all other nodes. Let us consider node i to be initiator and node j is attainable from node i. Here node i is said to a knot iff there is a directed path from node i to node j and every reachable node from i lies in a cycle. The initiator sends *detect_knot* message to all the reachable nodes in the graph. Node j will be immediate successor of node i iff there is a directed edge from i to j (i -> j). Each node after getting *detect_knot* message will forward to its neighboring node. These all edges from first *detect_knot* message will form Directed Spanning Tree (DST) with initiator i as a root. The initiator node i will send *cycle_ack* to node j after the *detect_knot* message to inform that a cycle is formed with initiator node i. If k receives second *detect_knot* message from j confirms that it lies on a cycle, it will send *seen_ack* for the node j immediately.

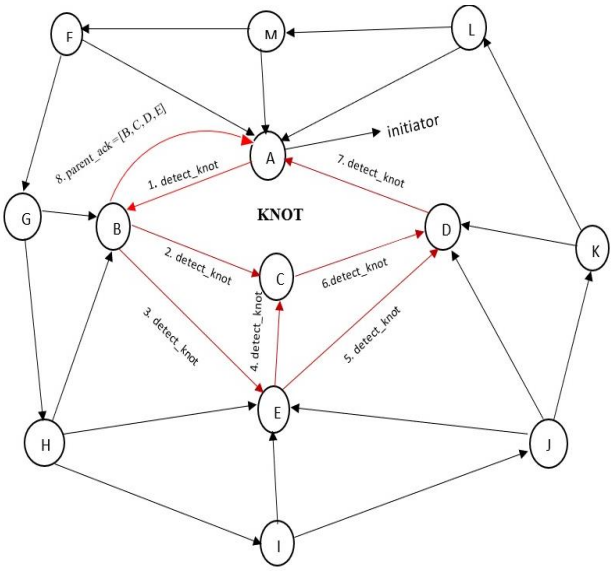

Fig 2.0 Diagrammatic view of Knot Detection Algorithm

Parent_ack acknowledge will be sent by node j to its parent, after receiving all acknowledges from its immediate successors. *Parent_ack* will list the nodes which will be on cycle with i, and order pairs (j, l). If i present in a knot, then cycle_*nodes* will contain precisely node ids associated in the knot. If i is not present in a knot, then the *cycle_nodes* have collection of nodes that are not in a cycle with i.

F. Edge – Chasing Algorithm

This algorithm *[8]* utilizes unique data packets called probes to recognize cycle in WFG. This probe information gets propagating only for transactions which are halted through edges of diagram and forwarded to all transactions on which halted transactions relies. At whatever point active transaction gets a probe it will be abandoned. At the point when Blocked transaction gets a probe, the path information will be modified and again it begins propagating. Limitation to this algorithm is, it unable to identify deadlock if starting transaction does not belongs to deadlock cycle.

Formal Description of Algorithm

Each probe consists of initiator_id (integer) and route-string (string) which comprises codes of passed edge. After getting probe, each node determines whether there is a message in its storage or not. If there is no message in its storage, then node will save message and sends to its successor by adding the string "r_i". If there is a message in the memory, it confirms the existing message with the initiator_id number of received message. If it is larger, message will be rejected and if it is smaller, message will be replaced with successor message. If the current message's route string is the route-string prefix of the receiving message, a deadlock is observed. Once the deadlock is detected it checks, if node is initiator node, If so 'clean-up' message is sent else 'permission' message.

```
Format of probe message: -
begin
    Init ID as an integer;
    Route string as a string;
End
Format of permission message: -
Begin
    ID as an integer;
End
Format of ok or deny message: -
Begin
    /*empty message*/
End
Format of node: -
Begin
    First time as Boolean;
    ID as integer;
    Memory as probe;

End
```

If request reaches the starting node before propagating clean-up message, 'ok' message is assigned otherwise 'deny' message is assigned.

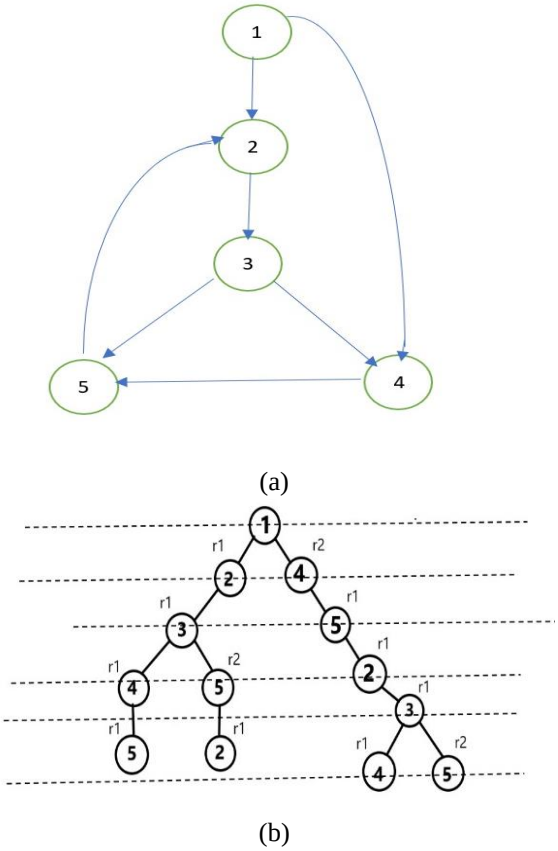

Fig. 3.0. Representing the implementation of algorithm.

We have shown the implementation of an algorithm in fig. 3.0 (a). let us assume node "1" as the initiator. In fig. 3.0 (b) shows the propagation path of probe message.

G. B. M. Alom Algorithm

This [9] approach utilizes LTS and DTS deadlock detection techniques and also uses priorities to recognize and resolve deadlock. This technique contains two tables, list of transactions in different sites in first table and transaction Initiator_id's and its priorities in second table. Transactions with least priority are terminated and hence the data items will be accessible for different transactions in waiting state. Limitation for this algorithm is, it unable to recognize deadlock if sequence of priorities gets changed.

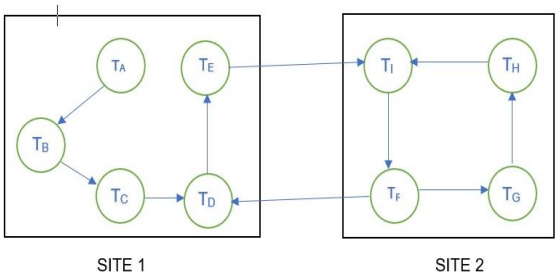

Fig 4.0 WFG containing local and global deadlock cycles.

H. Resolution of Local Deadlock and its Detection

It is observed that from fig. 2, in site 1 there is no cycle formation and hence no deadlock.[5] Based on fig 2 and table 1 it is confirmed that there exists a cycle in site 2 that leads to deadlock. As shown in table 2, G is having lowest priority so the edge from T_G to T_H is aborted and the resources are allocated to other waiting states.

Table 1: LTS for Site 2

T_x	T_Y
T_I	T_F
T_F	T_G
T_G	T_H
T_H	T_I

Table 2: Order of Transactions at Site 2

Transaction ID	Priority
G	1
F	2
H	3
I	4

I. Resolution of Global Deadlock and its Detection.

It is observed that from fig. 2 and table 3 there exist a cycle between site 1 and site 2 which lead to deadlock. As shown in Table 4, F is having lowest priority so, the edge from T_F to T_D is aborted and the deadlock is resolved.

Table 3: Site 1 and Site 2 Showing DTS

T_X	T_Y
T_D	T_E
T_E	T_I
T_I	T_F
T_F	T_D

Table 4: Order Of Transactions Showing Part Of Deadlock

Transaction ID	Priority
F	1
E	2
I	3
D	4

J. Explanation of Optimization Technique: [B.M. Monjurul Alom]

Optimization technique *[9]* is used to detect the edges that forms Most Deadlock Cycles (MDC) in distributed database system (DDS). Once these edges are detected from all the sites, they are removed. The edges that cause least deadlocks will not be removed.

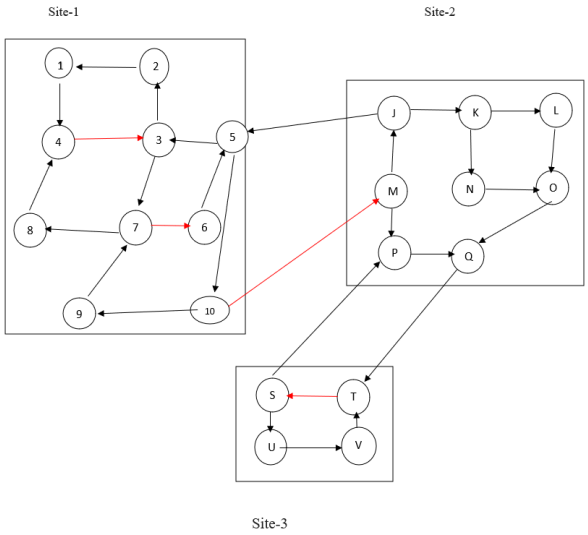

Fig 5.0 Complex TWFG consists local and global deadlock within sites s1, s2, s3.

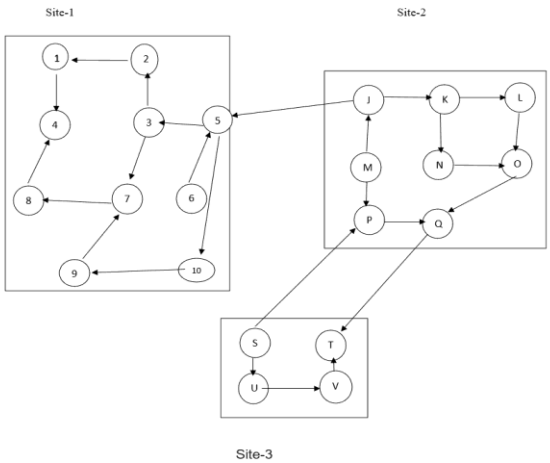

Fig 5.1 Optimized TWFG free from deadlocks within three sites s1, s2, s3.

In fig. 5.0 there is the possibility of forming deadlocks with the cycles (1, 4, 3, 2, 1), (4, 3, 5, 7, 4), (3, 5, 6, J, 3), (8, 5, 6, J, 9, 8) in site-1 among these cycles the edge 4→3 and

edge 6→7 causing most deadlock cycles (MDC), so we need to remove these edges from the site to avoid deadlock. Similarly, we should remove edges causing most deadlock cycles from site-2 and site-3. Fig. 5.1 is obtained after removing the MDC and now the system is free of deadlocks.

K. Safety Detection Algorithm

Momotaz Begum *et* al *[13]* proposed a Safety Detection algorithm, which depends on sorting and linked list data structure. The processes in this approach are arranged in ascending order depending on the amount of resources required. Linked list data structure is used to reserve resources of all processes in descending order depends on number of resource instances it required to complete its execution.

Let n represent number of processes and d types of resources. Max_Needed *[a, b]:* (a = 1, 2,3,4 n, b = 1, 2,3,4 …m) represent the maximum requirement of resources, *Current_allocation [a, b]*: (a=1, 2, n, b=1, 2, … m) represents currently allocated resources. Available_resources[k] is used to represent the number of resources which are available, *required_allocation* is used to specify the amount of resources required for each process to perform its execution. Max_Needed_Allocation [a, b]: (a = 1, 2,3,4 . . ., n, b = 1, 2,3,4, . . ., m) of integer data type is used to store maximum amount of resources required for each process. Complete process[j]:(j = 1, 2, . . ., n) of Boolean data type is used to intimate process completion status as 0 or 1, 0 refers process cannot be completed and 1 imply can be completed. Only when Max_Needed_Allocation ≤ Min_available the system will be in safe state and return safe sequence after completing execution of all processes, otherwise the system will return *unsafe_state*. Time complexity for this safety algorithm is O(n) for the best case and O(nd) for the worst case, which is much better than Banker's algorithm, for space complexity this algorithm uses arrays for storing, Current_allocation[] and Needed_allocation[]. This algorithm used Linked list a high dimensional data structure which have very fast access time and doesn't require pre-allocated memory. By this we can say that memory utilized by safety detection algorithm is very low compared to Original banker's algorithm.

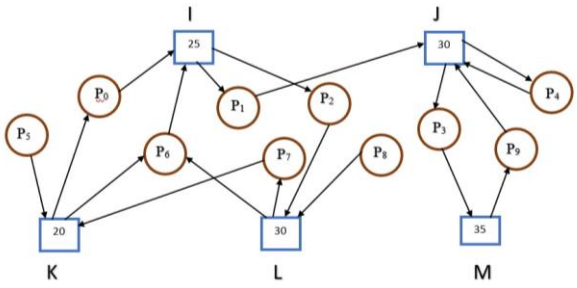

Fig 6.0 Graph showing Allocation of resources.

TABLE I: Representation of Needed resources and Currently allocated resources of each processes.

Process No	Needed_Allocation					Current_Allocation				
	I	J	K	L	M	I	J	K	L	M
P_0	7	1	8	4	6	4	0	4	2	3
P_1	6	6	4	3	3	4	0	2	2	0
P_2	1	6	4	7	1	0	3	0	4	0
P_3	1	3	6	2	4	0	1	2	0	1
P_4	4	0	0	1	0	1	0	0	1	0
P_5	5	2	4	0	1	1	0	1	0	0
P_6	3	1	1	5	2	1	0	0	1	0
P_7	0	0	5	0	0	0	0	2	0	0
P_8	0	1	6	0	0	0	0	1	0	0
P_9	0	0	8	0	7	0	0	2	0	2

According to resource allocation graph as shown in fig 6.0, the table1 and table2 are updated. The table1 shows maximum resources required and Current_Allocation of each process. On subtracting Max_Needed and Current_Allocation we get Needed_allocation of each process which is as shown in table2. Now, all the processes are arranged in ascending order with the Max_Needed_*Allocation as shown in table3*.

TABLE II: Representation of resources which are Needed and Max Needed of each process.

Process No	Needed_Allocation					Max_Need_Allocation
	I	J	K	L	M	
P_0	3	1	4	2	3	4
P_1	2	6	2	1	3	6
P_2	1	3	4	3	1	4
P_3	1	2	4	2	3	4
P_4	3	0	0	0	0	3
P_5	2	1	1	4	2	4
P_6	2	1	1	4	2	4
P_7	0	0	6	0	0	6
P_8	0	1	5	0	0	5
P_9	0	0	6	0	5	6

TABLE III: Representation of Sorted list of processes using maximum need.

Process No	Needed_Allocation					Max_Need_Allocation
	I	J	K	L	M	
P_4	3	0	0	0	0	3
P_0	3	1	4	2	3	4
P_2	1	3	4	3	1	4
P_3	1	2	4	2	3	4
P_5	4	2	3	0	1	4
P_6	2	1	1	4	2	4
P_8	0	1	5	0	0	4
P_1	2	6	2	1	3	5
P_7	0	0	6	0	0	6
P_9	0	0	6	0	5	6

The algorithm is explained using the following steps.

Step 1: Let us consider, the given available resources for all resource types are [I J K L M] is [6 7 5 3 4].

Step2: For process P_4 Max_needed_*Allocation* ≤ Min_Available is true because the minimum available resource for this process is 3. Since Available resource = Available resource + Current_allocation, so [I J K L M] = [7 7 5 4 4].

Step 3: By following the process explained in step2 for process P_0 the available resources change to [I J K L M] = [11 7 9 6 7].

Step 4: Similarly, for process P_2 the available resources [I J K L M] = [11 10 9 10 7] and the same operation is performed on all processes.

Finally, <P_4, P_0, P_2, P_3, P_5, P_6, P_8, p1, P_7, and P_9> is the final Safe Sequence. In this algorithm we require only 10 comparisons to determine safety whereas in Banker's algorithm we need 50 comparisons to check same operation.

L. Wait-die and Wound-wait algorithms

Wait-die: When a transaction is created, timestamps are attached to each transaction. Timestamps are used for ordering. In this approach if older process waits for the younger process then the older process continues to wait, otherwise if the younger process waits for the older process then younger process get expired and restarts with same timestamp. So, this algorithm is based on timestamps rather than priorities.

Algorithm:

```
//Transaction and timestamp are user defined datatypes.
Transaction Ti, Tj;
timestamp Ts (Ti), Ts (Tj);
    if (Ts (Ti) < Ts (Tj))
        Ti:=wait; Ti is older than Tj, Ti is allowed to wait
    else
        Ti:=terminate; // Ti younger than Tj, so Ti is terminate
                          and is resume later with the same timestamp
```

Wound-wait: In this approach older process waiting for the younger process destroys the younger process and acquire its resources. Otherwise, if the younger process waits for the older process then the younger process continues to wait.

```
Algorithm: // Transaction and timestamp are user defined
datatypes.
Transaction Ti, Tj;
timestamp Ts (Ti), Ts (Tj);
    If (Ts (Ti) < Ts (Tj))
        Tj := wound; // Ti is older than Tj so Ti wound Tj and
                        acquire it resources
    else
        Tj := wait; // Ti earlier than Ti so Tj is allow to wait
```

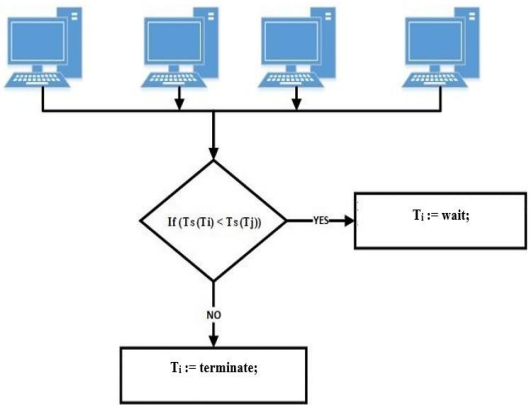

Fig 7.0 Wait-die algorithm

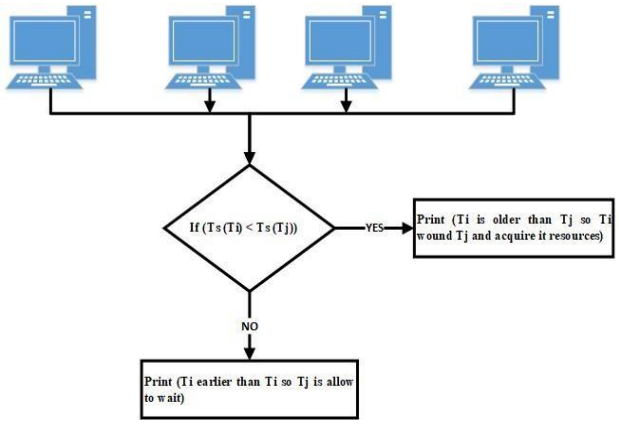

Fig 7.1 Wound-wait algorithm

III. COMPARISON OF DISTRIBUTED DEADLOCK DETECTION AND AVOIDANCE ALGORITHMS

In this paper, different deadlock detection and avoidance algorithms are discussed with their pros and cons. Comparison of these algorithms with different parameters like number of messages, delay, message size, etc. are tabulated below. This comparison helps the readers to choose appropriate deadlock detection or avoidance algorithm in their applications.

As shown in table v the kawazus algorithm needs 4m(n-1) messages, nT ms delay time,O(n) messages but unable to detects all deadlocks. Similarly, obermacks algorithm needs m(n-1)/2 message, delay time and message size is same as kawazus algorithm. Other algorithms shown in the table can detect all deadlocks without false detection except kawazus and obermacks algorithm.

Table V. Performance comparison between the different algorithms.

Algorithm	Number of messages	Delay	Message Size	Detect all deadlock	Not detect false deadlock
Kawazu's [1]	4m(n-1)	nT	O (n)	no	yes
Obermacks's [2]	m(n-1)/2	nT	O (n)	yes	no
Ho's [4]	Not Applicable	O(n)	Not Applicable	yes	yes
Sinha's scheme[6]	2(n-1)	2(n-1)T	Constant(small)	yes	yes
Knot detection [7]	2e	2(d+1)	$O(n^2)$	yes	yes
Edge chasing [3]	m(n-1)/2	O(n)	3 integer words	yes	yes
B.M. alom [9]	Depends on number of sites	O(n)	Constant(small)	yes	yes
Optimization Technique [9]	Depends on number of sites	O(n)	Constant(small)	yes	yes
Safety detection [13]	$O(nd^2)$	n	O(n)	yes	yes

The notations used in the table are N = number of nodes, n= number of sites in deadlock cycles, m= number of process involves in deadlock, e = number of edges, d = depth of WFG, T= Time in msec.

IV. Conclusion

The serious problem of distributed environment is resource starvation which is caused by deadlocks.[12] To overcome these problems choosing an efficient deadlock detection and resolution algorithm is necessary. In this paper, we have discourse on the performance of various deadlock detection and deadlock resolution approaches and also illustrated deadlock prevention approach.

On comparing, it is clearly depicted that algorithm which used optimization technique gave a better performance with more efficient time complexity and memory management. These algorithms used dynamic memory allocation as data structures that decreases the space needed. Based on the studies it is observed that safety detection algorithm needs only $O(nd^2)$ messages, n ms delay time, it can be able to detect all deadlocks but not false deadlocks.

In future, new algorithms can be designed which will avoid the deadlocks more efficiently compared to these present algorithms. An advanced research work is inevitable on distributed environment to increase resource sharing, concurrency control, fault tolerance and transparency.

References

[1] Kawazu, Seiichi, et al. "Two-phase deadlock detection algorithm in distributed databases." *Fifth International Conference on Very Large Data Bases, 1979*. IEEE, 1979.

[2] Obermarck, Ron. "Distributed deadlock detection algorithm." *ACM Transactions on Database Systems (TODS)* 7.2 (1982): 187-208.

[3] Chandy, K. Mani, and Jayadev Misra. "A distributed algorithm for detecting resource deadlocks in distributed systems." *Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing*. 1982.

[4] Ho, Gary S., and C. V. Ramamoorthy. "Protocols for deadlock detection in distributed database systems." *IEEE Transactions on Software Engineering* 6 (1982): 554-557.

[5] Mitchell, Don P., and Michael J. Merritt. "A distributed algorithm for deadlock detection and resolution." *Proceedings of the third annual ACM symposium on Principles of distributed computing*. 1984.

[6] Sinha, Mukul K., and N. Natarajan. "A priority based distributed deadlock detection algorithm." *IEEE Transactions on Software Engineering* 1 (1985): 67-80.

[7] Manivannan, D., and Mukesh Singhal. "An efficient distributed algorithm for detection of knots and cycles in a distributed graph." *IEEE Transactions on Parallel and Distributed Systems* 14.10 (2003): 961-972.

[8] Farajzadeh, Nacer, et al. "An efficient generalized deadlock detection and resolution algorithm in distributed systems." *The Fifth International Conference on Computer and Information Technology (CIT'05)*. IEEE, 2005.

[9] Alom, BM Monjurul, Frans Alexander Henskens, and Michael Richard Hannaford. "Optimization of detected deadlock views of distributed database." *2010 International Conference on Data Storage and Data Engineering*. IEEE, 2010.

[10] Tao, Zhi, et al. "A semi-centralized algorithm to detect and resolve distributed deadlocks in the generalized model." *2014 IEEE 17th International Conference on Computational Science and Engineering*. IEEE, 2014.

[11] Kate, Vandana, Akansha Jaiswal, and Ambika Gehlot. "A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock." *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. IEEE, 2016.

[12] Haque, Waqar, Matthew Fontaine, and Adam Vezina. "Adaptive Deadlock Detection and Resolution in Real-Time Distributed Environments." *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2017.

[13] Begum, Momotaz, et al. "An Improved Safety Detection Algorithm Towards Deadlock Avoidance." *2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, 2020.

[14] Jain, Sumika, Nitin Kumar, and Kuldeep Chauhan. "An Overview on Deadlock Resolution Techniques."