

901

**STUDY OF DEADLOCK DETECTION AND RESOLUTION
ALGORITHMS IN DISTRIBUTED DATABASE SYSTEMS**

670

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE

by

H. C. JOSHI

**SCHOOL OF COMPUTER AND SYSTEM SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI 110067
JANUARY 1991**

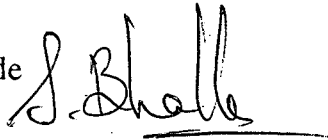
CERTIFICATE

This work titled **STUDY OF DEADLOCK DETECTION AND RESOLUTION ALGORITHMS IN DISTRIBUTED DATABASE SYSTEMS** has been carried out by Mr. H.C.Joshi bonafide student of the School of Computer and System Sciences, Jawaharlal Nehru University.

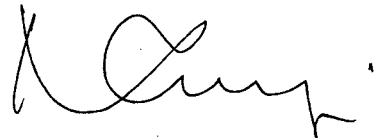
This work is original and has not been submitted so far in part or full for any degree or diploma in any other University or Institute.

Candidate

Guide



SCSS, JNU



Dean

SCSS, JNU

ACKNOWLEDGEMENTS

I thank my guide **Dr. S. Bhalla** for guiding me on this dissertation, constantly encouraging me, providing a lot of research materials and friendly behaviour.

I thank the **Dean and faculty** of School of Computer and System Sciences, for teaching me Computer Science.

I thank **Dr. Sudhir Kaicker**, the director of computer center and the staff in the center for the facilities provided in the Computer Center.

I thank my friends **Mr. Mahesh Prasad Baranwal** and **Mr. D.K. Lobiyal** for their help in editing the dissertation.



(H. C. Joshi)

January 1991

ABSTRACT

In distributed database systems deadlocks occur due to locking of data items by executing transactions. A deadlock exists in the system if a cycle is detected in transaction wait for graph (TWFG). However, due to communication delay, it is difficult to construct a consistent TWFG within the distributed systems

A number of deadlock detection algorithms have been suggested that detect the existence of a deadlock within the distributed environment. A few distributed deadlock detection algorithms are discussed and compared. Our comparison of these algorithms is based on number of message required for deadlock detection, inherent delays, data structures used in the algorithms and limitations.

Modifications in the algorithms are suggested for deadlock resolution. We have identified a deadlock avoidance algorithm that provides a matrix based approach to detect a deadlock as it occurs. This novel idea has further been studied. A deadlock resolution technique is suggested by us. A new matrix based model has been suggested for deadlock resolution.

CONTENTS

	PAGE NO.
ABSTRACT	
1. INTRODUCTION	1
1.1 Deadlock in databases	2
1.2 Database Deadlock Models	4
1.2.1 One Resource Model	4
1.2.2 AND Model	4
1.2.3 OR Model	5
1.2.4 AND-OR Model	5
1.3 Resource and Communication Deadlock	6
1.4 Locking	7
1.4.1 Two-Phase Locking	8
1.5 Deadlock Handling Approaches	9
1.5.1 Deadlock Prevention	10
1.5.2 Deadlock Avoidance	13
1.5.3 Deadlock Detection	13
1.6 Overview	14
2. DISTRIBUTED DEADLOCK DETECTION ALGORITHMS	16
2.1 TWFG Based Algorithms	17
2.1.1 Goldman's Algorithm	17

2.1.2	Isloor - Marshland's Algorithm	18
2.1.3	Menasce - Muntz's Algorithm	19
2.1.4	Obermarck Algorithm	22
2.1.5	Ho-Ramamurthy's Algorithm	24
2.1.6	Discussion	26
2.2	Non TWFG/Probe Based Algorithm	28
2.2.1	Chandy-Mishra's Algorithm	29
2.2.2	Sinha-Natarajan's Algorithm	30
2.2.3	Discussion	39
3.	COMPARISON OF DEADLOCK DETECTION ALGORITHMS AND DEADLOCK RESOLUTION.	41
4.	DEADLOCK RESOLUTION	48
4.1	Sinha-Natarajan's Algorithm	48
4.2	Ho-Ramamurthy Algorithm	51
4.3	Goldman's Algorithm	52
4.4	Deadlock Detection in TWFG Based Algorithms	53
5.	MATRIX BASED DEADLOCK RESOLUTION MODEL	56
5.1	Deadlock Detection	57
5.2	Edge Detection and Addition	58
5.3	The Proposed Deadlock Detection and Resolution Algorithm	59
6.	SUMMARY AND CONCLUSION	61
	REFERENCES AND BIBLIOGRAPHY	64

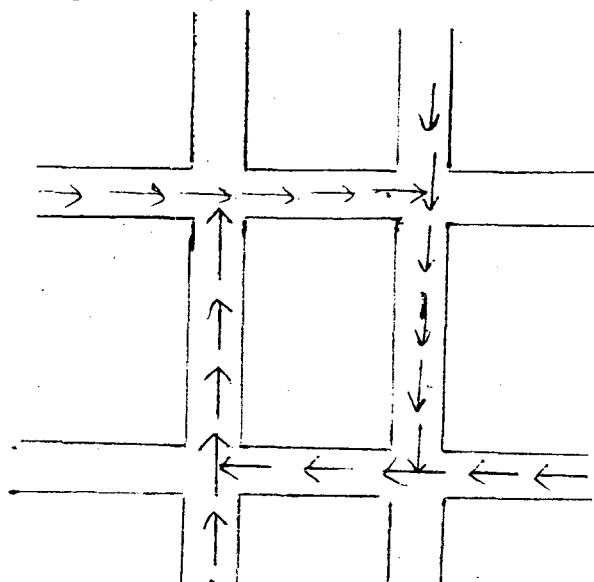
CHAPTER I

INTRODUCTION

Modern database systems are designed to support high degree of parallelism by ensuring that as many processes are permitted to run as possible. The benefits of such systems are two-fold; one, to maximize resource utilization and two, to minimize CPU waiting time (overall). The processes are local in the sense that these utilizes only local resources (i.e. the databases). In case of distributed database systems, processes make access to many databases and hold resources for the period of execution of transaction. It increases possibility of existence of transactions, waiting for resources. In such an environment many processes refer to databases at different physical locations.

Deadlock is not unique to database environment. In our day to day life, there are many situation where deadlock occurs. Take the following (Fig 1.1) example of road traffic. Cars are coming from all directions (west to east, north to south etc.). Cars are moving in a straight line only.

fig 1.1



Cars going towards east are blocked by cars going towards south which are blocked by cars going towards west and so on. This is an example of a deadlock. This deadlock cycle involves four crossings A, B, C and D. In fact this is example of distributed deadlock. Where individual sites (crossings A, B, C and D) have knowledge of local blockade, but no knowledge.

1.1 DEADLOCK IN DATABASE

The system is said to be in deadlock state if there exists a set of transactions, such that every transaction in the set is waiting for resources locked by another transaction in the set. More precisely, let the set of waiting transactions be $(T_0, T_1, T_2, \dots, T_n)$. T_0 is waiting for some resource held by T_1 , T_1 is waiting for a resource held by T_2 and so on and finally T_n is waiting for a resource held by T_0 . In this situation entire set of transactions is deadlock and can not proceed. The only solution of this deadlock is to abort a transaction or rollback it to some previous stage.

The following four conditions must hold simultaneously for a process to be deadlocked [PET85].

- i) Resources are shared in mutually exclusive fashion (at least one exclusive lock must be on some data item or exclusive lock is requested);
- ii) Transactions hold a lock on data item and wait for a resource held by other transactions;
- iii) The resources are usually not taken away from transactions; and
- iv) A cyclic wait condition exist.

However only fourth one is important in database systems. In databases mutual exclusion must be ensured by designer. A process can not release a resource without entering in shrinking phase, if two-phase locking is used (no pre-emption) and it has to wait for another resource.

As far as pre-emption is concerned, it is applied when breaking of deadlock is required either by partial or full rollback. However the circular wait condition must exits.

In distributed database environment, let us consider three sites S1, S2, and S3. Three transaction T1, T2 and T3 are executing at site S1, S2, and S3 respectively. Resources r1, r2 and r3 are at site S1, S2 and S3 respectively. In the beginning T1 holds lock on r1, T2 on r2 and T3 on r3. There is no resource wait condition. Now, T1 requests for r2, T2 for

r3 and T3 for r1 in any order. None of transaction will get lock on requested resource. This is a cyclic wait condition (T1 – T2 – T3 – T1) No transaction can make any progress, unless an action of roll back of any one of transaction in the cycle.

1.2 DATABASE DEADLOCK MODELS

In database systems many types of deadlocks are possible depending upon the processing environment. Edger Knapp, [EGR87] has discussed deadlock models. The few relevant models are as follows.

1.2.1 One Resource Model

It is the simplest model. One transaction can make request for only one resource at a time. So each transaction has at the most one outstanding request at a time. The request for another resource can be made only after previously requested resource is granted.

1.2.2 AND Model

A transaction can make request for many resources at a time. The transaction remains blocked until all the requested resources are granted

to it. Existence of a cycle is sufficient for deadlock detection in AND model.

1.2.3 OR Model

A transaction can request many resources at a time, if any of requested resource is granted, the transaction becomes active. The transaction remains blocked if none of requested resource is granted. This model fits for replicated database with read only transactions. Existence of a cycle is not sufficient in OR model for existence of a deadlock.

1.2.4 AND-OR Model

AND-OR model is generalization of AND model and OR modal. In this model requests are made as combination of AND and OR. For example request ((x or y) and z) or r is possible. Existence of cycle not sufficient for existence of a deadlock.

In distributed database environment a transactions make request for a number of resources. So deadlock if occurs fits in the AND model. In chapter II the deadlock detection algorithms deal with AND model.

1.3 RESOURCE AND COMMUNICATION DEADLOCK

Two type of deadlocks are described in the literature: communication deadlocks and resource deadlocks. A deadlock is a situation in the system in which each process is waiting for a resource held by another process in the set. N processes in a set are said to form a deadlock if each process in the set is waiting for a resource held by another process in the set. In the resource deadlock resources are data object, records, files, databases etc. A process can not proceed unless it acquires the requested resource. In the deadlock situation no process releases a resource. In the communication deadlock resources are messages to be sent by processes in the set, each of which is waiting to receive a message from another process which is part of deadlock cycle. If the processes are deadlocked no process sends a message.

Deadlock appearing in database systems are different from what we study in Operating Systems. In database systems, database files are considered as resources. There can not be multiple entries of resources.

A resource is unique even it is replicated. In distributed systems a write lock (exclusive lock) must be achieved from each replicated site. However in the Operating System environment, request for a tape drive is

met just by allocating any tap drive to the process. If process requires a write lock (exclusive lock) on a file X and many copies of file X are in the system then by allocating any copy of X is not sufficient.

1.4 LOCKING

Resource allocation to the transaction in the database system is based on locking data items [KOR86]. Two types of lock are possible on data items, shared lock (for reading the value of data items only) and exclusive lock (for updating the data items). Locks are necessary for consistency of database. The following example demonstrates the need of locking a data item.

Let T1 and T2 are two transaction running in parallel in the system. T1 and T2 read a data item x. T1 increases its value by a, and new value $x + a$ is written back. T2 increases its value b and new value $x + b$ is written back. Transaction T1 prints that it has increased its value by a and T2 prints that it has increased its value by b. User expects that final value should be $x + a + b$, but the final value will be either $x + a$ or $x + b$ depending upon which transaction has written back the value first. This is because transaction are not running one after another. But running of transaction one after another is not necessary. It is necessary that the

accesses to common data items should be serial. Multi-programming systems are based on simple principle of serializability. If many processes are running concurrently, they may access many data items. The serializability can be achieved by two phase-locking.

1.4.1 Two-Phase Locking

In two-phase locking a transaction has two phases, growing phase and shrinking phase. Initially, a transaction remains in the growing phase, acquires locks as per the requirement. Locks can be upgraded (read lock to exclusive lock) but these can not be released in this phase. In the second phase locks are either down graded (Exclusive lock to read lock) or released. No fresh lock can be acquired by transaction in this phase. Two-phase locking ensures serializability [KOR86].

In a single user single tasking system, a new process can start only when a running process has been completed/aborted/terminated. In a multi-programming system, many processes may started simultaneously. But the overall effect should be same as these process are executed serially (in some order). So ensuring serializability is essential in multi-programming systems.

A resource lock is used to achieve serializability. If a process P wants a resource X to read/update its value, it must get appropriate lock on it. Once a resource X is locked by a process P, it can release the lock only after completion/termination of process P. No process can get a lock on X (the lock which is incompatible with earlier lock on X) before completion/termination/abortion of process P. Two-phase locking is used in the most of database systems.

Use of two-phase locking may result in a deadlocks. A resource remains locked even after all the works regarding its use it over. Suppose a transaction P locks a resource X which is a particular bank A/C. The only operation regarding A/C X is, withdrawal of Rs. 100/-, in the beginning. The transaction completes all other remaining task in time t after making entire period of execution of transaction P although no further operation is required on X. Two-phase locking will keep resource X locked for duration of execution of transaction T. Suppose any other transaction requires resource X, it has to wait till completion of transaction T. So there is a possibility of a deadlock.

1.5 DEADLOCK HANDLING APPROACHES

There are three approaches to deal with this problem: deadlock prevention, deadlock avoidance and deadlock detection combined with

recovery [FER87]. First two approaches ensures that system never enters in deadlock situation but the third one allows the system to enter deadlock situation and recover from it. Deadlock prevention prevents a deadlock by dictating the way request are made, but this puts the additional restrictions on concurrency and on the throughput of the system [PET 85]. Deadlock detection and deadlock avoidance technique provide better concurrency. Waiting transactions (Waiting for a locked resources) are reported. A cycle in TWFG (Transaction Wait For Graph) is searched.

1.5.1 Deadlock Prevention

Deadlock prevention is a technique which ensures that there will not be any deadlock, but it will affect concurrency All the resource required by the process are locked before the process actually starts. But there are many problems in this technique other than the concurrency. Resources may be databases, to acquire all the resources at one time the process must lock all the resources which may be required by the process later on. But this will have three main problems [PET85]: first, there is no way to findout the all resources which the process may use in its duration of execution and mode (read/write mode) in which those must be acquired; second, a tremendous waste of resources, which result in loss of concur-

rency. Starvation is another major problem. A transaction which needs some popular data items has to wait indefinitely because at least one of the data item in general will be locked by some other transaction; and the transaction will have to wait for long time.

Another approach for preventing deadlock is to use pre-emption and transaction rollback. Two different deadlock prevention scheme using time stamped have been proposed [PET85].

The wait-die scheme is based on non-pre-emptive rollback scheme. If T_i (i th transaction) has requested a data item currently held by transaction T_j and $i > j$, then T_i will rollback itself, otherwise T_i will wait.

The second approach is called wound-wait scheme based on pre-emptive technique. If T_i has requested some data item currently held by T_j ; it would initiate rollback of T_j (pre-emption) if T_j is younger than T_i (i.e. $j > i$) otherwise it will wait.

In the both these techniques a transaction with smaller time stamp will not be rolled back. After a transaction is rolled back, the value of time stamp associated with transaction is preserved. So the transaction is not

rolled back again and again hence this technique removes any possibility of starvation.

In distributed database management system the time-stamp technique can be used as in [SIN 85]. A transaction entering in the system will have two values assigned to it i.e. (t,s) where t is actual time of entering the system and s is the identity of site where the transaction entered in the system. The ordering of two transactions $T_1(t_1,s_1)$, $T_2(t_2,s_2)$ is done as follows: T_1T_2 if $t_1 < t_2$; or when $t_1 = t_2$, T_1T_2 if $s_1 < s_2$. The same technique can be used here. But few efforts [FER87] have been made in this direction, because prevention technique may increase rollback overhead and system performance will go down.

The above two techniques look similar but these have remarkable difference. In wait-die scheme, if T_i requests for a data item which is locked by T_j and t_i is an older transaction, it will wait. But if T_i is not older than T_j , than it will be rolled back. T_i restarts and rolls back unless T_j releases the resource held (i.e. T_j completes) by it. However in wait-wound an older transaction never waits for younger one. Younger one is allowed to wait. So there are less number of rollbacks.

1.5.2 Deadlock Avoidance

Deadlock prevention algorithms prevent deadlocks by restricting how requests can be made. A major drawback of this approach is that it reduces system throughput. In deadlock avoidance technique resources are allocated to process. The maximum number of resource of each type are declared in the beginning. A safe sequence is decided. In the safe situation each process can request the resource and resources are granted to processes. If the resources are not available and system is in safe state processes wait. However there must be an algorithm to allocate resources and declare unsafe situation. The unsafe situation if ignored the processes may complete without trouble but there is no guarantee of freedom from deadlock. This technique is used and may be useful in Operating System environment. In the case of database systems, the technique has been suggested by [FER87].

1.5.3 Deadlock Detection

Deadlock avoidance and deadlock prevention are conservative approaches. If these approaches are used in a systems, it will reduce system throughput drastically which can lead to a big loss in the case of large systems. Deadlock detection approach is a risk taking approach. It allows

the system to enter deadlock situation, and then tries to recover from it. The resources utilisation in this case is maximum.

Most of the efforts have been made in this area. A lot of algorithms have been proposed. The algorithms for deadlock detection in centralised database system are simpler than that of distributed database systems. But in the distributed system, efforts are being made to find out better algorithms. A number of issues must be resolved. Some of these are communication problem, delays, no centralised memory and inconsistent TWFG.

1.6 OVERVIEW

Issues related to distributed deadlocks is described in this chapter. More details about deadlock detection and recovery follows in the next few chapters. Out of two type of deadlocks (Resource deadlock and communication deadlock), only resource deadlock has been chosen for study. Deadlock environment, safety criterion (through locking) and preservation of consistency are also discussed.

In this report, Chapter II deals with deadlock detection algorithms. These algorithms are described briefly. In chapter III, a comparative

study has been made. In chapter **IV** deadlock resolution criteria is discussed, Taking some representative algorithms into consideration. In chapter **V** a deadlock resolution technique is discussed. A new matrix based approach is described in detail in this chapter. Chapter **VI** covers summary and conclusion.

CHAPTER II

DISTRIBUTED DEADLOCK DETECTION ALGORITHMS

Deadlock detection and resolution is an important problem in distributed database management systems. A number of efforts have made in this area, in the past. A deadlock is a result of holding locks on resources, by transactions during the period of their execution. The two phase locking technique is used in the database environment to acquire and release locks on data items. Locks on data items are released only when transaction reaches to its final stage (i.e. in the shrinking phase).

The techniques of deadlock prevention and deadlock avoidance, if applied reduce concurrence, which is an undesirable result. A lot of work has been done on deadlock detection, which allows better concurrence. Deadlock problem is a part of concurrence control, when many transactions want an access to data items in a mutually exclusive fashion.

In a distributed database system the database is distributed over many sites. These sites may be involved in a deadlock, which is harder to detect because every site has local information. Information transfer among sites is by sending messages. There are delays in message communica-

tion, and it is possible that the message may reach in an arbitrary order. Thus, the sites may have an inconsistent view of the system.

Distributed deadlock detection algorithms are proposed by a number of researchers. It is a preferred approach because the entire system does not depend on a single site for deadlock detection. A few representative algorithms are discussed in the following sections. Based on certain features, algorithms are broadly divided into two major classes.

2.1 TWFG BASED ALGORITHMS

Algorithms of this class prepare/maintain TWFG (Transaction Wait For Graph) in some form or other. TWFG is updated periodically. The graph is checked, for existence of a cycle after every update. Different data structures are used by the algorithms for maintaining transaction related information. All the algorithms of this class don't use TWFG directly. Goldman's algorithm [GOL77] uses OBPL (ordered blocked Process List), which is similar to TWFG.

2.1.1 Goldman's Algorithm [GOL77]

This is based on sending deadlock related information to other sites, so that the other sites may detect the deadlock or send such information to

the next connected site. This algorithm does not prepare/maintain TWFG (Transaction Wait For Graph) at any site. When the process for deadlock detection starts, a transaction is picked up. A global OBPL (Ordered Blocked Process List) is prepared as follows. A waiting process P1 is chosen, it is the first member of OBPL. It sends OBPL to the process P2 which is holding a resource required by P1. P2 sends OBPL to a process P3 which is holding a lock in inconsistent mode on a resource required by p2 and so on. If the last process is not blocked by any other process, no action is taken. The deadlock is not detected but it does not mean that no deadlock exists. If OBPL reaches a process already in OBPL, this indicates existence of a deadlock.

2.1.2 Isloor-Marshland's Algorithms [ISL78]

This method detects deadlock at the earliest possible instant. This is based on the concept of reachable sets. A reachable set for the process P is the set of the all the processes which can be reached from process P in TWFG, which is a directed graph (this set does not contain P itself unless we reach process P after starting from P). For every process, each of the sites of the system prepares/maintains a system graph and a reachable set for each node. If it contains the process for which it is maintained then there is a deadlock. Whenever a resource is released or

there is a request for a resource, which is already locked by some other transaction, the information is broadcasted to all the participating sites

2.1.3 Menasce Muntz Algorithms [MEN79]

In this algorithm only two ends of TWFG are transmitted to concerned sites. Each site maintains TWFG. On receiving information from other sites, the cycle may be completed at a participating site, then the site declares a deadlock. Any further activity in detection will remain suspended unless the deadlock is resolved. The following definitions and data structures are used.

- a. Each site k has a data controller S_k which maintains TWFG (K) as per protocol.
- b. The arc $T \rightarrow T'$ in TWFG denotes that T is waiting for T' to release a resource (not necessarily required by T).
- c. A non blocked transaction is one which does not have an outgoing edge.
- d. A blocking set(T) of transaction T is set of all non-blocked transactions which can be reached following T in TWGF. There may be different blocking sets of a transaction T in different sites.
- e. The pair (T, T') is a blocking pair if T' is in blocking set(T).

- f. $Sorg(T)$ is the site where T enters in the systems.

There are two rules of the communication protocol.

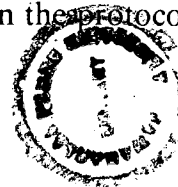
- Rule 1. Site k receives a request for resource R for transaction T . S_k can not grant the resource because it has been held by the transactions $S = (T_1, T_2, T_3, T_4, \dots, T_n)$ in inconsistent mode. Add an arc $T \rightarrow T'$ in TWFG (k) (Transaction Wait For Graph at site k) for each T' in S . If there is a cycle, it is a deadlock. Else for each T' in blocking set of t , send blocking pair (T, T') to originating site of transaction T $sorg(T)$ if $sorg(T)$ is not S_k and to $sorg(T')$ if $sorg(T')$ is not S_k .
- Rule 2. A blocking pair (T, T') is received by site k . Add an arc $T \rightarrow T'$. If a cycle exists it is a deadlock. Else if T' is blocked and $sorg(T)$ is not S_k for each transaction T' in the blocking set (T) blocking pair (T, T') to $sorg(T')$ if $sorg(T')$ is not S_k .

The protocol is incomplete in the sense that it does not tell what should be done in the case of release of a blocked resource. However one more flaw was detected and corrected by Gligor and Shattuck [GLI80]:

Gligor and Shattuck [GLI80] pointed out that the communication protocol defined in [MEN79] will work perfectly if there is no delay. If

there is delay in receiving the information the out of order graph updates will take place and deadlocks will not be detected.

Suppose T1 is waiting for a reply from a remote site about a requested resource. Meanwhile a request for a resource locked by T1 arrives, since the resource can not be granted so two ends to graph will be sent to the concerned site after adding edge to T1. Suppose now T1 receives message that the requested resources is not granted, the edge will be added but the earlier sent blocking pair will not be corrected. This will lead to non detection of a deadlock. The following corrections in the protocol are suggested.



All the data structures used in [MEN, 1979] are also used here. The following additional structures are introduced.

- a. A potential blocking set(T) is a set of all non-blocked waiting (waiting for reply from remote) transactions which can be reached from T in TWFG.
- b. A pair (T,T') is said to be a potential blocking pair if T' is in potential blocking set(T).

Dissertation

681306

J776

st

TH-3636

Rule 0:- When a transaction requests a non local resource, it must be marked as waiting. Rule 2 must be modified as follows.

Rule 2 A blocking pair (T, T') is received. Add an arc $T \rightarrow T'$ in TWFG (k). If a cycle is formed, then a deadlock exists.

Rule 2.1 If T' is blocked and $sorg(T)$ is not S_k , then for each transaction T'' in blocking set(T), send blocking pair (T, T'') to $sorg(T'')$ if $sorg(T'')$ is not S_k .

Rule 2.2 If T is waiting and $sorg(T)$ is S_k , then for each potential blocking pair (T'', T) send blocking pair (T'', T') to $sorg(T')$ if $sorg(T'')$ is not S_k . Then discard the potential blocking pair (T'', T) and erase the waiting mark from T .

2.1.4 Obermarck Algorithms [OBR82]

In Obermarck algorithm a path (part of cycle) is transmitted to a site where another part of cycle is expected. The algorithm performs as per following rules.

1. Each site maintains its local TWFG. If a non-local resource is requested, an agent representing the transaction is created at requested site. The communication link between two is established.
2. If there exists a cycle in TWFG which includes transactions and their agents, then there is a deadlock.
3. If there exists a cycle including external wait and external request (both are represented as EX in the TWFG), the following action is taken. This cycle is called potential wait cycle.
 - 3a. If the transaction/agent which is waiting to get resource/reply has lesser number in lexical ordering (lexical number is allotted when transaction enters in the system) than the first transaction in the string then
 - (i). Make a string starting with node EX and terminating with last transaction which is waiting for a remote resource i.e. a cycle EX-4-2-3-EX will become string EX-4-2-3.

- (ii) Send this string to all sites for where last transaction in string is waiting to receive resource, if lexical order of first transaction in the string lexical order of last transaction n the string.
- 3b. Receiving site updates TWFG in that site.
4. If there is a deadlock break this by arbitrary selection of the victim. Broadcast this news to all the sites, which must be remembered by the sites till next iteration.
5. Discard the node (all the edges going out and coming in) which has been selected as a victim.

2.1.5 Ho-Ramamurthy Algorithm [HOR82]

Ho-Ramamurthy algorithm is distributed algorithm but it is closer to centralised algorithms. This does not use TWFG but finally a graph is made for detection of a deadlock. Two algorithms one phase/two phase are described. Number of messagewise and also timewise one phase algorithm is better than two-phase algorithm. The algorithm works as follows. Each site maintains two tables:

Resource Status Table (RST)

It has entries for all local resource/indicating by which process the resource is locked and all the other processes which are waiting for these resource. The entry in RST is made at the time of allocation/denial of resource.

Process Status Table (PST)

Is has entries for all local processes. The process status table indicates the locked resources by the transaction and the resources for which the transaction is waiting. The entry/update in PST is made at the time a request is made or resource lock is received by transaction.

Periodically detection of deadlock starts. A site is chosen as a control site. Both the tables from each site are sent to the chosen site. Taking common entries from both the tables the site makes a TWFG; if there is a cycle within the TWFG, it indicates a deadlock. This site releases its control and another site gets control. The main advantage of algorithm is that it is simple to implement.

2.1.6 Discussion

Five algorithms of this class are discussed in this section. Goldman's algorithm [GOL77] appears to be the first algorithm for distributed deadlock detection. Some efforts in this area were made earlier, this includes [ESA76] and [PEE78].

Goldman's algorithm for deadlock detection is flawless. One of the plus points of this algorithm is that there is no need to prepare TWFG and maintain it. So the memory requirement for this algorithm is small. But there are many disadvantages, such as, even if no cycle is detected there is no guarantee that the deadlock does not exist. If in parallel many processes start independently (for detecting a deadlock), some of them may end up detecting the same deadlock. It is also not an efficient algorithm, the possibility of a deadlock can not be decided. So searching for existence a cycle among transactions is started even if there is no possibility of a deadlock, so a lot of CPU time will be wasted if this algorithm is implemented for deadlock detection. The algorithm has limitation that this can be used only for single resource model. No deadlock situation can be decided unless all $n-1$ (n is total number of the process in the system) processes are checked.

Another algorithm for deadlock detection in the distributed environment is the Isloor Marshland's algorithm [ISL78]. This algorithm detects deadlocks as fast as they occur in the system. Even if many sites fail the system may still be able to detect deadlock. But there are some disadvantages; because it requires a broadcast based communication network.

In the case of Menasce Muntz algorithm [MEN79] many issues are left unresolved and unexplained. What action should be taken, when a resource is released, is not mentioned. Gligor and Shattuck [GLI80] have suggested a small modification in the protocol.

Obermarck algorithm [OBR82] is a path-pushing algorithm. Only potential multisite cycles and victims are transferred, potential cycle is transmitted in one direction only. So it reduces message transfer to half. Failure of sites also does not affect this algorithm.

There are certain disadvantages of this algorithm. The same deadlock cycle may be detected by two sites which in turn may decide two different victims. Hence for breaking a single deadlock two transactions will be rolled back. Even with two different deadlock cycles, it is possible to break both the cycles by just rolling back a single transaction; in this

algorithm there are no such possibilities. The algorithm is iterative therefore it takes a lot of time to detect a deadlock.

Ho-Ramamurthy algorithm [HOR82] a (One phase algorithm) is broadcast based algorithm. It uses broadcast based communication heavily. This algorithm detects distributed deadlock. Also all the sites share this responsibility periodically. But at a time only one site does the entire effort of deadlock detection. In this way this algorithm is closer to centralized deadlock detection algorithm. A flaw in the algorithm is explained in [JAG83].

Only a few a algorithm are discussed in this section. These algorithm are representative algorithms. Obermarck's algorithms, [OBR82] is the best among TWFG based algorithms because it uses the least number of messages in comparison of others. But there are certain drawbacks of this algorithms. This is an iterative algorithm therefore it is slow algorithm.

2.2 NON TWFG/PROBE BASED ALGORITHM

The second class of algorithms is probe based algorithm. Probe is a special message which is send by one transaction to another. Probe

travels from transaction to transaction unless either it is discarded or a deadlock is detected. The size of a probe message and the condition for discarding a message varies from algorithm to algorithm.

2.2.1 Chandy-Misra-Haas Algorithm

Chandy-Misra-Haas algorithm [CHA83] (which is a version of [CHA82], is the first algorithm to use probe for deadlock detection. The other algorithms, [HAA83] and [SNH85], are variations of this algorithm. Each site has a copy of algorithm. A probe (x,y,z) is initiated for idle process T_x which is waiting for a resource from process T_y , which in turn is waiting for T_z . A probe message (x,y,z) is sent to controller of T_z by controller of T_y . If $z=x$, this is a deadlock. The probe is discarded if T_z is active. An example with intersite probe messages is shown in Fig 2.1.1

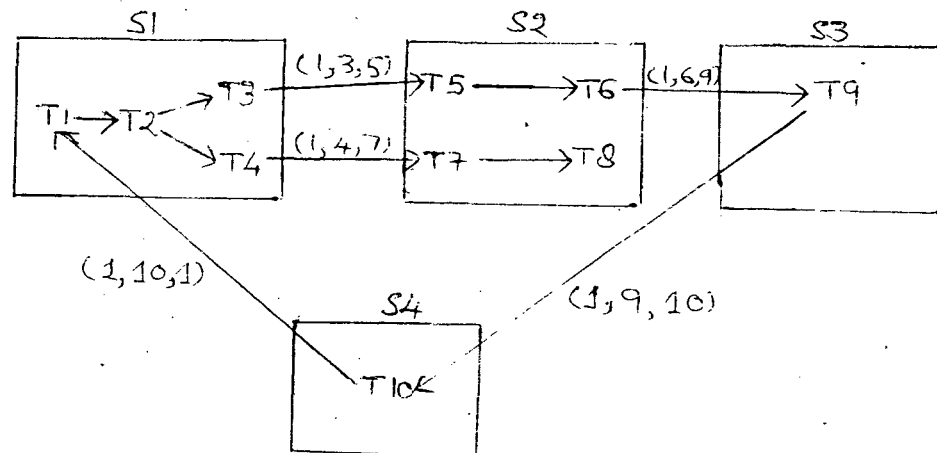


fig. 2.1.1

T1 initiates probe. Controller C1 sends (1,4,7) to C2. C2 discards (1,4,7) and propagates (1,6,9) to C3. In turn C3 sends (1,9,10) to C4 and C4 send (1,10,1) and deadlock is detected at that site.

2.2.2 Sinha-Natarajan Algorithm [SNH85]

Sinha-Natarajan's algorithm uses new data structures, probe and probe-Q. No TWFG is used anywhere in the algorithm. Probes are transmitted according to some rules; when a probe reaches its starting node a deadlock is detected. Priority is simply seniority of transaction. An older transaction has higher priority; in the case transactions are entered at the same time, the site identity is taken as a decisive factor. A transaction is said to be facing antagonistic conflict when it is waiting for a lower priority transaction. The following data structures and definitions are used.

Antagonistic Conflict

When a higher priority transaction is waiting for lower priority one for a resource, this is called antagonistic conflict.

Priority

Two sites T1 and T2 enter in the system at site S1 and S2 respectively. T1 enters at time t1 and T2 at t2. Then the priority of transaction is defined as follows.

Priority(T1) > Priority (T2) if $t1 < t2$
if $t1 = t2$ then
Priority (T1) > Priority (T2) if $S1 > S2$

Probe

When there is an antagonistic conflict, the higher priority transaction sends a message (i,j); i is the identity of higher priority transaction and j is identity of lower priority transaction. The message is called probe message or probe sometimes.

Probe-Q

Every incoming probe (if not discarded) is stored in Probe-Q, which may be any data structure like array, linked list etc.

The actual protocol has following three rules:

1. A data manager initiates probes in one of the following situations.

- a. If a locked resource is requested and $\text{requester} > \text{holder}$, then data manager sends the probe to holder.
- b. When a holder releases a resource and it is allocated to a transaction (say a new holder) the if more requests are pending s.t. $\text{requester} > \text{holder (new)}$, the data manager initiates probe. This probe is sent to new holder. In the probe the following assignments are made.

$\text{junior} = \text{holder};$

$\text{initiator} = \text{requester}.$

2. Each transaction maintains a probe-Q. A transaction sends the probe to data manager, where it is waiting in one of the following cases:
 - a. When transaction receives a probe it performs the following:

$\text{if } \text{junior} > T \text{ then } \text{junior} = T;$

Save it into probe-Q;

If T is in wait state send this probe to data manager where it is waiting.

- b. when a transaction goes to wait state after requesting a resource. It sends a copy of all probes stored in probe-Q to data manager.
3. When DM receives a copy of probes it does the following: if holder > initiator then discard it; else if holder < initiator then send it to holder; Else (when initiator = holder) report deadlock.

Alok Choudhary et. al. [ALO89] have pointed out flaws with the algorithm. It does not detect all existing deadlocks, and sometimes deadlocks detected are false deadlock. These drawbacks are discussed below.

Undetected Deadlock

The error can best be explained by an example. The situation is as drawn below in Fig. 2.2.1 initially.

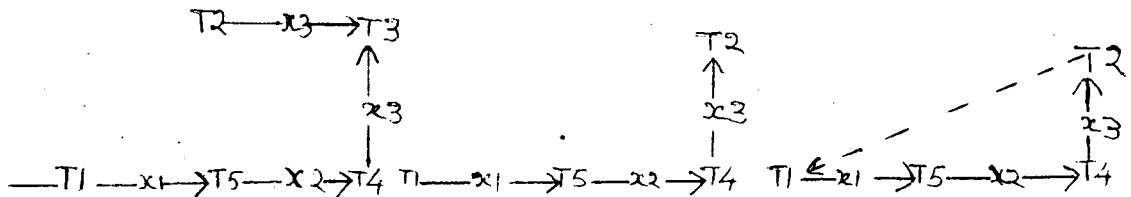


fig 2.2.1

fig 2.2.2

fig. 2.2.3

DM (x1) initiates probe (T1,T5) that propagates to T5, from there it will be passed to DM (x2) and so on. Finally it will reach T3 and stop there. The probe (T1,T5) will be stored in the transactions T5, T4 and T3 in their probe-Qs. Now let T3 abort and T2 get a lock on x3. The situation now will be the same as in Fig 2.2.2. Now, let us assume that T2 requests for a locked resource by T1 (Fig. 2.2.3), the probe-Q of T2 is empty so no probe will be sent to T1. T2 is also not facing antagonistic conflict. But this is a deadlock cycle which will never be detected.

False Deadlocks Due to External Probes

This algorithm also detects false deadlocks. Let the situation be like in Fig 2.3.1.

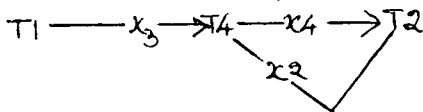


fig. 2.3.1



fig.2.3.2

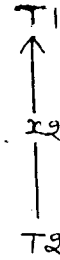


fig.2.3.3

Transaction T1 has locked data item x1 (which is not shown in the figure, because this data item is not requested by any other transaction), T2 has locked x4 and T4 has locked x2 and x3. In addition T1 has requested x3, T2 has requested x2 and T4 has requested x4. A probe (T1,T4) will be initiated by T1 and subsequently will be stored in probe-Q of T4 and T2. The deadlock T4 — T2 — T4 will be detected and T4 will be rolled back as a victim. The clean message will be sent within the cycle and finally it will reach the initiator T2, where it will be discarded. After T4 aborts T1 and T2, both get the requested resource and become active. Now let us assume T2 makes a request for the resource held by T1. The probe (T1,T4) is already in probe-Q of transaction T2; this probe will be transferred to T1 which will detect deadlock. By Fig. 2.3.3 it is clear that this is not a deadlock situation.

False Deadlocks Due to Old Information

Consider the situation if as in Fig. 2.4.1, there is deadlock between T2 and T4. Transaction T1, T2, T3 and T5 are waiting directly or transitively on T4. The probe (T1,T5) initiated by T1 will be stored in the probe-Q of T4 and T2. After detection of deadlock T4 — T2 — T4, T4 will be rolled back and clean message will be sent. T2's probe-Q will have probe message (T1,T5).

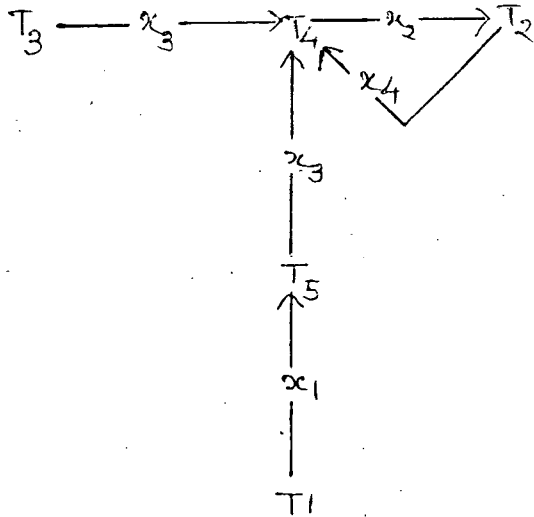


fig. 2.4.1

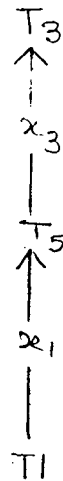
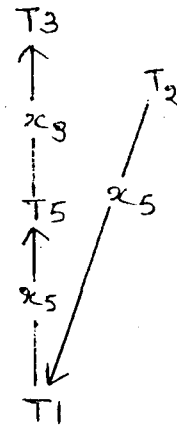


fig.2.4.2



2.4.3

The wait for relationship is shown in 2.4.2, after resolution x_3 is granted to T_3 and x_4 . Now let us assume, T_2 requests for some data item x_5 locked by T_1 (Fig 2.4.3). It is clear from Fig 3.3 that there is no cycle but DM (x_5) will declare a deadlock because it will receive probe (T_1, T_5). Clearly this is a false deadlock. The modified algorithm has been suggested as follows [ALO89]:

1. A data manager initiates, propagates, or reinitiates probes in one of the following situation:
 - a. If a locked resource is requested and requester $>$ holder, then data manager sends the probe to the holder.

- b. When the holder releases a resource and it is allocated to a transaction (say new holder) then if more requests are pending s.t. requester $>$ holder (new), this probe is sent to the new holder.

In the probe the following assignments are made.

junior = holder;

initiator = requester.

- c. When a transaction is completed or aborted it releases its locks. The data manager associated with each released data item assigns the lock for the data item to some transaction waiting for data. Each data manager then requests each the remaining transactions waiting on the new lock to send its probe-Qs to itself. After receiving Probe-Q data manager forwards probes to new holder for which initiator $>$ new holder.

2. Each transaction maintains a probe-Q. A transaction sends the probe to data manager, where it is waiting in one of the following cases.

- a. When transaction receives a probe it performs the following:

If junior $>$ T then $j = T$;

Save it into probe-Q;

If T is in wait state send this probe to data manager where it is waiting.

- b. When a transaction goes to wait state after requesting a resource, it sends a copy of all probes stored in probe-Q to data manager.
 - c. If a transaction is waiting and receives a request for its Probe-Q from a data manager where it is waiting, it sends a copy of its Probe-Q to the data manager.
3. When DM receives a copy of probes it does the following:

If holder > initiator then discard it;

else if holder < initiator then send it to holder.

Else (when initiator = holder) report deadlock.

In Sinha-Natarajan algorithm [SNH85] in the best case the total number of message transferred is $2*(n-1)$ and in the worst case it is $n(n-1)$. The minimum priority transaction (the youngest one) is taken as the victim to be rolled back. Only once probe is sent from one antagonistic conflict. The probe messages are stored in probe-Q of every transaction wherever

it reaches, and used later on whenever the transaction makes a request for a locked resource.

2.2.3 Discussion

Sinha-Natarajan algorithm [SNH85] is best among probe the based algorithms, it passes the minimum number of messages. The message sent earlier is also used later on.

Many other algorithms [NAT86],[MAR88], [HAA83], [ELA86a] and [BDL86] are not discussed here. These algorithms are similar to algorithms discussed in this chapter.

A number of corrections have been suggested in deadlock detection algorithms. [MEN79], [HOR82], [OBR83], [SNH85] are modified. Some of these algorithms have been proved correct using intuitive logic. This is because there no formal method for proving an algorithm correct.

Generating a proof is also difficult because a deadlock cycle may be formed in many ways. However a cycle of length two is most probable [Gra,1981]. Delays create more severe problems. Because of arbitrary

delays the order of receiving messages is lost. An old edge deletion message may reach late, which may lead to a false deadlock.

The main problem in developing efficient deadlock detection algorithm for distributed system is lack of global memory. The sites are distributed; so the transaction wait for response. The wait is uncertain in the sense that the transaction does not know, that whether the resource will be allocated or not. The wait state (wait for response) can change at any time to either active, or blocked state. This is the cause of error in the most of the algorithms.

CHAPTER III

COMPARISON OF DEADLOCK DETECTION ALGORITHMS AND DEADLOCK RESOLUTION

A few selected algorithms for deadlock detection in distributed database systems have been discussed in the previous chapter. The algorithms have many common features and these differ in many ways. Sinha-Natarajan algorithm [SNH85], Chandy-Misra algorithm [CHA83] and Menasce-Muntz algorithm [MEN79] detect a deadlock and but full deadlock cycle is not reported. It is insufficient information for selecting a victim for roll back purpose. However [SNH85] reports a deadlock with lowest priority transaction in the deadlock cycle, which is selected as the victim. Algorithms also differ in terms of the messages are transferred, maintenance of TWFG at each site, size of message (information), data structure used. Based on these factors the algorithms are classified broadly in two groups, latter in this chapter.

A better way of comparison of different algorithms is by listing out all the components viz delays, number of messages, data structure and type of communication used in algorithms. Delay imposed by algorithms affects the efficiency of a deadlock detection algorithm. Number of messages increase the cost of communication between sites. Use of complex data structures require more

memory and processor time for the algorithm. Such a comparison is shown in table 3.1. It is not possible to compute exact number of messages in some cases, so upper bound is given in those cases.

Goldman's algorithm [GOL77] has limitation that transaction may have at the most one outstanding request. It detects deadlock by passing messages to transactions. Such messages are discarded many times, before a deadlock is detected. If deadlock exists in the system and involves n sites, if there are m transaction in the deadlock, the deadlock will be detected before $m.n$ messages are sent. But this does not includes the number of messages sent before existence of deadlock. Deadlock does not occur very frequently in database. Goldman's algorithm if implemented will always try to detect a deadlock without any such possibility, and a number of messages will be sent and discarded. So the communication line will heavily be used without any result.

The algorithm proposed by Isloor and Marshland [ISO78] maintains reachable sets for each transaction. Each site also maintains TWFG. A message is transferred whenever a resource is acquired/released updates TWFG. But this algorithm requires a broadcast based network. However this algorithm does not introduce any delay. A deadlock if one exists is reported immediately.

The Mense-Muntz algorithm [MEN79] (modified by Gligor and Shattuck [GLI80]) sends only two ends of graph to concerned sites. This infact reduces communication of unnecessary detail. The cycle detected at any site is not the actual cycle which exists in the system but it is a curtailed cycle. It also does not support deadlock resolution. All the transactions involved a in deadlock cycle are not known. So deadlock is resolved by arbitrary aborting a transaction in the reported cycle.

COMPARISON TABLE

S. No	Name of Algorithm	Message Communication Type	Message Size	Inherent Delay in Processing	No. of Message	Data Structures Used	Limitation and Applicability
1.	Goldman's Algorithm	Mess- age based	Small (vari- able)	Delay in Creation and send	n to m.n	OBPL	Process May Have Only One Outstanding Request
2.	Isloor Marshland's Algorithm	Broad- cast Based	Small (vari- able)	No Delay	r(n-1)	Reachable set, TWFG	Requires Broadcast Based Communi- cation System contd...

S. No	Name of Algorithm	Message Communication Type	Size	Inherent Delay in Processing	No. of Message	Data Structures Used	Limitation and Applicability
3.	Menasce Muntz's Algorithm	Mess- age based	Small (cons- tant)	Delay Beca- use a Mess- age Creates a New Message	$m(n-1)$	Blocking Pair	TWFG
4.	Obermarck's Algorithm	Broad- cast Based	Medium (Vari- able)	Delay Impo- sed by algo- next Messa- ge Must be Transferred Only After First Iteration is Over	Less Than $n(n-1)/2$	TWFG	Requires Broadcast Based Communi- cation System. Strings
5.	Ho Ramamurthy Algorithms	Broad- cast Based	Large (Vari- able)	Control Site Waits to Receive Tables	$2(n-1)$	RST PST	Requires Broadcast Based Communi- cation System.
6.	Sinha Natarajan algorithm	Probe Based	Small (Cons- tant)	Delay in Communi- cation Probe	$2(m-1)$ to $m(m-1)$	Probe Probe-Q	Identifies a victim for rollback

m = Number of Processes in Deadlock Cycle

n = Number of Sites in Deadlock Cycle

r = Total Number of Updates in TWFG Before Deadlock is Detected.

PST = PROCESS STATUS TABLE

RST = RESOURCE STATUS TABLE

The algorithm proposed by Ho and Ramamurthy [HOR82] is a broadcast based algorithm. Responsibility of deadlock detection is shared by all sites, in turns. The algorithm requires a number of messages to be broadcasted, without knowing any possibility of a deadlock. If communication cost between sites is ignored, this is a good algorithm, as it is simple to implement. It uses consistent information for preparation of a TWFG

Obermarck's algorithm [OBR82] is also a broadcast based algorithm. This algorithm maintains TWFG at each site. A string of waiting transactions is sent to concerned site if there is a possibility of a multisite cycle. The string of waiting transactions is sent to only one direction, in order to reduce number of messages to be sent to half. The algorithm is iterative algorithm and can not detect a deadlock immediately, as processing introduces delays in message transfer.

The algorithm proposed by Sinha and Natarajan uses a special message called probe (a small, fixed size message). A probe message is transmitted only when

there is a possibility that no other probe message can detect a deadlock. Probes are also stored in probe-Q. Stored probes are used later on. So this algorithm cuts down un-necessary message communication. Probe also carries a possible victim in the case a deadlock is detected.

Probe based algorithms use minimum resources of the system for deadlock detection because no data structures like TWFG and tables are maintained. Sinha-Natarajan's algorithm also resolves deadlock by rolling back minimum priority transaction.

In Goldman's algorithm creation and updation (sending to other sites) of OBPL is uncertain, frequent creation and updation of OBPL increases communication cost. The number of messages (n to $m.n$) is not an exact bound, the bounds are correct only when deadlock exists. In Isloor-Marshland number of messages transferred are also uncertain, it depends on number of updations in TWFG.

than are in Menasce-Muntz algorithm, but there is delay in detection of deadlocks as the algorithm is iterative. Being a broadcast based algorithm it is not suitable for all the environments. Ho-Ramamurthy algorithm seems good by number of messages, but every site gets control periodically. A number of messages are received by control sites but can not be used later on. It is an efficient algorithm with high communication cost. Sinha-Natarajan's algorithm, requires minimum number of messages among all six algorithms. These messages are stored and used later on. It also indicates the victim transaction in the case of deadlock is detected. It is the best algorithm.

CHAPTER IV

DEADLOCK RESOLUTION

In the past, many techniques have been suggested for deadlock detection in distributed systems. Many algorithms have been further modified after studies. Once a deadlock cycle has been detected, the deadlock must be broken. A deadlock can be broken by rolling back any transaction in deadlock cycle. The selection of a good victim transaction for breaking the deadlock cycle is called deadlock resolution. Considering in the roll back of a transaction, the main criterion is that processing overhead should be minimum. Some of the factors which decide the cost of roll back are, CPU time, database resources and user's time. It is possible to consider these factors for rollback of a transaction. Few algorithms take up resolution of a deadlock and selection of victim in the post detection stage. The algorithms are discussed below with suggestion of selection of a victim to be rolled back.

4.1 SINHA-NATARAJAN ALGORITHM [SNH85]

This algorithm talks about deadlock resolution. Every transaction is given a priority number as it enters in the systems. A probe travels in the chain of processes keeping record of junior the minimum priority transaction among travelled processes. When it reaches back to originating

process, a deadlock is detected. The junior which is minimum priority transaction among all the transactions in deadlock cycle, is chosen as victim transaction.

However, there is a problem. The priority is assigned at time when transaction enters in the systems. Minimum priority only indicates that the transaction is the most recently entered transaction among all deadlocked transactions. Being the most recently entered transaction is not sufficient criterion for selection of victim. An old transaction may get blocked immediately as it enters in the system. The most recently entered one may be the transaction which has executed for longest time among all the transactions. If this is the case the most recently used transaction should not be taken as victim. The following criterion may be chosen to select a victim.

The probe message should be a triplet (initiator, junior, weight). The weight of transactions is taken to decide the best victim transaction. The selection of the junior must be based on the weight. The weight can be calculated as follows

$$\begin{aligned} \text{weight of junior} &= C1 * \text{Priority of junior} \\ &+ C2 * \text{Total CPU time taken by junior} + \\ &C3 * \text{The number of resources held by junior;} \\ C1 + C2 + C3 &= 1 \dots\dots\dots \text{Equation 1.} \end{aligned}$$

The priority of transactions is calculated as in [SNH85]. The CPU time of execution of a transaction is either calculated by DBMS or taken from Operating Systems' utilities. Resources held by transaction are known by checking lock information of the transaction. If weights of two transactions are same than priority is taken in consideration.

C1, C2, C3 are weight factors. The value of each factors can be decided as according which part is more important. The following change in the algorithm (complete protocol is described in chapter II) is sufficient for implementation.

Rule 2(a).When a transaction T receives a probe (initiator, junior,weight) it performs following.

if weight of junior > weight of T

then junior = T

rest part of algorithm is same.

CPU time must be taken in consideration, only a process which has been executed very short time must be aborted. Resources held is necessary to take in consideration because transactions wait for resources (in the case that resources are locked by other transactions in incompatible mode) before acquiring them.

4.2 HO-RAMAMURTHY ALGORITHM [HOR82]

In this algorithm each site maintains two tables. Periodically each site becomes control site. Control site asks to each site to send its table. It waits till all the tables are received. The common entries in the both the tables are used for constructing TWFG. If a cycle is found in TWFG it is a deadlock.

Algorithm does not mention how to resolve the deadlock. The modification will give the criterion to select a victim. The following modifications are sufficient for deadlock resolution.

Each process table keeps following additional information about transaction.

- i). Time when transaction entered in the system
- ii). CPU time for each process. Every time a process goes to its wait state, process status table is updated. The priority of two processes (p_1, t_1) and (p_2, t_2) (t_1 and t_2 are entry time of two process respectively), is calculated as follows

$$P(p_1) > P(p_2) \text{ if } t_1 < t_2$$

or

$$\text{if } t_1 = t_2 \text{ then } P(p_1) > P(p_2) \text{ if } s_1 > s_2;$$

S_1 and S_2 are site identity values.

Weight of each process is calculated when cycle is formed as follows.

Weight calculation is necessary only for the processes in the cycle.

$$\begin{aligned} \text{Weight of a process} &= C1 * \text{Priority of Transaction} \\ &+ C2 * \text{CPU Time} \\ &+ C3 * \text{Resource held.} \end{aligned}$$

Resources held can be obtained from any of the tables. The victim is a process of with least weight. When weight of two transactions is same and it is minimum also, then their priority can be taken to decide victim.

A lower priority transaction is rolled back.

4.3 GOLDMAN'S ALGORITHM [GOL77]

Goldman's algorithm which is based on transfer of OBPL, can also be modified to resolve a deadlock. The proposed suggestions are as follows.

Each Transaction when enters in the system is given a priority number. P1 and P2 are two processes entered in the system at time t1 and t2 respectively, their priorities are

$$\text{if } t1 < t2 \text{ } P(P1) > P(P2)$$

or

$$\text{If } t1 = t2 \text{ then if } s1 > s2$$

then $P(P1) > P(P2)$.

Each process keeps record of resources held by it and total CPU time.

Then the weight of process is calculated as follows.

$$\begin{aligned} \text{Weight} &= C1 * \text{Priority of Transaction} \\ &+ C2 * \text{CPU Time} \\ &+ C3 * \text{Resources Held.} \end{aligned}$$

When OBPL completes a cycle, deadlock exists. OBPL takes with it the process identity, priority, CPU time and resources held. At the time when a cycle is detected, weights of different processes are calculated and minimum weight process is taken as victim. In the case when the weight of two transaction is same minimum weight; then minimum priority transaction out of these two is rolled back.

4.4 DEADLOCK DETECTION IN TWFG BASED ALGORITHMS

Two algorithms were proposed for deadlock detection in early 80's. These algorithms work satisfactory after a slight modification. The main overhead of TWFG algorithms is graph remains partially replicated in different sites. Deadlock is detected at any site where cycle is found in the graph, but the cycle is not always the full cycle. (in [MEN79] the cycle

is subcycle of full deadlock cycle). The problem in selecting a good candidate for rollback (victim) is that, each site where the cycle is stored must have the necessary detail required for deadlock resolution.

A simple solution to this problem is that, each time a transaction requests a resource, the details of cost calculation parameters must be sent with the request. This may increase communication cost of algorithm. Cost parameters need to be stored with TWFG. The parameters are sent much before than they are required and must be stored only when process waits. Each time when a part of TWFG is transferred all the parameters can also be transferred. As the deadlock cycle is detected the cost of rollback is calculated to know minimum cost transaction. The cost can be calculated using following formula.

$$\text{Cost} = C1 * \text{Priority of transaction} + C2 * \text{CPU time} \\ + C3 * \text{Resources held by transaction.}$$

where $C1 + C2 + C3 = 1$Equation 1.

Priority of transaction is decided at the time of its entering in system. It is calculated as it was calculated in previous sections.

C1, C2 and C3 are weight factors. The value of weight factors is assigned, according to which part is more important, which may vary as per requirement. Equation 1 puts restrictions on the values, so it make assignment of values easy. However C1, C2 and C3 are all positive. Priority of transaction is basically age of transaction.

In the case weight of two transaction is same then decision can be taken based on priority of transaction.

CHAPTER V

MATRIX BASED DEADLOCK RESOLUTION MODEL

Efficiency of any system depends upon how efficiently the resources within the system are utilized. In a computer system, memory, processor, peripherals, communication system and databases constitute the resources. computer system is tightly connected in such a way that inefficient use of one resource leads to inefficient use of other resources. For example if deadlock exists in the system because of resource allocation graph is cyclic, it halts the processes. The processes hold many resources. If deadlock detection algorithm is not efficient, it results in the loss of throughput of the system.

A deadlock can be broken by aborting any transaction in the deadlock cycle. But aborting any transaction in the deadlock cycle is not an elegant way to resolve the deadlock. Deadlock resolution is technique of selection of a victim transaction (to be rolled back either full or partial). The selection of victim transaction is done, keeping in the view, the minimum waste of the system resources.

Few algorithms are discussed in chapter II. Comparative study is also done in chapter III. These algorithms perform properly within their limitations. The deadlock avoidance technique is taken in this chapter for study. The algorithm uses an elegant representation of transactions waiting for resource. The matrix

representation of TWFG is used. The deadlock avoidance technique is modified by us for deadlock detection and resolution, later in this chapter.

5.1 DEADLOCK DETECTION

Ferenc Belik [FER87] has presented a matrix based deadlock avoidance technique. TWFG is presented in a matrix form. If there are m processes and n resources in the system, the path matrix P_g (the matrix which represents a TWFG) is $m \times n$ matrix

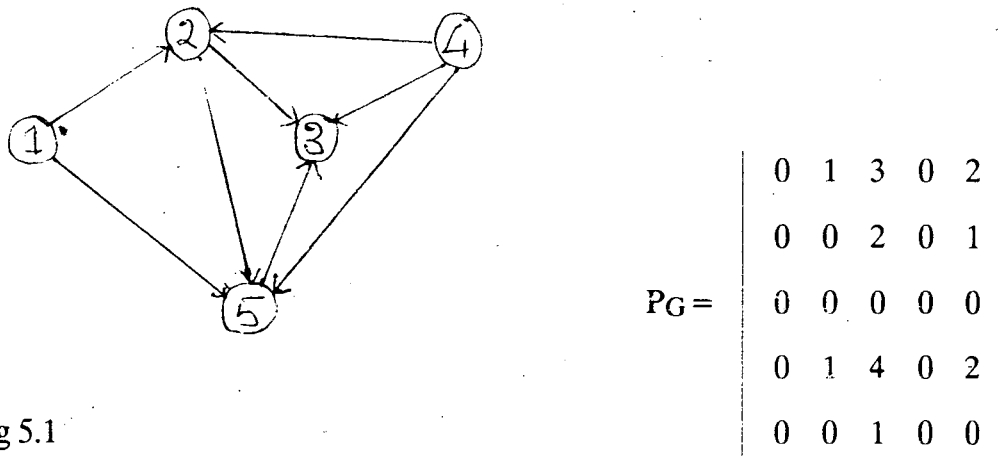


Fig 5.1

A matrix corresponding to TWFG is shown in the fig. 5.1. An entry (i,j) in the matrix represents number of paths from process p_i to process p_j in TWFG. A deadlock in the system is known even before its existence

as follows. For a request, by process p_i for the resource r_j , the entry (j,i) in P_g is checked if the entry (j,i) in P_g is nonzero entry, the request leads to a deadlock situation.

5.2 EDGE DELETION AND ADDITION

The matrix P_g is regularly updated, for every request/release of a resource. For any update of the matrix, outer product is required. An entry (i,j) represents the number of path from process p_i to process p_j . The outer product is calculated as follows. To delete an edge $(2,5)$ in the matrix shown in fig 5.1, the following calculation are done.

$$P(2,5) = (P_g[-,2] + V2[-]) \times (P_g[5,-] + V5[-])^T$$

Where

$P_g[-,2]$: Column 2 in P_g

$V2[-]$: Vector of unity

$P_g[5,-]$: Row 5 in P_g

$V5[-]$: Vector of Unity

$P(2,5)$: Number of paths passing through $(2,5)$

$$\begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline 0 & 0 \\ \hline 1 & 0 \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline \end{array} * ((00100) + (00001)) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$

The outer product matrix is subtracted from the matrix (P_g) to get resultant matrix after deletion of the edge. To add an edge (i,j), the entry (j,i) in the path matrix P_g is checked. If this entry is zero the outer product is calculated. The outer product matrix is added to get resultant matrix

5.3 THE PROPOSED DEADLOCK DETECTION AND RESOLUTION ALGORITHM

The matrix based approach proposed by Ferenc Belik detect a possible deadlock. The transactions involved in the deadlock are not known. To select a victim transaction, to resolve deadlock, the knowledge of the set of transaction involved in the deadlock is a must. The proposed algorithm use two matrices for deadlock detection and resolution. The first matrix is a path matrix. It is similar to the path matrix used by Ferenc Belik. Deadlock is detected just by checking (j,i) entry in the path matrix for any request (i,j) (process p_i request for a resource held by process p_j). The path matrix is also maintained in the similar fashion. The edge matrix represents only requested edges.

5.3.1 The Algorithm

This algorithm detects deadlock cycle when existence of deadlock is reported. There may be more than one deadlock cycle. The algorithm

detects all of them. A victim transaction is selected for rollback. When a request for adding an edge (i,j) arises and the entry (j,i) in the path matrix is nonzero, it is a deadlock situation.

Step 1. Set $@ = j$. Send a list containing only j to step 2.

Step 2. Make the n copies of the received list (n : total number of non-zero entries in the $@$ th row). Include first non zero entry to first list, second non zero entry to second list and so on. Send these list to step 3. Pass a signal to step 3

Step 3. For each received list check if last entry in the received list is send this list to step 5, else after receiving signal from step 2, set $@ =$ last entry in the list and send these list (one list at one signal) to step 2.

Step 4. Stop.

Step 5. The received list is a deadlock cycle. Calculate row and column total in the edge matrix for each transaction in the list. Rollback a transaction for which the sum of these two is maximum. The tie can be broken by rolling back a transaction which has higher column total among two.

CHAPTER VI

SUMMARY AND CONCLUSIONS

A study of deadlocks in the distributed database system has been done. A chapterwise summary of this report is as follows.

Chapter I deals with the issue of deadlock. A deadlocks in distributed database environment is defined with example. Deadlock in database environment differs from deadlock in Operating System environment. Different deadlock model are discussed with their applicability in different environments. Three approaches of dealing with deadlock problems are discussed.

In the chapter II following algorithms are discussed [GOL77], [ISL78], [MEN79], [HOR82], [OBR82], [CHA83] and [SNH85]. The later development, modification and other research work on algorithms is also discussed.

In chapter III the main points (number of messages transferred, data structures used in the algorithm, delays and communication type required by the algorithms of some of algorithms are presented in tabular form. A comparative study is also done.

In chapter IV small modifications in some of algorithms has been suggested to make algorithm efficient for deadlock resolution. The issue of deadlock resolution is left to implementor in the most of algorithms.

In chapter V a new matrix based algorithm for deadlock detection and resolution is presented.

A number of issues related to deadlock detection remain untouched. There is no formal method to prove the correctness of the algorithm. The informal proofs of correctness given for some of algorithm, have been later found incorrect [GLI80] and [ALO89].

There is no formal method for performance analysis of algorithms [MUK89]. The comparison of algorithm has been done based on number of messages required by the deadlock detection process. But there is a message transfer even before existence of deadlock. The number of messages are not sufficient for comparison of algorithm.

The issue of deadlock resolution in the post detection stage has not received much attention of researchers. The persistence of a deadlock for longer period affects the system throughput. The selection of a victim transaction has been left untouched in some of algorithms. In the case of others a simple criterion is taken

to select a victim, which many times does not take into consideration all the relevant factors.

REFERENCES AND BIBLIOGRAPHY

- [BDL86] **D.Z. Badal**, "The Distributed Deadlock Detection Algorithm", *ACM Transaction on Computer System*, Nov 1986, pp. 320-327.
- [CHO89] **A.L. Choudhary et. al.**, "A Modified Priority Based Algorithm for Distributed Deadlock Detection and Resolution", *IEEE Transaction of Software Engineering*, Jan 1899 pp. 10-17.
- [CHA83] **K.M. Chandy et. al.**, "Distributed Deadlock Detection" *ACM Transaction on Computer Systems*, May 1983, pp. 144-156.
- [EGR87] **Edger Knapp**, "Deadlock Detection in Distributed Database System", *ACM Computing Survey*, Dec. 1987, pp 303-328.
- [ELA86a] **Ahmed K. Elmagarmid et. al.**, "Deadlock Detection Algorithm in Distributed Database Systems", *Proceeding International Conference on DATA ENGINEERING 1986*, pp. 556-564.

- [ELA86b] **A.K. Elmagarmid**, "A survey of Distributed Deadlock Detection Algorithms", SIGMOD Record, 15 (3), 1986.
- [ELA88] **A.K. Elmagarmid et. al.**, "A Distributed Deadlock Detection and Resolution Algorithm and its Correctness", IEEE Transaction on Software Engineering, Oct 1988, pp. 1443-1452.
- [ESA76] **K.P. Eswaran et. al.**, "The Notion of Consistency and Predicate Locks in a Database Systems", Communications of ACM, Nov. 1976, pp. 624-633.
- [FER87] **Ferenc Belik**, "A Distributed Deadlock Avoidance Technique", Lecture Notes in Computer Science, Number 312, pp. 144-154.
- [Gli, 1980] **V.D. Gligor and S.H. Shattuck**, "On Deadlock Detection in Distributed Systems". IEEE Transaction in Software Engineering, Sept 1980, pp. 435-440.
- [GOL77] **B. Goldman**, "Deadlock Detection in Computer Networks" Tech. Report MIT/LCS/TR-185, MIT Cambridge, Massachusetts, Sept 1977.

- [GRA81] **J. Gray et. al.** "A Straw Man Analysis of The Probability of Waiting and Deadlock in Database Systems", IBM Research 1981.
- [HAS83] **L.M. Hass and C. Mohan**, "A Distributed Deadlock Detection Algorithm for Resource Based System", Research Report, IBM Research Laboratory, San Jose California, 1983.
- [HOR82] **G.S. Ho and C.V. Ramamurthy**, "Protocol for Deadlock Detection in Distributed Database Systems", IEEE Transaction on Software Engineering, Nov 1982, pp. 554-557.
- [HSU86] **Meichun Hsu and Arvola Chan**, "Partitioned Two Phase Locking" ACM Transaction on Database Systems, Dec. 1986 pp. 431-446.
- [ISO80] **Sreekanth S. Isloor and T. Antony Marshland**, "The Deadlock Problem: An Overview", Computer, Sept. 1980, pp. 58-78.
- [JAG83] **J.R. Jagannathan and R. Vasudevan**, "Comment on Protocol for Deadlock Detection in Distributed Database System", IEEE Transaction on Software Engineering, May 1983.

- [KOR86] **H.F. Korth and A. Silverschatz**, "Database System Concepts", McGraw Hill Book Company, New Delhi, 1986.
- [LAM78] **L.Lamport**, "Time, Clock and Ordering of Events in Distributed Systems", Communication of ACM, pp. 558-565, July 1978.
- [MEN79] **D.E. Menasce and R. R. Muntz**, "Locking and Deadlock Detection in Distributed Databases", IEEE Transaction on Software Engineering, May 1979, pp. 195-202.
- [MEN80] **Daniel A. Menasce**, "A Locking Protocol for Resource Coordination in Distributed Databases", ACM Transaction on Database Systems, March 1980, pp. 103-137.
- [MAR88] **Marina Roesler and Walter A. Burkhard**, "Deadlock Resolution and Semantic Lock Models in Object-Oriented Distributed systems" ACM SIGMOD 1988, pp. 361-370.
- [MUK89] **Mukesh Singhal**, "Deadlock Detection in Distributed Systems", Computer, Nov 1989, pp. 37-48.

- [NAT86] **N.Natarajan**, "Distributed Scheme for Detecting Communication Deadlocks", IEEE Transaction on Software Engineering, April 1986, pp. 531-537.
- [OBR82] **R. Obermarck**, "Distributed Deadlock Detection Algorithm", ACM Transaction on Database Systems, June 1982, pp. 187-210.
- [PEE78] **R. Peebles and E. Manning**, "System Architecture for Distributed Database Management, Computer, June 1978, pp. 40-47.
- [PET85] **J.L. Peterson and A. Silverschatz**, "Operating System Concepts", Addison Wesley Publishing Company, Reading, Massachusetts, 1885.
- [RIE79] **Daniel R. Ries and Michael R. Stonebraker**, "Locking Granularity Revisited", ACM Transaction on Database Systems, June, 1979, pp. 210-227.
- [SNH85] **M.K. Sinha and N. Natarajan**, "A Priority Based Distributed Deadlock Detection Algorithm", IEEE Transaction on Software Engineering, Jan 1985, pp. 67-80.

- [TAY85] Y. C. Tay et. al., "Locking Performance in Centralized Database"
ACM Transaction on Database Systems, Dec. 1985, pp. 415-462.
- [TRA82] Irvin L. Traiger et. al., "Transaction and Consistency in Dis-
tributed Database Systems", ACM Transaction on Database Sys-
tems, Sept. 1982, pp. 323-342.