Deadlock Detection Techniques in Distributed Database System

Swati Gupta
Amity University, Gurgoan
swattigupta@gmail.com

Suresh Kumar
FET, MRIU, Faridabad
enthuvs@gmail.com

Abstract: *Distributed database system provides resource sharing environment for optimal performance of various database activities, especially when data is spread over a large number of sites. Distributed nature of transactions occurring at different sites and requiring resources from diverse sites pose various operational problems, such as deadlocks, concurrency and data recovery. A deadlock may occur when a transaction enters into wait state which request resource from other blocked transactions. The deadlocks are handled in three phases namely deadlock detection, deadlock avoidance and deadlock detection. Various algorithms have been discussed in the literature for deadlock detection and resolution. These algorithms quite often fail to detect deadlock over distributed database. In this paper an attempt has been made to develop an algorithm for distributed deadlock detection at local and global levels. We have developed local transaction structure to deal with deadlock at local level and distributed transaction structure at global level*

Keywords*: Wait-For-Graph, Deadlock, Local Transaction Structure.*

1. INTRODUCTION

Distributed database systems (DDBS) comprises of database which is distributed over several sites interconnected by a communication network. It provides resource-sharing environment where database activities can be performed optimally in global as well as local framework. The distributed nature database demand full proof control structure for its proper and effective functioning. Therefore if the allocation of the resources is not properly controlled then it may lead to several anomalies such as concurrency of transaction, synchronizing of events and deadlocks. In Distributed database system model, the database is considered to be distributed over several interconnected computer systems. Users interact with the database via transactions. A transaction is a sequence of actions, which can be read, write, lock, or unlock operations. If the actions of a transaction involve data at a single site, the transaction is called local, on the other hand a distributed transaction involve resources at several sites.

A deadlock may occur when a transaction enters into wait state, i..e. the request is not fulfilled due to non-availability of the requested resources as these resources are held by another waiting transaction.

In such a situation, waiting transaction may never get a chance to change its state. Deadlock representation techniques for their easy detection has been discussed widely in the literature and graphical representation has been found to be suitable and effective technique.

A deadlock can be indicated by a cycle in the directed graph called Wait-for-Graph (WFG) [4] that represents the dependencies among the processes. A node in the graph G represents a transaction and a directed edge from vertex i to vertex j exist in G, if T_i (Transaction i) needs a resource, which is being held by T_j (Transaction j). For example, in Fig 1 a transaction T1 has locked data item X and needs to lock item Y, T2

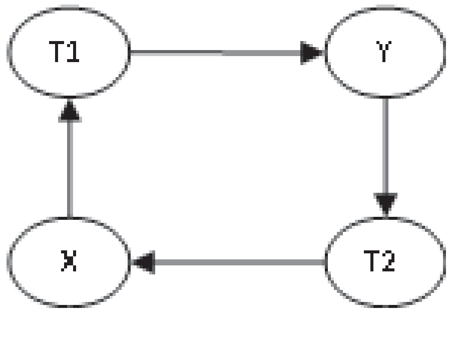

Fig. 1. Transaction Wait for Graph

has locked item Y and needs to lock item Z, T3 has locked item Z and needs to lock item W, while T4 has locked item W and needs to lock item X. In this case the transactions are waiting for each other, no transaction can continue resulting into a deadlock.

In distributed database system three techniques are generally used for handling the deadlocks: Deadlock avoidance, Deadlock prevention and Deadlock detection

Deadlock Avoidance: Deadlock avoidance is an approach in which deadlocks are dealt before they occur. When a transaction requests a lock on a data item that has already been locked by some another transaction in an incompatible mode, the deadlock avoidance algorithm decides if the requesting transaction can wait or if one of the waiting transactions need to be aborted.

Deadlock Prevention: It is an approach that prevents the system from committing an allocation of locks that will eventually lead to a deadlock. This technique requires pre-acquisition of all locks. The transactions are required to lock the entire data item that they need before execution. Deadlock prevention deals with deadlock ahead of time.

Deadlock Detection: In this approach, deadlock may have already occurred and the deadlock detection technique tries to detect it and gives the process by which it can be resolved. Thus the system periodically checks for them. The existence of a directed cycle in the Wait-for-Graph indicates a deadlock. One transaction in the cycle called victim is aborted, thereby breaking the deadlock.

In this paper we have proposed an algorithm that is based on the concept of creating a Local Transaction Structure (LTS) and Distributed Transaction Structure (DTS) to find and resolve local and distributed deadlocks respectively

2. DISTRIBUTED TRANSACTION MODEL

We next take up a distributed transaction model [1, 3] its general structure is shown in Fig 2. In this each node has the following modules: a Transaction Manager (TM), a Data Manager (DM), a scheduler (S), and a Transaction Process (T). The Transaction Manager (TM) present at each distributed site controls the execution of each transaction process (T). The transactions communicate with TMs, and in turn TMs communicate with Data Managers (DMs), the Data

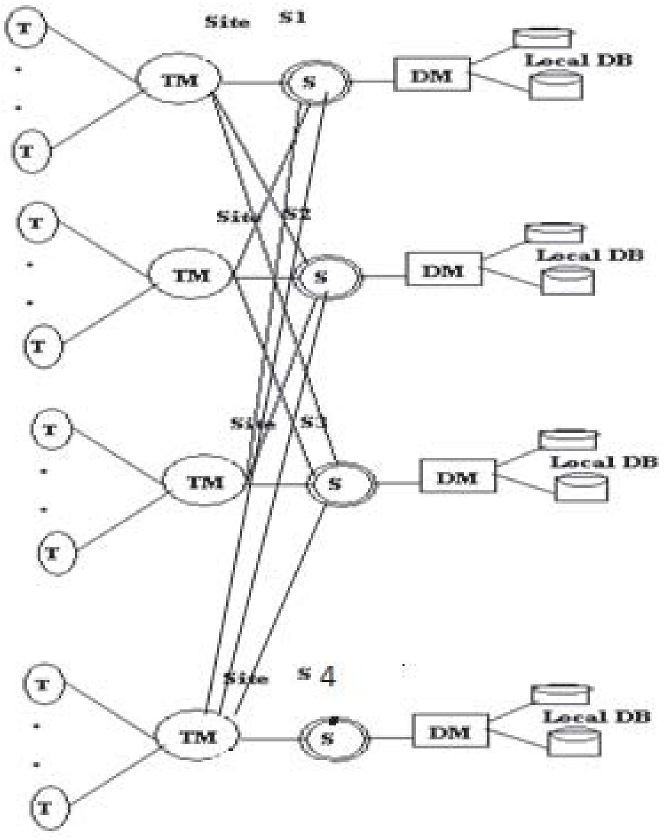

Fig. 2. Distributed Transaction Model

Manager, manages the actual data at each distributed site. A single TM supervises each transaction executed in the DDBMS. The transaction sends all of its database operations to a particular transaction manager.

The Transaction manager controls the execution of the transaction process by providing the necessary data item required by the transaction process. It does so by contacting with the data manager present at that particular site. But if the transaction process requires a data item, which is not present at the site where it originates, the transaction manager contacts the data manager of the other site where the required data item actually resides. The scheduler in turn, at each site, synchronizes the transaction requests and performs deadlock detection. A transaction may request multiple data objects simultaneously.

3. RELATED WORK

Different distributed deadlock detection and resolution algorithms have been proposed in the literature. Here we discuss the contributions of other researchers and the algorithms they have used for dealing with deadlocks.

Chandy et. al. [3] used a Transaction Wait-for-Graph (TWFG) to represent the status of transaction at the local sites and probes to detect global deadlock. They called the algorithm, as probe computation by which a transaction Ti determines if it is deadlocked or not. A probe is generated if a transaction starts waiting for another transaction and the probe gets propagated from one site to another based on the status of the transaction that received the probe. The probes are meant only for deadlock detection. A transaction sends at most one probe in any probe computation. If the initiator of the probe gets back the probe, then that transaction is involved in a deadlock. They found that this scheme does not detect any false deadlock.

Menasce D. A. et. al. [8] describes two protocols for the detection of deadlocks in distributed data bases: a hierarchically organized and a distributed. A graph model, which depicts the state of execution of all transactions in the system, is used by both protocols. A cycle in this graph is a necessary and sufficient condition for a deadlock to exist.

Qinqin et. al. [13] have used the principle of adjacency matrix, path matrix and strongly-connected component of simple directed graph in graph theory. They have proposed a model for detecting deadlock by exploring strongly-connected component from resource allocation graph. The experiment shows that it can detect resources and processes involved in deadlock effectively.

Mehdi et. al. [7] have proposed a distributed deadlock detection algorithm in which the chance of phantom deadlocks detection is minimized by using a new approach and some improvements to resolution of deadlocks. The algorithm manages simultaneous execution by the nodes involved in deadlocks, thus prevent the detection of same deadlock.

4. PROPOSED TECHNIQUE

We have developed a deadlock detection and resolution technique which can detect both local and global deadlocks in a distributed database system.

A Local Transaction Structure (LTS) is maintained for all the transactions executing at each local site. The existence of cycle in LTS represents the local deadlock.

The Distributed Transaction Structure (DTS) is used to handle the global deadlocks among the distributed sites. Each transaction is assigned a unique timestamp by the local transaction manager (TM).

The proposed technique uses Transaction Wait-for-Graph (TWFG) to represent the transaction data request and to indicate the deadlock situation in a distributed database environment. This technique assures that global deadlock is not dependent on local deadlock. The victim transaction is a youngest transaction based on the timestamp value and is aborted in order to resolve the deadlock.

In the proposed technique if any transaction Tp requests a data item that is held by another transaction Tq, values of p and q are stored on the local transaction structure (LTS), where p and q represent their corresponding transaction numbers.

The Global Transaction Manager (GTM) will keep record of the transactions whose request for the resource is not satisfied at a single site. The GTM will create a distributed transaction structure and find the global deadlock cycle using the same procedure as LTS

Distributed Transaction Structure (DTS) stores all the transactions that are interconnected from one site to another site. The transactions in both LTS and DTS are arranged in the increasing order of their timestamp value. The algorithm for the distributed deadlock detection and resolution is presented as:

4.1 Detection and Resolution Technique

Local Deadlock Detection:

1) Create an input LTS table for each transaction Tp requesting for a data item Tq in the increasing order of their timestamp value. The following data structures are used:
    a. Array P to store the p value of LTS
    b. Array Q to store the q values of LTS
    c. Array vector that is initially empty is used to store the scan values of LTS.
    d. Stack Temp.
2) Take the first value from the Array P and store the selected first value in the vector array.
3) Choose corresponding q value and assign the value of q to the variable temp.
4) Search temp in array P. For each successful search put value of P in a stack.

5) Repeat until stack is empty:

   a. Pick the top most value from the stack, select the corresponding index in array q, compare it with value stored in the vector array if the match occurs (same value) than there exist a deadlock and put the corresponding index of array P in the vector array. For each deadlock cycles that are detected the victim is selected based on lowest timestamp value. It is then aborted from the old LTS and vector array so as to resolve the deadlock and a new LTS structure is created.

   b. Else if comparison fails than store the value of P at that index in the vector array and than goto step 3.

6) For unsuccessful search if temp is not found in array P then make the stack empty.

Global Deadlock Detection: Create an input DTS table to record different transaction request communicating among different sites, Apply the local deadlock detection technique in each DTSi to resolve the global deadlock cycle

4.2 Illustrative Example

The proposed technique can be explained with the help of an example as shown in Fig 3. In this example there are three sites S1, S2 and S3. The TWFG in the Fig 3 shows local deadlocks and global deadlock cycles. The transaction managers assign a unique timestamp to each of the transactions at site S1, S2 and S3 as shown in Table 1.

Table1: The Timestamps of Transactions at different sites

Site 1	T4→2	T1→3	T2→4	T3→5	
Site 2	T6→2	T8→3	T7→4	T9→5	T5→6
Site 3	T12→2	T14→3	T11→4	T10→5	T13→6

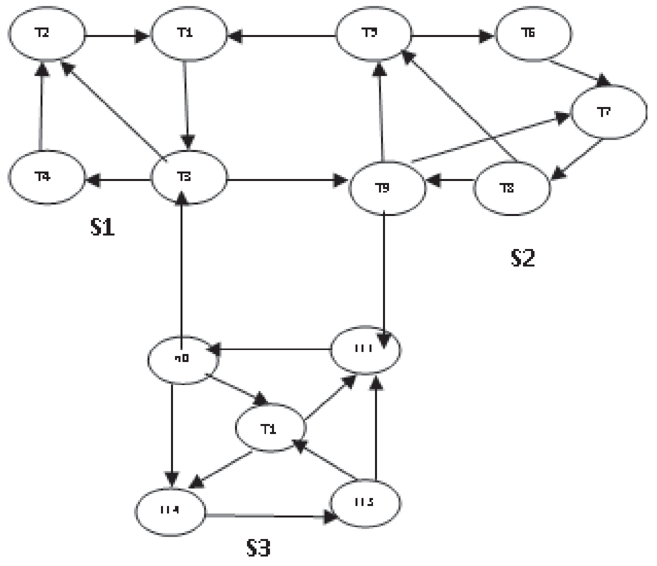

Fig 3. TWFG with three Sites S1, S2 and S3

The Local Transaction Structure for the sites S1, S2 and S3 have been created in the increasing order of their timestamp value from the Transaction Wait-for-Graph (TWFG) as shown in the Table 2.

Table 2: LTS at Site S1, S2 and S3

Site S1		Site S2		Site S3	
P	q	p	q	p	q
4	2	6	7	12	11
1	3	8	9	12	14
2	1	8	5	14	13
3	4	7	8	11	10
3	2	9	5	10	14
		9	7	10	12
		5	6	13	12
				13	11

a) At Site S1

The first (p, q) pair (4, 2) is selected. The P value is placed in the vector array and Q value is assigned to variable temp. Now temp is searched in array P and for each successful search the pair (p, q) is placed on a stack .Now Q value of the (p, q) pair on the top of the stack is compared with values on the vector array. If no match found then P value of the (p, q) pair is stored on the vector array and Q value is assigned to variable temp. If match found it means a deadlock is present.

The vector array is shown in Fig 4 depicts a deadlock cycle (4→2→1→3→4).

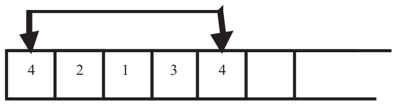

Fig. 4: Vector array for the deadlock cycle at site S1

Finally the transaction pair with lowest timestamp (3→4) is aborted to resolve the deadlock. The same procedure is repeated until stack is not empty. Since {3→2} value exist in the stack so, next we check whether any deadlock cycle exist using {3→2}, we find the deadlock cycle {2→1→3→2} as shown in Fig 5.

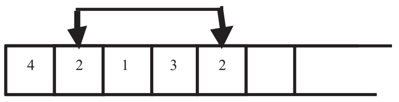

Fig. 5: Vector array for the deadlock cycle at site S1

Now the transaction pair 3→2 (having the lowest timestamp value) is aborted to resolve the deadlock.

a) At Site S2

Next we take up deadlock detection algorithm to detect the local deadlock cycles and create the vector array for each deadlock cycle present at the site S2. The deadlock cycle (6→7→8→9→5→6) is detected as shown in the vector array in Fig 6. Finally the transaction pair with lowest timestamp (5→6) is aborted so as to resolve the deadlock.

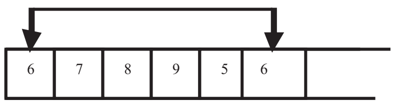

Fig. 6: Vector array for the deadlock cycle at site S2

Next we find the deadlock cycle (6→7→8→9→7) as shown in the vector array in Fig 7. We abort the transaction pair 9→7 (having the lowest timestamp value) so as to resolve the deadlock.

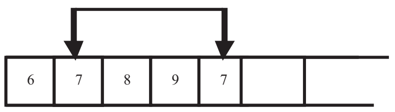

Fig. 7: Vector array for the deadlock cycle at site S2

Next we select the last value (8→5) present in the stack since no deadlock cycle exist using this transaction pair, we delete (8→5) from the stack.

a) At Site S3

Next we detect the local deadlock cycle and create the vector array for each deadlock cycle present at site S3. The deadlock cycle (12→11→10→14→13→12) is detected as shown in vector array in Fig 8. Finally the transaction pair with lowest timestamp (13→12) is aborted to resolve the deadlock.

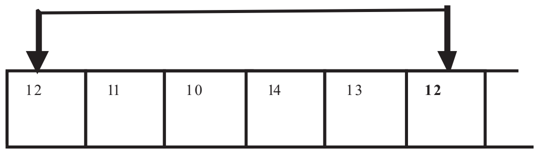

Fig. 8: Vector array for the deadlock cycle at site S3

Next we find the deadlock cycle (11→10→14→13→11) as shown in vector array in Fig 9. We abort the transaction pair 13→11 (having the lowest timestamp value) so as to resolve the deadlock.

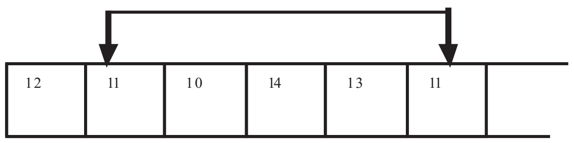

Fig. 9: Vector array for the deadlock cycle at site S3

Next we select the last value (10→12) present in the stack and check for the deadlock cycle using this transaction pair we find the deadlock cycle (12→11→10→12) as shown in the vector array in Fig 10. We abort the transaction pair 10→12 (having the lowest timestamp value) so as to resolve the deadlock.

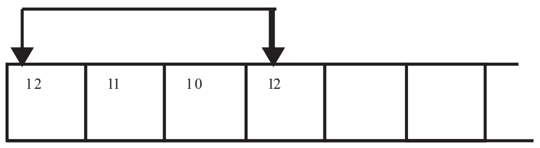

Fig. 10: Vector array for the deadlock cycle at site S3

We finally create a new deadlock free LTS for site S1, S2 and S3 after all the deadlock cycle are detected and the victim from all the deadlock cycles is aborted as shown in Table 3.

Table 3: Deadlock Free LTS for Site S1, S2 and S3

Site S1		Site S2		Site S3	
p	Q	p	q	p	q
4	2	6	7	12	11
1	3	8	9	12	14
2	1	8	5	14	13
		7	8	11	10
		9	5	10	14

To find global deadlock we have created Distributed Transaction Structure (DTS) as shown in Table 4. After creating the DTS same procedure is repeated to find the global deadlock as it is done to find the local deadlock cycle by creating the LTS.

Table 4: DTS at Site S1, S2 and S3

Site S1, S2		Site S1, S2 and S3	
P	q	p	q
1	3	11	10
3	9	3	9
9	5	9	11
5	1	10	3

We detect the global deadlock cycle and create the vector array for each deadlock cycle present at site S1 and site S2. The deadlock cycle (1→3→9→5→1) is detected as shown in vector array in Fig 11. Finally the transaction pair (5→1) is aborted so as to resolve the deadlock.

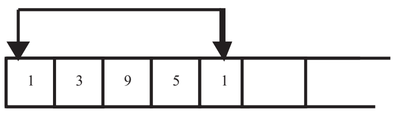

Fig. 11: Vector array for the deadlock cycle at site S1 and site S2

We detect the global deadlock cycle and create the vector array for each deadlock cycle present at site S1, site S2 and site S3. The deadlock cycle (11→10→3→9→11) is detected as shown in vector array in Fig 12. Finally the transaction pair

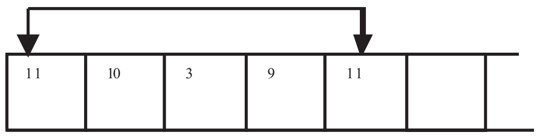

Fig. 12: Vector array for the deadlock cycle at site S1, S2 and site S3

5. CONCLUSIONS

In this paper we have presented an approach to detect both local deadlocks and global deadlocks by creating the LTS and DTS structure for local and global deadlock cycles. This technique assures that global deadlock detection is not dependent on local deadlock detection. The proposed algorithm eliminates the dependency of LTS and DTS on the directed edges of transaction wait for graph by assigning unique timestamp to each transaction.

We have compared results of proposed algorithm with that of Alom B. M. et. al.. We find that the algorithm proposed by them have dependency on the WFG when creating LTS and DTS. The transaction pairs have to be placed on LTS and DTS in a particular order to detect deadlock cycles. The algorithm proposed by us is free from any such dependency.

6. REFERENCES

1. Alom B.M. Monjurul, Frans Alexander Henskens, Michael Richard Hannaford, Optimization of Detected Deadlock Views of Distributed Database, International Conference on Data Storage and Data Engineering , pp. 44-48, ISBN: 978-0-7695-3958-4, 2010.

2. Carlos F. Alastruey, Federico Fariña, Jose Ramon Gonzalez de Mendivil, “A Distributed Deadlock Resolution Algorithm for the AND Model”, IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 5, pp. 433-447, May 1999.

3. Chandy K. M., Hass L. M and Misra J, “Distributed Deadlock Detection”, ACM Transactions on Computer Systems, vol. 1, no. 2, pp. 144-56, 1983.

4. Elmagarmid A. K., “A Survey of Distributed Deadlock Detection Algorithms”, SIGMOD RECORD, vol. 15, no. 3, pp. 37-45, 1986.

5. Gray J., “A Straw Man Analysis of the Probability of Waiting and Deadlocks in Database Systems”, IBM Research Report, 1981.

6. Ho G. S. and Ramamoorthy C. V., “Protocols for Deadlock Detection in Distributed Database Systems”, IEEE Transaction on Software Engineering, vol. 8, no. 6, pp. 554-557, 1982.

7. Mehdi Hashemzadeh, Nacer Farajzadeh, Abolfazl T. Haghighat, “Optimal Detection and Resolution of Distributed Deadlocks in the Generalized Model,” 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP’06), pp.133-136, 2006.

8. Menasce D. A. and Muntz R. R., “Locking and Deadlock Detection in Distributed Data Bases”, IEEE Transaction on Software Engineering, vol. 5, no.3, pp. 195-202, 1979.

9. Merritt M. J. and Mitchell D. P., “A Distributed Algorithm for Deadlock Detection and Resolution,” ACM Transactions, vol. 2, no. 3, pp. 95-99, 1984.

10. Singhal Mukesh “Deadlock Detection in Distributed System” IEEE Transaction on Software Engineering, vol. 4, no. 3, pp. 195-199, 1989.

11. Jain Kamal, MohammadTaghi Hajiaghayi and Kunal Talwar, “The Generalized Deadlock Resolution Problem”, Autoomata Languages and Programming, Lecture Notes in Computer Science, 2005, vol. 3580/2005, 103, DOI: 10.1007/11523468_69

12. Nacer Farajzadeh, Mehdi Hashemzadeh, Morteza Mousakhani, Abolfazl T. Haghighat, “An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems”, Fifth International Conference on Computer and Information Technology (CIT’05), pp. 303-309, 2005.

13. Qinqin Ni, Weizhen Sun and Sen Ma, “Deadlock Detection Based on Resource Allocation Graph”, Fifth International Conference on Information Assurance and Security, vol. 2, pp.135-138, 2009.

14. Selvaraj S. and R. Rajaram, “A Decentralized Deadlock Detection and Resolution Algorithm for Generalized Model in Distributed Systems”, Journal of Distributed and Parallel Databases, vol. 29, No. 4, pp. 261-276, DOI: 10.1007/s10619-011-7078-7

15. Soojung L. and Junguk L. Kim, “Performance Analysis of Distributed Deadlock Detection Algorithms”, IEEE Transactions in Knowledge and Data Engineering, vol. 13, no. 4, pp. 623-636, August 2001.

16. T. Ozsu and Valduriez P., “Principle of Distributed Database Systems”, Prentice Hall, 1999.

❐