ANALYSIS OF DEADLOCK DETECTION AND RESOLUTION TECHNIQUES IN DISTRIBUTED DATABASE ENVIRONMENT

HIMANSHI GROVER, SURESH KUMAR
Department of Computer Science and Engineering, Manav Rachna International University
Faridabad, India
himanshigrover22@gmail.com, enthuvs@gmail.com

ABSTRACT
Deadlock is one of the most serious bottlenecks in multitasking concurrently running systems and it has become further complicated in distributed environment. Deadlock detection and their resolution is quite difficult in distributed systems as it involve data located at different sites. The deadlock problem is intrinsic to a distributed database systems which employs locking process in its concurrency control system. In literature various techniques have been discussed and used to prevent, detect and resolve the deadlocks. In this paper we have analyzed some deadlock detection and resolution techniques used. We have reviewed in detail the algorithms presented by B. M. Alom and Brian M. Jhonston for deadlock detection and resolution in distributed environment and found that when the order of the transactions is changed in the algorithm by Aloms then it fails to detect deadlocks whereas algorithm by B. M. Jhonston aborts a transaction for which the intersection values of Wait-for and Request-Q values is not nil. We have introduced the concept of time stamping to avoid these deficiencies.

KEYWORDS: *Deadlocks, Distributed Databases, Stand Alone Environment, Distributed Environment.*

1. INTRODUCTION

A Distributed database system (DDBS) is a collection of databases distributed over several sites interconnected by a communication network. It provides a resource-sharing environment where database activities can be performed optimally both in global and local framework. The distributed nature of database demand full proof control structure for its proper and effective functioning. Therefore the allocation of the resources should be properly controlled otherwise 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 located at several sites.

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

In such a situation, waiting transaction may never get a chance to change its state. A deadlock representation technique 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.

At functional level, a distributed database management system ('DDBMS') is a software system that permits the management of a distributed database and makes the distribution transparent to the users. A distributed database is a set of databases stored on multiple computers that typically appears to applications on a single database. To ensure that the distributed databases are up to date and current, there are two processes: replication and duplication.

Replication involves using specialized software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be very complex and time consuming depending on the size and number of the distributive databases. This process can also require a lot of time and computer resources.

Duplication on the other hand is not as complicated. It basically identifies one database as a master and then duplicates that database. The duplication process is normally done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, changes to the master database only are allowed. This is to ensure that local data will not be overwritten. Both of the processes can keep the data current in all distributive locations.

Deadlocks in distributed systems are similar to deadlocks in centralized system, but the situation is rather worse. They are harder to avoid, prevent or even detect. They are hard to cure when tracked down because all relevant information is scattered over many machines. But in spite of being complicated to be removed we need to overcome this problem. Looking in this aspect we have analyzed various algorithms presented by different authors to detect and resolve deadlocks in a distributed database environment. We have analyzed in detail the algorithms presented by B. M. Alom and B. M. Jhonston for detecting and resolving deadlocks in distributed environment.

In section 2 we take up the recent work and analyze the contributions made by several researchers dealing with prevention, detection and resolving of the deadlocks. In section 3, we take an example to compare the working of techniques by B. M Alom[14] and B. M Jhonstons[13] in details. In section 4, we reviewed the performance analysis of various techniques discussed in the previous sections and in section 5, we give the concluding remarks.

2. RECENT WORK

As mentioned earlier deadlock occurs whenever waiting transactions need access to the resources held by each other and none is able to complete their execution. In the literature various techniques have been discussed to deal with deadlocks for both centralized as well as distributed systems. Here we are discussing several deadlock detection and resolution techniques.

Chandy & Mishra [6] have proposed an algorithm which uses transaction wait-for-graphs (TWFG). This technique makes the use of probe computation for deadlock detection. Probe computation is a method by which transaction Ti determines if it is deadlocked or not. It checks the status of transactions at the local sites and uses probes to detect global deadlocks. A probe is issued whenever a transaction begins to wait for another transaction and is directed from one site to another based on the status of the transaction that received the probe. The probes are meant only for deadlock detection but they do not have any relation with requests and replies. A transaction send at most one probe in any probe computation. If the initiator of the probe computation gets back the probe, then it is involved in a deadlock. This scheme does not suffer from false deadlock detection even if the transactions do not obey the two-phase locking protocol.

Sinha and Natrajan [8] have developed an algorithm which is an extension over Chandy's scheme. It is based on the priorities of transactions. Using priorities, the number of messages required for deadlock detection is reduced considerably. An advantage of the scheme is that the number of messages in the best and worst cases can be easily determined. In this scheme we have a number of transactions and data managers. The data managers have the responsibility to grant and release locks for various resources. A transaction's request for a lock on a data item is sent to the data manager for the item. If the request can not be granted, the data manager initiates deadlock computation by sending a probe to the transaction that holds a lock on the data item, if the priority of the holder is greater than that of the requestor. The transaction inserts this probe in a probe-q that it maintains. The probe is then sent to the data manger of the data item it is waiting for. At this stage of deadlock computation, priorities of transaction are used to decide whether to propagate or not. The probe is propagated only if the priority of the holder of the data item it manages is greater than that of the initiator. When a transaction begins to wait for a lock, all the probes from its queue are propagated. When a data manager gets back the probe it had initiated, deadlock is detected. Since the probe contains the priority of the youngest transaction in the cycle, the youngest transaction is aborted.

Chim-fu yeung [12] presented an improvement over the algorithm proposed by Chandy & Mishra in which Deadlock is found by passing special messages called probe messages along the edges of a wait-for graph. This process has several advantages as compared to Chandy's algorithm, such as:

- This algorithm is error free.
- It suffers very little performance degradation as compared to the original one.
- Even for large values of multi-programming level, probe-based algorithm can outperform time–out.
- The rate of probe initiation is a dominant factor in determining system's performance.

In this algorithm we focus on clearing the dependency table regularly. If the dependency tables are cleared periodically then the number of probe messages will be great as many probes needed to be propagated. As the controller initiates the probes periodically, the number of probe messages could be used to estimate the total blocking time period of a process.

Obermack [3] presented an algorithm which at each site builds and analyzes directed TWFG and uses a distinguished node at each site. This node is called "external" and is used to represent the portion of TWFG that is external (unknown) to the site. This algorithm does not work correctly; rather it detects false deadlocks because the wait-for graphs constructed to represent a snap-shot of the global TWFG at any instant.

Menasce [7] presented the first algorithm which uses a condensed transaction-wait-for graph (TWFG) in which the vertices represent transactions and edges indicate dependencies between transactions. This algorithm can fail to detect some deadlocks and may discover false deadlocks.

Ho [9] proposed an approach, in which the transaction table at each site maintains information regarding resources held and waited on by local transactions. The resources table at each of the sites maintains information regarding the transactions holding and waiting for local resources. Periodically, a site is chosen as a central controller responsible for performing deadlock detection. The drawback of this scheme is that it requires 4n messages, where n is the number of sites in the system.

Kawazu [10] presented an algorithm based on two phases: In the first phase local deadlocks are detected and in the second phase global deadlocks are detected in the absence of local deadlocks. This scheme suffers from phantom deadlocks, because each local wait-for graph is not collected at the same time due to communication delays. Phantom deadlocks are the ones which occur due to some system delays. Also, in case a transaction simultaneously waits for more than one resource, some global deadlocks may go undetected since the global deadlock detection is initiated only if no local deadlock is detected.

Brian M. Johnston [13] proposed an algorithm that uses an update message The function of update message is two fold: first to modify the Wait-for variables and second to check the occurrence of deadlock. As compared to many recent algorithms in this field, the proposed algorithm can detect the most frequent deadlocks with minimum message passing. With message complexity defined as the number of messages transmitted between initiating a update message and detecting the cycle. In the worst case, in a system of n transactions, the overall message complexity of the proposed algorithm is O (n).

B. M. Alom [14] has introduced a deadlock detection and resolution technique using the concept of priorities. In this technique a priority based table is used. This algorithm maintains a list of all the transactions, and whenever a deadlock cycle is detected, the priorities of the transactions constituting the deadlock is checked. The transaction with the least priority is aborted so that the resources held by it can be set free and can be granted over to the waiting transactions. But it has been found that if the order of the priorities is changed this algorithm fails to detect deadlocks.

3. ILLUSTRATIVE EXAMPLE FOR PERFORMANCE ANALYSIS

Let us take two sites, $Site_1$ and $Site_2$. A number of transactions are running on both the sites. In this example we have considered transactions T1, T2, T3, T4 executing on $Site_1$ and transactions T5, T6, T7, T8 executing on $Site_2$ as shown in Figure 1. We have analyzed two techniques in the following sub sections.

3.1 Using B. M. Alom Technique

In this technique two transaction structures are used: one is linear transaction structures (LTS) which is used to detect deadlock for each site locally and distributed transaction structures (DTS), which is

used to detect deadlocks in distributed environment. If there is an edge from transaction Tx to Ty, then a corresponding entry is done in its LTS and similarly if there is an edge from Ta to Tb, both belonging to different sites, then a corresponding entry is done in their DTS. Apart from these transaction structures, they have used a transaction queue which consists of transaction id and their priorities. The LTS for both the sites is shown in Table 1.

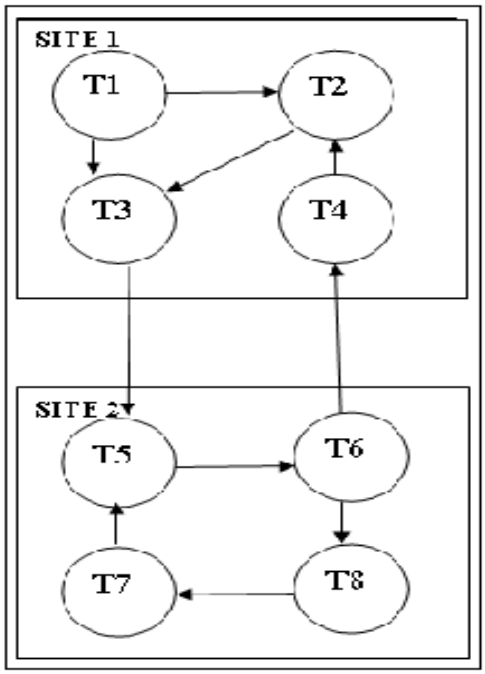

Figure 1: A Distributed environment having 2 sites

Table 1: LTS for $Site_1$ and $Site_2$

Table 1(a) : LTS1 (For site 1)

T_ID	Request_ At
1	2
1	3
2	3
4	2

Table 1(b): LTS2 (For site 2)

T_ID	Request_ At
5	6
6	8
8	7
7	5

Table 1(c) : TQ1

T_ID	priority
1	1
4	2
3	3
2	4

Table 1(d) : TQ2

T_ID	priority
6	1
8	2
5	3
7	4

To find local deadlock, transaction's requests for data item are stored in LTS, from any transaction in any site. The priority transaction id of transactions which form cycles in LTS are stored in TQ as shown in Table 1. It is recommended that any starting transaction in the cycle has the highest priority Id. For detecting global deadlock, the intra connected transaction's (those are connected to other sites) request are stored in DTS. If a cycle is formed then we conclude that deadlock has occurred. Since

there is no cycle in LTS1 so we can say that its deadlock free. There is a cycle in LTS2; therefore, the transaction with lower priority will be aborted. In this case transaction 6 has the lowest priority as shown in TQ2.

The DTS (Distributed transaction structure) for $Site_1$ and $Site_2$ is shown in Table 2. Table 2(a) shows the DTS for both the sites and Table 2(b) shows the corresponding transaction queue.

Table 2: DTS for $Site_1$ and $Site_2$

Tx	req_at
1	2
1	3
2	3
3	5
5	6
6	4
4	2
8	7
7	5
6	8

Tx	req_at
1	1
5	2
3	3
7	4
2	5
8	6
4	7
6	8

According to TQ3, the transaction 5 will be aborted as it is the youngest transaction according to the priority queue.

Table 3: Deadlock free DTS1

Tx	Req_at
1	2
1	3
2	3
6	4
4	2
8	7
6	8

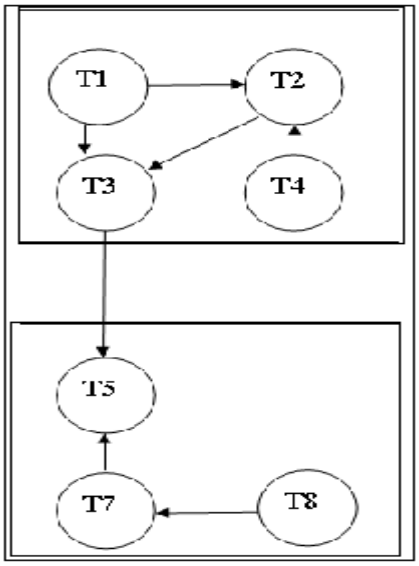

Figure 2: Deadlock free TWFG

The deadlock free TWFG after aborting the transactions involved in deadlock is shown in Figure 2.

3.2 Using Brian M. Jhonston Technique

Next, let us consider the technique suggested by Brian. M. Jhonston[13]. In this technique each transaction has a transaction id, T-ID. And there are three more variables associated with it: Wait-for, Held-by and Request-Q. Suppose a transaction Ti makes a lock request for a data object Dj**.** If Dj is free then Dj is granted to Ti and Locked_by (Dj) is set to Ti. If Dj is not free then Dj sends a not granted message to Ti along with the transaction identifier locking Dj (henceforth, called Ti). Ti joins the Request-Q (Tj) and sets its Wait-for equal to Wait-for (Ti). Now Ti initiates a update message to modify all the Wait-for variables which are affected by the changes in Locked by variable of the data objects. Update message is a recursive function call that will continue updating all elements of every Request-Q in the chain. When a transaction Tj receives the update message it checks if its Wait-for value is the same as the new Wait-for value. If it is not the same then the value is modified. Now, a check for deadlock is performed. If a deadlock is not detected, then the update message is forwarded, else deadlock is declared and deadlock resolution process is initiated.

Table 4: Transaction Structure for Site1

T-ID	Wait-for	Held-by	Request-Q
T1	T2	T2 ,T3	NIL
T2	T2	T3	T4
T3	NIL	NIL	T1 , T2
T4	T2	T2	NIL

Wait-for(T0) ∩ Request-Q (T0) = nil
Wait-for(T1) ∩ Request-Q (T1) = nil
Wait-for(T2) ∩ Request-Q (T2) = nil
Wait-for(T3) ∩ Request-Q (T3) = nil

Since all the intersection values from Table 4 are nil, So there is no deadlock in the system.

Table 5: Transaction Structure for $Site_2$

T_ID	TS	Wait-for	Held-by	Request-Q
T5	**3**	**T6**	**T6**	**T7**
T6	**1**	**T6**	**T8**	**T5**
T7	**2**	**T6**	**T5**	**T8**
T8	**4**	**T6**	**T8**	**T6**

Wait-for(T5) ∩ Request-Q (T5) = nil
Wait-for(T6) ∩ Request-Q (T6) = nil
Wait-for(T7) ∩ Request-Q (T7) = nil
Wait-for(T8) ∩ Request-Q (T8) = T6

Since intersection values in table 5 , for transaction T8 is not nil. So we will abort T8 .

Table 6: Transaction Structure for $Site_1$ and $Site_2$

T-ID	TS	Wait-for	Held-by	Request-Q
T1	7	T2	T2, T3	NIL
T2	1	T2	T3	T1 , T4

T3	5	T2	T5	T1 , T2
T4	2	T2	T2	T6
T5	6	T6	T6	T7
T6	3	T6	T8	T5
T7	8	T6	T5	T8 NIL
T8	4	T6	T7	T6

Wait-for(T0) ∩ Request-Q (T0) = nil
Wait-for(T1) ∩ Request-Q (T1) = nil
Wait-for(T2) ∩ Request-Q (T2) = NIL
Wait-for(T3) ∩ Request-Q (T3) = T2
Wait-for(T5) ∩ Request-Q (T5) = nil
Wait-for(T6) ∩ Request-Q (T6) = nil
Wait-for(T7) ∩ Request-Q (T7) = nil

Since intersection values in table 6, for transaction T3 is not nil.
So we will abort transaction T3.

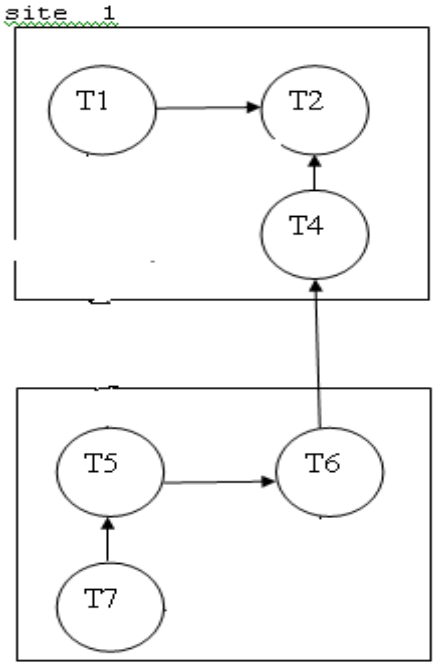

Figure 3: Deadlock free TWFG

Figure 3 represents deadlock free system for the example shown in figure 1.

4. RESULTS AND DISCUSSIONS

As we have seen above in section 2, a lot of deadlock detection techniques have proved to be successful but there are a number of flaws too related to these techniques. Like in case of Chandy and Mishra's algorithm, it may detect false deadlock. Even in this algorithm a lot of performance degradation is seen whereas if we compare it to the algorithm proposed by Chim –fu-Yeung, we see

that this algorithm is error free. It suffers very little performance degradation as compared to the original one. The rate of probe initiation is a dominant factor in determining system's performance
Similarly, when we move on to Kawaju's algorithm (section 2.6), it is realized that it has problem of detecting phantom deadlocks. Phantom deadlocks are those which occur due to system delays but this problem is not seen in any other technique.
In section 3.1, in the algorithm proposed by B.M. Alom he has made use of the priority based technique and we have analyzed that If any other order of priorities of the transactions is taken instead of the one he has taken in his paper [14] then deadlock cannot be detected or it detects false deadlocks.
In section 3.2, in the algorithm proposed by Brian M Jhonston [13] we make use of update message but in this there are no criteria of deciding that which transaction needs to be aborted. They are simply aborting the transaction for which the intersection values of Wait_for variable and Request_Q variables are not nil. But we think it is not a fair decision t abort any transaction like this, without even realizing how old that transaction is and to what extent is the system dependent on it and thus we think that sme other concept has to be used here so that we abort the youngest transaction.

5. CONCLUSION AND FUTURE SCOPE

Deadlocks in a distributed system drastically reduce the performance of the system and therefore have to be detected and resolved as soon as possible for the efficiency f the systems. After analyzing various techniques we have found that the technique presented by B. M. Alom (section 3.1)[14] is detecting deadlocks correctly if the priorities are taken in the same order as taken by him in his paper but if we take any other order then it detects false deadlocks. In the technique presented by B. M. Jhonstons (section 3.2)[13] we found that they are simply aborting the transaction for which the intersection values of Wait-for and Request-Q variables is not nil but this may not result into a fair decision by the author and hence concept of time stamping could be used to abort the younger transaction in our future work.

REFERENCES

[1] P. A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," ACM, vol. 13:2, pp. 186-221, 1981.
[2] H. T. Kung and J. T. Robinson, "Optimistic Methods for Concurrency Control," ACM Transaction on Database Systems, vol. 6, pp. 213-226, 1981.
[3] R. Obermarck, "Distributed Deadlock Detection Algorithm," ACM Transaction on Database Systems, vol. 7:2, pp. 187-208, 1982.
[4] J. N. Gray, "A discussion on distributed systems," IBM Research Division, 1979.
[5] K. M. Chandy, J. Misra, and L. M. Hass, "Distributed Deadlock Detection," ACM Transaction on Computer Systems, vol. 1:2, pp. 144-56, 1983.
[6] Chandy X. M. and Mishra J., "A Distributed Algorithm for Detecting Resource Deadlocks in Distributed Systems " in ACM, 1982.
[7] D. A. Menasce and R. R. Muntz, "Locking and Deadlock Detection in Distributed Data Bases" IEEE Transaction on Software Engineering, vol. 5:3, pp. 195-202, 1979.
[8] Sinha M. K. and Natarjan N., "A Priority Based Distributed Deadlock Detection Algorithm" IEEE Transaction on Software Engineering, vol. 11:1, pp. 67-80, 1985.
[9] G. S. Ho and C. V. Ramamoorthy, "Protocols for Deadlock Detection in Distributed Database Systems" IEEE Transaction on Software Engineering, vol. 8:6, pp. 554-557, 1982.
[10] S. Kawazu, S. Minami, K. Itoh, and K. Teranaka, "Two-Phase Deadlock Detection Algorithm in Distributed Databases" in IEEE, 1979.
[11] H. Wu, W.-N. Chin, and J. Jaffar, "An Efficient Distributed Deadlock Avoidance Algorithm for the AND Model," IEEE Transactions on Software Engineering, vol. 28:1, pp-18-29, 2002.
[12] Chim fu Yeung , "A new distributed deadlock detection algorithm for distributed databases systems", Department of Computer science, pp-506 -510, 1983.
[13] Brian M. Johnston, Ramesh Dutt Javagal: Ajoy Kumar Datta, Sukumar ghosh, "A Distributed Algorithm for Resource Deadlock Detection", Department of Computer Science , IEEE , pp-252 -256 ,1991

[14] B.M. Monjurul Alom, Frans Henskens, Michael Hannaford "Deadlock Detection Views of Distributed Database", IEEE Sixth International Conference on Information Technology: New Generations, Page(s):730–737, 2009
[15] Sheung-lun Hung ,Chim-fi Yeung, Kam-yiu Lam and Chee-keung Law "A new distributed deadlock detection algorithm for distributed database systems" IEEE , Ninth International conference, Department of Computer Science City Polytechnic of Hong Kong, vol 1 ,page(s) 506 – 510 , 2004.
[16] R. Akbarinia, E. Pacitti, and P. Valduriez, "Data Currency in Replicated DHTs1," in *SIGMOD* Beijing, China, 2007.
[17] N. Farajzadeh, M. Hashemzadeh, M. Mousakhani, and A. T. Haghighat, "An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems," in *International Conference on Computer and Information Technology*, 2005.
[18] A. G. Olson and B. L. Evans, "Deadlock Detection For Distributed Process Networks," in *ICASSP*, 2005, pp. 73-76.
[19] B.M. Monjurul Alom, Frans Henskens, Michael Hannaford, Optimization of Detected Deadlock Views of Distributed Database, International Conference on Data Storage and Data Engineering, pp: 44-48, 2010

BIOGRAPHY

Himanshi Grover is an M. Tech student of Faculty of Engineering and Technology, Manav Rachna International University. She is working in the area for deadlock detection and resolution in distributed environment. She is working under Dr. Suresh Kumar, professor in the university.

Suresh Kumar is a Professor in Faculty of Engineering and Technology, Computer science department, Manav Rachna International University. His areas of interests are DDBMS, Networking,