Volume 3, Issue 7, July 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Analysis for Deadlock Detection and Resolution Techniques in Distributed Database

Swati Gupta , Meenu Vijarania
CSE Dept, Amity University, Haryana
Haryana, India

Abstract: In a distributed database environment, where the data is spread across several sites there are many concerns to deal with such as concurrency control, deadlock. Deadlocks impact the overall performance of the system. A deadlock is a condition in a system where a process cannot proceed because it needs to obtain a resource held by another process which it itself is holding a resource that the other process needs. In literature various techniques have been discussed which are used to prevent, detect and resolve the deadlocks. In this paper we have analyzed the deadlock detection and resolution techniques that are used. We have reviewed in detail the algorithms presented by B. M. Alom[2] for deadlock detection and resolution in distributed environment and found that when we rearrange the order of the transactions pair in the algorithm by Aloms[2] then it completely fails to detect the deadlocks

Keywords: Distributed database system, Wait-For-Graph, Deadlock detection, Timestamp, Wait-for-graph

1. Introduction

Distributed database systems (DDBS) consist of different number of sites which are interconnected by a communication network. In such a resource sharing environment the database activities can be performed both at the local and global level so if the allocation of the resource is not properly controlled than it may lead to a situation that is referred to as deadlock. 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 activities such as read, write, lock, or unlock operations. If the actions of a transaction involve data at a single site, the transaction is said to be local, on the other hand a global transaction involve resources 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. Deadlock representation techniques for their easy detection have been discussed widely in the literature and graphical representation has been found to be most 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 P and needs to lock item Q, T2 has locked item Q and needs to lock item P. In this case the transactions are waiting for each other and no transaction can continue resulting into a deadlock.

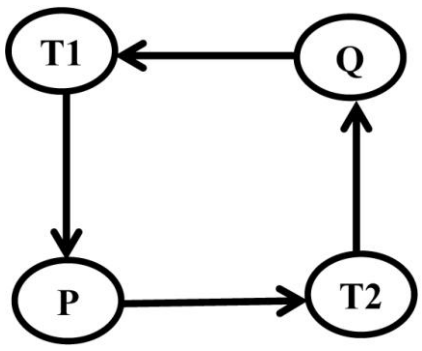

Fig 1: Transaction Wait for Graph

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.

We have analyzed in detail the algorithms presented by B. M. Alom[2] for detecting and resolving deadlocks in distributed environment. In section 2, we discuss about the transaction model. In section 3, 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 4, we take an example to compare the working of techniques by B. M Alom[2] in details. In section 5, we give the concluding remarks.

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

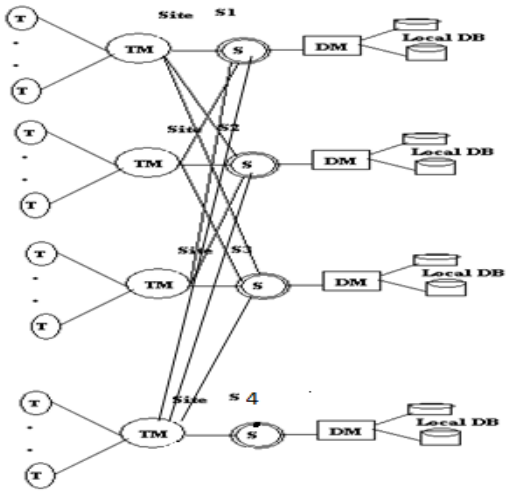

Fig 2: Distributed Transaction Model

The Transaction manager controls the execution of the different transaction which requires the data item for their execution. 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. In the paper we discuss the contributions of other researchers and the algorithms they have used for dealing with deadlocks. Chandy et. al. [4] 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 issued if a transaction begins to wait for another transaction and 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 computation gets back the probe, then it is involved in a deadlock. They found that this scheme does not suffer from false deadlock detection. Menasce D. A. et. al. [9] 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. [14] 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. B. M. Alom [2] has introduced the deadlock detection and resolution technique which uses the concept of creating the structure table named LTS and DTS for detecting the local and global deadlock. In this algorithm whenever a deadlock cycle is detected, the priorities of the transactions constituting the deadlock are 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 we rearrange the order of transaction pair in LTS and DTS than this algorithm completely fail to detect the deadlocks.

4. Illustrative Example For Performance Analysis

Let us take two sites, Site1 and Site2. A number of transactions are running on both the sites. In this example we have considered transactions T1, T2, T3, T4 executing on Site1 and transactions T5, T6, T7 executing on Site2 as shown in Fig. 3. We have analyzed the techniques in the following sub sections.

4.1 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 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 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.

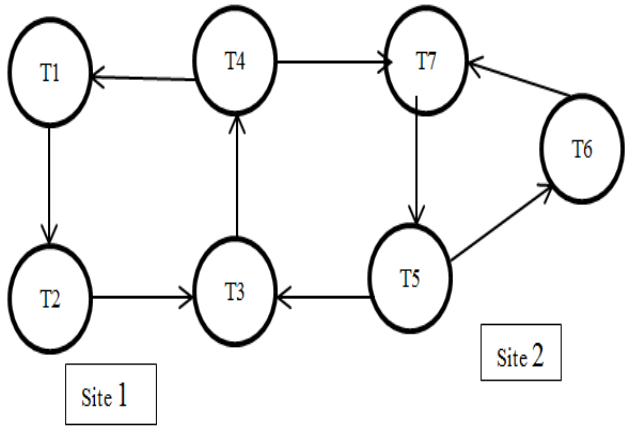

Fig 3: A Distributed environment having 2 site

Table 1: LTS for Site1 and Site2

p	q
1	2
2	3
3	4
4	1

p	q
5	6
6	7
7	5

Table 2: Transaction queue with the priority_id

Tx_No	P_id
1	1
2	2
3	3
4	4

Tx_No	P_id
5	1
6	2
7	3

In LTS_1 (Site1) there is one deadlock cycle {1→2,2→3,3→4,4→1}and in LTS_2 the deadlock cycle is {5→6,6→7,7→5,}.According to table 2 the youngest transaction with lowest priority id is selected and it is aborted to make the system deadlock free. In the given scenario according to table 2 for LTS_1 and LTS_2 the transaction 4 and 7 is selected as they have the lowest priority. The transaction pairs {4→1} and {7→5} are aborted. Now for detecting global deadlock, the intra connected transaction's (those are connected to other sites) are seen to find out if there exist a global deadlock cycle. The DTS (Distributed transaction structure) for Site1 and Site2 is shown in Table 3.

Table 3: DTS for site 1 and site 2

p	q
3	4
4	7
7	5
5	**3**

The transaction queue for DTS is considered as it is created for LTS.As a result of which the transaction pair {5→3} is aborted to free from global deadlock.
According to this approach, the TWFG without having any local or global deadlock cycle is shown in fig 4

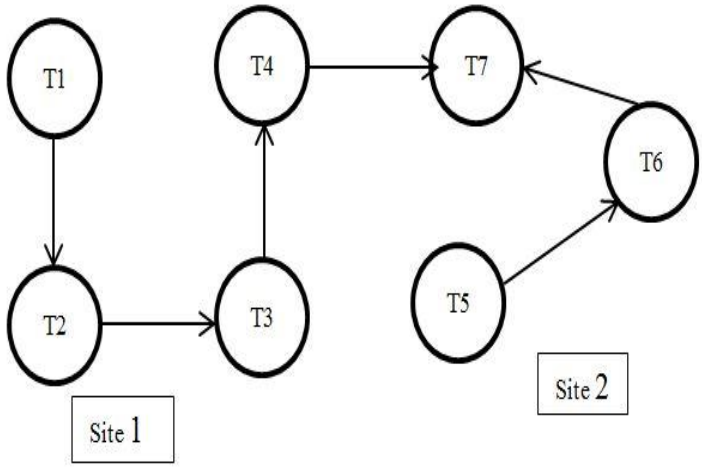

Fig 4: A deadlock free TWFG of the two sites

We analysed that if we rearrange the transaction pair in LTS and DTS the BM alom [2] technique completely fails to detect the local and global deadlock as there is a dependency of LTS and DTS structure on the directed edges of transaction wait for graph. We can eliminate this problem by assigning unique timestamp to each transaction.

5. Conclusions

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 of the systems. After analyzing various techniques we have found that the technique presented by B. M. Alom (section 4.1)[2] 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. It means that there is a complete dependency on directed edges of wait for graph in the B.M Alom technique [2]. The concept of time stamping could be used to abort the younger transaction in our future work.

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]. Alom B.M. Monjurul, Frans Alexander Henskens, Michael Richard Hannaford,"Deadlock Detection Views of Distributed Database", IEEE Sixth International Conference on Information Technology: New Generations, Page(s):730–737, 2009

[3]. 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.

[4]. Chandy K. M., Hass L. M and Misra J, Distributed Deadlock Detection, ACM Transaction on Computer Systems, Vol.1, No.2, pp.144-56, 1983.

[5]. Elmagarmid A. K. A Survey of Distributed Deadlock Detection Algorithms, SIGMOD RECORD, Vol. 15, No.3, pp. 37-45, 1986.

[6]. Gray J., A Straw Man Analysis of the Probability of Waiting and Deadlocks in a Database Systems, IBM Research Report, 1981

[7]. 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.

[8]. 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

[9]. 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.

[10]. Merritt M. J. and Mitchell D. P. "A Distributed Algorithm for Deadlock Detection and Resolution," in ACM, Vol. 2, No. 3, pp. 95-99, 1984.

[11]. Singhal Mukesh "Deadlock Detection in Distributed System" IEEE transaction on Software Engineering, Vol. 4, No. 3, pp. 195-199, 1989.

[12]. Jain Kamal, MohammadTaghi Hajiaghayi and Kunal Talwar, The Generalized Deadlock Resolution Problem, autoomata, Languages and Programming, Lecture Notes in Computer Science, 2005, Volume 3580/2005, 103, DOI: 10.1007/11523468_69

[13]. 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

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

[15]. Selvaraj Srinivasan, R. Rajaram, A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems, Distributed and Parallel Databases, Vol. 29, No. 4,pp. 261-276, DOI: 10.1007/s10619-011-7078-7

[16]. Soojung Lee, 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

[17]. T. Ozsu and Valduriez P., "Principle of Distributed Database Systems," Prentice Hall, 1999.

