



Approaches for Deadlock Detection and Prevention in Distributed Database System

Swati Gupta

Lecturer, Department of Computer Science
Amity University Haryana ,Gurgaon, India

Abstract: A distributed database system consists of different number of sites that are inter connected by a communication network. 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. The efficiently resolving of a deadlock in a system is important because the persistence of deadlock has two major disadvantages: First, all the resources held by deadlocked processes are not available to any other process. Second, the deadlock persistence time gets added to the response time of each process involved in the deadlock. In this paper, we discuss deadlock detection techniques and present approaches for detecting deadlocks in Distributed Systems.

Keywords: Wait-For-Graph, Deadlock, Transaction, Distributed database, Deadlock detection, lock manager

I. Introduction

A distributed database can be defined as a database environment consisting of a collection of data with different parts under the control of separate DBMSs running on independent computer systems. All the computers are interconnected and each system has autonomous processing capability serving local applications. Each system participates, in the local as well as in the execution of one or more global applications. Such applications require data from more than one site. The distributed nature of the database is hidden from the users.

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.

In distributed systems, [9] several transactions may compete for a finite number of resources. Upon requesting a resource, a transaction enters a wait state if the request is not granted due to non-availability of the resource. A situation may arise wherein waiting transactions may not ever get a chance to change their states. This can occur if the requested resources are held by other similarly waiting transaction. This situation is called deadlock.

Once a deadlock occurs in the system the processes that are involved in a deadlock are blocked so a deadlock detection technique should be carried out so that deadlock can be detected in finite time. In the deadlock detection technique one of the transaction known as victim transaction is selected and it is aborted so as to resolve the system from the deadlock situation. *Victim selection can be based on many parameters such as:*

- Choosing the youngest transaction as the victim
- Choosing the shortest transaction (transaction with fewest required data items) as the victim
- Choosing the transaction with the lowest restart overhead as the victim

The deadlocks in distributed database are of two types: the local deadlock where deadlock occur among the transactions present at a single site and global deadlock occur among transaction present at different sites.

A deadlock can be indicated by a cycle in the directed graph called Wait-for-Graph (WFG) [3] 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 K has locked data item R and needs to lock item M , transaction L has locked item M and needs to lock item R . In this case the transactions are waiting for each other and no transaction can continue resulting into a deadlock.

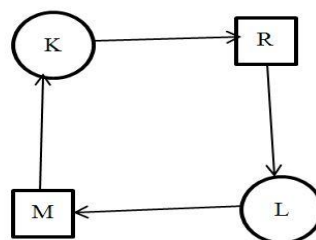


Figure 1: Transaction Wait for Graph

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

- **Deadlock Avoidance:** Deadlock avoidance [16] 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,[15] 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, I have discussed deadlock detection techniques and present approaches for detecting deadlocks in Distributed Systems.

II. Approaches for Deadlock Detection for Distributed Systems

In deadlock detection approach if the data item required by the transaction is not locked, the lock manager grants the requested lock to the requesting transaction but if the data item needed is already locked by some other transaction and the lock type is not compatible with the new lock request, the lock manager makes the requesting transaction wait. It is obvious that, without precautions, some of these transactions may end up in a deadlock after a while. A transaction, known as victim, needs to be rolled back to break the deadlock cycle. Some of the basic approaches for deadlock detection are as follows

- **Edge Chasing Algorithm:** This approach uses transaction wait for graphs (TWFG) to represent the status of transactions at the local sites and uses probes to detect global deadlocks. The algorithm by which a transaction T_i determines if it is deadlocked is called a probe computation. A probe is issued if a transaction begins to wait on 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 and are distinct from requests and replies. 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. This scheme does not suffer from false deadlock detection even if the transactions do not obey the two-phase locking protocol.
- **Path-pushing algorithms:** The basic idea underlying this class of algorithms is to build some simplified form of global WFG at each site. For this purpose each site sends its local WFG to a number of neighboring sites every time a deadlock computation is performed. After the local data structure of each site is updated, this updated WFG is then passed along, and the procedure is repeated until some site has sufficiently complete picture of the global situation to announce deadlock or to establish that no deadlocks are present. The main features of this scheme, namely, to send around paths of the global WFG, have led to the term path-pushing algorithms.

III. Distributed Deadlock Prevention

Deadlock prevention is an approach that prevents the system from committing to an allocation of locks that will eventually lead to a deadlock. In other words, it is impossible for a deadlock to occur in a system using a deadlock prevention approach. Basically it states that “when two or more transactions require conflicting locks for the same data item, only one of them will be given the lock.” One example of such implementation is the preacquisition of all locks algorithm.

The Preacquisition of All Locks Algorithm: The preacquisition of all locks is an implementation example of deadlock prevention. In this scheme, transactions are required to lock all the data items they need before they are allowed to start their work and are also required to hold onto these locks for the entire duration of the transaction.

For example, suppose a transaction needs three data items. The transaction will ask the lock manager to lock all three items before it starts executing any commands inside the transaction. If any data item that is needed by this transaction is locked by some another transaction, then the transaction will have to wait. As soon as all the locks needed by the transaction are obtained, the transaction then proceeds. This approach is to say that deadlock prevention deals with deadlocks “**ahead of time**”.

This approach is suitable for transactions that know all the data items they need a priori. This is, of course, somewhat limiting with respect to the types of transactions that can be processed. The deadlock prevention approach can often lead to **Transaction starvation** which usually happens to large/long transactions that need to

lock many data items. Because a long transaction needs to lock many data items, it is possible that while it is waiting, some other smaller/shorter transactions (requiring fewer data item locks) will succeed in locking the items that it needs. If the shorter transactions keep locking items required by the longer transaction than in this case the longer transaction might be forced to wait for a longer time and ultimately will never receive the chance to acquire its needed resource.

IV. Conclusion

Deadlock is referred as a condition when two or more processes are each waiting for the other to release a resource. Deadlock is a common problem in multiprocessing system where many processes share a specific type of mutually exclusive resource known as a software lock or soft lock.

Several deadlock detection algorithms have been proposed for distributed system, a number of issues remain to be addressed. Future research should focus on efficient resolution of deadlocks. In this paper I have discuss distributed deadlock detection and prevention approaches which are being used to handle the deadlock situation.

V. 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 Transaction 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 a 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," in ACM, 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, automata, Languages and Programming, Lecture Notes in Computer Science, 2005, Volume 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, 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 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
- [15]. 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
- [16]. T. Ozsu and Valduriez P., "Principle of Distributed Database Systems," Prentice Hall, 1999.