A Survey on Database Deadlock Detection

Gowdhami D¹ and Aruna Devi P²

¹Research Scholar, Dr SNS Rajalakshmi College of Arts and Science, Coimbatore
gowdhami.d@gmail.com

²Assistant professor, Computer Technology Department, Dr SNS Rajalakshmi College of Arts and Science, Coimbatore
aruna7825@gmail.com

Abstract
Now days many industries are developing an application program using database by sending Structured Query Language (SQL) statements to them and also finding data by executing the SQL Statements. If the applications are using the same database concurrently sometimes database deadlock will be occurred. Testing applications to determine how they cause database deadlocks is important as part of confirming correctness, reliability, and performance of these applications. But it is very difficult to reproduce database deadlocks.

Keywords: Deadlock, Software Testing and Database.

1. Introduction

A deadlock is defined as the situation where two or more processes permanently block each other by acquiring the lock on a resource that has already been requested by another process. This means one process is trying to lock the resource while the other one has already locked it.

In concurrent programming, a deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.

In a transactional database, a deadlock happens when two processes each within its own transaction updates two rows of information but in the opposite order.

In Operating System, Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.

2. Various Modes of Deadlocks

The transaction in deadlock waits until it is broken by some external process. There are distinct states of Deadlock; also known as Coffman Conditions and these include:

A. Mutual Exclusion: At a given period of time one resource is assigned to a single process.

B. Resource Holding: A resource is held by a process which is further waiting for another resources held by other executing process in the loop.

C. No Preemption: When a process is being completed, it can release the resource that it holds while execution phases.

D. Circular Wait: In this state, a process is waiting for a resource to be released by another process which is in turn waiting for some other resource. At the same time, Scrum Master also understands the time wasted and impact on the team when individuals sit on or ignore problems.

3. Database Deadlock

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks.

For example, Transaction A might hold a lock on some rows in the student table and needs to update some rows in the result table to finish. Transaction B holds locks on those very rows in the result table but needs to update the rows in the student table held by Transaction A. Transaction
A cannot complete its transaction because of the lock on result. Transaction B cannot complete its transaction because of the lock on student. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transaction.

Currently database deadlocks are typically detected within the database engine using special algorithms that analysis whether transactions hold resources in cyclic dependencies, and these database engines resolve database deadlocks by forcibly breaking the hold-and-wait cycle.

4. Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermined a deadlock situation.

5. Wait For Graph

This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction Ti requests for a lock on an item, say X, which is held by some other transaction Tj, a directed edge is created from Ti to Tj. If Tj releases item X, the edge between them is dropped and Ti locks the data item.

The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there’s any cycle in the graph.

6. Process Of Detecting Deadlock

Deadlock Detection scheme first allows the deadlock to occur and then examine it for detecting that it has been occurred; to further correct it. An algorithm is been utilized to detect the allocation of resources that are further restarted or rolled back some of the processes for removing the detected deadlock.

After detecting the deadlock, the Database Engine dismisses the existing batch and the transaction is rolled back and displays error 1205 to the respective application. When a transaction for deadlock is rolled back, the victim then releases all the locks that the specified transaction holds.

Error 1205 records the information in the error log for all the processes/threads or the resources that are involved; or participated in the formation of particular deadlock. Now, when the deadlock is been detected, it can be corrected by implementing either of the given methods:

**Process Termination:** The chain of executing and the waiting processes in the deadlock can be aborted. Either some or entire processes in the queue may be dismissed to ensure fast and speedy resolution.

**Resource Allocation:** The allocated resources are successfully released by the processes and are further allocated to the waiting processes in the loop.
7. Approaches for Deadlock Detection

STEPDAD: Systematic Testing in Presence of Database Deadlocks that enables testers to Instantiate database deadlocks in applications with a high level of automation and frequency. STEPDAD represents relational databases as sets of resources (e.g., database tables) and transactions that DCAs issue to databases as sets of abstract operations.

Lock graphs are used in this approach to detect hold and-wait cycles in transactions. Cycle detection algorithm has been applied in this approach. In some cases, STEPDAD reproduced a database deadlock after running an application only two times, while no database deadlocks were reproduced after ten runs using the baseline approach.

Edge-chasing algorithms: The presence of a cycle in a distributed graph structure can be verified by propagating special messages called probes along the edges of the graph. Probes are assumed to be distinct from resource request and grant messages. When the initiator of such a probe computation receives a matching probe, it knows that it is in cycle in the graph. A nice feature of this approach is that executing processes can simply discard any probes they receive. Blocked processes propagate the probe along their outgoing edges.

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.

Blocked-List Deadlock Detection Algorithm: Based on the concept of a message passing through the entire blocked processes during the execution. It is actually presented for distributed systems that consist of processes and resource managers. The deadlock would be detected by one of the blocked processes and that process sends the list of all blocked processes on the deadlock cycle to the coordinator for recover.

Transaction Wait-for-Graph (TWFG): This technique represents 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.

8. The Concept of Minimizing Deadlocks

Though it is not possible to avoid or eradicate the occurrence of Deadlock but this can be reduced to some extent. The following coding conventions can be used to lower the occurrence of Deadlock state.

View Objects in Actual Order: The transactions access object in exactly same order in which they were generated. When the executing process is completed or rolled back, the second process in the queue will be started and the same procedure continues for all the following processes.

Avoid User Interference in Transactions: Avoid transaction that involves user interaction such as the processes that requires user’s manual response. The queries or processes that run without manual interaction are much faster than those that are bounded with user interventions.

Minimizing the Volume of Transactions: The major cause behind the occurrence of Deadlock situation is the large size of transactions. Minimizing the volume of transactions and retaining them under one batch can help in resolving this issue as the longer transactions holds resources for longer period of time till the process gets completed or rolled back. Keeping the transactions under single batch reduces the load over network as then it does not have to take huge round trips.

Ample Resources for the Transactions: Ensure that there are ample of resources available corresponding to the processes so that each transaction is allocated with at least one resource. More the resources, faster will be the process execution time.

Prevent Locking Contention: By running the transaction at lower isolation level, the occurrence of deadlock situations can be minimized. Implementation of READ_COMMITTED on transactions allows them to read the data that is previously read but not modified; by succeeding
transaction without the completion of previous transaction.

9. Conclusion

Deadlock is a major problem in operating systems, Database and also concurrent programming. However there are several techniques to deal with deadlock such as deadlock avoidance, prevention etc. but still deadlock can occur. In this paper we have seen about the deadlock, database deadlock and how to detect deadlock detection and also various approaches used for deadlock detection. And also we discussed how to minimizing deadlock I deadlock in database.

References

[6] https://db.apache.org/derby/docs/10.0/manu als/develop/develop75.html