Developed S2PL in Transaction Concurrency with Emphasis on Deletion of Convoy Phenomenon and Improvement of Starvation of Transactions

Mehdi Asadi, Mehdi Khanbabaie

1Department of Computer Science, Islamic Azad University - Khamneh Branch, Khamneh - Iran
2Department of Computer Science, Ap lied and Science University - Shabestar Center, Shabestar - Iran

Abstract: The investigation on the concurrency control has led to develop more algorithms for concurrency control. Most of these algorithms are based on three basic mechanisms like locking, timestamp and optimistic concurrency control. The key issue of this paper is the method of strict two phase locking and the aim is to improve this method. The S2PL method under particular circumstances has reached a deadlock and it leads to problems such as convoy phenomenon. The proposed method and the results obtained from the implementation of transaction show that this method not only resolves the convoy phenomena, it also increases the efficiency of process in transaction and paves the way for starvation issue of transaction to win the required resources.

Key words: Concurrency Control, 2PL, Strict Two Phase locking (S2PL), Convoy Phenomenon, Starvation

INTRODUCTION

The concurrency means implementing two or more non-consecutive transactions on some common data in order that access to data has occurred simultaneously on the condition that concurrent implementing of them does not lead to wrong conclusion. This wrong conclusion is reached when there is no fine control and management over interleaving and then creating of impact event leads to incorrect implementation of concurrent transaction, even though each of transactions is correct independently. In principle, every transaction may cause a problem and fail to complement due to two reasons (Garcia et al, 2009):

1. Revealing of problem or failure in simple operations like reading bad sector data or main operations such as failure in telecommunication links.
2. Transaction may lead to no problem but under no fine control may cause an incorrect result. For instance, T2 transaction may use effects and alterations of T1 transaction, but after a while and under any reason transaction is aborted and is not committed and all alterations that were implemented over common data return to first state. In such case we will encounter an incorrect result.

Types of implementing concurrent transactions
1. Interleaved Concurrency: operations of transactions are interleaved.
2. Simultaneous Concurrency: operations of transactions run simultaneously.

Problems of the Concurrency:

1. Lost Updates:
   T1 transaction changes the Q data at q time and then, T2 transaction changes the same Q data. Afterwards T1 transaction reads an amount of Q data and expects to read the same amount that it has written without notice that T2 transaction has changed data to new extent.

2. Dirty Data Read:
   T1 transaction changes the Q data at any point of the time e.g. q. Then, T2 transaction reads the same altered data and uses that in its operation but subsequently, under any reason, T1 transaction is aborted and failed to commit and it must turn all alterations to first state over Q data. But T2 transaction has used this altered data in its operation that it is wrong. This problem is regarded as the Dependency on Non-Stabilized Phenomenon (Conn, 2002) and often called Ad Hoc Co-occurrence (Elma, 2000).
3. Inconsistent Analysis:
This problem reveals when T1 transaction changes and co-occurs Q1, Q2 data and T2 transaction reads an amount of Q1 and Q2 before and after co-occurring, respectively and its operation is wrong. Sometimes this problem is called Incorrect Obtained Sum.

4. Unrepeatable Read or Fuzzy Read:
This problem occurs when T1 transaction wants to read Q data again, but Q data just for once after reading is altered by T2 transaction, therefore T1 transaction cannot read the same amount of Q data again and it is a critical problem.

Consecutiveness:
Given there are n transactions here, their concurrent performing design is consecutive if concurrent performing of n transactions has the result like consecutive implementing of the same n (Chaitanya, 2005). Concurrency control of transactions is accomplished by a mechanism called Scheduling. In this method, each transaction is composed of a body of operations. These operations do not have similar performing time in different transactions and following the way that concurrent implementing of them on each other has not the negative effect. This method should be based on three maxims (Silberschatz, et al., 2001):
1. Each transaction should be corrected independently, it means that independent implementation leads not to change the state of compatibility of data base; it does not happen if all transactions reach their implementing stages.
2. Each design of transactions implementing in consecutive way is correct too, because transactions are correct and they work independently from each other, in other words, the impact of function in different transactions does not exist.
3. A design for concurrent implementing of transactions is correct if it has an impact and effect similar to a consecutive design means that it is consecutiveness.

As mentioned earlier, consecutiveness is a feature for a series of transactions, in other words if a set of transactions in a consecutive design is able to be performed (JOHN 97), we could say that this set has the feature of consecutiveness. The data base system must control the concurrent implementation of transactions to guarantee that the data base condition after implementing of transactions remains compatible.

Concurrency Control:
Concurrency control is a mechanism which makes a compromise among access to database in the database management system by several users. Concurrency control gives permission to users to work on multi-programmer system; from user’s point of view, the system behavior is likes that the user supposes his activities take place in one-programmer environment (Bhargava, 1999). To cope with torment reveals in extracting information by another user is the most difficult problem into this system. The issue of concurrency control has considered during years ago the aim is to prevent difficulties when users enjoy a series of common data for several users (Thomasian, 1998).

In order to concurrency control different algorithms in database are offered which you can see on figure 1. To survey, we can reach this fact that all of them are the combinations of some limited sub-algorithms. In fact these sub-algorithms are the different copies of 2 main techniques of concurrency control such as Locking and Timestamp (Najadat, 2006).

Rules of Locking:
One of the ways we can be assured of consecutiveness is that we need to access information by the method of Mutually Exclusion. It means when a transaction has the right to access information, none of other transactions has the right to edit or alter the same data. General method to win this need (Mutual Exclusion) is that we give permission to transactions to access the data just if they have locked the same data in advance.

Locks:
In order to lock data, several methods have been used. Here, let us review two methods:

1. Shared Method:
If Ti transaction does a shared lock on Q data, so Ti transaction can read the Q data but has not the right to write on, this kind of locking is represented by S.
Exclusive Method:

If Ti transaction does an exclusive lock on Q data, Ti transaction can either read or write on. We need that each transaction requests a suitable locking stated on the kinds of orders which implements on Q data. Transactions offer their lock requisitions to concurrency control manager. Transactions can implement their orders on data if the concurrency controls manager leaves a lock to them in advance. In different kinds of locking, there is a series of functions and rules of compatibility we will consider them afterwards. If we mention A and B as treaties for locking modes suppose that Ti transaction makes a request from mode A for a lock on Q data which this Q data is locked by Tj (Ti≠Tj) transaction from mode B in advance. If transactions can get a lock on Q data immediately and instead of available mode of lock B: we can say that mode A is compatible with mode B. In this case a function can be presented in a matrix simply. Compatibility relationship between two locking mode is take under debate here that is appeared in comparison matrix on figure2. The value of a cell in this chart will be true if and only mode A is compatible with mode B.

Remember that the shared mode can be compatible with other shared mode but in the case of the exclusive data it cannot be compatible. At once several shared locking modes from different transactions can place on a particular data simultaneously. The next requisition of exclusive lock is suspended until the shared locks give up. A request of transaction for does a shared lock on Q data is presented by the order of LOCK-S (Q). Thus, the request of transaction for exclusive lock is shown by LOCK-X (Q). For data access, transactions must get a lock on data in advance, but if a data was locked by an incompatible mode, the currency control manager would not permit transactions to lock if locker transaction releases its lock. Therefore, Ti transactions must suspend until locks release by other transaction.

One transaction can release a lock if it does a lock in advance (Figure 3). Remember that transactions must access to data until they maintain the lock on it and immediately after concluding the functions leave it, because it may fail the consecutiveness of the system, to maintain a lock on a data is not a suitable task.

![Fig. 1: Classification of Algorithms in the Concurrency Control](image)

![Fig. 2: Comparison Matrix](image)
Granting Of Locks:
If a transaction makes a request about a lock on a particular mode in a data and none of the transaction has a lock on data in a hittable mode the lock can be gotten by transaction, but this solution is threatening and we must eschew it due to following reasons. Suppose that T2 phenomenon has a shared lock on a data and another transaction likes T1 requests an exclusive lock on the same data. As such, T1 must wait for T2 to release its lock on data. Besides, we may have T3 transaction which wants to make a shared lock on the same data. The request of T3 lock is compatible with Q lock is in hand of T2, so T3 can get a shared lock on specified data too. In this part of time, T2 may release its lock but T1 must wait for T3 to release its shared lock that gets T1 and T2 during suspending time. In fact it is possible that we have a condition in which a series of transactions get a shared lock on a data such as Q and after a while they get it, want to release but T1 which requests an exclusive lock on Q data never reaches a success to get a lock forever and in this case we could say T1 has no progress and strikes by starvation.

Two-Phase Locking Protocol:
Two-phase locking is one of the protocols that provide consecutiveness. This protocol needs to implement the requests of locking and unlocking in phase2 (figure 4).

1. Growing Phase:
One transaction may get a lock but perhaps it may release none of them.

2. Shrinking Phase:
One transaction may release a lock, but perhaps it gets no new lock.
At first one transaction is in growing phase and it gets locks that need them. When transaction releases a lock, it enters shrinking phase and cannot get a lock. We can show that two-phase locking in each phenomenon provides encounter able consecutiveness (Jumah, 2002). The point that gets the scheduling design of the last lock (end of growing phase) is called Lock Point of transactions. Now, transactions can be arranged by using lock point. In fact, it is a consecutive combination of phenomenon. Two-phase locking cannot guarantee your saving in deadlock.
Cascading Rollbacks may occur on two-phase locking protocol but we can eschew the cascading rollbacks by limited two-phase locking which is a reformed two-phase locking. This protocol needs to occur in two phases but all obtained locks of a transaction are maintained until it is implemented. This method guarantee that no data is written by an uncommitted transaction which has exclusive lock unlike this transaction is implemented correctly and it prevents from reading a data before successful implementing of writing task.
Strict Two-Phase Locking Protocol:

Lock method for concurrency control of transactions uses a two-phase mechanism. In two phases mentioned for two-phase locking method, the way for getting and releasing locks may cause following problems:

Cascading Abort:

This problem in two-phase lock is explained like this: If a transaction is aborted, some other processes that use the data which is altered by aborted transactions must be aborted too. This condition is called Cascading Abort.

Dirty Read:

This problem reveals if suppose that T1 and T2 are implemented by interleave and T1 does a exclusive lock on a data and releases the lock after finishing its work and T2 continues its task on data and then before committing T1 transactions under unclear reasons leads to fail and turns the data to its first amount.

In this case T2 continues its task by an invalid data and this causes an incompatibility into implementation. To prevent a cascading abort in releasing the locks, the S2PL method (Xiaopeng, 2003) is used. In this case specified transaction maintains all locks to end, it means that releasing task happens if it is aborted or committed. To fence Dirty read problem in the phase of releasing, transaction does not give permission to release locks if the exclusive lock is aborted or committed.

The S2PL method plays a great role in resolving the two-phase locking problem. Through this method in the phase of Growing, locks are obtained in order. It means that at first point a transaction needs a lock, it grants to transaction but locks remain lock and at last they release altogether and the waste of resources reduces. This method in comparison with other methods needs to more simultaneity and implementing of it is simpler. So, this method is used more than other two-phase locking algorithms.
Problem of Convoy Phenomenon:

In the S2PL method, we can pave the ways of two-phase locking by management in the phase of releasing the locks. Little by little allocation of locks may lead to a deadlock. One of these problems is Convoy phenomenon. In this case locking task is implemented step by step; it means that T1 transaction tries to lock S1 source which wastes more time, T2 transaction locks S2 source and waits for releasing of S1 and T3 process locks S3 and waits S2 to release and this circle is continued. All transaction wait for a lock which is in hands of other transaction. This problem in S2PL method is called Convoy Phenomenon. Therefore, a body of transactions and resources is chained together in following figure 6 and none of them can continue its task (Joudi Begdillo et al., 2007). This chain leads to a deadlock. If we can break this chain in particular point, we win to resolve this problem and unlock the locked link.

Fig. 6: The Problem of Convoy Phenomenon.

Suggested Solution

To resolve the problem of convoy phenomenon and to break this chain in a particular location, we suppose that the database system has a body of transactions and resources and the turns for Run Count and Run Time Estimate are registered for every transaction in database and given to scheduler. Every transaction which needs to a resource announce its demand to scheduler and one of the following tasks is implemented by system scheduler:

The suggested source is free and is in hands of no transaction. In this case through a Timestamp (e.g. system watch) a numerical value is given to resource and afterwards is given to transaction. The requested resource is allocated to other transaction in advance. In this case system scheduler by means of below formula considers that which transaction (resource requested transaction, or resource owner transaction) has the right for granting the resource.

\[
P(T) = \frac{C_1 \times \text{TimeStamp}(T)}{\text{MaxTimeStamp}(T)} + \frac{C_2 \times \text{RunCount}(T)}{\text{MaxRunCount}(T)} + \frac{\text{EstRunTime}(T)}{\text{MaxEstRunTime}(T)}
\]

\[
C_1 + C_2 + C_3 = 1, \quad \text{MaxEstRunTime}(T) = \text{MaxEstRunTime\{T_1, T_2, \ldots, T_n\}}, \quad T = \{T_1, T_2, T_3, \ldots, T_n\}
\]

\[
\text{MaxRunCount}(T) = \text{MaxRunCount\{T_1, T_2, \ldots, T_n\}}, \quad \text{MaxTimeStamp}(T) = \text{MaxTimeStamp\{T_1, T_2, \ldots, T_n\}}
\]

In above formula, Timestamp(Ti) is the allocated timestamp by Ti, MaxTimeStamp(T) is the maximum time among timestamp, RunCount(Ti) is the number of times for implementing Ti from beginning of database task, MaxRunCount (T) is the number of times for implementing among transactions, EstRunTime(Ti) is the estimated time for implementing Ti and MaxEstRunTime(Ti) is the estimated maximum time for implementing.
the transactions. We can choose a value of C1, C2 and C3 stated on the kind of database. Every transaction which gets a little amount of above formula can grant and implement its source. This function is performed for all resources and resource requested transactions, therefore, at least one of the available transactions in chain cannot maintain it’s granted or requested resource so that the locked link changes to unlock and it prevents of causing convoy phenomenon in S2PL method. For example, we suppose the evaluation for two transactions such as T1 and T2 with abstract parameters which perform concurrently. Figure 8 shows the locked link and creating of a deadlock.

In accordance with the suggested method, T1 transaction requests S2 resource and because this source is obtained by no transaction, for example, a scheduler allocates timestamp 100 to S2 source and T1 phenomenon. Afterwards T2 phenomenon requests S1 and because this source is frees, the scheduler allocates timestamp 110 to it and T2 phenomenon. Then, T1 phenomenon requests S2 source. In this case scheduler by means of suggested formula for every source makes a competent function and calculates a value for every phenomenon and makes a decision for allocating a source to transaction, it means that the source is given to a phenomenon which gets the least value in calculating the \( P(T_i) \) function. Here we allocate the most of C value to time of estimated performing of transaction and Timestamp.

\[
C_1 = 0.4 \quad C_2 = 0.2 \quad C_3 = 0.4
\]

\[
p(T_1) = 0.4 \times \frac{110}{110} + 0.2 \times \frac{4}{4} + 0.4 \times \frac{12}{12} = 0.36 + 0.2 + 0.4 = 0.963
\]

\[
p(T_2) = 0.4 \times \frac{110}{110} + 0.2 \times \frac{1}{4} + 0.4 \times \frac{5}{12} = 0.4 + 0.1 + 0.16 = 0.616
\]

Now, high competence of T2 transaction changes its state from unsecured deadlock to a secure one in figure (9).

Therefore, it prevents of a locked link, in this case because of high influence of time of estimated implementing transactions is mentioned in formula and the influence of turns of implementing and starvation will reach to its least amount.

Fig. 7: Preventing of Convoy Phenomenon.
Fig. 8: A deadlock for two phenomena and essential parameter chart.

Fig. 9: Influence of Suggested Method for 2 Transactions.

**Conclusion:**
One of the most important issues in implementing several transactions is database simultaneity which must prevent from an impact in different mechanisms. However, lock method prevents this impact by suspending the transaction, but it has special problems in substance. Several algorithms are suggested for improving lock method which S2PL is one of them and its simultaneity and implementing is simple. But S2PL itself has the problem of convoy phenomenon and starvation, it means that transactions are waited for a source which is in hands of other transactions and sometimes they wait for a source until suffer from starvation. To resolve this problem, in this paper a method is suggested to give priority in obtaining a source with special rules in granting-phase of source is S2PL prevents from convoy phenomenon and reaches the starvation value to its least amount.

**ACKNOWLEDGMENT**
This work is supported by Research Center of Islamic Azad University, Khamneh Branch in Iran under Grant 54004881223001. The author would like to thank Dr. Ali Ghafari for constructive comments and suggestions.

**REFERENCE**