Volume 2, Issue 5, May 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

An Innovative Approach for Two Way Waiting Algorithm in Databases for Concurrency Control

Sunil Ahuja
Department of Computer Science & Engineering
Doon Valley Institute of Engineering & Technology
Karnal, India
E-mail: ahujaksunil@gmail.com

Ms. Reena Sharma
Department of Computer Science & Engineering
Doon Valley Institute of Engineering & Technology
Karnal, India
E-mail: er.sharma.reena@gmail.com

Abstract—**Database is the gathering of information that is stored in such a way that it can be accessible to each and every user. The Main purpose of database system is to retrieve information; perform some operation on the information and storing complete information back to the database. It should contain the efficient information so that each user can access the final result of the operations. Database is the well-organized collection of data in a meaningful way that user can perform the transaction. Transaction is a sequence of many actions considered to be atomic. When the transactions are executing in parallel then it can lead to concurrency in database and also the deadlocks in the transactions. This approach will take us prevent the deadlocks in the transaction that will also increase the efficiency of the system by reducing the number of restart of the transaction.**

***Keywords*— concurrency, transaction, serializability, deadlock, direction, lock, timestamp**

I. INTRODUCTION

Database consists of a group of organized data and a set of programs to access that data. Database is the gathering of information that is stored in such a way that it can be accessible to each and every user. A Database is a gathering of information and structured of data in such a way that it can be accessed, updated and managed by the user in a very simple way. A database should not contain any duplicate data; if there is such a case then it should be abolished or minimized. When there is a collection of data and this data is accessed, updated then this concept is known as Database Management System (DBMS). When the data is stored at different locations or servers then this type of database system is known as Distributed Database. It is the collection of logical related data that work together in the crystal clear mode. This mode means that the user can access the data from the database of any system; this looks like that user is working on the sole database. When the data is distributed to different system; is accessed by users then this will lead to the concept of Distributed Database Management System (DDBMS). When there is a very fast growth of the computer networks then we have the requirement of the database application. Applications are required to maintain the database in the system, so that the efficiency of the system can be increased. A Database represents the aspects of the genuine humanity application. A few changes in the genuine humanity are revealed in the database. It is intended and put up with data for a definite purpose. It is a well-organized collection of data in a meaningful way that user can perform the transaction. A Transaction is a sequence of many actions considered to be atomic. When the transaction is successfully completed then it said to be committed otherwise it said to be aborted. When we want to increase the throughput of the system then we have to execute the transactions in parallel. When the transactions are accessing the same data at the same time then this is said to be concurrency. When the transactions are executed in parallel, there will be the problem of interfacing of transaction with each other. Then this concurrency will result into the lost-update problem, dirty read problem and inconsistent retrievals [1]. To solve out the above problems we have to make use of the concurrency control mechanism. The concurrency control is that in which a number of transactions are executed in parallelism in the shared database that will ensure the serializability of the transactions. It permits users to access a database in a multiprogrammed method while preserving the false impression that each user is executing alone on a committed system. It is an essential element for correctness in any system where two or more database transactions are executed with time overlap and can access the same data. Concurrency control mechanisms for transaction schedulers can be classified into two wide modules: pessimistic control techniques and optimistic control techniques [2]. With pessimistic techniques, the execution of conflicting transaction operations is delayed so as to synchronize transactions early on in their execution lifecycle (locking mechanisms).With optimistic techniques, execution of transaction operations is not delayed and the objects are accessed freely, but as a transaction attempts to commit a validation procedure determines whether committing the transaction would violate consistency of object's state thereby requiring the transaction to rollback.

II. PROBLEMS FOR TRANSACTIONS IN CONCURRENCY

When the transactions are executing in parallel mode then there will be problems of concurrency can occur and it will create the problems for the users to access that database for their transaction or work. When the transactions are performed without the concurrency control mechanism then there will be some problems like:

A. Dirty Read

This problem is also recognized as Uncommitted Data. This problem occurs when one transaction is performing and updates the database with the new result but later on fall short for any reason. The updated data is access (which is called dirty read) by any transaction then it will give the incorrect outcome or result. In this, if T1 and T2 are transactions if the transaction T1 updates the data in the database and this updated result is read by the transaction T2 but later on transaction T1 does not committed due to any reason then the data read by the transaction T2 will result into an incorrect outcome.

B. Lost Update

When the transactions are performed simultaneously on any data and update the database by their results then the first result is overwritten by the result of the other transaction. In this, if T1 and T2 are transactions and reads the record from the database and T1 update the record by its result after that the transaction T2 completes and it also update its result in the database then the result of first transaction is overwritten by second transaction.

C. Inconsistent Retrieval

This Problem is also known as Unrepeatable Read. When one transaction is performing a few works on the database and the other transaction is performing an update process on the database. Then this problem occurs because one transaction reads some results before they are changed and some after they are changed, then results in inconsistent retrieval of data. In this, if T1 and T2 are transactions working on the database. The transaction T1 is to get the total salary of all the employees in any organization and the transaction T2 is incrementing the salary of the employees in the organization then there will be problem that the transaction T1 gets some values before the completion of transaction T2 and some after the completion of transaction T2. Then T1 will result in inconsistent retrieval.

III. SYSTEM MODEL

The distributed database system (DDBS) consists of a set of data items ($X_{i,j,...}$ (Say)). A data item is the smallest available item of data. It may be a page, a record, an object or a file. In this paper, a, b, c.., stand for data items. Each data item is stored at one site only. However, this hypothesis does not put a ceiling on the algorithm and can be hassle-free in a global case. Transactions are represent by T_i,$T_{j...;}$ and sites are represented by S_i, S_j, .. .; where, i , j . . . are numeral values. The data items are stored at various database sites associated by a computer group. Each site supports a transaction manager(TM) and a data manager (DM).The transaction manager controls execution of the transactions and there exist some local transaction manager at each distributed site that is responsible for the transaction management at that site. The data managers manage individual databases and there exist some local data manager at each distributed site that is responsible for the managing the data at that site. The group is assumed to detect failures, as these occurred respond to the server. When a site fails, it simply prevents working and other sites sense this fact. The communication medium makes available the competence of message convey between dissimilar sites. A site always passes a message to the communication medium, which forwards and delivers it to the destination site in fixed time [3].For any couple of sites S_i and S_j, the communication medium always delivers the messages to S_j, in the same order in which they were conveyed to the medium by S_i.

Each site is implicit to run the strict two-phase locking concurrency control. Whether a data object ‘a’ is a page, a file or a record, the data object is crushed by the distributed system. A transaction that needs to access a given data objects must first get hold of the lock coupled with that data object. The strict two-phase locking requires a transaction to wait to release all its locks after it commits or aborts [4].

IV. TRANSACTION MODEL

Transaction is a sequence of many actions considered to be atomic. A transaction is a way by which an application programmer can enclose together a sequence of database operations, so that the database system can provide guarantees to the user that the transaction will strictly follow the transaction properties, these transaction properties are known as the ACID properties [5]. The transaction proceeds in three phases namely Read, Validate and Write phases.

Read: The transaction reads values from the database and writes them into a private system.

Validation: When the transaction is about to commit, it will be checked whether the committing transaction does not conflict with other parallel transactions. If a conflict exists then the transaction is aborted and restarted later on. Its terminal is also cleared.

Write: When the validation phase determines that the transaction does not conflict with other transactions then its private work space is copied into the database [6].

The performance of transaction processing systems with two-phase locking (2PL) can be degraded by transaction blocking due to lock conflicts and aborts to resolve deadlocks [7].An operation is either a read or a write. For any T_i and data item X, r[X] denotes a read executed by T_i on X. Similarly, w[X] denotes a write executed by T_i, on X. The notation O_i denotes an operation of transaction (T_i). In general, a transaction does not have to be a totally ordered in sequence. When two operations are not in order relative to each other, then these operations can be executed in any order. However, a read operation and a write operation on the same element must be ordered, so that there will not any conflict occurred between the transactions. Suppose there are two operations O_i[X] and O_j[X], these operations will not conflict with each other, if these operations are of read but they will conflict to each other if they operate on data item (X) and at least one of them is a Write operation.

A. SERIALIZABILITY IN A DATABASE

Let T = T_1…. T_i is a set of active transactions in a distributed database system where 'i' is the numeral. Correctness criteria for transaction execution in a distributed database are described below:

The term Serializability can be defined as "A serial schedule is a totally ordered schedule, if for every pair of transactions T1 and T2, either all of T1's operations precede T2's operations, or vice versa. In a distributed database, transactions perform operations at several sites. A sequence of operations performed by transactions at a site is a local schedule".

In this we assume that each transaction must maintain the database consistency. In serial implementation of set of transaction maintains the database consistency. It is the property of transaction history which relates to the isolation policy. The Serialization is of two forms:

1. *Conflict Serialization*: In this, when there are different overlapping transactions with read and write operation on the same database.
2. *View Serialization*: In this, when there are different non-overlapping transactions with read and write operation on the same database.

There will be testing of serialization [3] on the basis of precedence graph. Two phase locking may lead to the deadlock. Deadlock occurs when one transaction T_i in a set of number transaction is waiting for any resource which is locked by some other transaction T_j in the set.

B. TRANSACTION NUMBER

Every site S_i has a logical clock C_i, which takes a monotonically non-decreasing integer value [7]. A Transaction Number is assigned to the transaction T_i, on its arrival, by a site S_i. When the transaction has occurred on the site then the server has to assign the transaction number to the transaction. The transaction number can be assigned to the transaction on the basis of the counter value that will be incremented by one when the transaction has occurred and transaction number can also be assigned on the basis of the system clock on which the transaction has occurred.

V. AN INNOVATIVE APPROACH FOR TWO WAY WAITING ALGORITHM

Many methods for concurrency control exist. The major methods, which have each many variants,are:

A. Locking - Strict Two Phase Locking

The widely used commercial Strict Two Phase Locking guarantees serializability by first acquiring locks on all accessed resources (phase 1), performing the requested operations if all locks were granted and releasing the locks afterwards (phase 2)[8].

B. Serialization graph checking (also called Serializability, or Conflict, or Precedence graph checking) –

The nodes of WFG are labelled with active transaction names[6].In a WFG there exist an edge from Ti to Tj iff transaction Ti is waiting for transaction Tj to release some lock. Is there exist a cycle in WFG, it means deadlock has occur and broken by aborting a transaction[9]. Detecting distributed deadlocks ultimately relies on sending enough information to a single site to allow the deadlock detection algorithm to be run. The transaction chosen for abort is called the victim [10].

C. Timestamp Ordering (TO)

Timestamp based concurrency control is also called Timestamp Ordering (TO)[11]. This is an different approach to locking that make use of the timestamps [6][7].Timestamps are assigned to the transaction in a well ordered manner. The general way is to provide each transaction a timestamp which specify when the transaction began. If conflicts are occurred then they resolved by timestamp as each operation is performed. The timestamp can be generated by assigning sequential number to the transaction or by the system clock value which is equal to the value of the clock when the transaction, T, entered into the system. If the timestamp is assigned sequentially then after each transaction the timestamp is increased by counter so that a new value is assigned to the next transaction.

In this algorithm, besides the time stamp direction of a transaction is introduced. It will use both time stamp and direction to determine which transaction should wait and which transaction should be restarted when conflict exists between transactions. The direction of a transaction T, denoted as D(T), has three values: neutral, forward, and backward. We have the following direction determination rules for our system:

Rule 1: The initial direction of a transaction is 'neutral'.

Rule 2: When the T_i request for T_j, if $TS(T_j)<TS(T_i)$ and T_i can wait for T_j, then $D(T_j)=D(T_i)=$ 'backward'. This is called as backward waiting.

Rule 3: When the T_i request for T_j, if $TS(T_j)>TS(T_i)$ and T_i can wait for T_j, then $D(T_j)=D(T_i)=$ 'forward'. This is called as forward waiting.

Rule 4: When the T_i request for the T_j, but T_i is not allowed to wait for T_j then one of the transactions is rolled back and restarted. The timestamp of restarted transaction does not change but its direction is changed to 'neutral'.

So this algorithm will reduces the number of restarted transaction and also saves the time and cost for the transaction. The can-wait and cannot-wait conditions in Rules 2-4 are discussed. These rules tell us that a transaction entering the system will be assigned a time stamp and a direction. The time stamp remains unchanged but the direction will change, which enables us to incorporate the wait-die and wound-wait protocols into one concurrency control protocol. The standard wait-die and wound-wait protocols are easy to be implemented, so here we are going to maintain the basic characteristic of the standard wait die and wound wait in our protocol that is when wait is not allowed we always let the older rollback the younger [12]. Here there are two cases. One is that the requester is older than the holder and waiting is not allowed, the holder will be wounded and finally it may be rolled back. The other is that the requester is younger than the holder and the waiting is not allowed, then the younger dies and will be rolled back. The condition can wait and cannot wait are derived from the decision table. Table I is built by time stamp order and by direction order, and is the core of our new algorithm. This decision table has integrated the standard wait-die and wound-wait protocols so that our improved algorithm has the properties of these protocols. The basic idea depends on the waiting approach and the restart approach as follows:

The waiting approach for the new algorithm is:

*forward waiting (Fig. 1) If both T_i and T_j have the same direction 'forward' or 'neutral' or one of them is 'neutral' and the other is 'forward' under the condition $TS(T_i) < TS(T_j)$,Then we allow T_i to wait T_j (i.e., T_i can wait T_j).

*backward waiting (Fig. 2) If both T_i and T_j have the same orientation 'backward' or 'neutral' or one of them is 'neutral' and the other is 'backward' under the condition $TS(T_i)> TS(T_j)$,Then we allow T_i to wait T_j (i.e., T_i can wait T_j).The restart approach for the new algorithm is when wait is not allowed (the preceding conditions are not satisfied or simply say T_i cannot wait T_j), the younger is restarted. Algorithm will follow the following steps:

STEP 1. If a transaction T_i requests a lock that is now available, T_i gets the lock and need not look up the decision table.

STEP 2. If a transaction T_i requests a lock on a resource (data object) that is already locked by another transaction T_j (that is the two transactions are in conflict), look up the decision table and choose a decision for T_i or for T_j or for both.

Assuming that initially $D(T_1)$ is 'neutral'. At that time if T_2 requests R1 and $TS(T_1) < TS(T_2)$, we then have the following cases:

CASE 1. $D(T_2)$='neutral'

T_2 will wait until T_1 commits or aborts; $D(T_1)$ changes to 'backward'. $D(T_2)$ changes to 'backward';

CASE 2. $D(T_2)$ ='backward'

T_2 waits until T_1 commits or aborts; $D(T_1)$ changes to 'backward';

CASE 3. $D(T_2)$ = ' forward '

T_2 will be rolled back and $D(T_2)$ is changed to 'neutral'.

TABLE I

DECISIONTABLE

If $TS(T_j)<TS(T_i)$		
$D(T_j)$	$D(T_i)$	Decision
backward	backward	T_i waits
forward	forward	T_i is wound
forward	backward	T_i is wound
backward	forward	T_i is wound
neutral	neutral	T_i waits;$D(T_j)=D(T_i)$=backward
forward	neutral	T_i is wound
backward	neutral	T_i waits;$D(T_i)$=backward
neutral	forward	T_i is wound
neutral	backward	T_i waits;$OT(T_j)$=backward

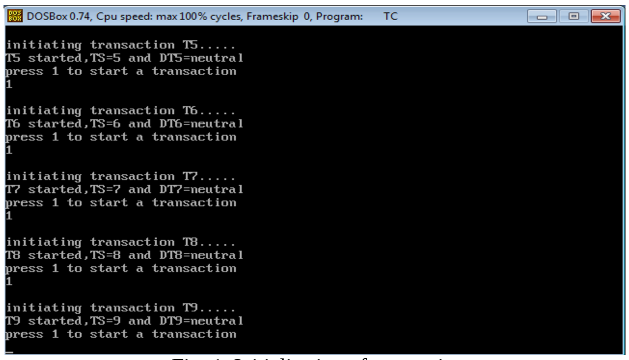
TABLE II

DECISIONTABLE

If $TS(T_j)>TS(T_i)$		
$D(T_j)$	$D(T_i)$	Decision
backward	backward	T_j is wound
forward	forward	T_i waits
forward	backward	T_j is wound
backward	forward	T_j is wound
neutral	neutral	T_i waits;$D(T_j)=D(T_i)$=forward
forward	neutral	T_i waits;$D(T_i)$=forward
backward	neutral	T_j is wound
neutral	forward	T_i waits;$D(T_j)$=forward
neutral	backward	T_j is wound

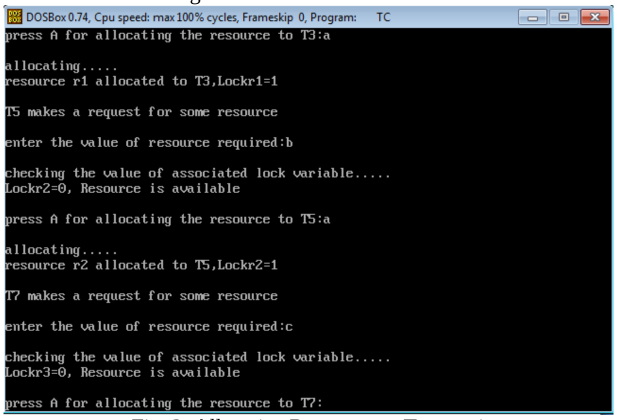
VI. RESULTS

When the above algorithm is implemented in the 'C' language. This language was developed by Dennis Ricthie in 1972 at Bell's Laboratories. The results are shown in the following figures:



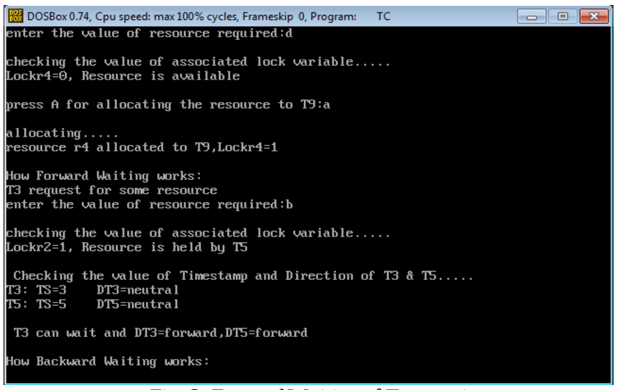
```
initiating transaction T5.....
T5 started,TS=5 and DT5=neutral
press 1 to start a transaction
1

initiating transaction T6.....
T6 started,TS=6 and DT6=neutral
press 1 to start a transaction
1

initiating transaction T7.....
T7 started,TS=7 and DT7=neutral
press 1 to start a transaction
1

initiating transaction T8.....
T8 started,TS=8 and DT8=neutral
press 1 to start a transaction
1

initiating transaction T9.....
T9 started,TS=9 and DT9=neutral
press 1 to start a transaction
_
```

Fig. 1: Initialization of transactions



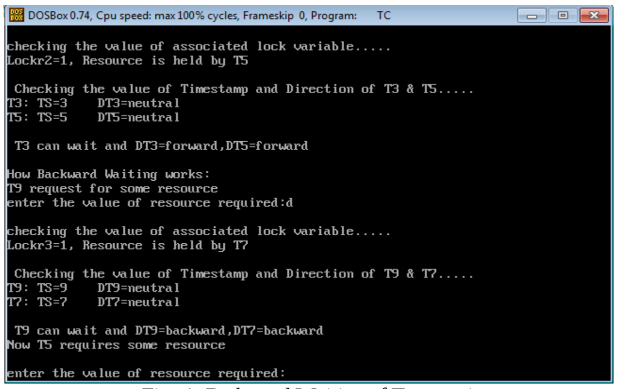
```
press A for allocating the resource to T3:a

allocating.....
resource r1 allocated to T3,Lockr1=1

T5 makes a request for some resource

enter the value of resource required:b

checking the value of associated lock variable.....
Lockr2=0, Resource is available

press A for allocating the resource to T5:a

allocating.....
resource r2 allocated to T5,Lockr2=1

T7 makes a request for some resource

enter the value of resource required:c

checking the value of associated lock variable.....
Lockr3=0, Resource is available

press A for allocating the resource to T7:
```

Fig. 2: Allocating Resources to Transactions



```
enter the value of resource required:d

checking the value of associated lock variable.....
Lockr4=0, Resource is available

press A for allocating the resource to T9:a

allocating.....
resource r4 allocated to T9,Lockr4=1

How Forward Waiting works:
T3 request for some resource
enter the value of resource required:b

checking the value of associated lock variable.....
Lockr2=1, Resource is held by T5

 Checking the value of Timestamp and Direction of T3 & T5.....
T3: TS=3    DT3=neutral
T5: TS=5    DT5=neutral

 T3 can wait and DT3=forward,DT5=forward

How Backward Waiting works:
```

Fig. 3: Forward Waiting of Transactions



```
checking the value of associated lock variable.....
Lockr2=1, Resource is held by T5

 Checking the value of Timestamp and Direction of T3 & T5.....
T3: TS=3    DT3=neutral
T5: TS=5    DT5=neutral

 T3 can wait and DT3=forward,DT5=forward

How Backward Waiting works:
T9 request for some resource
enter the value of resource required:d

checking the value of associated lock variable.....
Lockr3=1, Resource is held by T7

 Checking the value of Timestamp and Direction of T9 & T7.....
T9: TS=9    DT9=neutral
T7: TS=7    DT7=neutral

 T9 can wait and DT9=backward,DT7=backward
Now T5 requires some resource

enter the value of resource required:
```

Fig. 4: Backward Waiting of Transactions

VII. CONCLUSION

The two standard methods i.e wait-die and wound-wait only provides the forward or backward direction for the transaction. In this new algorithm, we have both type of direction for the transaction i.e backward as well as forward. When the older transaction is requesting the younger transaction and the older transaction has been waiting for the younger transaction then it is said to be forward waiting and when the younger transaction is requesting the older transaction and the younger transaction has to wait for older transaction to complete then this will said to be backward waiting. There are number of transaction are working in the system then this algorithm will provide some transaction with forward waiting and some with backward waiting. This algorithm will also diminish the number of restarts transaction and the system will attain the high throughput. This also saves the time and the cost for performing the transactions in the system as there is very less amount of transaction are restarted. This is algorithm is a deadlock free. Deadlock is a condition where two or more transactions are waiting for one or more of the others to complete but none of them can progress further. It will improve the efficiency of the distributed system. There should also the research work for future, after finding a transaction conflicting with another transaction how much time should have to wait to restart the aborted transaction.

REFERENCES

[1] Bharat Bhargava, “Concurrency Control in Database Systems”,IEEE transactions on knowledge and data engineering, vol. 11, no. 1 january/February 1999.

[2] S.C.Shyu, V .O.K.Li, and C.P. Weng, “Performance analysis of static locking in distributed database system”IEEE transactions on Computers, vo1.39, no.6, pp.741-751, June 1990.

[3] A. Bernstein and Nathan Goodman, “Concurrency Control in Distributed Database Systems”, *Computing Surveys, Vol. 13, No 2, June 1981.*

[4] P.A. Bernstern, D.W. Shipman, and J.B. Rothnie, Concurrency Control in a system for distributed databases, Database System 5(1): pp. 18-51, June 1980

[5] Yuetang Deng,Phyllis Frankl “Testing Database Transaction Concurrency” Proceedings of the 18th IEEE International Conference on Automated Software Engineering 2003.

[6] A. Buckmann, M.T. Ozsu, D.Georgakopoulos, "Towards a Transaction Management System for DOM", Internal Doc. GTE Lab. Inc. , June 1991.

[7] M.H. Eich, and D.L Wells, “Database concurrency control using data flow graphs,” AGM Transactions on Database Systems, vo1.13, no.2,pp.197-227, June 1988.

[8] P.I.Reddy, and S.Bhalla, “Deadlock prevention in a distributed database system,” ACM Vol. 22, NO. 3, pp. 10-46,September 1993.

[9] P.K. Reddy, and S. Bhalla, “A Non-Blocking Transaction Dataflow Graph Based Protocol for Replicated Databases,” IEEE T”actions on Knowledge and Data Engineering, to appear 1995.

[10] O.Bukhres, “Performance comparison of distributed deadlock detection algorithms,” Eighth International Conference on Data Engineering, pp.210-217, February 1992.

[11] A.N.Choudhary, Tost of distributed deadlock detection: A performance study,” Proceedings of the Sixth International Conference on Data Engineering,
pp.174-181, February 1990.

[12] M.H.Eich and S.H. Garad, “The performance of flow graph locking,” IEEE Transactions on Software Engineering , vo1.16, no.$ pp.477-483, April 1990.