ISSN:2249-5789
Monica Gahlyan et al , International Journal of Computer Science & Communication Networks,Vol 3(3),177-181

# Waiting Algorithm for Concurrency Control in Distributed Databases

Monica Gahlyan
M-Tech Student
Department of Computer Science & Engineering
Doon Valley Institute of Engineering & Technology
Karnal, India
E-mail: monicagahlyan25@gmail.com

Sandeep Jain
Asstt. Professor
Department of Computer Science & Engineering
Doon Valley Institute of Engineering & Technology
Karnal, India
E-mail: sandeepjain4891@gmail.com

***Abstract* — Database is the well-organized collection of data in a meaningful way and stored in such a way that it can be accessible to each and every user so that user can perform the transaction. 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. Transaction is a sequence of many actions considered to be atomic. The throughput of the transactions can be increased by performing the transactions in parallel. Transactions are performed in parallel without infringing the data integrity. When the transactions are executed in parallel then this will lead to concurrency in database and deadlock of the transaction. This work is done to prevent the occurrence of deadlock and also improves system performance by diminishing the number of restarts transaction which also saves the time and cost for performing the transaction in the system. So, it improves the efficiency of the distributed system.**

**Keywords- concurrency; transaction; timestamp; lock; serialization; consistency ;deadlock ;direction.**

## I. INTRODUCTION

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 is the collection of information, which is stored in such a way that it can be accessible to each and every user for their purposes. Database consists of a group of organized data and a set of programs to access that data. Database should contain the update information so that the user can access up-to-date data. 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). The main purpose of database system is to retrieve information; perform some work on the information and storing complete information back to the database. When the data or information is stored at different location or system then this is the concept of the 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 is the concept of Distributed Database Management System (DDBMS). In Databases, we work on the transactions. A Transaction can be observed as a program; whose implementation will maintains the consistency of the database. Transaction is a sequence of many actions that are 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 transactions then they are executed 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. This interfacing will result in lost-update problem, dirty read problem, inconsistent retrieval problem. To remove this type of problems, we have to use the concurrency control mechanism. Concurrency Control is the process of running concurrent execution of transactions in a shared database, to ensure the serializability of transactions. It is the activity of coordinating parallel accesses to a database in a multiuser database management system. 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. The major technical complexity in attaining this objective is to avoid database updates performed by one user frFom interfering with database retrievals and updates performed by another. The concurrency control problem is exacerbated in a distributed DBMS (DDBMS) because (1) users can access data stored in several different computers in a distributed system, and (2) a concurrency control

device at one computer cannot instantaneously know about interactions at other computers [1].
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 life- cycle (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. REQUIREMENT FOR CONCURRENCY CONTROL

When the transactions are performed in order without any time overlap in the transaction then there will not be any requirement of the concurrency control. But when the transactions are performed in parallel mode then there will be the requirement for the concurrency control and if the transactions are performed in parallel mode without the concurrency control the their will be some problems like:

1. 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.

2. 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.

3. 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. POLICY FOR ATRANSACTION IN DATABASE

The theory of a database transaction has evolved in order to enable both a well implicit database system performance in a out of order situation where crashes can occur any time, and recovery from a crash to a well understood database state. A Transaction in database has four policies that leads to constant and trustworthiness of the database management system. The Policies are:

1. Atomicity: It refers that each transaction is a single logical unit in database consists of number of operations. Either all the operations of the transaction are completed successfully or not carried out completely. This policy is also known as the all or nothing policy of the transaction. In other ways, the completed transactions have appeared by its effect in database and an aborted transaction does not have any effect on the database.

2. Consistency: It refers to the accuracy of the data. This policy state that the transaction must change from one consistent state to another consistent state. This policy will leave the database in the steady state. It provides flexibility to define transaction that can operate.

3. Isolation: Isolation means that there is no interference of the transaction with each other. It is the main goal of the concurrency control in database. This means that if one transaction is performing any read or modifying any data then other transaction cannot do any operations on that data.

4. Durability: It means that effects of all the transaction completed or committed will be store in a non-volatile memory permanently, so that it can be access by any transaction even if there will be any problem of power failure or system crashes.

So these are the ACID policies which are required by the transactions in the database.

## IV. FICTION ANALYSIS OF CONCURRENCY CONTROL METHODS

Many methods for concurrency control be present [2][4][5][6][8][9].The major methods, which have each many variants, are:

1. Locking: Locking is the common type of concurrency control mechanism. It provides the access control to data by assigning the locks to transactions. It is mechanism commonly used to solve the problem of coordinate access to shared data [4]. There are many types of locks are used in concurrency control such as Shared or Exclusive locks , Binary locks (0 or 1), each transaction has a lock related with it. When one transaction T, is going to perform some work on the data then they have to first examine that the related lock. If there is no transaction that holds the lock then the scheduler obtains the lock on the behalf of transaction T. If another Transaction T1, holds the lock then the Transaction T has to wait until the transaction T1 gives up the lock. The scheduler will not give to transaction T until it will be releases by Transaction T1. When the lock is associated with one transaction then it is not assigned to another transaction, so that the isolation policy remains valid. When there are more number of transactions are performing concurrently, then there will be the greater probability that transaction will be blocked, which leads to more response time and very small throughput of the transaction. The locking protocol ensure serializability in two phase locking (2PL) protocol [7]. This protocol requires that each and every transaction issues request in two phases:

   1. Growing Phase: In this phase, a transaction can obtain locks but cannot release these locks.
   2. Shrinking Phase: In this phase, a transaction can release locks but cannot obtain any new locks.

2. Timestamp Ordering: 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.

3. Serialization: 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. A precedence graph is that in which transaction is shown by vertices and arcs show the resources required by the transaction. There will be arc between $T_i$ and $T_j$ if and only if $T_i$ is waiting for the resource which is locked by the $T_j$. If there is a cycle in the precedence graph or the wait for graph then deadlock has been occurred. So, this deadlock can be busted by aborting that particular transaction.

## V. ANALYSIS OF TWO-WAY WAITING ALGORITHM IN DATABASES

The concurrency control method is based on the concept of the timestamp ordering [6][7]. The Transaction timestamp is assigned to each transaction on the basis of the system clock or any sequential order of transaction occurring in the system. The timestamp is allocated to each and

every transaction in a monotonically increasing order. The transaction Ti comes early will have lower timestamp and the transaction Tj comes later will have the higher timestamp i.e TS (Ti) < TS ($T_j$). When the two or more transactions are waiting for the resources that are hold by them and shown in the precedence graph. If the graph is having the cycle then there will be deadlock. The Deadlock can be prevented by the two methods. These two methods will follow the timestamp ordering. These are:

1. Wait-die: If the older transaction ($T_i$) is requesting for the resource which is hold by the younger transaction ($T_j$) i.e TS($T_i$) < TS($T_j$). Then transaction with lesser timestamp has to wait but if the younger transaction is requesting for the resource then younger transaction will be rolled back and restarted the transaction. This rolled back transaction is known as Victim.

2. Wound-wait: If the younger transaction ($T_i$) is requesting for the resource which is hold by the older transaction ($T_j$) i.e TS ($T_i$) > TS ($T_j$). Then transaction with larger timestamp has to wait but if the older transaction is requesting for the resource then older transaction will give some time to younger to complete its work in a definite amount of time and younger transaction is restarted if it is not complete in definite time.

These two above methods was having the one way waiting for the transactions. The transaction (T) will have the direction which is denoted by D(T) are having the values like: „neutral", when the transaction is started, „forward", when the transaction is in wait-die method and „backward", when the transaction is in wound-wait method. The Rules for the direction of the system transaction are:

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.

## VI. CONCLUSION

The two approaches 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. 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. A. Bernstein and Nathan Goodman, "Concurrency Control in Distributed Database Systems", *Computing Surveys, Vol. 13, No 2, June 1981.*

2. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistence and predicate locks in a database system,"Commun. ACM, vol. 19, no. 11, pp. 624-633, Nov. 1976.

3. Victor 0. K. LI, "Performance Models of Timestamp-Ordering Concurrency Control Algorithms in Distributed Databases" IEEE Trans. on Computers,Vol. C-36, No. 9, September 1987.

4. J.F.Pons and J.F. Vilarema, "A Dynamic and Integrated Concurrency Control for Distributed Databases"IEEE Journal on Selected Areas in Comm. Vol. 7, No. 3, April 1989.

5. Ugur Halici & Asuman Dogac," An Optimistic Locking Technique for Concurrency Control in Distributed Databases", IEEE Trans.on Software Engineering, vol. 17, no. 7. July 1991.

6. F. Bukhari and Sylvia L. Osborn, "Two Fully Distributed Concurrency Control Algorithms" IEEE Trans. on Knowledge and Data Engineering, vol. 5, no. 5, October 1993.

7. Philip .Alexander Thomasian,"Concurrency Control: Methods, Performance, and Analysis" ACM Computing Surveys (CSUR) Survey Volume 30 Issue 1, March 1998.

8. Transaction Processing" IEEE Trans. on Knowledge and Data Engineering, Vol. 10, No. 1, January/February 1998.

9. Alexander Thomasian,"Distributed Optimistic Concurrency Control Methods for High-Performance.
10. Theo Harder, Kurt Rothermel, "Concurrency Control Issues in Nested Transactions," VLDB journal Vol. 2(1) page(s) 39-74, July 1993.
11. O.Bukhres, "Performance comparison of distributed deadlock detection algorithms," Eighth International Conference on Data Engineering,pp.210-217, February 1992.
12. W. Perrizo, Request order linked list (ROLL): A concurrency control object, in: Proceeding of the IEEE International Conference on Data Engineering, Kobe, Japan,Apr. 1991.
13. Alexander Thomasian, In Kyung Ryu. Performance Analysis of Two-Phase Locking; IEEE Transactions on Software Engineering, Vol. 17, no. 5, May 1991.