Multiversion Concurrency Control With The Precedence Graph Generation Algorithm

Prateek Vikram, Salman Abdul Moiz, G R Anil

Abstract— Simultaneous access of a shared record by multiple transactions leads to conflicts while writing. Such scenario generates the problems like lost update, dirty read, non-repeatable read etc . In such a case, transaction need to be rolled back to get the system into a consistent state. To handle these conflicts Multiversion Concurrency control (MVCC) is used. MVCC has the ability to avoid the read-write conflicts by performing the read operations using the older version when the write request is in progress. When the multiple write operation concurrently executes then the transaction is aborted or rolled back. To address the problem of rollbacks, a methodology using the Precedence Graph generation based algorithm to reschedule the sequence of the transaction is proposed. In the proposed methodology whenever there is a system failure or network failure the transaction need not be started again, rather it is partially rolled back. The proposed methodology is executed and compared with the other MVCC approached on same set of transactions defined on shared and non shared data items. It is observed that the proposed methodology achieves better execution time as compared to the methods available in the literature..

Index Terms—Concurrency control, Database, Multiversion timestamp protocol, Precedence graph generation, Partial rollback

I. INTRODUCTION

Concurrency control is extensively used in the Database management system (DBMS) in order to achieve better throughput and resource utilization of the transaction. A transaction must possess the properties like Atomicity, Consistency, Durability, Isolation(ACID). These properties are violated while performing the concurrent transactions which leads to the problems like Lost update, read Dirty data, Non Repetition. When multiple transactions perform their operation on the same dataitem, isolation is not preserved.. There are techniques like Concurrency control mechanism to retain the isolation property.

The concurrency control mechanisms are divided into two categories Pessimistic and Optimistic. The pessimistic method has the time stamped ordering. The optimistic method in concurrency control uses a timer based method as well as the locking methods.

Multiversion concurrency control(MVCC) maintains a single logical address and multiple physical version in the database. The procedure is explained below

Revised Manuscript Received on July 18, 2019.

Prateek Vikram, School of Computer and Information Sciences, University of Hyderabad, Hyderabad,Telangana, India.(email: prateek.uohyd@gmail.com)

Salman Abdul Moiz, School of Computer and Information Sciences, University of Hyderabad, Hyderabad,Telangana, India.(email: salman@uohyd.ac.in)

G R Anil, School of Computer and Information Sciences, University of Hyderabad, Hyderabad, Telangana, India.(email: anilgrcse@gmail.com)

(1) Keep the older version of data item during the update.

(2) Maintain the serializability while reading the older version of the data item.

Multi Version Concurrency Control exists in multiple versions of Timestamp ordering and 2 phase locking protocol. Basic Multiversion Timestamp ordering protocol uses read and write time stamps.

- Read timestamp(TS): The read timestamp for version xi is the highest timestamp amongst all the transaction that read the version successfully.
- Write timestamp(TS): Once a transaction T performs the write operation on a data item(x) it will create the new version xk+1.

Correspondingly, when a Read operation is performed by Transaction T it read the version xi is the highest timestamp amongst all the transaction that the read version successfully.

The following two rules are used to ensure the serializability

1. If transaction T issues a write item(X) operation and version i of X has the highest writeTS(Xi) of all version of X which is also less than or equal to TS(T) and readTS>TS(T), then abort and roll back transaction Ti otherwise create a new version Xi of X with read TS(Xi)=writeTS(Xi)=TS(T)
2. If transaction T issues read(X) operation, find the version i of X that has the highest writeTS(Xi) of all version of X which is also less than or equal to TS(T) then return the value of Xi to transaction T and set the value of read(Xi) to the larger of TS(T) and the current read TS(Xi).

One key challenge in MVCC is when two concurrent write operations in a transaction occurs, only the first write operation can be rolled back. Then after transaction rollback, system only increases the starting time of the transaction to achieve the purpose of concurrent.

The remaining part of this paper is organized as follows: Section II summarizes the survey of existing techniques, Section III specifies the proposed concurrency control strategy, Section IV describes the Experimentation and Results and Section 5 concludes the paper.

II. RELATED WORK

Significant research is done in the area of multi version locking [1,7,8,9,12]. Wang et al[1] algorithm is very well known work in this area. Wang[1] proposed the idea to decrease the overload of a system by turning the rollback problem into the waiting state.

To realize this concept, a read write table is created along with the timestamp of each read write operation of the transactions. If there exists any WRITE WRITE operation than older transaction has to commit first and the younger transaction has to wait until the older transaction is committed.

Yingjun Wu et al [2] discusses the key management decisions of transaction in MVCC DBMS. Version storage scheme is the key criteria of this method. Whenever a dataitem is updated, its new version is created. With this schema tracing the updates is convenient. There are 3 approaches to store the version in the database.

1) Append only storage: Tuple versions are stored in the same storage space. Now for updating the tuple DBMS acquires empty free slot to store the new version in the table. The key idea of this approach is how the DBMS order the tuple. Maintaining the doubly linked list is not possible in the Append only storage. So version chain is pointing only in one direction. There are two scenarios on which append only works:

- Oldest to Newest(O2N): In this version, the HEAD is pointed to the oldest version in the chain. The advantage of this O2N version is no need to update the index for pointing out the newer version of the tuple when it gets modified. But during the query processing DBMS traverse a lot to find the latest version.
- Newest to Oldest(N2O): In N2O the DBMS does not have to traverse for the latest version of a tuple However the chain HEAD changes on every modification of the tuple. DBMS modified all the tables indexes both primary as well as secondary for pointing the new version.

2) Time Travel storage: The storage schema is very similar to the append only storage the only difference is that the older version is stored in the separate table.

It maintains the master version of every tuple in the main table and also maintains multiple versions of the same tuple in the different time travel table.

In SQL Server the current version of the tuple is called the master version and system like SAP HANA store older version of the tuple is considered as the master version.

While updating a tuple DBMS needs a slot in the time travel table and then the master version is copied to this location and it will not affect the indexes because they are pointing to master version.

3) Delta Storage: Delta version Sequence is stored in separate delta storage. While updating the tuples the DBMS needs a separate continuous space from the delta storage to create the new delta version. This approach reduces the memory allocation by doing update operation that modified the subset tuple.

But overhead should be higher for reading intensive workloads while using this approach Garbage collection While Updating the tuple new version is created so older version is no longer needed for the system The execution time of the query also increased because DBMS spend a lot of time to traverse the long version chain. Due to which MVCC performance is highly dependent on Garbage collection. This Garbage collection is divided into three steps.

(1) Detect the version which is expired

(2) Storage space is reclaimed

(3) Unlink the version that has indexes and associated chain

Tuple level Garbage Collection: The DBMS check visibility of every tuple version in one of two ways: Background Vacuuming (VAC): DBMS periodically check the expired version using background threads and this is one of the most common approaches in MVCC. DBMS is easier to work and implement with all type of version storage schemes. But this mechanism does not work for large scale databases, especially it works for the small number of threads.

Cooperative Cleaning (COOP): When executing a transaction, the DBMS navigates the adaptation chain to find the obvious version. During the traverse, it finds out the expired versions and records them in the data structure. This approach works only for O2N append only storage. But there is a challenge in this approach when the transaction does not traverse the version chain for a particular tuple, then the system is not able to remove the expired version.

Transaction level Garbage collection: The DBMS retrieves storage space at transaction level granularity in this GC system. It is consistent with all storage systems for versions. The DBMS believes a transaction to have expired when no active transaction shows the versions it produced. After an epoch finishes, all the versions produced by the transactions of that epoch can be securely produced.

MVCC[5][6][7] keep separate indexes and Database version information. An index entry is defined as key/value pair, where the key is a tuple index and value is the pointer to that tuple. For tuple version chain DBMS follow the pointer and then Scans to locate the visible version for the transaction. Kaloian Manassiev et.al [3] proposed a novel distributed concurrency control algorithm. For each update in a transaction which creates a new version that is being broadcast per page version to all sites where replicas exist. The two approach namely master update with scheduler support and update anywhere with no scheduler support has been discussed.

Several algorithms proposed in the literature addressing concurrency in distributed services [8][10], with the multi version range control[11] and time stamp range conflict management[9][12].

Salman Abdul Moiz et al[14] proposed the analytical strategy as a variation in time based commit protocols in which a transaction takes place only when the anticipated execution time is within the present time value. The choice to rollback or abort is produced when the timer expires in the timeout-based processes. The decision on the transaction status is know at the start in the suggested approach. This can be done if the expected execution time is known.

Concurrency control in mobile environments are addressed by Moiz et al[13][14][17]. Single lock manager approach is used for concurrency control in mobile environments [15]. There exists a method for concurrency control without locking [16].

Mohammad Sadoghi et al [4] proposed the KV direction method which reduces the burden of input/output updates. This should be done by maintaining historical data and current data stored as a version enabled table. It also contains multiple physical and logical representations of records. In order to differentiate between the logical record and physical records identifies. It allows the reader to read concurrently every commit record without interfering with writers.

The scheme proposed by Wang et al[1] suffers from following drawbacks:

The scheme is used when conflict occurs from multiple write operation on the concurrent transaction. The younger transaction waits if the older wishes to perform write on the same data item at some later point of time. We will have an enlarged waiting queue and it will, later on, become problematic to decide the waiting order of the transaction.

- It can be possible that transaction can be rollback due to network or power failure in this case we have to rollback the complete transaction.

The drawbacks of these methods derived the motivation to solve this concurrency problem with new approach.

III. PRECEDENCE GRAPH BASED ALGORITHM

In Graph based algorithm, every write-write conflict is realized by generating a graph. Here, every transaction is considered as a node. For example, If there is a write write conflict among two transactions A,B (First A is to be executed followed by execution of B), then A->B is the graph generated. Every node in the graph would be having in-degree(number of edges incoming), out-degree(number of edges outgoing).

The Graph based algorithm is processed as follows.

1. Create nodes and draw directed edges if the write write conflicts occur.
2. Calculate the in-degree of each node in the graph.
3. Sort the nodes of a graph in the increasing order on the basis of their in-degree.
4. The node with a small in-degree must be executed first and the node that had large in-degree must be executed last.
5. If the transaction fails then do particle rollback.

Figure 1 explains the procedure of Graph based algorithm such that A, B, C, D are the transaction that having both read and write operation. If there exists write write conflicts among transactions A with {B, C, D}, B with {C, D}, C with D then the Graph based algorithm will be as in Figure 1. Now the in-degree and out degree of the nodes must be computed, which is shown as label on the node. Calculate the in-degree of each node of the transaction. The node A having lesser in-degree 0 and node D having higher in-degree 3. According to the graph based algorithm the transaction has less in-degree proceed first and the transaction has large in-degree proceed next.

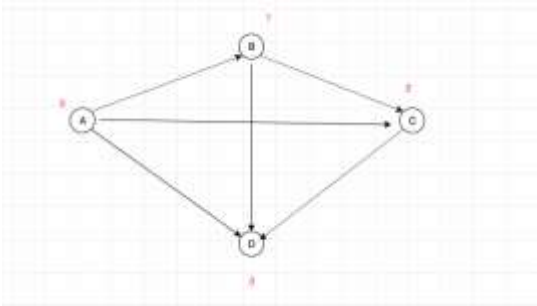

Figure 1 Procedure of Graph based algorithm

```
Input: T : Set of transactions
Output: EP : Transactions lis
1.  begin
2.    G ← Graph Generation(T);
3.    SV ← Sort Vertex(G) ; // Sort V ∈ G
        based on in-degree in increasing order
4.    EP ← SV ;
5.    return EP;
6.  end
7.  Graph Generation(T);
8.  {
9.    G(V, E) ← φ, Flag ← 0;
10.   for all ti ∈ T do
11.     if Ti ∈/ V then ; // If the node is not
          created before
12.       Add Vertex G, Ti;
13.     endif
14.   for all tj ∈ {T} − Ti do
15.     Flag ← Check WW Conflict(Ti, Tj ); //
        Check the Write-Write Conflict between Ti
        and Tj
16.     if Flag == 1 then
17.       Add Vertex Tj;
18.       Add Directed Edge Ti, Tj; // Add a directed
          edge between Ti and Tj
19.     end if
20.   end for all
21.  end for all
22.  return G;
23. }
```

Algorithm1: Graph based algorithm

Algorithm 1 the GraphGeneration(T) function create a graph if there is a write write conflict. Check_WW_Conflict(Ti,Tj) function checks whether a write write conflict occurs between Transaction Ti,,Tj. Sort_Vertex(G) sort the node of the basis of their in-degree.

The proposed method differs from Wang et al[1] algorithm in the following aspect.

- The write write conflict occurs then according to the proposed algorithm by Wang et al [1] younger transaction has to wait until the older transaction is commit the graph based algorithm , it is better to reschedule the transaction rather than waiting for the transaction.
- If there is any failure of network or power problem the transaction is completely roll back according Wang et al [1]. Thus it can make partial roll back which can prevent the complete roll back situation.

IV. EXPERIMENTATION AND RESULTS

Dataset Description:

A database containing 100 transaction are created that have both shared and non shared data items. Four different experiments are performed on this transaction data set.

Experiment 1:

25 transactions which defines on the same shared data are considered in this experiment, these transaction contain both read and write Operation.

Initially 5 transaction are consider in Experiment 1 and executed with Wang Yujun LI [1] Junke and the graph based algorithm. The execution times of both methods are recorded. The Experiment is continued by, adding 5 more transaction to the initial set and experiment is continued, results are noted. This process is continued with the experimentation with 15, 20, 25 transactions. The results are depicted as in figure 2.

The observed results clearly shows that Graph based algorithm performed better than the Wang et al [1] algorithm.

Figure 2: Experiment 1

In first 5 transactions the execution time of graph based algorithm is 1.75 sec and that of Wang et al [1] is 1.86 sec, which is improved by 5.91%. 10 transactions the execution

time of graph based algorithm is 2.09 sec and that of Wang et al [1] is 2.65 sec, which is improved by 21.13%. 15 transaction the execution time of graph based algorithm is

3.56 sec and that of Wang et al [1] is 3.98 sec, which is improved by 10.55%. 20 transactions the execution time of graph based algorithm is 4.47 sec and that of Wang et al [1] is

4.98 sec, which is improved by 10.55% . 25 transactions the execution time of graph based algorithm is 6.42 sec and that of Wang et al [1] is 6.78 sec, which is improved by 5.30%.

Experiment 2:

25 transactions that are defined on shared data as well as the data that is accessing other item. In the Experiment 2 initially 5 transaction are executed with Wang Yujan LI junke [1] and the graph based algorithm. The execution time of both the methods are recorded and the process is continued with the experimentation with 10,15,20,25 number of transactions.

The results are depicted as in figure 3. The results show Graph based algorithm Execution time is less than the Wang et al [1] algorithm.

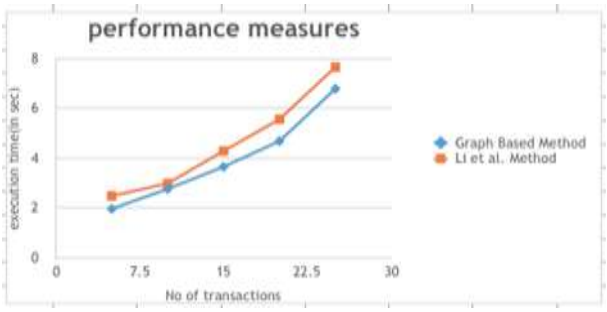

Figure 3: Experiment 2

In 5 transactions the execution time of graph based algorithm is 1.98 sec and that of Wang et al [1] is 2.11 sec, which is improved by 6.61%. 10 transactions the execution time of graph based algorithm is 2.41 sec and that of Wang et al [1] is 2.59 sec, which is improved by 6.94%. 15 transactions the execution time of graph based algorithm is

2.98 sec and that of Wang et al [1] is 4.16 sec, which is improved by 28.36%. 20 transactions the execution time of graph based algorithm is 4.48 sec and that of Wang et al [1] is

5.51 sec, which is improved by 13.24%. 25 transactions the execution time of graph based algorithm is 5.98 sec and that of Wang et al [1] is 6.80 sec, which is improved by 12.05%.

Experiment 3:

25 transaction having both read and write operation where 60% of transaction are writes and 40% are reads on same shared item. In the Experiment 3 initially 5 transactions are grant for the execution with Wang Yujan LI junke [1] and the graph based algorithm. The execution times of both the algorithm are recorded and the process is continued with the experimentation with 10,15,20,25 transactions.

The results are depicted as in figure 4. The results show Graph based algorithm Execution time is less than the Wang et al[1] algorithm.

Figure 4: Experiment 3

In 5 transactions the execution time of graph based algorithm is 1.76 sec and that of Wang et al [1] is 2.08 sec, which is improved by 6.16%. 10 transactions the execution time of graph based algorithm is 2.76 sec and that of Wang et al [1] is

2.89 sec, which is improved by 6.94%. 15 transactions the execution time of graph based algorithm is 3.89 sec and that of Wang et al [1] is 4.42 sec, which is improved by 28.36%. 20 transactions the execution time of graph based algorithm is

4.56 sec and that of Wang et al [1] is 5.65 sec, which is improved by 13.24%. 25 transactions the execution time of graph based algorithm is 5.78 sec and that of Wang et al [1] is

6.56 sec, which is improved by 12.05%.

Experiment 4:

25 transactions that contain only write operation where 100% transactions are writes on shared items. In the Experiment 4 initially 5 transactions are grant for the execution with Wang Yujan LI junke [1] and the graph based algorithm. The execution times of both the algorithm are recorded and the process is continued with the experimentation with 10,15,20,25 transactions.

The results are depicted as in figure 5. The results show Graph based algorithm Execution time is less than the Wang et al[1] algorithm.

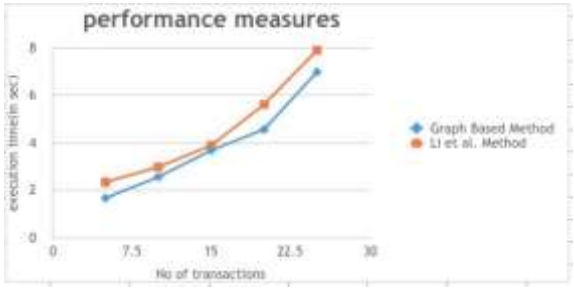

Figure 5: Experiment 4

In 5 transactions the execution time of graph based algorithm is 1.67 sec and that of Wang et al [1] is 2.12 sec, which is improved by 15.38%. 10 transactions the execution time of graph based algorithm is 2.56 sec and that of Wang et al [1] is 2.65 sec, which is improved by 4.49%. 15 transactions the execution time of graph based algorithm is 3.56 sec and that of Wang et al [1] is 4.21 sec, which is improved by 11.99%. 20 transactions the execution time of graph based algorithm is 4.45 sec and that of Wang et al [1] is 5.76 sec, which is improved by 19.29%. 25 transactions the execution time of graph based algorithm is 5.87 sec and that of Wang et al [1] is 6.45 sec, which is improved by 11.89%.

V CONCLUSION

The multi version concurrency control is addressed with the Graph based algorithm. The proposed algorithm is realized on the set of transactions defined on shared data item and non shared data item. Wang et al algorithm is also executed on the same set of transactions in the same environment. It has been observed that Graph based algorithm execution time is less if there are a large number of transactions compared to Wang et al [1] algorithm. It is observed that complete roll back is followed in the Wang et al algorithm, where as with the partial rollback algorithm, better execution time is achieved. This graph based algorithm can be extended further to achieve much enhanced execution time.

REFERENCES

1. Wang Yujun, LI Junke The Solution to the Roll Back Problem in Multi version Concurrency Control Timestamp Protocol International Conference on Computer Science and Network Technology,pp 2803-2806,IEEE 2011
2. Wu, Y., Arulraj, J., Lin, J., Xian, R. and Pavlo, A., 2017. An empirical evaluation of in memory multiversion concurrency control. Proceedings of the VLDB Endowment, 10(7), pp.781-792.
3. Kaloian, Manassiev, MadalinMihailescu, Cristiana Amza Exploiting Distributed Version Concurrency in a Transactional Memory Cluster pp 198- 208, New York, ACM 2006
4. MohammadSadoghi,MustafaCanim,BishwaranjanBhattacharjee,Fabian Nagel, Kenneth A. Ross Reducing Database Locking Contention Through Multiversion Concurrency pp 1331-1342, Proceedings of the VLDB Endowment, Vol. 7, No. 13, 40th International Conference on Very Large Data Bases, September 1st 5th 2014, Hangzhou, China.
5. Juchang Lee, Hyungyu Shin, Chang Gyoo Park Hybrid Garbage Collection for Multiversion Concurrency Control in SAP HANApp 1307-1318, SIGMOD, ACM. June 26-July 01, 2016, San Francisco, CA, USA.
6. Joao A. Silva, Joao M. Lourenco and Herve Paulino Boosting Locality in Multi version Partial Data Replication pp 1311-1314, SAC15, ACM, April 1317, 2015, Salamanca, Spain.
7. Eran Chinthaka Withana, Beth Plale, Roger Barga, Nelson Araujo Versioning for Workflow Evolution, pp 756-765, HPDC'10, ACM, June 2025, 2010, Chicago, Illinois, USA.
8. Nirmit Desai and Frank Mueller Scalable Distributed Concurrency Services for Hierarchical Locking Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS03) IEEE, 2003
9. David Lomet, Alan Fekete, Rui Wang, Peter Ward Multiversion Concurrency via Timestamp Range Conflict Management pp 714-725, IEEE 28th International Conference on Data Engineering,2012
10. Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, Danny Dig How Do Centralized and Distributed Version Control Systems Impact Software Changes?pp 322-333, ICSE 14, ACM, June,2014
11. JustinLevandoski,DavidLomet,SudiptaSengupta,RyanStutsman,and Rui Wang "Multiversion Range Concurrency Control in Deuteronomy"pp 2146-2157, Proceedings of the VLDB Endowment, Vol. 8, No. 13, 42nd International Conference on Very Large Data Bases, September 5th September 9th 2016, New Delhi, India.
12. Moiz, Salman Abdul, and Lakshmi Rajamani. "Concurrency control strategy to reduce frequent rollbacks in mobile environments." 2009 International Conference on Computational Science and Engineering. Vol. 2. IEEE, 2009.
13. Moiz, Salman Abdul, et al. "Concurrency control in mobile environments: Issues & challenges." International Journal of Database Management Systems 3.4 (2011): 147.
14. Moiz, Salman Abdul, and Lakshmi Rajamani. "An Analytical Approach for Guaranteeing Concurrency in Mobile Environments." World Congress on Engineering and Computer Science. Vol. 1. 2010.
15. Moiz, Salman Abdul, and Lakshmi Rajamani. "Single lock manager approach for achieving concurrency control in mobile environments." International Conference on High Performance Computing. Springer,

Berlin, Heidelberg, 2007.

16. Moiz, Salman Abdul, and Mohammed Khaja Nizamuddin. "Concurrency control without locking in mobile environments." 2008 First International Conference on Emerging Trends in Engineering and Technology. IEEE, 2008.
17. Moiz, Salman Abdul, and Dr Lakshmi Rajamani. "An algorithmic approach for achieving concurrency in mobile environment." 1st National Conference on Computing for Nation Development, INDIACom. 2007.