PERFORMANCE ANALYSIS OF MULTIVERSION ONCURRENCY CONTROL ALGORITHMS

Sonal Kanungo1, Rustom. D Morena2

1 Smt.Z.S.Patel College of Computer Application, Surat,
2 Department of Computer Science
Veer Narmad South Gujarat University, Surat

ABSTRACT:

Concurrency control protocols based on multiversion have been used to give the serialized execution of transactions. This paper portrays a simulation study of the execution of different multiversion concurrency control algorithms. Different multiversion techniques are analyzed through simulations. A prototype dynamic environment is presented here for the evaluation of database for specific workload for each algorithm. We are comparing algorithms on the basis of a number of Commits, Rollbacks and Deadlocks in the mix predefined Read-only and Update transactions.

Keywords: Concurrency control, Multiversion, Rollback, Deadlock

[1] INTRODUCTION

Concurrency control of current database can be enhanced with various versions. Multiversion algorithms using numerous versions of a data ensure that Read only transactions will never Rollback. This implies
Read only transactions can read consistent committed latest version of the data while new versions are simultaneously made by Update transaction [5].

To examine the behavior of the algorithms under a mix of transactions, various experiments were performed. Since updates do not overwrite each other, old values are stored and new versions are created. In Multiversion protocol old values are never overwritten and are therefore always available for delayed Reads. Multiple versions item helps the scheduler to avoid rejecting operations that arrive too late. A Read would normally have rejected because the value it was supposed to read has already been overwritten. By simply having the old version, these rejections of Read can be avoided. In such cases the delayed Read can be given an old value of a data item, even though it was “overwritten” [3].

Two types of multiversion concurrency control methods are presented here; Timestamp based and Lock based.

[1.1.] MULTIVERSION TECHNIQUE BASED ON TIMESTAMP ORDERING

The Timestamp Ordering keeps different versions of every data item. Every version keeps the value of version which accompanying two timestamps: Read Timestamp and Write Timestamp. Read timestamp is the biggest of all the timestamps of transactions that have effectively read the version and Write timestamp is the timestamp of the transaction that composed the value of version. Read operation reads the version with the biggest timestamp. Every Commit operation makes a new version of data items. The value field of this version holds the value committed by transaction [6].

The transaction can be successful only when it's timestamp is greater than last Read and Write timestamp of data, else this transaction will abort and Rollback. Rollback and abort additionally happen when the transaction is endeavored to commit a version data that have been Read by another transaction [13]. It will abort and restart with new timestamp and this Rollback will bring about cascading Rollbacks. Old data are never overwritten therefore Reads are constantly available and accessible. Updates don't overwrite each other and create new versions, hence Reads can read any version which offers adaptability to Multiversion concurrency control [3].

[1.2.] MULTIVERSION LOCKING

Multiversion Locking technique adds the benefits of Multiversion concurrency control with two-phase locking [3]. When a transaction writes on a data item, it creates a new version of data items. This Write won't overwrite the old version of the item, however it creates another version and keeps both versions. The transaction requests to access the data item which is updating and new version is not committed yet, still this transaction permits to Read the previous committed version of this data item [14]. In conventional 2PL, Read-Write clashes block each other. Write lock on a data item prohibits transactions from getting Read locks on the same data item. In the standard locking methodology, the Read lock is shared and Write lock is an exclusive lock. Once a transaction gets a Write lock on an item, no other transactions can get to that item [1]. In Multiversion Mixed Method algorithm Multiversion Timestamping is utilized to process Read-only transactions (queries) and Multiversion locking is used to process general transactions (updaters). Questioning (Reading) data doesn't conflict with locks granted for writing data, hence Read never blocks Write and Write never blocks Read. In MVCC, querying data don't conflict with locks acquired for writing data. Queries never postpone nor abort updaters, and updaters never abort queries [2].
Multiversion 2PL convention separates between Read-only transactions and Update transactions. The Update transactions follow rigorous two-phase locking where all locks are discharged only at the end of the transaction at the committed time [3]. A single timestamp is kept for every version of a data item. When an Update transaction reads or writes a data item it locks the item similarly as it would in two-phase locking and it reads or writes the latest version of the item [12].

Read-only transactions will read the latest value and transaction’s timestamp is given at the start of the transaction. Multiversion timestamp technique is used for Reads [2]. Read-only transactions never wait for locks. Multiversion two-phase locking additionally guarantees that transactions are recoverable and cascade less [14].

[2] LITERATURE SURVEY

1) CHRISTOS H. PAPADIMITRIOU, PARIS C. KANELLAKIS, NATHAN GOODMAN
(DESCRIBED VARIOUS CONCURRENCY CONTROL METHODS USING MULTIPLE VERSIONS) Improved Multiversion concurrency control is given effective critical and sufficient conditions for an execution. 1-SR concurrency control and expanded concurrency control theory. Authors gave a chart structure named The Multiversion serialization graphs (MVSGs) that checks these conditions and the associated speculation to three Multiversion concurrency control algorithms. One algorithm uses time stamps, one uses locking and one joins locking with timestamps.

2) RICHARD E. STEARNS AND DANIEL J. ROSENKRANTZ

Authors deal with the Write of the database. Write of a database entity is both the "before" or old value, and the "after" or new value. Two schemes for producing such controls are given, first scheme works in a system where processes are committed on termination, and the other for systems where commitment happened later. They introduce some basic terminology and model of a concurrent, distributed database system. This paper developed a series of design principles which applied to all designs. Two concurrency control schemas are introduced, one is for the case COMMIT=CLOSE and the other for COMMIT=TERMINATE.

3) D. AGRAWAL, V. KRISHNASWAMY

Authors states the transactions should be divided into three groups: Read-only, Read-Write, and Write-only transactions. Multiversion data can also be used to eliminate or minimize the interference between the Read-Write and Write-only transactions. A different handling of Write-only transactions is given. In this paper, a version control mechanism is given that minimizes the interference between the Read-Write and Write-only transactions.

4) PHILIP A. BERNSTEIN and NATHAN GOODMAN

(Multiversion Concurrency Control-Theory and Algorithms) Authors presented a theory for analyzing the correctness of concurrency control algorithms for Multiversion database. They showed some new Multiversion algorithms separated these new algorithms. This research gives overviews of concurrency control assumption for nonmultiversion databases and extends the theory to Multiversion databases.
5) MICHAEL J. CAREY and WALEED A. MUHANNA

(The Performance of Multiversion Concurrency Control Algorithms) Authors gave simulation study on the performance of several multiversion concurrency control algorithms. These algorithms compare with each other and their single version counterparts on basis of Storage cost, CPU, I/O overall all throughput and response time.

[3] METHODOLOGY

We consider a dynamic database with \( D \) objects. These objects have Write timestamp and Read timestamp, which denotes timestamp of last Read or Write on the object. Random transactions will be generated which are either Read or Write request on objects. These transactions can be long update or single Read-only query transactions. Each transaction is assigned a unique timestamp when it starts. Transaction’s operations will generate randomly in the system. Transaction’s operations will come in serialized order. Each data item can have multiple versions, these versions will be created by successful commit of Update transaction. Read-only will read the latest version.

Simulation results are compared on the basis of a number of Commit, Rollback, Waits and Deadlocks generated by two-phase locking and timestamp methods against each other.

[3.1.] MV2PL

Transactions are predefined into Read-only and Update. Locks are requested by different update transactions in exclusive and shared mode. When an operation wants to request read lock this will check if there is any write lock on the object. If there is none, Read lock is granted in the shared mode. This will allow \( n \) number of shared operations can be applied to objects.

When a transaction request write lock on the object there must be no lock on the object as Write operations are exclusive, therefore only one operation can apply a lock on the data. Write lock will be incompatible with other Read and Write operations. When incompatible lock is found on object by requesting transaction, it will send to Wait. This operation will remove from the Wait as soon as the acquired lock is released by object. Transactions block when they can't get a lock, and deadlock must be managed in one of the standard ways.

The transaction can acquire locks and can release only when all the operations of transaction will be succeeded to get locks. When the transaction commits, all locks will be released. This commit will not overwrite the old object but create a new version of the same object.

Read-only transactions will never lock objects, they just read the latest version. The timestamp is provided for each transaction when it starts running. At the point when a Readonly transaction wishes to get to a data, no locking is required. Rather, the transaction just reads the latest version of the data with a timestamp. A simulation model of each transaction experiences queues requesting for the data item. In this procedure, a transaction might be rolled-back. A rolled-back transaction is restarted again.

Performance degradation with standard locking is blocking and deadlocks, which are resolved by restarts of the transaction. This causes a secondary effect on performance degradation.
[3.2.] MYTSO

Transactions are predefined in to Read-only and Update. Each version of an object X is set to write timestamp and read timestamp; the startup timestamp of its making of transaction. Read requests from a transaction for an object are yielded by allowing the transaction to read the most recent version of object. Every version has a read-timestamp. The transactions which have read request must have greater timestamp than read a write timestamp of version. A request by a transaction to read an object is always successful and the version of the object.

An attempt by a transaction to write an object is only fruitful if the current transaction's timestamp is greater than version's latest write timestamp and Read timestamp. The transaction is aborted and restarted with another timestamp. Timestamp protocols are free from deadlocks, but cascading Rollbacks are more in number.

[4] PERFORMANCE EVALUATION

Performance evaluation is important in light of the fact that it gives us a comparative analysis of various multiversion algorithms on

- How they work to produce an execution that has an indistinguishable impact from a serial one?
- How they enforce Isolation by mutual exclusion among conflicting transactions?
- How they safeguard database consistency through consistency preserving execution of transactions and resolve read-write and write-write conflicts?

At the point when a transaction performs conflicting operations on data objects among active transactions that are in execution and not yet committed, the conflict manager of simulation model resolves conflicting operations using one of the following two approaches; (i) Blocking (Transactions made to wait) (ii) Rollback where restart of transaction gives low CPU utilization. While a transaction waits for disk I/O or for another transaction to terminate resulting in slowing down of processing considerably and creates low throughput. The mix transactions are comprising of update transactions and read-only transactions. The major output parameters are the read throughput and the update throughput, both measured D object and N operations in overall session.

The level of multi-programming is set to 300 so that 100 transactions will be in every session of the framework (although some of these transactions might be blocked or experience a restart delay). The simulation is done for the following cases:

4.1. Experiment 1: Small set of objects and greater number of transactions (50% Readonly and 50% Update)

The workload parameters for Experiment 1 are shown in Table 1 50% Readonly and 50% Update. 50% of these transactions are updaters that read and afterward update the randomly picked objects, and the other 50% are read-only transactions. Each read-only transaction reads various items and the size of the object is from 1 to 15 as they are arranged for simulations of this experiment. With these parameter settings, clashes between update transactions and Read-only transactions will undoubtedly happen.

Sonal Kanungo, Rustom. D Morena
Figure-1 shows Read-only will always succeed in both case of MV2PL and MVTO. MVTO will suffer from more Rollback of Update transactions, while MV2PL gives better results with Update transactions. 

**Table-1 (50% Readonly and 50% Update)**

<table>
<thead>
<tr>
<th></th>
<th>MV2PL</th>
<th>MVTSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Transactions</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total Data</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Total No of Operations</td>
<td>300(100*3)</td>
<td>300(100*3)</td>
</tr>
<tr>
<td>No of Commit</td>
<td>75</td>
<td>35</td>
</tr>
<tr>
<td>No of Rollback</td>
<td>25</td>
<td>65</td>
</tr>
<tr>
<td>No of time Send to Wait</td>
<td>235</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure-1**

**4.2. Experiment-2 Small set of objects and greater number of transactions (100% Readonly)**

In **100% Readonly** transactions, all operation will Read Objects and no transition will have to wait (MV2PL) and Rollback (MVTO). Therefore, these transactions are always successful and no Read operation ever has to wait. Still MV2PL gives better results as compared to MVTO because reading is performed on the time of transactions commit while in MVTO reading takes place at the start of transaction.
In Experiment-2, Figure-2 shows that more number of Rollback is generated by MVTO as all operations are read only; while no waits and deadlocks are generated by MV2PL. This will improve performance of MV2PL.

All operations are locked in the shared mode by MV2PL, so it can obtain n number of Read locks. Read operation of MVTO reads object that have timestamp less than transactions, therefore operation does not have to wait for others.

**Table 2 (100% Readonly)**

<table>
<thead>
<tr>
<th></th>
<th>MV2PL</th>
<th>MVTSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Transactions</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total Data</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Total No of Operations</td>
<td>300(100*3)</td>
<td>300(100*3)</td>
</tr>
<tr>
<td>No of Commit</td>
<td>96</td>
<td>92</td>
</tr>
<tr>
<td>No of Rollback</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>No of time Send to Wait</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure-2**

**4.3. Experiment-3 Small set of objects and greater number of transactions (20% Readonly, 80% Update)**

The experiment has been given a large size of the update transactions. This experiment will experience almost all conflicts between Read-only and Update transactions. All Multiversion algorithms, Read-only
transactions to execute using older versions of object and requiring Update transactions to work same as normal 2PL.
MV2PL provides good throughput for the large Read-only transactions, but it also provides the worst throughput for the update transactions. In Figure-3 where 80% update and 20% Read-only transactions were there, MVTO had given better performance than MV2PL.
All the experiments show that when more number of Update transactions is present then Read-only MVSTO works better. MV2PL gives poor performance when number of transactions increases.

**Table-3 (20% Readonly, 80% Update)**

<table>
<thead>
<tr>
<th></th>
<th>MV2PL</th>
<th>MVTSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Transactions</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total Data</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Total No of Operations</td>
<td>300(100*3)</td>
<td>300(100*3)</td>
</tr>
<tr>
<td>No of Commit</td>
<td>24</td>
<td>56</td>
</tr>
<tr>
<td>No of Rollback</td>
<td>16</td>
<td>404</td>
</tr>
<tr>
<td>No of time Send to Wait</td>
<td>635</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure-3
4.4. Experiment-4 Small set of objects and greater number of transactions (100% Update)

Transactions have combinations of Read and Write operations. Write-Write and Read-Write conflicts are bound to occur. In Figure-4 We have pure update transactions. Here, using MV2PL gives very poor performance as when object is locked by write operation, write and read both operation has to wait until lock obtained of object is released. These conflicts further lead to deadlock and transaction has to be Rollbacked to remove the deadlock.

MVTO in this case has given much better performance as it does not wait for locks and deadlocks, but because of conflicts a large number of Rollback occurred.

**Table-4 (100% Update)**

<table>
<thead>
<tr>
<th></th>
<th>MV2PL</th>
<th>MVTSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Transactions</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total Data</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Total No of Operations</td>
<td>300(100*3)</td>
<td>300(100*3)</td>
</tr>
<tr>
<td>No of Commit</td>
<td>36</td>
<td>12</td>
</tr>
<tr>
<td>No of Rollback</td>
<td>678</td>
<td>4345</td>
</tr>
<tr>
<td>No of time Send to Wait</td>
<td>3427</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure-4**

![Figure-4](image-url)
[5] CONCLUSION

This research is focused to examine the impact of read-write proportions on the performance of the Concurrency Control mechanism. For the analysis, the behavior of performances for different Multiversion Concurrency Control mechanisms, Simulation modeling has been used. We briefly review these works and their findings and argue that more work in this area is required for a good understanding of the behavior of Concurrency Control mechanism based on multiversion schemes.

Both MVTO and MV2PL algorithm deliver a serialize schedule. MVTO algorithm doesn't bring about deadlock. In MV2PL transactions aborts to avoid deadlocks, this degrades the performance of the database system. It will bring about wastage of system resources.

MVTO is suffers from cascading rollbacks which also degrade systems performance. Performance can improve in both algorithm by allowing more number of Readonly transactions.

REFERENCE

[8] RONG SUN, GOMER THOMAS, “Performance Results on Multiversion Timestamp Concurrency Control with Predeclared Writesets”, 1987 ACM, Pages 177-184

Sonal Kanungo, Rustom. D Morena