Multiversion Thomas' Write Rule Timestamp-based Concurrency Control

Habes Alkhraisat[1], Hasan Rashaideh [2]

[1] Department of Computer Science, Al-Balqa Applied University
Al-Salt 19117, Jordan

[2] Department of Computer Science, Al-Balqa Applied University
Al-Salt 19117, Jordan

Abstract

One of the fundamental properties of a transaction is isolation. When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved. Concurrency control techniques are used to ensure the noninterference or isolation property of concurrently executing transactions. Most of these techniques ensure serializability of schedules. This paper addresses the problem of modifying the Thomas write rule to support Multiversion Timestaming concurrency control technique.

Keywords— Transaction, ACID properties, concurrency control, timestamp ordering, Multiversion concurrency control, Thomas write rule, serializability graph SG

1. Introduction

A transaction is an atomic unit of processing that should either be completed in its entirety or not done at all. Transaction processing systems are systems with large databases and hundreds of concurrent users executing database transactions. Concurrency control ensure the correct executions of transactions. The database management system maintain the ACID properties of the transactions, which should be enforced by the concurrency control and recovery methods. The ACID properties are: (1). Atomicity: A transaction is an atomic unit of processing; it should be either performed in its entirety or not performed at all. (2). Consistency preservation: A complete execution of transaction takes the database from one consistent state to another. (3). Isolation: A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently. (4). Durability or permanency: The changes applied to the database by a committed transaction must persist in the database and must not be lost because of any failure [1].

When several transactions execute concurrently in the database, the isolation property may no longer be preserved. To ensure the isolation, the system must control the interaction among the concurrent transactions; concurrency control techniques are used to ensure the isolation property of concurrently executing transactions. Several current trends in the field of computing are giving rise to an increase in the amount of concurrency possible. As database systems exploit this concurrency to increase overall system performance, there will necessarily be an increasing number of transactions run concurrently.

Multi-Version Concurrency Control (MVCC) is an advanced technique for improving database performance in a multi-user environment. Multiversion concurrency control algorithm keeps the old values of a data item when the item is updated, each Write on a data item x produces a new copy (or version) of X. The DM that manages x therefore keeps a list of versions of X, which is the history of values that the DM has assigned to X. For each Read(x), the scheduler not only decides when to send the Read to the DM, but it also tells the DM which one of the versions of x to read.
The benefit of multiple versions for concurrency control is to help the scheduler avoid rejecting operations that arrive too late. For example, the scheduler normally rejects a Read because the value it was supposed to read has already been overwritten. With multiversions, such old values are never overwritten and are therefore always available to tardy Reads. The scheduler can avoid rejecting the Read simply by having the Read read an old version. When a transaction requires access to an item, an appropriate version is chosen to maintain the serializability of the currently executing schedule, if possible. The idea is that some read operations that would be rejected in other techniques could still be accepted by reading an older version of the item to maintain serializability [2].

Maintaining multiple versions may not add much to the cost of concurrency control, because the recovery algorithm may need the versions anyway. An obvious cost of maintaining multiple versions is storage space. To control this storage requirement, versions must periodically be purged or archived. Since certain versions may be needed by active transactions, purging versions must be synchronized with

respect to active transactions. This purging activity is another cost of multiversion concurrency control [3].

2. Timestamp ordering Protocol

Each transaction Ti in the system has a unique fixed timestamp, denoted by TS (Ti). The database management system assigns timestamp before the transaction T_i starts execution. If a transaction T_i has been assigned timestamp TS (Ti), and a new transaction T_j enters the system, then TS (T_i) < TS (T_j).

The timestamps of the transactions determine the serializability order. Thus, if TS (T_i) < TS (T_j), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction T_j.

To implement timestamps ordering, each data item Q has two timestamp values [1, 2]:

- W-TS (Q) denotes the largest timestamp of any transaction that executed write (Q) successfully.
- R-TS (Q) denotes the largest timestamp of any transaction that executed read (Q) successfully.

W-TS (Q) and R-TS (Q) timestamps are updated whenever a new read (Q) or write (Q) instruction is executed.

2.1 Basic Timestamp Ordering Protocol

The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

This protocol operates as follows [2]:

1. Suppose that transaction Ti issues read (Q).
   a. If TS (T_i) < W-TS (Q), then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T_i is rolled back.
   b. If TS(T_i) ⩾ W-TS(Q), then the read operation is executed, and R-TS(Q) is set to the maximum of R-TS(Q)and TS(T_i).
2. Suppose that transaction T_i issues write (Q).
   a. If TS (T_i) < R-TS (Q), then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the write operation and rolls T_i back.
   b. If TS (T_i) < W-TS (Q), then T_i is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls T_i back.
   c. Otherwise, the system executes the write operation and sets W-TS (Q) to TS (T_i).

2.2. Thomas' Write Rule Timestamp Ordering Protocol

The protocol rules for read operations remain unchanged. The protocol rules for write operations, however, are slightly different from the timestamp-ordering protocol of Section 2.1. Thomas' Write Rule is a modified version of the timestamp-ordering protocol in which obsolete write operations can be ignored under certain circumstances [1].

Suppose that transaction Ti issues write (Q):

1. If TS (T_i) < R-TS (Q), then the value of Q that T_i is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls T_i back.
2. If TS (T_i) < W-TS (Q), then T_i is attempting to write an obsolete value of Q. Hence, this write operation can be ignored.
3. Otherwise, the system executes the write operation and sets W-TS (Q) to TS (T_i).

By ignoring the write, Thomas' write rule allows schedules that are not conflict serializable but are correct. Thomas' write rule makes use of view serializability by, in effect, deleting obsolete write operations from the transactions that issue them. This modification of transactions makes it possible to generate serializable schedules that would not be possible under the protocols presented in Section 2.1.

3. Multiversion Concurrency Control Techniques

The concurrency-control schemes discussed thus far ensure serializability by aborting the transaction that issued the operation. For example, a read operation may be may be rejected (that is, the issuing transaction must be aborted) because the value that it was supposed to read has already been overwritten. These difficulties could be avoided if old copies of each data item were kept in a system [1,3].

In multiversion concurrency-control schemes, each write (Q) operation creates a new version of Q. When a transaction issues a read (Q) operation, the concurrency-control manager selects one of the versions of Q to be read. The concurrency-control scheme must ensure that the version to be read is selected in a manner that ensures serializability. It is also crucial, for performance reasons that a transaction is able to determine easily and quickly which version of the data item should be read [1].

3.1 Multiversion Timestamp Ordering Protocol

The timestamp-ordering protocol can be extended to a multiversion protocol. With each transaction Ti in the system, associates a unique static timestamp, denoted by TS (Ti). With each data item Q, a sequence of versions <Q_1, Q_2... Qm> is associated. Each version Q_k has two timestamps:

- W-TS (Q_k) is the timestamp of the transaction that created version Q_k.
- R-TS (Q_k) is the largest timestamp of any transaction that successfully read version Q_k.

Multiversion timestamp-ordering scheme operates as follows: Suppose that transaction Ti issues a read (Q) or write (Q) operation. Let Qk denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS (Ti).

1. If transaction Ti issues a read (Q), then the value returned is the content of version Q_k.
2. If transaction Ti issues write (Q),
   a. If TS (Ti) < R-TS (Q_k), then the system rolls back transaction T_i.
   b. If TS (T_i) = W-TS (Q_k), the system overwrites the contents of Q_k.
   c. Otherwise TS (T_i) > R-TS (Q_k), it creates a new version of Q.

The multiversion timestamp-ordering scheme has the desirable property that a read request never fails and is never made to wait. In typical database systems, where reading is a more frequent operation than is writing, this advantage may be of major practical significance [1, 2].

3.2 Modified Multiversion Thomas' Write Rule (MVTWR)

In this section, we are going to explain the modified Thomas' Write Rule by applying the multi version timestamp-ordering protocol.
Multiversion Thomas' Write Rule operates as follows: Suppose that transaction T_i issues a write (Q) operation. Let Q_k denote the version of Q whose write timestamp is the largest write timestamp less than or equal to TS (T_i).

1. If transaction Ti issues a read (Q), then the value returned is the content of version Q_k.
2. If transaction Ti issues write (Q),
   a. if TS(T_i) = W-TS(Q_k), the system overwrites the contents of Q_k;
   b. if TS(T_i) < W-TS (Q_k), this write operation can be ignored;
   c. Otherwise, it creates a new version of Q, with W-TS(Q_{k+1})= R-TS(Q_{k+1}) = TS(T_i).

The Multiversion Thomas' Write Rule has the desirable property that a read request never fails wait, also in a the case when the transaction T_i issues write (Q), and the TS (T_i) < R-TS, then the system never rolls back transaction T_i.

4. Proof of Correctness

To prove MVTWR correct, we must describe it in serializability theory. The scheduler processes $r_i[x]$ by first translating it into $r_i[x_k]$, where x_k is the version of x with the largest timestamp less than or equal to TS(T_i), and then sending $r_j[x_k]$ to the DM. It processes $w_i[X]$ by considering three cases. If it has already processed a Read $T_j[X_k]$ such that TS(T_i) < TS(Tj), it translates wi[x] into $w_i[x_i$. If it TS(T_i) < TS(T_k), then ignore the Write Ti[x]. Otherwise, it translates wi[x] into $w_i[x_i]$. Finally, to ensure recoverability, DMNS must delayed the processing of c_i until it has processed c_j for all transactions T_j that wrote versions read by T_i.

The following properties describe the essential characteristics of every MVTO history H over (T_0, ... T_n}.

- $MVTWR_1$. For each T_i, there is a unique timestamp TS(T_i);
- $MVTWR_2$. For every $r_k[x_j] \in H$, $w_j[x_j] < r_k[x_j]$ and TS(T_j) ≤ TS(T_k).
- $MVTWR_3$. For every $r_k[X_j]$ and $w_i[x_i] \in H$, i≠j, either
  (a) TS(T_i) < TS(T_j) or
  (b) TS(T_k) < TS(T_i) or
  (c) i = k and $r_k[x_j] < w_i[x_l]$.
- $MVTWR_4$. If $r_j[x_i] \in H$, i≠j, and $c_j \in H$, then $c_i < c_j$.

Property $MVTWR_1$, says that transactions have unique timestamps. Property MVTWR2, says that each transaction T_k only reads versions with timestamps smaller than TS(T_k). Property $MVTWR_3$, states that when the scheduler processes $r_k[X_j]$, xj is the version of x with the largest timestamp less than or equal to TS(T_k). $MVTWR_4$, states that H is recoverable.

These conditions ensure that H preserves reflexive reads-from relationships. In other words, MVTWR is a correct scheduler.
Proof: Define a version order as follows: $x_i \ll x_j$ iff TS(T_i) < TS(T_j). We now prove that MVSG(H, ≪) is acyclic by showing that for every edge $T_i \rightarrow T_j$ in MVSG(H,≪), TS(T_i) < TS(T_j).
Suppose $T_i \rightarrow T_j$ is an edge of SG(H). This edge corresponds to a reads from relationship. That is, for some X, T_j reads x from T_i. By MVTO, TS(T_i) 5 TS(T_j). By $MVTWR_2$, TS(T_i) ≠TS(T_j). So, TS(T_i) < TS(T_j) as desired.

Let $r_k[x_j]$ and $w_i[x_i]$ be in H where i, j, and k are distinct, and consider the version order edge that they generate. There are two cases: (1) $x_i \ll x_j$, which implies Ti → Tj is in MVSG(H, $\ll$); and (2) $x_j \ll x_i$, which implies $T_k \rightarrow T_i$ is in MVSG(H, $\ll$). In case (l), by definition of $\ll$, $TS(T_i) < TS(T_j)$. In case (2), by $MVTWR_3$, either $TS(T_i) < ts(T_j)$ or $TS(T_k) < TS(T_i)$. The first option is impossible, because $x_j \ll x_i$ implies $TS(T_j) < TS(T_i)$. SO, $TS(T_k) < TS(T_i)$ as desired. Since all edges in MVSG(H, $\ll$) are in timestamp order, MVSG(H, $\ll$) is acyclic.

5. Conclusion

In this paper, we explained how to extend the Thomas' Write Rule timestamp-ordering protocol using the multiversion timestamp-ordering scheme. A multiversion concurrency-control scheme is based on the creation of a new version of a data item for each transaction that writes that item. When a read operation is issued, the system selects one of the versions to be read. The concurrency-control scheme ensures that the version to be read is selected in a manner that ensures serializability, by using timestamps. A read operation always succeeds. The multiversion Thomas' Write Rule, allows greater potential concurrency than does the basic Thomas' Write Rule.

References

[1] A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concept, 6th ed, New York: McGraw-Hill, 2010.

[2] R. A. Elmasri, S. Navathe, Fundamentals of Database Systems, 6th ed., Addison-Wesley, 2010.

[3] Philip A. Bernstein, Eric Newcomer, Principles of Transaction Processing, 2nd Ed, Morgan Kaufmann, 2009.

Dr. Habes Alkhraisat is a professor of computer science at Al-Balqa Applied University, Salt, Jordan. He received his Bsc. Degree in Information Technology from Al-Balqa Applied University, Jordanm in 2001, master degree of computer science from University of Jordan, Jordan, 2003, and Ph.D degree of Automated System from Saint Petersburg Electrotechnical University, 2008. In 2009, he joined the Department of Computer Science, Al-Balqa Applied University, as assistant professor. His current research interests include database management system, ear recognition, development methodology, and biometric.

Dr. Hasan Rashaideh is a professor of computer science at Al-Balqa Applied University, Salt, Jordan. He received his B.Sc. and master Degrees in Computer Science from Yarmok University, Jordan in 1999, 2001 respectively, and Ph.D. degree of Automated System from Saint Petersburg Electrotechnical University, 2008. In 2009, he joined the Department of Computer Science, Al-Balqa Applied University, as assistant professor. His current research includes image processing, ear recognition, and development methodology.