Validation concurrency control protocol in parallel real-time database systems

LEI Xiang-dong(雷向东), YUAN Xiao-li(袁晓莉)
College of Information Science and Engineering, Central South University, Changsha 410083, China

Abstract: In parallel real-time database systems, concurrency control protocols must satisfy time constraints as well as the integrity constraints. The authors present a validation concurrency control (VCC) protocol, which can enhance the performance of real-time concurrency control mechanism by reducing the number of transactions that might miss their deadlines, and compare the performance of validation concurrency control protocol with that of HP2PL (High priority two phase locking) protocol and OCC-TI-WAIT-50 (Optimistic concurrency control-time interval-wait-50) protocol under shared-disk architecture by simulation. The simulation results reveal that the protocol the author presented can effectively reduce the number of transactions restarting which might miss their deadlines and performs better than HP2PL and OCC-TI-WAIT-50. It works well when arrival rate of transaction is lesser than threshold. However, due to resource contention the percentage of missing deadline increases sharply when arrival rate is greater than the threshold.

Key words: parallel database system; real-time database; concurrency control; validation; transactions

1 INTRODUCTION

Real-time database is widely used in the process control of avionics, and robotics. In real-time database systems, concurrency control protocols must not only maintain the consistency constraints of the database, but also satisfy the timing requirements of the transactions accessing the database. In the past decade, parallel database systems have been advanced. The combination of parallel database system and real-time database becomes a new trend. There are several architectural models for parallel databases\(^1\), such as shared-memory, shared-disk, shared nothing and hierarchical. We will pay our attention to shared-disk architecture.

Real-time concurrency control mechanism has two broad streams. The first stream considers extensions to locking algorithms\(^2\). Two major drawbacks have been identified with locking-based strategies for real-time scenarios. First, a transaction might take lots of time to wait for data locks and thus miss its deadline. The other problem is that a high priority transaction is blocked by a lower priority transaction. The other stream is optimistic concurrency control\(^3\). But it is well-known that, in optimistic concurrency control, a large number of transactions restarts, which strongly affects real-time concurrency control mechanism. In this paper, we present a validation-based concurrency control protocol to reduce the number of transaction missing their deadlines for enhancing the performance of real-time concurrency control mechanism.

2 TIMESTAMP

For each transaction \(T_i\) in the system, we associate a unique fixed timestamp\(^4\), denoted by \(TS(T_i)\). This timestamp is assigned by the database system before the transaction \(T_i\) starts.
execution. If a transaction \( T_i \) has been assigned timestamp \( TS(T_i) \), 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 \( TS(T_i) < TS(T_j) \), the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction \( T_i \) appears before transaction \( T_j \). Each data item \( Q \) has a read and a write timestamps.

- \( W\text{-timestamp}(Q) \) denotes the largest timestamp of any transaction that executed write (\( Q \)) successfully.
- \( R\text{-timestamp}(Q) \) denotes the largest timestamp of any transaction that executed read (\( Q \)) successfully.

These timestamps are updated whenever a new read (\( Q \)) or write (\( Q \)) instruction is executed.

Each transaction \( T_i \) executes in three different phases in its lifetime, in order, as follows:

1) **Read phase.** During this phase, the execution of transaction \( T_i \) takes place. The values of the various data items are read and stored in variables local to \( T_i \). All write operations are performed on temporary local variables, without updates of the actual database.

2) **Validation phase.** Transaction \( T_i \) performs a validation test to determine whether it can copy to the database from the temporary local variables that hold the results of write operations without causing a violation serializability.

3) **Write phase.** If transaction \( T_i \) succeeds in validation, then the actual updates are applied to the database. Otherwise, \( T_i \) is rolled back.

Each transaction must go through the three phases mentioned above. However, all three phases of concurrently executing transactions can be interleaved.

### 3 TIMESTAMP-ORDERING IN READ PHASE

We determine the serializable order by the timestamp-ordering technique using the value of the timestamp \( TS(T_i) \) in read phase.

1) Suppose that transaction \( T_i \) issues read (\( Q \)).

   a. If \( TS(T_i) < W\text{-timestamp}(Q) \), then \( T_i \) needs to read the value of \( Q \) that is already overwritten. Hence, the read operation is rejected, and \( T_i \) is rolled back.

   b. If \( TS(T_i) \geq W\text{-timestamp}(Q) \), then the read operation is executed.

2) Suppose that transactions \( T_i \) issues write (\( Q \)).

   a. If \( TS(T_i) < R\text{-timestamp}(Q) \), then the value of \( Q \) that \( T_i \) is producing was needed previously, and the system assumed that the value would never be produced. Hence, the write operation is rejected, and \( T_i \) is rolled back.

   b. If \( TS(T_i) < W\text{-timestamp}(Q) \), then \( T_i \) is attempting to write an obsolete value of \( Q \). Hence, this write operation is rejected, and \( T_i \) is rolled back.

   c. Otherwise, the write operation is executed, and \( W\text{-timestamp}(Q) \) is set to \( TS(T_i) \).

   If a transaction is rolled back, it is assigned a new timestamp and restarted.

### 4 VALIDATION

The validation checking for transaction \( T_i \) requires that, for all transactions \( T_j \) with \( TS(T_j) < TS(T_i) \), the following condition be satisfied.

The set of date items written by \( T_j \), denoted by \( WS(T_j) \), does not intersect with the set of data items read by \( T_i \), denoted by \( RS(T_i) \), and \( T_j \) completes its write phase before \( T_i \) starts validation phase. This condition ensures that the writes of \( T_j \) and \( T_i \) do not overlap. Since the writes of \( T_j \) do not affect the read of \( T_i \), and \( T_i \) cannot affect the read of \( T_j \), the serializability order is maintained.

For all transactions \( T_j \) with \( TS(T_j) > TS(T_i) \), the validation checking for transaction \( T_i \) also satisfies the following condition:

The set of data items read by \( T_i \) does not intersect with the set of data items written by \( T_j \). Since the writes of \( T_i \) do not affect the read of \( T_j \), and \( T_j \) cannot affect the writes of \( T_i \), the serializability order is indeed maintained.

If validating transaction \( T_i \) is committed, for