American Journal of Applied Sciences 3 (2): 1706-1710, 2006
ISSN 1546-9239
© 2006 Science Publications

On Optimistic Concurrency Control for Real-Time Database Systems

Amer Abu Ali
Faculty of Information Technology, Philadelphia University, Jordan

Abstract: The performance of database transaction processing system can be profoundly affected by the concurrency control method employed since it is necessary to preserve database integrity in a multi-user environment. In addition to satisfying the consistency requirement as in traditional database system, real-time database systems must also satisfy timing constraints. In this study we present a virtual run policy for the restarted first run optimistic transactions and compare its performance with optimistic concurrency control in firm real-time database system.

Key words: Database transaction system, real-time database system, optimistic concurrency control

INTRODUCTION

The problem of concurrency in database systems has been considered by many researchers and several concurrency control mechanisms have been introduced[1-4]. Concurrency means that different users have access to the database at the same time. In such a system each user must be protected against others. We must avoid the situation in which one user is altering an object in the database, while another user is reading it[1]. The task of a concurrency control mechanism is to ensure the consistency of the database while allowing a set of transactions (i.e., user's programs) to execute concurrently.

During the last few years the interest in the study of real time database system (RTDBS) has increased considerably because of their importance in a wide range of applications. A real time database is a database system where transactions have explicit timing constraints such as deadlines[5-7]. Concurrency control is one of the main issues in the study of real time database systems. In addition to satisfying consistency requirements as in traditional database systems, a real time transaction processing system must also satisfy timing constraints. To support real time transaction processing the new criteria and issues to be considered in design and implementation of real time database systems are scheduling of CPU and I/O and the requirement that conflict resolution schemes used should be time cognizant[7].

Terms and definitions: The *database* is viewed as a set of distinct data objects. An *object* has a name and a value. Associated with the database is a set of assertions called *integrity constraints*. The database is in a correct state if the set of objects satisfies the integrity constraints. The *state of the database* undergoes changes because of the actions performed by the users. The sequence of actions of one user is called *Transaction*. *A transaction* is a program that issues reads and writes to a DBMS and it represents a unit that preserves integrity of the database. Transaction when executed alone transforms the database from one correct state to another correct state, but during intermediate stages of the execution of a transaction the integrity constraints may be violated, that is why concurrency control mechanisms prevent other transactions from seeing these transient stages. A transaction is executed *atomically* even in the face of failures, the database system either executes all of its actions or performs none of them[2]. *Consistency* deals with the correct processing of concurrent transactions. The *concurrent execution* of the transactions T1,......,Tn must produce the same effect as the execution of some *serial schedule*. *A serial schedule* is a schedule consisting of a sequence of transactions without any interleaving between their reads and writes (if Ti precedes Tj in the serial schedule, then all of Ti's operations precede all of Tj's operations). Since each transaction is a correct computation, a serial schedule is correct. *An interleaved schedule* (concurrent execution of transactions) is considered to be correct if its effect on the database is equivalent to that of a serial schedule and it is called *a serializable schedule*. *Serializability* is the main correctness criterion for concurrency control[1]. *Serializability* requires that the execution of each transaction must appear to every other transaction as a single atomic step.

Two *transactions conflict* if they access the same data object and one or both of them does (do) a write operation or update on that data object. The order in which operations execute is computationally significant if and only if the operations conflict.

Characteristics of data and transactions in RTDBS: Transaction characterization in RTDBS is based on the manner in which data is used by the transaction, nature of time constraints and the significance of executing a transaction by its deadline or more precisely the consequence of missing specified time constraints. In

Corresponding Author: Amer Abu Ali, Faculty of Information Technology, Philadelphia University, Jordan

hard RTDBS, missing deadlines of transactions may result in catastrophic consequences, i.e., they are safety critical transactions. In soft RTDBS, transactions have time constraints but there may be some value in completing the transactions even after their deadline and this value drops to zero at a certain point past the deadline. When this value drops to zero by missing the transaction deadline it is referred to as *Firm RTDBS* but catastrophic consequence do not result if their deadlines are missed[8]. In these simulation experiments we assume firm real time database system model where transactions that missed their deadlines are aborted and permanently discarded from the system.

CONCURRENCY CONTROL

Conflict detection: Conflicts between transactions can be detected in two ways. Pessimistic method detects conflicts before making access to the data object[2]. When a transaction requests access to some data item, the concurrency control manager will examine this request and will determine whether to grant the request or not (if a conflict will occur or not). The optimistic method detects conflicts after transactions have accessed the data object when checking for serializability is done later at the certification time[4].

Optimistic schemes are designed to get rid of the locking overhead. They are optimistic in the sense that they take into account the explicit assumption that conflicts among transactions are rare events. They rely on the hope that conflicts will not occur. Since locks are not used in pure optimistic concurrency control they are deadlock free (one of the disadvantages of lock-based schemes). The task of concurrency control is deferred until the end of transaction when some checking for potential conflicts has to take place and will be resolved accordingly, taking into consideration the amount of progress that has been done and the nature of conflict with transactions

Resolving conflicts among concurrent transactions: When concurrency control detects a conflict among some concurrent transactions accessing the same object, a conflict resolution mechanism needs to be put on. Concurrency control manager decides which transaction (victim) to penalize (the lock holder or the requester) and chooses an appropriate action and suitable timing. Two possible actions are most used: Blocking (wait) and abort (restart). In pessimistic concurrency control either blocking or abort can be used to resolve the conflict[1]. However, in optimistic concurrency control only aborting is appropriate since conflict has been detected after the transaction has accessed the data object and performed some computation[4]. As for timing of action, it is immediate for blocking but it can be immediate or deferred (delayed) for aborting.

OPTIMISTIC CONCURRENCY CONTROL

The basic idea of an optimistic concurrency control mechanism is that the execution of a transaction consists of three phases: read, validation and write phases as in Fig. 1. For all optimistic concurrency control (OCC) schemes a conflict is detected after the data object has been accessed. In the OCC, conflict detection and resolution are both done at the certification time when a transaction completes its execution; it requests the concurrency control manager to validate all its accessed data objects. If it has not yet been marked for abort, it enters the commit phase where it writes all its updates to the database. Backward-oriented OCC (BOCC) checks during the validation test of Tj whether its readset RS(Tj) intersects with any of the write sets WS(Ti) of all concurrently executed transactions Ti having finished their read phases before Tj. Forward-oriented OCC (FOCC) checks during the validation phase of Tj whether its write set WS(Tj) intersects with any of the read set RS(Ti) of all transactions Ti having not yet finished their read phases[4, 9,10].

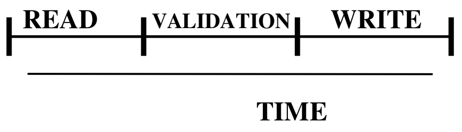

Fig. 1: The three phases of an optimistic transaction

Extension of optimistic concurrency control for RTDBS: Ideally optimistic concurrency control (OCC) should be non-blocking and deadlock free. These properties make OCC attractive in real-time transaction processing. OCC may be in a better position to be integrated with priority driven CPU scheduling. To adapt OCC into RTDBS the main issue is how to incorporate priorities (time constraints) into conflict resolution[8,11,12]. The key component of optimistic concurrency control schemes is the validation phase where a transaction's destiny is decided. Transaction validation can be performed in one of two ways: forward validation and backward validation. In protocols that perform backward validation the validating transaction either commits or aborts depending on whether it has conflicts with transactions that have already committed. So this scheme does not allow us to take transaction characteristics into account and it is not suitable for real time database. In forward validation however, either the validating transaction or the conflicting ongoing transactions can be aborted to resolve conflicts. This scheme can be extended to real time database since the timing characteristics of transaction can be considered and proper decision can be taken in aborting, delaying the committing transaction or aborting the conflicting ongoing transactions. We explain below the three schemes used in our simulation experiments.

OCC-forward validation with virtual run policy: In this scheme (OCC_FV) the transaction that reaches its validation phase is allowed to commit if it is not a virtual first run transaction and all the active conflicting transactions which are in their read phases are immediately aborted and restarted if they are rerun transactions. In case some of the conflicting read phase transactions are in their first run, instead of aborting them they enter their virtual run and continue their read phase so as to bring data objects required to buffer, assuming the system buffer has a high retention effect, then a transaction in its second run and onward does not need to access the disk since the data objects are already in memory. When the virtual run transaction completes its read phase, it is aborted and resubmitted to the system to start its real second run. It is clear that there is no point to allow restarted rerun transaction to complete its read phase in virtual mode since all its data items are already in memory. This scheme does not take the transactions timing constraints into account and favours the validating one to save the amount of progress done by the validating transaction since it is near completion and will definitely complete if it is not restarted.

OCC-sacrifice with virtual run policy: It is an optimistic protocol which uses a priority-driven abort for conflict resolution. In this protocol (OCC_OS) when a transaction reaches its validation phase, it is aborted if one or more conflicting transactions have higher priority than the validating one; otherwise it commits and all the conflicting read phase transactions are restarted immediately. This protocol uses transaction priority (timing constraints) in such a way that the validating transaction sacrifices itself for the sake of conflicting ones with higher priority.

If some of the restarted read phase transactions are in its first run, it enters the virtual run phase as explained above to complete its read phase, so its access pattern will be known and brought to buffer. On completing its virtual run, it is aborted and restarts its real second run.

OCC-abort50 with virtual run policy: In this scheme (OCC_A50) when a transaction reaches its validation phase, its priority is checked against those conflicting transactions in the read phase. If more than 50 percent of the transactions in their read phase have higher priority than the transaction in its validation phase, the validating transaction is aborted and all other transactions are allowed to continue. If the number of transactions in the read phase having higher priority than validating transaction is less than or equal to 50 percent, the validating transaction is allowed to commit and all the other transactions are restarted.

If some of the restarted read phase transactions are in its first run, it enter the virtual run phase as explained above.

SIMULATION MODEL

Our program to simulate a RTDBS system was written in C. For each of the following experiments the simulation was run with the same parameter values for 10 different random number seeds. Each run continued until 2000 transactions were executed. For each run the statistics gathered during the first few seconds were discarded in order to let the system stabilize after initial transient condition.

The simulation model for RTDBS is a single-site disk resident and memory resident database system operating on shared memory multiprocessors. CPUs share a single queue and the service discipline used for the queue is priority scheduling without preemption. Each disk has its own queue and is also scheduled with priority scheduling. Figure 2 shows the RTDBS queuing model.

In this model, the execution of a transaction consists of multiple instances of alternating data access request and data operation steps until all of the data operations in it complete or it is aborted for some reason. When a transaction completes its data access requests, it requests the concurrency control manager to validate them. if it is validated it enters the commit phase with raised priority to maximum so it can complete its write phase as fast as possible; otherwise it is aborted and enters the deadline test, if it missed its deadline it is terminated and discarded from the system since with the firm deadline assumption, transactions that have missed their deadlines are aborted and permanently discarded from the system, or it is restarted if there is a time to complete before missing its deadline. The data operation consists of disk access and CPU computation and the transaction passes through disk queue and CPU queue.

The database is modeled as a collection of data objects. A transaction consists of a mixed sequence of read and writes operations. We assume that a write operation is always preceded by a read, that is, the write set of a transaction is always a subset of its read set. A data item that is read is updated with the probability *Update probability*.

When a transaction attempts to read a data item, the system determines whether the object is in memory or disk using the probability *DISK ACCESS PROB*. If the data item is determined to be in memory, the transaction can continue processing without disk access. Otherwise, an I/O service request is created and placed in the input queue of the appropriate disk. The database is partitioned equally over the disks.

Transactions arrive in a Poisson stream, i.e., their inter-arrival times are exponentially distributed.

The assignment of deadlines to transaction is controlled by the parameters : *minimum slack factor* and *maximum slack factor* which set a lower and upper bound, respectively , on a transaction's slack time and it

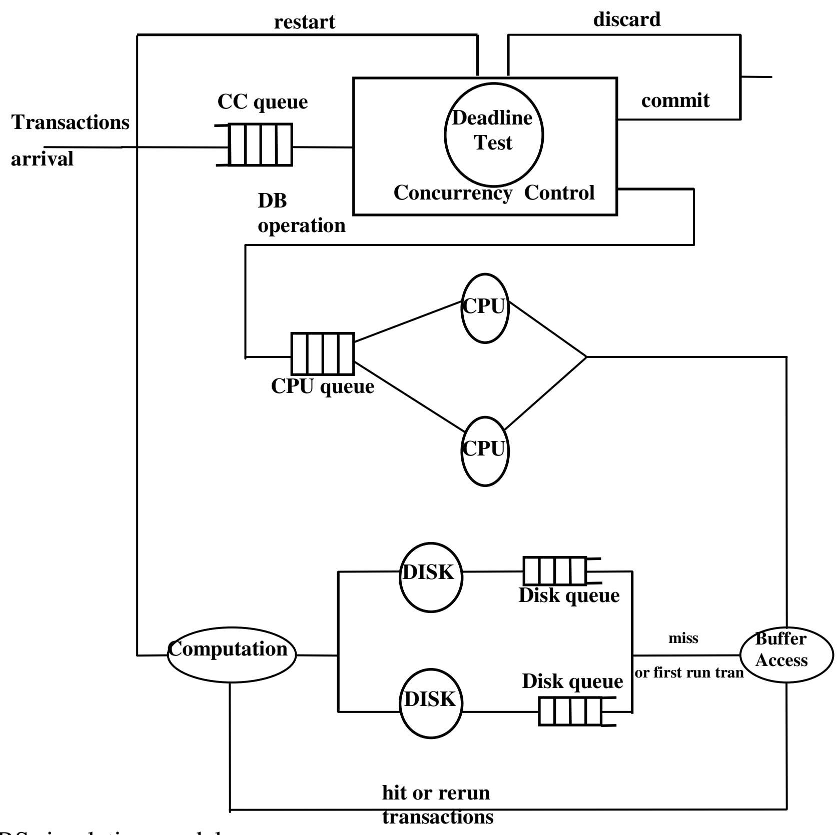

Fig. 2: RTDBS simulation model

is generated uniformly using the arrival time, transaction length, CPU time and disk time.

In this system, the priorities of transactions are assigned by the *Earliest Deadline First* policy, which uses only deadline information to decide transaction priority, but not any other information about transaction execution time.

Table 1: System resources and workload parameters

Parameter	Value
Number of data objects in database	500
Number of processors	4
Number of disks	8
Mean CPU time for processing a data object	15
Mean disk service time for a data object	25
Disk access probability	0.5
Update probability per accessed object	0.5
Mean transaction length (in accessed objects)	10
Minimum slack factor	2
Maximum slack factor	8

The important goal of RTDBS is to meet the time constraints of the transactions, therefore the primary performance metric used is the percentage of transactions which miss their deadlines, referred to as *Miss Percentage. Miss Percentage* is calculated with the following equation:

Miss Percentage = 100 * (no. of tardy transactions / no. of transactions arrived).

We show also the average number of restarts per transaction which is referred to as restart count.

RESULTS AND DISCUSSION

Here we present the performance results of our experiments for extending the optimistic concurrency control to real-time database systems and investigate the performance gain while incorporating its technique with virtual run policy for aborted first run transaction assuming sufficient buffer so that data blocks referenced by aborted transactions continue to be retained in memory and be available for access during reruns without I/O by the aborted or restarted rerun transaction. The optimistic concurrency control scheme OCC_FV with virtual run policy does better than the scheme without this policy under low system workload level up to 20 transactions/sec where system workload is controlled by the arrival rate of transactions in the system, but as the number of arriving transactions increases its performance is somewhat degraded. This is because the restarted first run transactions under this policy continue their read phases to bring the required data objects in memory in virtual run mode, increase the already high system resources contention since they compete for system resources and waits in system queues to complete their read phases in contrast to the other policy where the aborted first run transactions are restarted immediately. We get similar results for OCC_OS and OCC_A50 but are not shown due to space limitation.

It is clear that the schemes using virtual run policy outperform significantly the other schemes for a wide range of system workload due to the elimination of I/O operations for rerun transactions since all the required data blocks are already in memory, brought by the first run of the restarted transaction in virtual mode. The virtual run policy helps transactions to complete fast and reduce the average number of restarts transactions encounter before completion. Similar results for OCC_OS and OCC_A50 are obtained.

At low arrival rate there is no much difference among the three protocols. However, as the arrival rate increases, OCC_A50 does better than OCC_OS and OCC_FV does even better than OCC_A50. The improvement in performance of OCC-FV can be done if it avoids wastage of work done by transactions as every transaction which reaches its validation phase is allowed to complete, unlike the case of OCC-OS where a transaction in its validation is aborted for the sake of a higher priority transaction still in its read phase which may later be killed. The OCC-A50 gives better performance than OCC-OS as we are aborting the transaction in its validation phase only if there are more than 50 percent of the transactions in the read phase of higher priority than the validating transaction.
The restart counts of all the three schemes decrease after a certain workload point when system resources contention dominates data contention in discarding deadline missing transactions.

CONCLUSION

A major difference between conventional database and real-time database transaction processing is their approach to resolving data and resource conflicts. Conventional database attempts either to be fair in data and resource allocation or to maximize resource utilization. In real-time databases, timely transaction execution is more important and both fairness and maximum resource utilization become secondary goals. Also, in contrast to conventional databases that use transaction response time and throughput as performance measures, real-time databases use the percentage of transactions that complete within their deadlines. In this study we presented some features which can be added to concurrency control, virtual run policy for restarted first run transaction and we show that it improves the performance of real-time optimistic concurrency control schemes especially under moderate system workload level and in systems with disk resident databases.

REFERENCES

1. Bernstein, P. and N. Goodman, 1981. Concurrency control in distributed databasesystems. Comp. Sur., 13: 185-221.
2. Franaszek, P. and J. Robinson, 1985. Limitation of concurrency in transaction processing. ACM Trans. Database Syst., 10: 1-28.
3. Rosenkrantz, D., R. Strearns and P. Lewis II, 1978. System level concurrency control for distributed database systems. ACM Trans. Database Syst., 3: 178-198.
4. Kung, H. and J. Robinson, 1981. On optimistic method for concurrency control. ACM Trans. Database Syst., 6: 213-226.
5. Song, X. and J. Liu, 1995. Maintaining temporal consistency: Pessimistic vs. optimistic concurrency control. IEEE Trans. Knowledge and Data Engg., 7: 786-796.
6. Ozsoyoglu, G. and R. Snodgrass, 1995. Temporal and real-time databases. IEEE Trans. on Knowledge and Data Engg., 7: 513-532.
7. Ramamritham, K., 1993. Real-Time Databases. Distributed and Parallel Databases I , pp: 199-226.
8. Haung, J., J. Stankovic, D. Towsley and K. Ramamritham, 1990. Real-time transaction processing: Design, implementation and performance evaluation. COINS Technical Report May 1990. Department of Computer Science, University of Massachusetts at Amherst.
9. Harder, T., 1984. Observations on optimistic concurrency control schemes. Information Systems, 9: 111-120.
10. Lee and S.H. Son, 1993. Using dynamic adjusting of serialization order for real-time database systems. Proc. the 14th Real-Time Systems Symp., pp: 66-75, Raleigh-Durham, NC.
11. Lee and S.H. Son, 1995. Performance of CC Algorithms for Real Time Database Systems. Prentice-Hall.
12. Huang, J., J. Stankovic, D. Towsley and K. Ramamritham, 1989. Experimental evaluation of real-time transaction processing. Real-Time Systems Symp.