Incorporating real-time scheduling methods into database management systems

Özgür Ulusoy

Bilkent University, Ankara, Turkey

Received 14 January 1995
Revised 15 February 1995

Abstract.

Many database applications today are characterised by the requirement of timely access to data. This requirement leads to an increasing trend towards adapting real-time scheduling techniques to the management of data access requests. In this paper, we summarise and stimulate developments of time-cognisant scheduling techniques for database management systems. In particular, we review briefly the methods used in mapping timing constraints of transactions into priorities, and the priority-based protocols used for concurrency control. We also suggest useful directions for future research.

1. Introduction

An increasing number of database applications today are characterised by the requirement to access and manipulate data in a timely manner. Among those application areas are information retrieval systems, computer-integrated manufacturing, airline reservation systems, stock market, banking, and command and control systems. As opposed to conventional database management systems (DBMSs), the DBMS of such applications not only has to maintain the consistency of the underlying database, but also satisfy timing constraints associated with transactions. Consider a transaction that is executed in a stock market to update the database with new information. The transaction needs to satisfy certain timing constraints to ensure that the database contains an accurate representation of the current market [1]. As another example, a transaction may be executed to learn the price of a particular stock. The result of the transaction should return as quickly as possible since the prices can change very quickly.

The major challenge posed to the researchers is to adapt real-time scheduling methods to DBMSs. However, the scheduling algorithms used in real-time systems assume in general *a priori* knowledge about the characteristics of transactions, such as arrival time, data/resource access pattern, worst case execution time, etc. Thus, it is predictable whether the time constraints of a transaction can be satisfied. In a database system, on the other hand, there exist a number of sources of unpredictability [7].

(1) Transactions might have conflicting accesses on data and hardware resources. Access conflicts usually lead to blocking of transactions.
(2) The execution path of a transaction is dependent on the current values of data.
(3) Delay due to dynamic paging and I/O might be experienced.
(4) To maintain database consistency, it might be necessary to abort and later restart a transaction.

All these factors make it virtually impossible to predict computation times of database transactions. As a result, real-time scheduling methods cannot be directly applied to database systems. However, it is quite possible to use some ideas from real-time scheduling in extending traditional database management techniques to observe timing constraints of transactions.

Correspondence to: Dr O. Ulusoy, Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey. Fax: +90 312 266 4126. E-mail: oulusoy@bilkent.edu.tr

Our goal in this paper is to summarise and stimulate developments of time-cognisant transaction scheduling techniques in DBMSs. We outline approaches to various aspects of processing transactions that are associated with timing constraints. We also suggest useful directions for future progress. Throughout the paper, we explain the concepts at an intuitive, rather than at a detailed technical, level. The next section provides an examination of methods used in mapping timing constraints of transactions into priorities. Section 3 discusses priority-based concurrency control techniques that control the interaction among concurrently executing transactions to satisfy both the consistency requirement of the database and the timing constraints of transactions. Section 4 provides some concluding remarks.

2. Priority assignment

The timing constraint of a transaction typically takes the form of a *deadline*. The deadline of a transaction indicates that it is required to complete the transaction before a certain time in the future. A typical categorisation of transactions concerns the strictness of the deadlines assigned.

- *Hard deadline transactions* are associated with strict deadlines and the correctness of transaction operations depends on the time at which the results are produced [12]. The system must provide schedules that guarantee deadlines.
- *Soft deadline transactions* are scheduled based on their deadlines, and satisfaction of deadlines is still an important performance goal in scheduling transactions; however, in this case, there is no guarantee that all deadlines will be met. A soft deadline transaction is executed until completion, regardless of whether its deadline has expired or not.
- *Firm deadline transactions* also do not carry strict deadlines, i.e. missing a deadline may not result in a catastrophe, but, unlike soft deadline transactions, they are aborted by the system once their deadlines expire. Typically, no value will be imparted to the system if a firm deadline transaction misses its deadline.

Processing hard deadline transactions in a database system is generally considered to be infeasible because, as we discussed earlier, it is difficult to predict computation times and thus to provide schedules that guarantee deadlines. Real-world examples of applications supporting soft or firm deadline transactions are provided in [1]. Banking systems and airline-reservation systems usually process soft deadline transactions. When a customer submits a transaction, if the system cannot generate a response to the transaction within its deadline, the customer prefers getting the response late to not getting it at all. Stock market trading is an example of applications supporting firm deadline transactions. If, for instance, a transaction is submitted to learn the current price of a particular stock, the system should either return the result in a specified time period or not perform the operation at all, because conditions in the stock market can change very quickly.

As stated before, one of the primary scheduling goals in processing time-constrained transactions is to meet transaction deadlines. The scheduler thus assigns a priority to each transaction based on its deadline. Two of the most popular priority assignment schemes based on transaction deadlines are:

(1) *earliest deadline first* (EDF): a transaction with an earlier deadline has higher priority than a transaction with a later deadline;

(2) *least slack first* (LSF): the *slack time* of a transaction is defined as the maximum length of time the transaction can be delayed and still satisfy its deadline. The LSF policy assigns the highest priority to the transaction with the least slack time. When a transaction T arrives at the system, its slack time ST_T can be evaluated using the following formula:

$$ST_T = D_T - (AT_T + ET_T)$$

where D_T, AT_T, and ET_T denote the deadline, the arrival time, and the estimated execution time of transaction T, respectively. The LSF policy assumes that each transaction provides its execution time estimate. The dynamic version of the LSF deadline assignment scheme requires the evaluation of transaction priorities at each decision point [6]. Let PT_T and $ST_T(t)$ denote the processing time spent so far by T and the slack time of T at time t, respectively. The slack time of T at decision point t can be determined by the following formula:

$$ST_T(t) = D_T - (t + ET_T - PT_T(t))$$

The EDF policy is usually preferred to LSF because the estimate of execution times is often unavailable for database transactions.

Some applications may assign different values to transactions, where the *value* of a transaction reflects the return the application expects to receive if the

transaction is completed before its deadline [4]. The scheduling goal for such applications is to maximise the value realised by the completed transactions. Some algorithms were provided to establish a priority ordering among transactions that are distinguished by both values and deadlines [2, 4]. A range of trade-offs between value and deadline has been covered in those algorithms. One common algorithm gives equal weight to deadline and value in determining the priority of transactions. The priority P_T of transaction T is specified by $P_T = V_T/D_T$, where V_T denotes the value of transaction T. A variation of this algorithm uses the relative deadline instead of the absolute deadline in assigning priorities. The relative deadline is defined as the difference of the transaction deadline and the transaction arrival time, i.e. $P_T = V_T/D_T - AT_T$.

3. Time-cognisant concurrency control

If the transactions processed in a database system are associated with deadlines, implementation of concurrency control protocols in that system is difficult due to the conflicting requirements of meeting deadlines and maintaining data consistency. Concurrency control protocols proposed so far to preserve data consistency in conventional database systems are all based on transaction blocking and transaction restart, which makes it difficult to satisfy deadlines. Thus, a need has arisen for developing concurrency control protocols that take the timing constraints into account while scheduling transactions. Some scheduling techniques have been borrowed from real-time systems to be used in developing such protocols.

There is a growing literature about development/evaluation of time-cognisant concurrency control protocols for database systems (e.g. [1, 3, 5, 9, 13]). In this section, we give an overview of the protocols available in the literature.

In a lock-based concurrency control protocol, a situation that needs to be carefully handled is *priority inversion*. Priority inversion can be defined as uncontrolled blocking of high priority transactions by lower priority transactions [8]. Two main approaches have been pursued to solve the priority inversion problem: *priority inheritance* (PI) and *priority abort* (PA). They are both time-cognisant extensions of the conventional two-phase locking (2PL) protocol. Variations of these approaches have been the basis for the other lock-based concurrency control protocols.

PI, proposed by Sha *et al.* [9], ensures that when a transaction blocks higher priority transactions, it is executed at the highest priority of the blocked transactions; in other words, it inherits the highest priority. Due to the inherited priority, the transaction can be executed faster, resulting in reduced blocking times for high priority transactions.

PA prevents priority inversion by aborting low priority transactions whenever necessary [1]. In resolving a data lock conflict, if the transaction requesting the lock has higher priority than the transaction that holds the lock, the latter transaction is aborted and the lock is granted to the former one. Otherwise, the lock-requesting transaction is blocked by the higher priority lock-holding transaction. A high priority transaction never waits for a lower priority transaction. This condition prevents deadlocks if we assume that the real-time priority of a transaction does not change during its lifetime and that no two transactions have the same priority.

Huang *et al.* [5] developed a combined priority abort and priority inheritance protocol, called *conditional priority inheritance*, to capitalise on the advantages of both schemes. The protocol attempts to reduce the blocking times with respect to PI, and to reduce the abort rate with respect to PA. When a transaction T is blocked by a lower priority transaction T', if T' is near completion, it inherits the priority of T; otherwise, T' is aborted. The protocol assumes that the length of a transaction (i.e. the number of data items accessed by the transaction) is known in advance. The protocol has a threshold parameter h. At the time of a data conflict, if the remaining number of data items to be accessed by the lock-holding transaction is less than or equal to threshold h, then PI is applied; otherwise, PA is used.

An extension to PI is the *priority ceiling* protocol which bounds the blocking time of high priority transactions to no more than one transaction execution time [9, 10]. It eliminates the deadlock problem from PI and attempts to reduce the blocking delays of high priority transactions. The 'priority ceiling' of a data item is defined as the priority of the highest priority transaction that may have a lock on that item. In order to obtain a lock on a data item, the protocol requires that a transaction T must have a priority strictly higher than the highest priority ceiling of data items locked by the transactions other than T. Otherwise, transaction T is blocked by the transaction which holds the lock on the data item of the highest priority ceiling.

In a more recent work, we provided a new concurrency control protocol, called *data-priority-based locking protocol*, to prove that the real-time performance provided by PA, which appears to be a good

locking protocol, can be further improved if the data access requirements of transactions are known in advance [13]. Similar to the priority ceiling protocol, the proposed protocol is based on prioritising data items; each data item carries a priority equal to the highest priority of all transactions currently in the system that include the data item in their access lists. In order to obtain a lock on a data item D, the priority of a transaction T must be equal to the priority of D. Otherwise (if the priority of T is less than that of D), transaction T is blocked by the transaction that is responsible for the priority of D.

Some variants of the optimistic concurrency control protocol have also been developed and evaluated for time-critical applications. Haritsa *et al.* developed an optimistic protocol, called *WAIT-50*, which allows for the use of priorities to improve decision making in resolving conflicts [3]. The protocol uses a '50 per cent' rule as follows: the validation check for a committing transaction is performed against the other active transactions. If a conflict exists and half or more of the transactions conflicting with the committing transaction are of higher priority, the transaction is made to wait for the high priority transactions to complete; otherwise, it is allowed to commit while the conflicting transactions are aborted.

The priority inversion problem that was defined for locking protocols can also exist in a system that maintains data consistency through use of a time-stamp-ordering concurrency control protocol. It is possible that a high priority transaction T is aborted at its access to a data item, since a lower priority transaction T', carrying a time-stamp higher than the time-stamp of T, has accessed that data item previously. We proposed a time-cognisant concurrency control protocol that attempts to control the priority inversion problem of the time-stamp-ordering scheme [13]. The new protocol categorises the transactions into time-stamp groups based on their arrival times. The time is divided into intervals of a certain length and the transactions that arrive at the system within the same interval are placed in the same time-stamp group. The basic idea is to schedule the transactions of the same time-stamp group based on their real-time priorities. Each transaction is assigned a two-level time-stamp made up of a group time-stamp and a real-time time-stamp. The transactions within the same time-stamp group are assigned the same group time-stamp which is the arrival time of the first transaction in that group. Real-time time-stamps of transactions within the same group are determined based on the real-time priorities of transactions. The transaction with the highest priority obtains the largest real-time time-stamp, so it cannot be aborted by any other transaction in the same group in the case of a data access conflict.

An extensive exploration of the issues in concurrency control and other time-cognisant scheduling concepts, such as buffer management, I/O scheduling, commitment, etc, is provided in [14].

4. Discussion

There is a growing interest in applying the principles and techniques of real-time scheduling to transaction management in DBMSs. Today, many application areas supported by a DBMS (e.g. information retrieval systems, airline reservation systems, stock market, banking, etc) are characterised by the requirement of timely access to the underlying database. In addition to maintaining database consistency, an essential scheduling goal in those applications is to satisfy timing constraints associated with transactions accessing the database.

In this paper, we introduced the research efforts in time-constrained transaction scheduling. We briefly reviewed the basic methods used in mapping timing constraints of transactions into priorities and the priority-based concurrency control techniques proposed to control the interaction among concurrently executing transactions. We believe that, although some progress has been made towards the development of time-cognisant concurrency control protocols, more general empirical work needs to be performed to demonstrate the practicality of those protocols.

As a final remark, main memory databases are expected to be economically feasible in the near future, due to falling memory prices and growing memory sizes [11]. With memory-resident databases, transaction execution time will become more predictable, and thus the adaptation of real-time scheduling techniques to DBMSs will become much easier. Trends in the technology of main memory suggest that research for the time-critical database management should be focused more on main memory database systems.

Acknowledgement

This work was supported by TÜBITAK under grant number EEEAG–137.

References

[1] R. Abbott and H. Garcia-Molina, Scheduling real-time transactions: a performance evaluation, *ACM Transactions on Database Systems* 17 (1992) 513–560.

[2] J. R. Haritsa, M. J. Carey and M. Livny, *Value-Based Scheduling in Real-Time Database Systems* (TR-1204) (Department of Computer Science, University of Wisconsin-Madison, 1991).

[3] J. R. Haritsa, M. J. Carey and M. Livny, Data access scheduling in firm real-time database systems, *Real-Time Systems* 4 (1992) 203–241.

[4] J. Huang, J. A. Stankovic, D. Towsley and K. Ramamritham, Experimental evaluation of real-time transaction processing. In: *Proceedings of the 10th Real-Time Systems Symposium* (1989) pp. 144–153.

[5] J. Huang, J. A. Stankovic, K. Ramamritham, D. Towsley and B. Purimetla, Priority inheritance in soft real-time databases, *Real-Time Systems* 4 (1992) 243–268.

[6] E. D. Jensen, C. D. Locke and H. Tokuda, A time-driven scheduling model for real-time operating systems. In: *Proceedings of the 6th Real-Time Systems Symposium* (1985) pp. 112–122.

[7] K. Ramamritham, Real-time databases, *International Journal of Distributed and Parallel Databases* 1 (1993) 199–216.

[8] L. Sha, R. Rajkumar and J. Lehoczky, Concurrency control for distributed real-time databases, *ACM SIGMOD Record* 17 (1988) 82–98.

[9] L. Sha, R. Rajkumar and J. Lehoczky, Priority inheritance protocols: an approach to real-time synchronization, *IEEE Transaction on Computers* 39 (1990) 1175–1185.

[10] L. Sha, R. Rajkumar, S. H. Son and C. H. Chang, A real-time locking protocol, *IEEE Transactions on Computers* 40 (1991) 793–800.

[11] M. Singhal, Issues and approaches to design of real-time database systems, *ACM SIGMOD Record* 17 (1988) 19–33.

[12] J. A. Stankovic and W. Zhao, On real-time transactions, *ACM SIGMOD Record* 17 (1988) 4–18.

[13] Ö. Ulusoy and G. G. Belford, Real-time transaction scheduling in database systems, *Information Systems* 18 (1993) 559–580.

[14] Ö. Ulusoy, *Research Issues in Real-Time Database Systems* (BU-CEIS-94–32) (Department of Computer Engineering and Information Science, Bilkent University, 1994).