***ELECTRONIC* WORKSHOPS IN COMPUTING**

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

Two Phase Locking-Based Algorithm with Partial Abort for Firm Deadline Real-Time Database Systems

P. Krzyzagórski and T. Morzy

Published in collaboration with the
British Computer Society

Springer

ISBN: 3-540-76227-2

Two Phase Locking-Based Algorithm with Partial Abort for Firm Deadline Real-Time Database Systems

Piotr Krzyżagórski*
Institute of Computing Science, Poznań University of Technology
Poznań, Poland

Tadeusz Morzy
Institute of Computing Science, Poznań University of Technology
Poznań, Poland

Abstract

Locking-based concurrency control algorithms are often proposed as an alternative to optimistic algorithms for real-time database systems. Many of them suffer from the costs of transaction restarts, usually having influence on the system performance. We propose a new method - called partial abort - aimed to minimize the costs of restarts by saving the part of previously performed work, which is independent of conflicting data. The modification of well known 2PL-HP algorithm is presented. Through simulation experiments we evaluate the performance of the modified version, and show that the improvement is significant.

1 Introduction

Real-time systems (RTSs) represent an important class of computer systems which is characterized by existence of timing requirements concerning task execution times. In addition, RTSs process data items which values and validity change in time. The growing importance of RTSs' applications is accompanied with both the increase of data volumes processed by the systems and the requirements concerning durability, security and consistency of the processed data. So far, such requirements have been typical for traditional database systems (DBSs). Now, there is a requirement to create systems which could integrate the features of both RTSs and DBSs. Such systems are called real-time database systems (RTDBSs). A real-time database system (RTDBS) is a database system designed to process transactions having timing constraints associated with them, accessing data items which values and validity change in time, supporting data descriptions (which help avoiding data redundancy), data correctness and integrity maintenance methods, efficient data access and management techniques, and guarantee the correct execution of transactions in spite of concurrency and failures [24].

The applications of RTDBSs concern Computer Integrated Manufacturing (CIM), factory automation and robotics, workflow systems, military command and control, aerospace systems, telecommunication and computer network management, telephone directory service systems, medical monitoring, air traffic control, traffic control, stock arbitrage systems, multimedia systems [4, 6, 8, 16, 28, 29]. All the above applications make substantial use of databases, but also require support for timeliness, which is not provided by traditional databases.

To realize the nature of real-time constraints let us consider an example of an RTDBS which supports air traffic controllers. The system collects data concerning aircraft flight parameters (flight direction, position, altitude, velocity) and weather conditions over a particular area (sunshine, wind, rain, temperature), etc. The data are collected from different sensors and radar systems, and periodically written into the database. Each data item is characterized by a value, a timestamp, and a validity interval. The data items correspond to the

*This work was partially supported by the State Committee for Scientific Research (KBN), Poland, under grant KBN 8 T11C 052 08.

external objects of the environment, so their values are up-to-date for a certain amount of time - the quicker the value changes the shorter is the validity interval. The data items are then analyzed and used to provide the air traffic controllers with necessary information. As we can see, the timing requirements have to be imposed on the processing times to guarantee the availability of the information to the controllers - which allows them to make appropriate controlling decisions in time. The other requirement concerns the maximum timestamp difference between any two data items in a set to guarantee that any data derived from these items may be considered reliable.

The wider application of RTDBSs in practice depends on solving several important problems concerning the performance requirements of such systems, i.e. meeting transaction deadlines. The performance can be increased by allowing concurrent execution of transactions having access to the same data items. The activity of coordinating concurrent access of transactions to shared data in DBSs is called a *concurrency control*. Concurrency control algorithms for RTDBSs are in general some extensions or combinations of traditional concurrency control techniques, i.e. two phase locking (2PL), optimistic concurrency control (OCC) or time-stamp ordering (TO), which guarantee a serialization order among conflicting transactions.

2PL is the most widely used concurrency control algorithm in traditional DBSs. The disadvantages of the basic 2PL algorithm for real-time applications are: priority inversion, possibility of deadlocks, and unpredictable blocking times. In the literature several extensions of 2PL algorithm have been proposed, e.g. 2PL High Priority (2PL-HP) [1], 2PL Conditional Restart, 2PL Wait Promote (2PL-WP) [2], 2PL Priority Ceiling (2PL-PC) [25, 27].

The common problem faced in RTDBSs concerns restarts caused by concurrency control protocols trying to resolve conflicts between transactions. These restarts significantly degrade the system performance as the previously performed work is lost. In the paper, we propose a new approach - a partial abort, which is aimed to minimize the costs of transaction restarts, by saving the part of previously performed work which is independent of the conflicting data. The approach is applied here to 2PL-HP algorithm but can be easily incorporated to other locking based algorithms. The performance of the proposed algorithm, called 2PL-HP/PaA, has been tested in a model of disk-resident multiprocessor RTDBS. The results show that applying partial abort approach to the original locking-based algorithm significantly improves its performance.

The organization of the remaining parts of the paper is as follows. In Section 2 real-time database systems are briefly described. Locking-based algorithms for firm deadline RTDBSs are presented in Section 3. The details of the algorithm with partial abort and the discussion of its properties are presented in Section 4. Section 5 describes our RTDBS simulation model and presents experimental results. Finally, in Section 6 we summarize the main conclusions of the study.

2 Real-Time Database Systems

A real-time database is a collection of data items. Some of the items are temporal which means that their values are up-to date only for a certain amount of time. For such data items a temporal consistency criterion is defined. The criterion has two components: absolute and relative. Absolute consistency specifies the length of the time interval during which the current value of the data item is considered to be valid. The relative consistency defines the maximum timestamp difference between any two data items in a set. In addition, the traditional database consistency has also to be maintained, which means that the database is always mapped from one consistent state to the another consistent state.

The data items can be accessed by invoking transactions. A transaction is a sequence of read and write operations, executed atomically on data items, followed by the commit or abort operation. A transaction T processed in an RTDBS may be characterized by the following attributes:

A_T - arrival time,
D_T - deadline,
E_T - execution time,
S_T - slack time,
V_T - value function,

C_T - criticalness,

P_T - priority.

The arrival time of a transaction is the moment of time in which the transaction enters the system. The deadline of a transaction indicates the requirement to complete the transaction before a certain time in the future. The above parameters are the timing constraints of the transaction.

The execution time of a transaction is a runtime estimate which approximates the amount of computation required by the transaction. This parameter is rather rarely known. The data requirements of a transaction (its read set and write set) and resource requirements (expected CPU usage and number of I/O operations) are data dependent. Hence, they are usually known only during its execution.

The slack time is defined as the maximum length of time the transaction can be delayed and still satisfy its deadline. These timing attributes are related by:

$D_T = A_T + E_T + S_T$.

The value function of a transaction measures how valuable is to complete the transaction at some point in time after its arrival. The criticalness (criticality) of a transaction is the value of the value function at the critical time, i.e. the deadline. It measures how critical to the system is that the transaction meets its deadline [15, 21].

The priority is assigned to each transaction to indicate the service precedence among transactions to the system hardware resources (processors, memory buffers, disks) and logical resources (data items). A number of *priority assignment policies* have been proposed in the literature, e.g. *earliest deadline first*, *highest value first*, *highest value per unit computation time first*, *longest executed transaction first* [24]. Most popular in RTDBSs is *earliest deadline first* which so far has been widely used in RTSs. It was proved [20] that *earliest deadline first* policy is optimal for periodic task sets with hard deadlines executing on a single processor. However, there are no optimal on-line scheduling policies for arbitrary task sets executing on multiprocessor systems [22].

There are some other transaction parameters that sometimes may also be known and influence transaction processing in RTDBSs. These parameters include: arrival pattern (periodic, sporadic or aperiodic), data access pattern (random, i.e. unknown, or predefined: write-only, read-only, update), accessed objects properties (continuous or discrete), resource requirements (CPU and I/O time knowledge) [23].

Two application environments may be distinguished in traditional RTSs, i.e. hard deadline RTSs and soft/firm deadline RTSs. In hard deadline systems the violation of timing constraints is equivalent to a catastrophe. In soft deadline systems missing deadline leads only to degraded performance. The term firm deadline RTS is used to distinguish specific soft deadline RTS that derive no benefit from completing transactions after their deadlines. With the respect to the consequences of missing a deadline two categories of transactions can be distinguished: hard deadline and soft/firm deadline transactions.

Hard deadline transactions are those, for which missing deadlines may be equivalent to a catastrophe. As shown in Figure 1(a), they deliver constant values if completed before their deadlines, but large negative values if their deadlines are missed. Transactions have soft deadlines if they are still valuable after missing deadline (although the value is reduced) and may run to completion (Figure 1(b)). Firm deadline transactions have no value if the deadline is missed (Figure 1(c)). According to the above categorization there are hard deadline and soft/firm deadline RTDBSs. Hard deadline RTDBSs are typical for life-critical applications. Financial and manufacturing applications are usually soft or firm deadline RTDBSs.

The performance of a RTDBS depends on the system architecture, the speed of its components - hardware resources, and the access scheduling policy both to hardware and logical resources. In hard deadline RTDBSs all transactions have to meet their timing constraints. This requirement is difficult to satisfy as there exist many sources of unpredictability concerning transaction parameters. The only solution is to guarantee that various transaction parameters, which are important for scheduling decisions, are known *a priori*. It mainly concerns transaction arrival and data access patterns, and resource requirements. This knowledge is necessary to estimate the worst-case execution time of transactions and use real-time task scheduling techniques to guarantee timely transaction execution [8, 24].

The timing requirements for soft/firm deadline RTDBSs are less stronger than for hard deadline systems. The performance goals for firm deadline RTDBSs are to minimize the percentage of transactions that do not meet their deadlines, and for soft deadline systems to minimize total tardiness, mean tardiness and mean

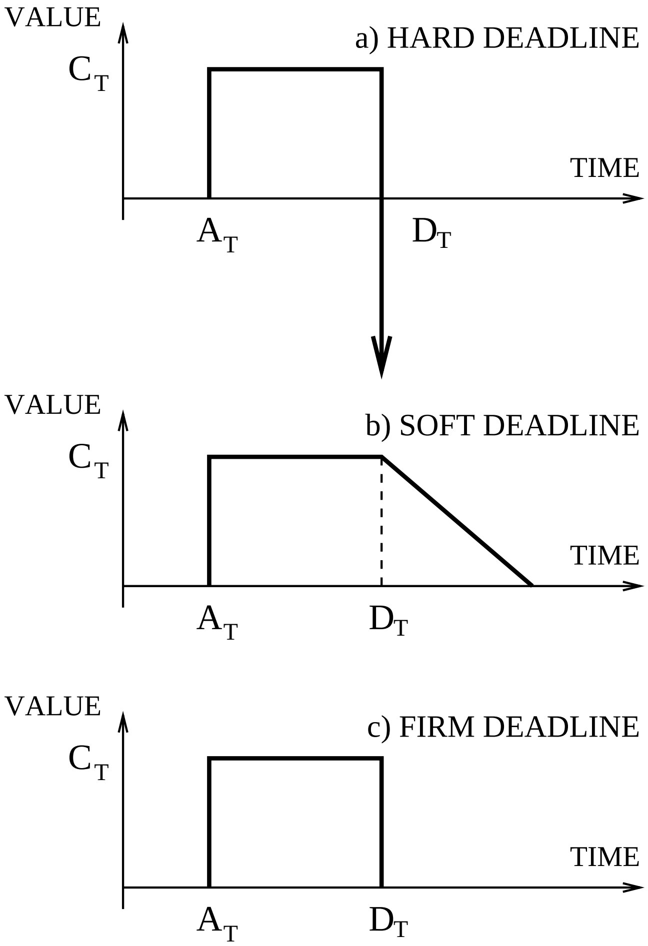

Figure 1: Value Function Model for Real-Time Transactions

weighted tardiness of transactions. These goals can be achieved by adopting time cognizant priority assignment and resource scheduling policies.

Scheduling algorithms used to coordinate multiple access of transactions to hardware resources are generally some extensions of traditional, i.e. non-real-time algorithms. The extensions of the algorithms consist in taking transaction priorities into account, which are aimed to optimize selected performance criteria. These algorithms are described e.g. in [3, 10, 12, 13]. However, we would like to emphasize that scheduling access to logical resources has greater influence on the overall system performance. There are several reasons. Data items have identity, so they cannot be substituted. A multiple access to the same data items may be performed concurrently by different transactions, and if according to the serializability criterion, a conflict among these transactions cannot be reconciled, some of them have to be aborted and restarted loosing the service already received.

The activity of coordinating concurrent access of transactions to shared data in DBSs is referred to as a *concurrency control*. Concurrency control protocols resolve data access conflicts in a manner that induces a serialization order among the conflicting transactions. We have already distinguished two application environments, i.e. hard deadline RTDBSs and soft/firm deadline RTDBSs. In the former, all timing constraints have to be met. In this case, real-time task scheduling techniques are used.

Several concurrency control protocols have been proposed for soft/firm deadline RTDBSs. However, most of them are addressed to firm deadline RTDBSs, because if transactions have different values it is not easy to make a tradeoff between their values and deadlines. For traditional DBSs three main concurrency control techniques have been proposed, i.e. two-phase locking, optimistic concurrency control, and timestamp ordering [7, 9]. The manner in which conflicts are resolved is called a *conflict resolution policy*. Concurrency control protocols for firm deadline RTDBSs additionally use priority information in the resolution of data conflicts, and try to resolve these conflicts in favor of the higher priority transaction. The timestamp ordering technique is not specially suitable for RTDBSs as transaction timestamps do not correspond to their deadlines or priorities. The proposed algorithms have been initially based on two-phase locking because this approach is well studied and widely used in traditional DBSs. Locking based algorithms for firm deadline RTDBSs are briefly described in the following section.

3 Locking-Based Algorithms

In traditional (non real-time) 2PL algorithm a transaction must obtain read (write) locks before executing read (write) operations. As a write lock cannot be shared with any other locks some transactions may by blocked until the conflicting locks are released. When applying this algorithm to RTDBSs a phenomenon called *priority inversion*, i.e. blocking a high priority transaction by a low priority one, may occur. The other disadvantages of 2PL for real-time applications are a possibility of deadlocks, and unpredictable blocking times. Several extensions of the 2PL algorithm have been proposed in the literature.

The following notation is used in the paper to describe algorithms. Let T_H be a transaction that holds a lock on a data object O, T_R a transaction that is requesting a lock on this object, $p(T)$ the current priority value of a transaction T.

The 2PL High Priority (2PL-HP) [1] algorithm is shown below:

IF $p(T_R) > p(T_H)$ **THEN**
    T_R aborts T_H;
ELSE
    T_R waits;
ENDIF

The algorithm uses priority abort mechanism to resolve conflicts between transactions. The conflict is immediately resolved by aborting the lower priority transaction(s) holding the conflicting lock. This mechanism may degrade performance of the system due to *wasted restarts* of transactions. A wasted transaction restart occurs when a transaction is aborted and restarted by a higher priority transaction that later misses its deadline and is discarded. However, this approach prevents priority inversion. If we also assume that the real-time priority of a transaction does not change, and any two transactions have not the same priority then 2PL-HP algorithm is deadlock free.

In 2PL Wait Promote (2PL-WP) [2] the priority inheritance mechanism [26] is applied to solve the problem of priority inversion. When a priority inversion occurs, i.e. the higher priority transaction requesting a lock is blocked by a lower priority one holding the lock, then the lock holder's priority is promoted to be equal to the requester priority. This technique allows the lower priority transaction to finish and release its resources quickly. The main drawback of this mechanism is that the duration of blocking transactions is uncertain; as the property of priority inheritance is transitive, under high data contention most of the transactions in the system are executed having the same priority. Hence, the performance of the algorithm degrades rapidly and it behaves like traditional 2PL.

The idea behind 2PL Conditional Restart algorithm is that aborting lower priority transactions holding conflicting locks should be avoided unless it is necessary. If it is possible to estimate the remaining running time of the lower priority transaction T_H holding a lock, and if this time is shorter than the slack time of the lock requester T_R, then T_H may be allowed to finish without missing T_R's deadline. Of course to properly estimate the running time and the slack time, some additional knowledge concerning transaction data requirements is necessary.

2PL Priority Ceiling (2PL-PC) [25, 27] algorithm, which also uses the priority inheritance mechanism, bounds the blocking time of the high priority transaction to one transaction execution time. The algorithm requires prior knowledge of transactions' data access patterns to totally order all active transactions. The above requirement is rather inapplicable to many RTDBSs as transaction read sets and write sets are data dependent.

There are several algorithms based on the ordered sharing rule between locks [5], which is a relationship between locks to eliminate blocking of read and write operations at the expense of a possible delay at transaction commitment. The variants of the basic 2PL Ordered Sharing (2PL-OS) algorithm, i.e.: 2PL Avoid Cascading Aborts/Ordered Sharing (ACA 2PL-OS), 2PL Ordered Sharing/Before Image (2PL-OS/BI) and Strict 2PL-OS/BI (ST 2PL-OS/BI) [6] differ in extra features concerning recoverability or avoiding cascading aborts.

It has been already mentioned that an important problem encountered in RTDBSs concerns restarts. We would like to propose a solution which is aimed to increase the system performance by reducing the cost of restarts. The approach can be easily incorporated into existing locking based algorithms. It will be described in the following sections of the paper.

4 Locking-Based Concurrency Control Algorithm with Partial Abort

4.1 Basic Concepts

To minimize the cost of transaction restarts we propose to save the part of previously performed work which is independent of the conflicting data. Each transaction T is an ordered sequence of read and write operations followed by the commit or abort operation. These operations are denoted by $op_1(T)...op_n(T)$. Let's consider a conflict between two transactions T_a and T_b concerning operations $op_k(T_a)$ and $op_j(T_b)$, respectively. In addition, let's assume that according to the conflict resolution policy T_a is restarted and the ordered sequence of just performed operations of T_a, i.e. $op_1(T_a)...op_{k-1}(T_a)op_k(T_a)op_{k+1}(T_a)...op_t(T_a)$, is lost. However, in the above sequence a subsequence of operations, i.e. $op_1(T_a)...op_{k-1}(T_a)$, is independent of the conflicting operation $op_k(T_a)$. Of course, in a particular case some of the operations $op_{k+1}(T_a)...op_t(T_a)$ are also independent of the conflicting operation $op_k(T_a)$. However, a detailed knowledge concerning semantics of these operations is necessary to prove this. Here, we assume that such information is unavailable, so we restrict the set of conflict-independent operations to the sequence $op_1(T_a)...op_{k-1}(T_a)$.

The idea of the approach proposed here is, that instead of detecting a conflict, and - as a consequence - aborting T_a, the information specifying the conflicting operation $op_k(T_a)$ is sent to T_a. Transaction T_a after obtaining the information resumes execution starting from the operation $op_k(T_a)$ instead of $op_1(T_a)$. It means that the previously performed sequence of operations $op_1(T_a)...op_{k-1}(T_a)$ is not lost.

The approach may be applied to different concurrency control algorithms - both optimistic as well as locking based. In the paper, we restrict our attention to modifications of locking-based concurrency control algorithms.

4.2 2PL High Priority Algorithm with Partial Abort

The approach may be applied to different 2PL algorithms. The algorithm given below, called 2PL High Priority with Partial Abort (2PL-HP/PaA), is an example of applying the approach to 2PL-HP algorithm. 2PL-HP algorithm was chosen because it often represents locking-based concurrency control algorithms in the previous studies [16, 19]. The 2PL-HP/PaA algorithm is shown below:

IF $p(T_R) > p(T_H)$ **THEN**
  T_R partially aborts T_H due to data conflict
    concerning data element x;
ELSE
  T_R waits;
ENDIF

In the algorithm, the procedure which detects a conflict between transaction T_H (holding the lock) and transaction T_R (requesting the lock) also delivers a detailed information indicating conflicting data item. It makes possible to partially abort T_H.

4.3 Example

To illustrate the 2PL-HP/PaA algorithm let us consider a simple example in which the behavior of the algorithm is compared to original 2PL-HP algorithm. To simplify the diagrams assume a single-processor memory-resident RTDBS.

Given are two transactions *A* and *B* with arrival time *a*, deadline *d* and data requests *rd* (read) and *wr* (written) shown in Table 1.

Transaction	a	d	rd	wr
A	0	300	x, y, z	y
B	30	200	y	y

Table 1: Transaction Parameters

The priority assignment policy is *earliest deadline first*, so $p(A) < p(B)$. The processor scheduling procedure always selects the higher priority transaction for execution. The schedules produced by both algorithms are shown in Figure 2.

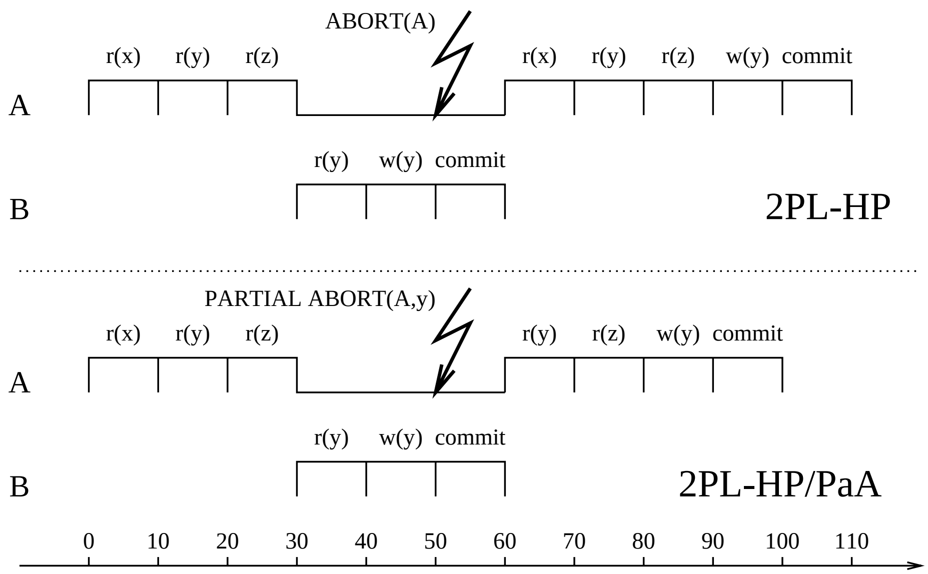

Figure 2: Example Transaction Schedules

Transaction *A* is the only job in the system at time 0. It gains the processor and reads data objects *x, y, z*. *B* arrives at time 30. Since *B* has an earlier deadline than *A*, *B* preempts *A* and begins to execute. It reads and writes data object *y*. There is a conflict between transactions concerning data element *y*. In the 2PL-HP transaction A is aborted at time 50 and is restarted. B commits at time 60, then A gains the processor and starts from very beginning. It finishes at time 110.

According to our algorithm, transaction B specifies the conflicting data element, i.e. element *y*. Transaction A is partially aborted. It resumes execution at time 60 starting with reading data item *y*. Previously read data item *x* is not reread. Due to this, in the algorithm 2PL-HP/PaA transaction A finishes earlier - at time 100.

5 Simulation Experiments and Results

The performance of 2PL-HP/PaA concurrency control algorithm was tested in a model of a disk-resident multiprocessor real-time database system. A database is modeled as a collection of data pages being uniformly distributed across all the disks. The *databaseSize* parameter defines its size in pages.

The source generates a Poisson stream of transactions at a mean rate specified by the $ArrivalRate$ parameter. A transaction consists of elementary calls to the database (read page, write page operations), generated once upon the transaction's arrival. Number of pages read by a transaction is uniformly distributed between *minTransactionSize* and *maxTransactionSize* pages. For each transactions a page that is read is updated with probability *writeProbability*. The same access pattern is also used when the transaction is restarted. The deadline assigned to each transaction is computed from the formula proposed in [17]:

DA1: $D_T = A_T + SF \cdot E_T$

D_T, A_T and E_T are the deadline, arrival time and execution time of transaction T, SF is a slack factor.

$$E_T = numReads * (pageCPU + pageDisk) + \\ + numWrites * pageCPU$$

where: $numReads$ - number of read pages, $numWrites$ number of written pages, $pageCPU$ - CPU cost for processing a page, $pageDisk$ - disk cost for processing a page.

A transaction which has not completed by its deadline is aborted and discarded from the system (terminated).

Tables 2 and 3 summarize the workload and system parameters, respectively. Their values are the same as in [16]. The $ArrivalRate$ changes from 10 to 100 transactions per second.

Workload Parameter	Value
minTransactionSize	8 pages
maxTransactionSize	24 pages
writeProbability	0.25
Deadline Assignment	DA1

Table 2: Simulation Workload Parameters

System Parameter	Value
Database Size	1000 pages
numCPUs	10
numDisks	20
pageCPU	10ms
pageDisk	20ms

Table 3: Simulation System Parameters

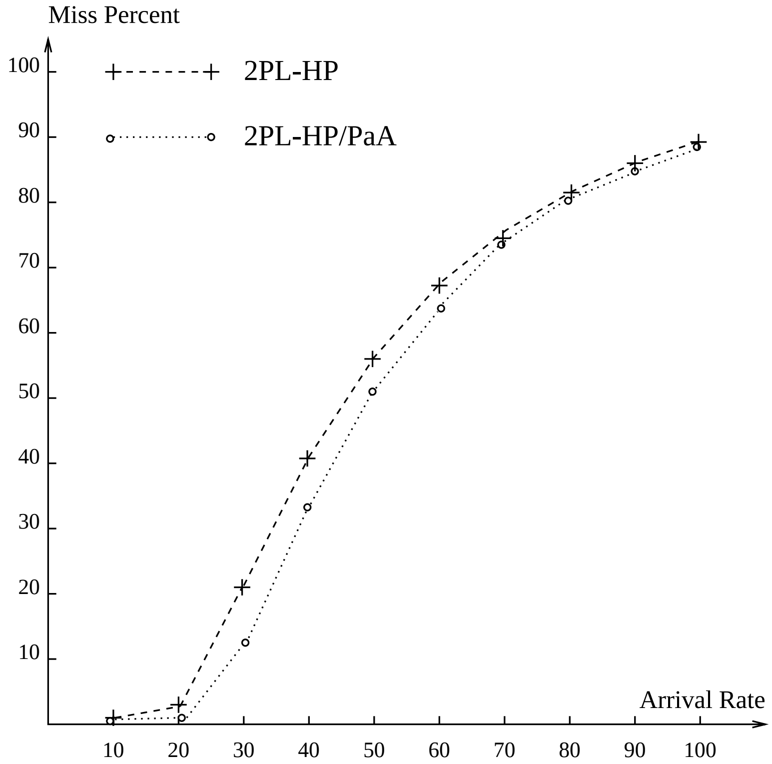

Figure 3: Sensitivity of Miss Percent to Mean Arrival Rate

The basic performance metric in the experiment is $MissPercent$ which is computed as:

$$MissPercent = \frac{number\ of\ tardy\ transactions}{number\ of\ transactions\ arrived} \cdot 100.$$

We also measure the average progress made by transactions before they are aborted or partially aborted due to data conflicts, and the average part which is saved according to the partial abort policy. These results are measured as the fraction of the total execution time (including disk write operations after commit). In the experiments, 2PL-HP algorithm is compared with its modified version, 2PL-HP/PaA.

Figure 3 shows the sensitivity of Miss Percent to mean arrival rate. It indicates that 2PL-HP/PaA always outperforms 2PL-HP - specially when the load is not very heavy. In this range of workload, the significant part of previously performed work is saved by partial abort method.

Of course, the part of previously performed work which can be saved is sensitive to the number of conflicts and the resource utilization. As shown in Figure 4, when a transaction is aborted it is possible to save about half of the work already performed. 2PL-HP/PaA transactions are aborted a bit later. which means that the system resources are better utilized, and the system throughput is higher. Of course, when the load is very high, transactions make little progress and the number of aborts increases. Most of the transactions miss their deadlines and are discarded from the system.

The results were obtained assuming that the concurrency control overhead was negligible compared to the CPU time for page processing and therefore set to 0.0. In real systems, the overhead includes finding the conflict-independent subset of operations and storing the data necessary to resume execution.

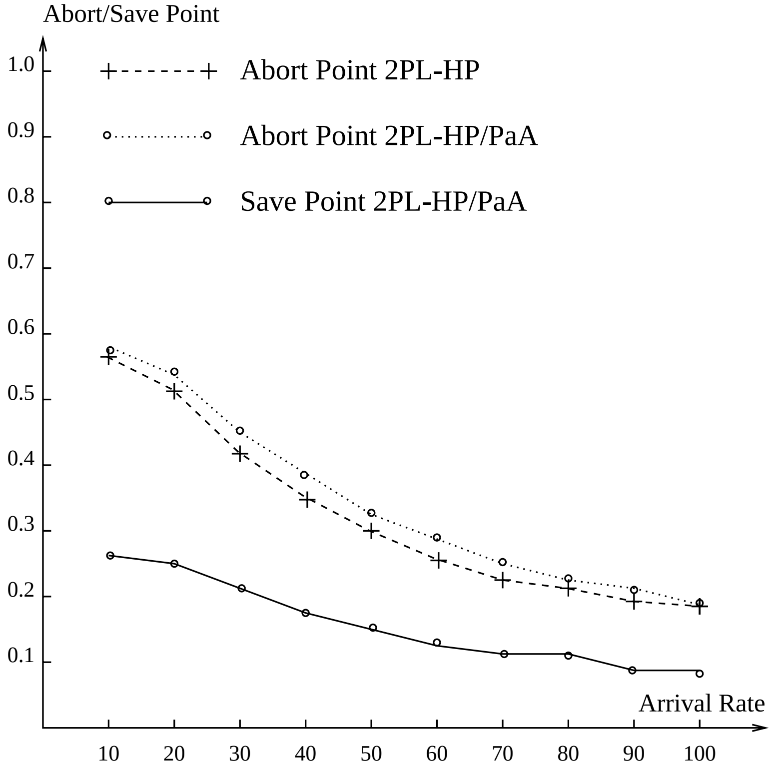

Figure 4: Sensitivity of Abort/Save Point to Mean Arrival Rate

6 Conclusions

In the paper, a new approach called partial abort is presented which is aimed to improve the system performance through decreasing the costs of transaction restarts. The transactions are restarted due to restart-based conflict resolution policy. In RTDBSs such policy is used by many concurrency control algorithms, both locking based and optimistic. Here, we proposed a modification of one well known algorithm called 2PL-HP. The results of our simulation experiments show significant improvement of the performance.

It is difficult to definitely estimate the prospective practical advantages of the method. The actual implementation costs have not been considered in our experiment. Of course, the effectiveness of the approach is closely related to its physical implementation. In the description of the method presented above, we have assumed that after the partial abort each transaction can resume execution starting from the operation just preceding the conflicting one. This is possible if before each database operation all the data necessary for resuming the operation are stored. The cost of storing the auxiliary data may be significant. A tradeoff has to be found between this cost and possible effects. In a compromise solution the data would be stored periodically. The experiences with checkpoints technique used for recovery might be very helpful [14].

Describing basic concepts of our method, we assumed that the detailed knowledge concerning transaction semantics is not available. So, we restricted the set of conflict independent operations. However, we hope to further improve the performance of concurrency control algorithms using knowledge concerning data semantics [9, 11, 30].

References

[1] Abbott R, Garcia-Molina H. Scheduling Real-Time Transactions: a Performance Evaluation. Proceedings of the 14th VLDB Conference, Los Angeles, California, 1988.

[2] Abbott R, Garcia-Molina H. Scheduling Real-Time Transactions with Disk Resident Data. Proceedings of the 15th VLDB Conference, Amsterdam, 1989, pp 385-395.

[3] Abbott R, Garcia-Molina H. Scheduling I/O Requests with Deadlines: a Performance Evaluation. Proceedings of the Real-Time Systems Symposium,1990.

[4] Abbott R, Garcia-Molina H. Scheduling Real-Time Transactions: a Performance Evaluation. ACM Transactions on Database Systems, vol. 17, no. 3, September 1992, pp 513-560.

[5] Agrawal D, El Abbadi A. Locks with Constraint Sharing. Proceedings of the 9th ACM Symposium on Principles of Database Systems, Nashville, TN, 1990.

[6] Agrawal D, El Abbadi A, Jeffers R, Lin L. Ordered Shared Locks for Real-Time Databases. VLDB Journal 1995; 4:87-126.

[7] Bernstain PA, Hadzilacos V, Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading MA, 1987.

[8] Buchmann AP, McCarthy DR, Hsu M, Dayal U. Time-Critical Database Scheduling: A Framework For Integrating Real-Time Scheduling and Concurrency Control. Data Engineering Conference, February 1989.

[9] Cellary W, Gelenbe E, Morzy T. Concurrency Control in Distributed Database Systems. North Holland Pub. Co., 1988.

[10] Chen S, Kurose JF, Stankovic JA, Towsley D. Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems. Technical Report UM-CS-90-077, Department of Computer & Information Science, University of Massachusetts, Amherst, 1990.

[11] Di Pippo LBC, Wolfe VF. Object-Based Semantic Real-Time Concurrency Control. Proc. 14th Real-Time Systems Symp., 1993.

[12] Effelsberg W, Haerder T. Principles of Database Buffer Management. ACM Transactions on Database Systems 1984; 9(4).

[13] Graham MH. Issues in Real-Time Data Management. The Journal of Real-Time Systems 1992; 4:185-202.

[14] Gray R, Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.

[15] Jensen ED, Locke CD, Tokuda H. A Time-Driven Scheduling Model for Real-Time Operating Systems. Proceedings of the 6th IEEE Real-Time System Symposium, 1985.

[16] Haritsa JR. Transaction Scheduling in Firm Real-Time Database Systems. Computer Sciences Technical Report #1036, University of Wisconsin-Madison, 1991.

[17] Haritsa JR, Carey MJ, Livny M. Data Access Scheduling in Firm Real-Time Database Systems. The Journal of Real-Time Systems 1992; 4:203-241.

[18] Lin Y, Son SH. Concurrency Control in Real-Time Database Systems by Dynamic Adjustment of Serialization Order. Proceedings of the 11th IEEE Real-Time Systems Symposium, Orlando, Florida, 1990.

[19] Lee J, Son SH. Performance of Concurrency Control Algorithms for Real-Time Database Systems. In: Kumar V. (ed) Performance of Concurrency Control Mechanisms in Centralized Database Systems, Prentice Hall, 1995.

[20] Liu C, Layland J. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. Journal of the ACM 1973; January.

[21] Locke CD. Best-Effort Decision Making for Real-Time Scheduling. PhD thesis, Carnegie-Mellon University, 1986.

[22] Mok A, Dertouzos M. Multi-processor Scheduling in Hard Real-Time Environment. Proc of the 7th Texas Conf. On Computing Systems, October 1978.

[23] Özsoyoğlu G, Sondgrass RT. Temporal and Real-Time Databases: A Survey. IEEE Transactions on Knowledge and Data Engineering 1995; vol. 7, 4:513-532.

[24] Ramamritham K. Real-Time Databases. Distributed and Parallel Databases 1993; vol. 1, 2:199-226.

[25] Sha L, Rajkumar R, Lehoczky JP. Concurrency Control for Distributed Real-Time Databases. ACM SIGMOD Record 1988; vol. 17, 1:82-98, .

[26] Sha L, Rajkumar R, Lehoczky JP. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers 1990, vol. 39, 9:1175-1185.

[27] Sha L, Rajkumar R, Son SH, Chang, C-H. A Real-Time Locking Protocol. IEEE Transactions on Computers 1991, vol. 40, 7:793-800.

[28] Son SH. Real-Time Database Systems: A New Challenge. IEEE Data Engineering 1990; vol. 13, 4:39-43.

[29] Ulusoy O. Research issues in Real-Time Database Systems. Technical Report BU-CEIS-9432, Bilkent University, Ankara, 1994.

[30] Weihl W.F. Data Dependent Concurrency Control and Recovery. Proc. 2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing, Cambridge, 1983.