International Journal of Innovation and Scientific Research
ISSN 2351-8014 Vol. 19 No. 1 Nov. 2015, pp. 190-200

http://www.ijisr.issr-journals.org/

Predictive speculative concurrency control for Real-Time Database Systems

Elyes Kooli[1] and Nacéra Madani Aissaoui[2]

[1]High Institute of Technological Studies of Ksar-Hellal, Tunisia

[2]Faculty of Sciences Monastir (FSM), Tunisia

ABSTRACT: Real-Time Database Systems (RTDBSs) are designed to manage the majority of current applications which manipulate a large volume of data and have a great need of real-time computation. One of the main issues in the DBMSs is to control the access to the same data items by the transactions in incompatible mode. In RTDBSs, this problem becomes more complicated since the transaction manager must not only avoid data access conflicts, but it has also to provide mechanisms that help transactions to meet their deadlines, to maximize the transactions success ratio. In this paper, we describe The SCC protocol (Speculative Concurrency Control) which is one of the first concurrency control protocol RTDBS. It is based on the transactions duplicating transactions. However, the SCC raises some problems; we propose a new extension of this protocol to solve these problems and to increase the number of transactions meeting their deadline.

KEYWORDS: RTDBS, priority assignment, speculative concurrency control, predictive.

1 INTRODUCTION

Real-time applications differ from traditional applications by the time constraints they must comply and which are expressed in the form of deadlines and periods of validity. Some of these applications manipulate large amounts of data. Using Systems Management Database (DBMS) may be necessary to handle such data effectively.

However, the traditional DBMS do not effectively meet the needs of these applications because they do not incorporate mechanisms to take into account the time constraints [13]. This imposed new challenges for computer scientists. It is developing a new generation of DBSs: the Real-Time database system RTDBS.

A real time database system is a database system which uses real-time processing that attempts to satisfy the timing constraints associated with each incoming transaction.

Typically, a time constraint is expressed in the form of a deadline handle. This differs from traditional databases containing persistent data, mostly unaffected by time RTDBS are systems that must meet a dual objective: to maintain the consistency of the database and have mechanisms that allow transactions to meet their time constraints, often given in the form of deadlines.

To maintain consistency of the currency database, the transaction manager must prevent data access from the conflict problem. A conflict occurs when two transactions not yet validated want to access on the same data with incompatible operation mode (read-write, for example). In a Database Management System (DBMS), *Database* concurrency *conflicts* are resolved by the concurrency control protocols [2]. These protocols can be classified into two families.

The first method, called pessimistic which uses locking as the basic serialization mechanism to prevent potential conflicts: reading or writing is validated before access to the data. Instead, the second, called optimistic, based on the idea of conflicts and transaction restart [12], allow transactions to run in competition and conflict are only checked in validation phase where conflicted transactions are abandoned and restarted.

Corresponding Author: Elyes Kooli

For Real-Time DBMS (RTDBS), the problem is more complicated: the RTDBS must respect not only the integrity constraints of the database but also the individual time constraints of transactions that are expressed by assigning a deadline each transaction. Transactions are correct if they are validated before theirs deadlines [13].

The performance of a RTDBS is mainly determined by a concurrency control algorithm used for scheduling concurrent accesses of transactions to hardware and logical system resources. The scheduling of transactions is based on priority order. Given these challenges, considerable research has recently been devoted to designing concurrency control algorithms for RTDBS and to evaluating their performance [1]

This paper, we investigate a control protocol real-time competition designed specifically for RTDBS: SCC Protocol (Speculative Concurrency Control) proposed by Bestavros [3]. This protocol combines both advantages of pessimistic and optimistic concurrency control (PCC and OCC) methods because it detects conflicts as they arise (pessimistic) method but it allows transactions to run in competition (optimistic) method. SCC protocol is particularly suitable for RTDBS in the sense that it reduces the negative impact of blocking and restarting which are the main drawbacks of pessimistic and optimistic methods respectively [3]. Thus we propose significant improvements to this method.

2 THE SPECULATIVE CONCURRENCY CONTROL PROTOCOL SCC

A. NOTATION

The major notations and symbols used in this paper are as follows:

T_i transactions running in competition
T_i' Each transaction can be duplicated and shadow transaction Ti
S: Start transactions
R_x: Operation to Read x data item
W_x: Operation to Write and update x data item
x_0: Value of x data item before updating by write operation
x_1: Value of x data item after updating by write operation
C: commit
A: aborted

In the remainder of this paper, we denote Ti transactions running in competition. Each transaction can be duplicated and phantom transaction Ti is denoted Ti. The operations performed by transactions are S (Start) to start, Rx (Read x) and Wx (Write x) to read and update the data x. C and A are used when the transaction is committed or aborted.

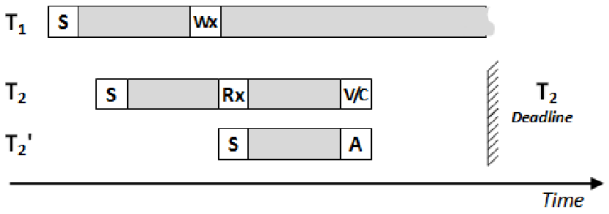
B. PRINCIPE

The Speculative Concurrency Control protocol (SCC) has been proposed to solve the problems of conflict type Read-Write or Writing- Reading [3] . At the time when a conflict of this type is detected (using locking data, for example), SCC protocol suggested to duplicate the read transaction. The new copy of the transaction is called "shadow transaction".

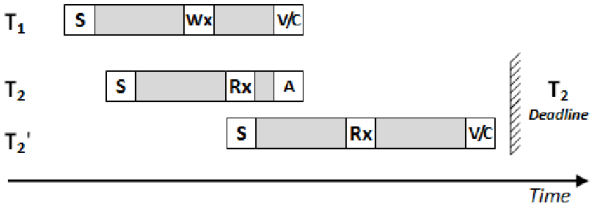
The original reader transaction continues to run optimistically whereas the shadow transaction remains blocked at the point of conflict. The original transaction and the shadow transaction are almost identical; they differ only in the data they handle. The original transaction runs with the image data before updating that by the writer transaction. However, shadow transaction is blocked with image data after updating by shadow transaction, updating the writer transaction will actually be taken into account. In this way, the shadow transaction may has been committed if the original transaction will be aborted to resolve the conflict.

To better illustrate this method, consider the following example. Assume that we have two transactions T_1 and T_2. T_2 reads item x after T_1 has updated it. We will have two possible scenarios shown in Fig. 1 and Fig. 2 depending on the time needed for transaction T2 to reach its validation phase. Each one of these scenarios corresponds to a different serialization order

- Case 1. If the reader transaction T2 reaches its validation phase before the writer transaction (T_1), then the shadow transaction (T2') is simply ignored and has to be aborted as shown in Figure 4.1 (a).
- Case 2. If the writer transaction (T_1) reaches its validation phase before T_2, then T_2 cannot continue to execute because it has accessed to incoherent item x. T_2 is aborted. The shadow transaction (T_2') is released and it runs taking the updating value of x item by T_1 (Figure 4.1 (b)). Thus, T_2 is not restarted from the beginning but simply from the point of conflict.



(a) Schedule with a developed potential conflict



(b) Schedule with a developed conflict

Fig. 1. Principe of SCC protocol

When the shadow transaction is released, it becomes a main transaction and may it be duplicated if a new conflict arises. In RTDBBS, many transactions can be executed in competition. SCC basic protocol creates a shadow transaction every appearance of a new conflict. Management of all transactions ghosts is not easy even if at a given time, only one copy of the transaction is actually executed, the others being stranded at their respective points of conflict. A class of protocols has been proposed to limit the number of copies of each transaction. This class is called SCC- kS (k-Shadow SCC) where each transaction can have at most k copies. Among this class of protocols, the most studied is the protocol SCC- 2S (Two Shadow SCC) which authorizes the creation of a single shadow transaction by primary transaction.

C. CONVENIENCE OF THE SPECULATIVE METHOD IN A REAL TIME CONTEXT

The main advantage of the protocol SCC is that it combines the advantages of pessimistic and optimistic methods while avoiding their main drawbacks

On the one hand, SCC resembles PCC in that potentially harmful conflicts are detected as early as possible, allowing a head-start for alternative schedules, and thus increasing the chances of meeting the set timing constraints, should these alternative schedules be needed (due to restart as in OCC). On the other hand, SCC resembles OCC in that it allows conflicting transactions to proceed concurrently, thus avoiding unnecessary delays (due to blocking as in PCC) that may jeopardize their timely commitment [3].

D. WRITE-WRITE CONFLICTS RESOLVING

At first, SCC protocol does not take into account the conflicts between transactions looking to update the same object data together (Write-Write conflict). It was then proposed to use the TWR method (Thomas Write Rule) to treat these conflicts [5].

TWR method is based on the timestamps of transactions using the following hypothesis: "Only the write operation of the transaction largest stamp (the youngest transaction) will be visible at the end of the execution of all transactions» [2]. TWR method is based on the timestamps of transactions using the following hypothesis: "Only the write operation of the transaction largest stamp (the youngest transaction) will be visible at the end of the execution of all transactions» [2]. Thus, the write operation of a transaction can be ignored if another younger transaction has already updated the some data. With this assumption, when a transaction T_1 wants to write about a given transaction while a younger T_2 has already written on this data, the writing of T_1 is simply aborted.

E. VALUE-COGNIZANT SCC

Speculative method seems interesting in a real time environment even if, the time constraints of transactions are overlooked on processing conflicts. A problem with SCC algorithms and other common concurrency control schemes is that

committing a transaction as soon as it finishes validating, may result in a value loss to the system. Another extension of the SCC protocol has been proposed to reflect the transactions deadlines and their criticality.

The value-cognizant SCC protocol uses deadline and criticalness information in resolving data conflicts or in making other scheduling decisions. In fact, these parameters are used to calculate a coefficient of penalty (penalty gradient) for each transaction. When a transaction finish, the penalty coefficient determines if the commit operation cannot be deferred to solve the conflict.

For example, in figure 2, committing T_1 as soon as it is validated causes T_2 to miss its deadline and a value penalty to be assessed to the system. Haritsa showed that by making a lower priority transaction wait after it is validated, the number of transactions meeting their deadlines is increased, which results in a higher value-added to the system [7].

Consider Figure 1 (b). If the deadline of T_1 is sufficiently far, it may defer its committing operation and allow time for T_2 to complete. One then obtains the situation shown in Figure 2: as the commit operation was delayed T_1, T_2 can finish and the conflict is resolved.

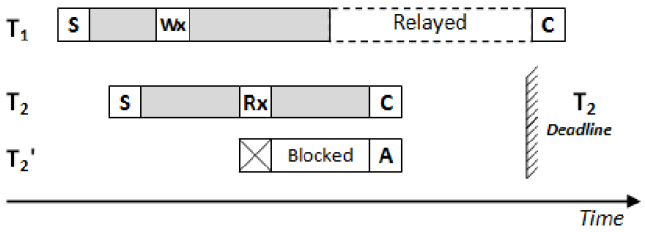


Fig. 2. A deferred commit under the Value-cognizant SCC

3 THE LIMITS OF SPECULATIVE CONCURRENCY CONTROL PROTOCOL SCC

SCC protocol has some interesting mechanisms for the management of real-time transactions. Unfortunately, despite the improvements that have been proposed, problems persist in its use. We will, in this section, illustrate the limits of the SCC protocol. Then, in the next section, we propose methods to overcome these problems.

First, consider the conflicts W-W category. They are solved by the TWR method that uses timestamps for transactions possibly ignore write operations. In a real-time context, this method is impractical. Indeed, the use of stamps is not adequate in RTDBS since it ignores the temporal constraints of the transactions. In addition, the fact that only the results of the youngest transaction are visible at the end of the execution is not applicable in a real time environment. Indeed, the results of a transaction in a RTDBS are important (reusable by other transactions) as soon as it has validated [13]. TWR method suffers from the problem of losing update that can be detrimental in a RTDBS.

There are also some problems in resolving Read-Write and Write-Read conflicts some problems. More precisely, we believe that the performance of SCC Memorandum is limited to the management of real-time transactions. For example, Figure 4.4 illustrates a situation where only one of the two transactions can meet its deadline. Indeed, the commit operation of T_1 cannot be deferred and when T_2 is released, it does not have enough time to run. If the deadlines of transactions are soft type only the QoS T_2 will decrease. However, if the transactions are strict deadlines, T_2 provides no income and therefore reduces performance RTDBS [10].

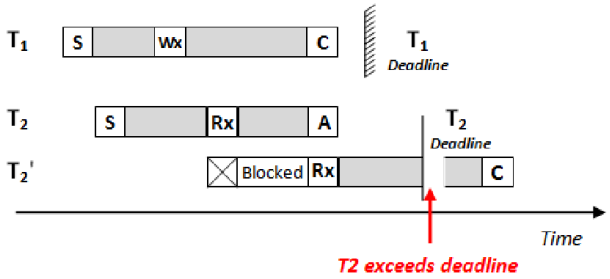


Fig. 3. Example of limitation of SCC protocol

4 PREDICTIVE SPECULATIVE CONCURRENCY CONTROL PSCC

In this section we introduce a new variant of SCC method called Predictive Speculative Concurrency Control PSCC. Our goal is *to minimize the number of transactions* that miss their *deadlines.*

A. READ-WRITE AND WRITE-READ CONFLICTS RESOLVING

At first, we proposed PSCC protocol to increase performance by optimizing protocol SCC conflict resolution type R-W and W-R; indeed we integrate a policy for scheduling transactions at the time of detection conflicts. In other words, unlike the speculative methods presented, we propose to consider transaction parameters upon detection of a conflict.

In fact, it is to choose, at the time of duplication, it is preferable that the originally transaction continues running with the image in front of the given conflict without considering the changes made by the transaction to writing in conflict, or with the image data after the conflict i.e. taking into account the modification of data provided by the writer transaction. This problem of decidability depends on two factors:

- In addition to scheduling two conflicting transactions, we should predict who will finish the first, the write transaction T_1 or the read transaction T_2? If the write transaction T_1 complete its first run, it would be preferable that the read transaction T_2 chooses the new values x1 updated by T_1. In the opposite case where the read transaction T_2 complement its first run it would be interesting that T_2 have used x0 value.
- On the one hand, the fate of the write transaction: is it can succeed scripts without missing the deadline? Indeed, if the write transaction T_1 has a great chance to validate then it is advantageous that the read transaction T_2 chooses new value x1. Conversely, if the write transaction T_1 has a low chance to validate scripts then it is better than read transaction T_2 selects the old value x0.

To resolve this decidability problem, we propose in the following sections two ways to manage the two factors in question.

B. ALLOCATION OF LUCKY POLICY

We use the following policy to calculate the hope that a transaction can do its job without exceeding its deadline. The idea is based on a metric that estimates the opportunities to success for each transaction in conflict with other transactions. The formula 1 presents the priority assignment policy.

$$Hope(T_i) \leftarrow \underbrace{\overbrace{[D(T_i) - T_A(T_i)] - E_{est}(Ti)}^{A}}_{B} * \underbrace{\frac{P(T_i)}{P_{\max}}}_{C}$$

Formula 1

$D(T_i)$: T_i Deadline
$T_A(T_i)$: T_i arrived time
$E_{est}(T_i)$: estimated execution time of the transaction T_i

$P(T_i)$: Priority T_i
P_{max} : highest priority assignment for transactions

Part A of the formula is slack time. The slack time is simply the difference between the time available and the execution time [1]. Part B expresses the time available, the available time is the time between the start time (time arrived) and expiration date. Finally Part C is a weighting that reflects the degree of resistance of the transaction conflicts with other transactions.

C. PRECEDENCE TERMINATION ORDER OF TRANSACTIONS

It is to know the order of termination of the two conflicting transactions. Just compare the remaining executions time of transactions. Indeed, when a conflict is detected, we compare the remaining execution time E_{rt} of transactions conflict.

So if $E_{rt}(T_1)<E_{rt}(T_2)$ then it is possible that T_1 complete before T_2 otherwise the termination of T_2 before T_1 is more feasible .

However, the second factor depends on the first, since you cannot properly determine termination of both transactions unsuspecting fate of the write transaction is that it could validate scripts or not.

This led us to determine a new metric called the coefficient precedence K, this metric provides information about the order of precedence of conflicting transactions which facilitates good decision making (the read transaction T_2 must use the image before or after the data at issue) , this coefficient is represented by the following formula :

$$\mathcal{K}(T_{i/j}) \leftarrow \frac{[D(T_i) - T_A(T_i)] - E_{est}(T_i)}{[D(T_i) - T_A(T_i)]} * \frac{P(T_i)}{P_{\max}} * \frac{E_{rt}(T_j)}{E_{rt}(T_i)}$$

Transaction T_j in conflict with the transaction T_i Thus, if K ($T_{1/2}$) and K ($T_{2/1}$) are the respective coefficients of T1 and T_2, then the duplication of T_2 is managed as follows :

- If K ($T_{1/2}$) < K ($T_{2/1}$), then we can assume that T_2 has a good chance to finish before T_1 (here T_1 ends before T_2, we can apply the protocol value- cognizant SCC to defer termination T_1). A write operation T_1 does not impact on the course of T_2. Optimistically, T_2 runs using the back image x0 of x conflict data while the shadow transaction T_2' is stuck with the front image x1 of x given conflict.

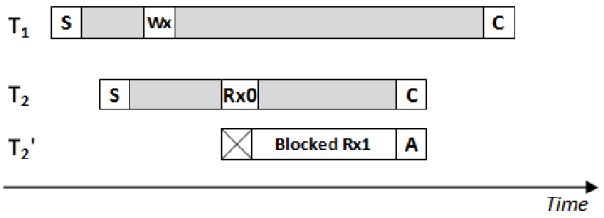


Fig. 4. T1 above T_2: T_2 uses the image before x

- If K($T_{1/2}$) > K($T_{2/1}$) , then for the same reasons as the previous case , T_1 has a good chance to finish before T_2 and any amendments will be considered by T_2. Running the SCC protocol may lead to the situation in Figure 4.4. Therefore, transaction T_2 must run after the image of x1 given x, so the ghost transaction T_2' is blocked with x0 before the given x. (Figure 4.6) we see that this time , the two transactions T_1 and T_2 can meet their deadlines

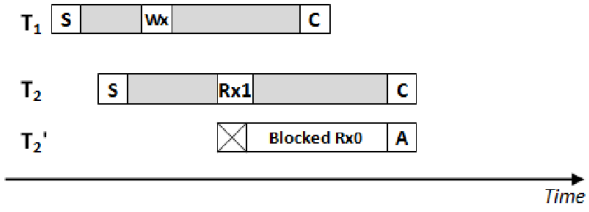


Fig. 5. T_1 precedes T_2: T_2 uses the after image of x

The PSCC protocol allows more transactions to complete before maturity by choosing a smart way the image taken by the main transaction and that the ghost transaction scheduling transactions between reading and writing based on temporal parameters of conflicting transactions. It is important to note that using the assumption that a transaction can access data not yet validated, the PSCC protocol relaxes the property transaction isolation.

D. RESOLUTION OF W-W CONFLICTS

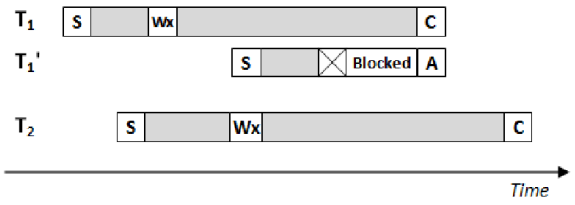
The PSCC protocol also supports conflict resolution W-W type without the use of stamps proposed by the TWR method or duplication of transactions that overload the system benefit transactions.

We adopt the basic SCC protocol for conflict resolution R-W/W-R where you create a ghost transaction for the read transaction at issue, to the point of conflict is a ghost proliferate transaction for one of the two write transactions in conflict.

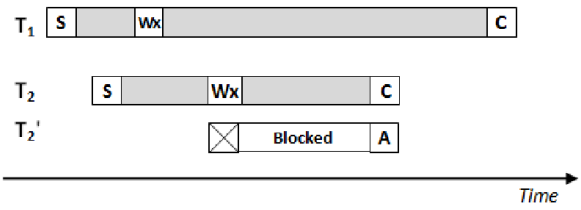
We propose to use the same predictive method we proposed for the management of type conflicts R-W and W-R for the choice of the write transaction to give him a ghost transaction. Indeed the transaction that most likely validate scripts and ended the first will be a new ghost transaction. The management of the main transaction and the shadow transaction is then identical to that of SCC protocol.

Indeed, if K ($T_{1/2}$) < K ($T_{2/1}$) then we can anticipate that T_2 has a good chance to finish before T_1, in which case it creates a shadow T_2'au point of conflict transaction which stores the point of conflict . The originally transaction T_2 is running with the front image of the data, that is to say that the update of the transaction T_1 is not counted. As against the T_2' remains blocked with the image data after this phantom transaction, that is to say, if the transaction is enabled phantom, the update of the transaction T_1 will be effectively taken into account . In this way, the ghost transaction may be released if the main transaction will be abandoned to resolve the conflict.

Figure 4.7 illustrates the two possible situations of conflict WW: when T_1 precedes T_2 (FIG. 5 (a)) and when T_2 precedes T_1 (FIG. 5 (b)).



(a) T_1 precedes T_2



(b) T_2 precedes T_1

Fig. 6. Principe of PSCC protocol W-W mode

The algorithm PSCC protocol defined above is as follows:

Algorithm: PSCC

Start:

Detection of a conflict: (Write (Ti, x), read (Tj, x))

```
1)   If K (Ti) <K (Tj) then
2)          Run (Ti)
3)          Tj ' ← duplicate (Tj, x1)
4)          Block (Tj ')
5)          Run (Tj, x0)
6)          If Tj ends before then Ti
7)                 Ignore (Tj ')
8)          otherwise
9)                 Ignore (Tj)
10)                Unlock (Tj ')
11)         end if
12)  else
13)         Run (Ti)
14)         Tj '← Duplicate (Tj, x0)
15)         Block (Tj ')
16)         Run (Tj, x1)
17)         If Tj ends after Ti then
18)                Ignore (Tj ')
19)         else
20)                Ignore (Tj)
21)                Unlock (Tj ')
22)         end if
23)  end if
```

Conflict detection: (Write (Ti, x) and Write (Tj, x))

```
1)   x0: initial value of x
2)   x1: Writing Ti
3)   If K (Ti) <K (Tj) then
4)          Run (Ti)
5)          Tj '← duplicate (Tj, x1)
6)          Block (Tj ')
7)          Run (Tj, x0)
8)          If (Tj ends before Ti) then
9)                 Ignore (Tj ')
10)         else
11)                Ignore (Tj)
12)                Unlock (Tj ')
13)         end if
14)  else
15)         Run (Tj)
16)         Ti '← Duplicate (Ti, x1)
17)         Block (Ti ')
18)         Run (Ti, x0)
19)         If Ti completes before Tj then
20)                Ignore (Ti ')
21)         else
22)                Ignore (Ti)
23)                Unlock (Ti ')
24)         end if
25)  end if
```

End.

5 PERFORMANCE EVALUATION

To evaluate the performance of PSCC method, we have developed a RTDBS Simulator of firm deadline transactions (transactions which miss their deadlines are immediately killed). The simulation model, workload parameters, and assumptions are similar to those in [4,7] to make the results compatible

A. SIMULATION MODEL

We assume a closed queuing model of a single site database system, which consists of multiple CPUs sharing the common memory and a memory-resident database. The transaction arrival rate follows a Poisson distribution and each transaction is associated with an arrival time, a deadline, and an estimated execution time.

A transaction will request a sequence of read and write operations. The scheduler uses the underlying priority assignment policy to selected the transaction with the highest priority in the ready queue for execution.

B. WORKLOAD MODEL

The workload model characterizes the transactions running in the system according to the number of pages they access and their execution time. Table 1 summarizes the key workload parameters used in our experiments.

Table I lists the workload model parameters used and their base values. The deadline of a transaction T_i is assigned as $D(T_i) = T_A(T_i) + SRatio * R_{max}$ where R_{max}, is the required resource time for the largest transaction in the workload.

Table 1. The Workload Parameters

Parameter	Meaning	Settings
DBSize	Database size in pages	1000 pages
TRANSize	Size of transactions in pages accessed	20 pages
WProb	Probability to update an accessed page	0.25
RSize(Ti)	Number of readied pages by transaction Ti reads	Randomly
WSize(Ti)	Number of written pages by transaction Ti reads	Randomly
SRatio	Slack Ratio	1.5
RTime	Average time to read a page	3 msec
WTime	Average time to update part of a page	15 msec

Performance metric used in this paper is the number of transactions that miss their deadlines, Missed Deadlines.

*MissRatio = (number of transactions missing the deadlines) / (total number of submitted transactions) * 100%*

C. EXPERIMENTAL RESULTS

FIG. 6 show the average number of transactions missing their deadlines. The both methods have the same performance for a small number of transactions in the system. But, when the multiprogramming level in the system increases, the superiority of the PSCC appears.

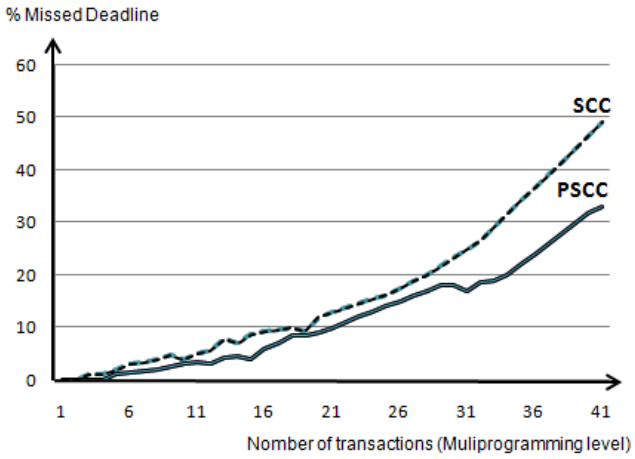


Fig. 7. SCC vs. PSCC Baseline Model (Missed Deadline)

Even though both speculative methods manage to preserve a large portion of the computation performed by each individual transaction, the reason that PSCC outperforms SCC can be explained by the fact that PSCC predicts and select for the transaction on conflict, the data value (before or after update) to avoid restarting.

This property of PSCC is especially advantageous when the number of data conflicts in the system increase.

6 CONCLUSION AND PERSPECTIVES

The SCC algorithm is interesting in the sense that it has the advantages of both pessimistic and optimistic methods of concurrency control methods for transactions. However, further study shows that disadvantages remain in its development. SCC protocol, for example, does not take into account the time constraints of transactions. We have therefore contributed to the evolution of the SCC method to reduce these drawbacks and allow more transactions to meet their deadlines.

We plan to extend this work in several ways. We will exploit semantics data for scheduling transactions. Further, we also plan to extend our work to manage multisite real time databases.

REFERENCES

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *In Proc. of the 14th VLDB Conf. ,* pp. 1-12 , San Mateo, Calif. , 1988.

[2] P. A. Bernstein, V. Hadzilacos, and N. Godman, Concurrency Control Recovery in Database Systems, *Addison-Wesly*, 1987

[3] A. Bestavros, Speculative Concurrency Control. *Technical Report TR-16-9*2, Boston University, Boston, MA, 1992.

[4] A. Bestavros, S. Braoudakis, E. Panagos, Performance Evaluation of Two-Shadow Speculative Concurrency Control. *Technical Report 1993-001*, Boston University, Boston, MA, 1993.

[5] A. Bestavros, S. Braoudakis, Timeliness via Speculation for Real-Time Databases. *In 14th IEEE Real-Time System Symposium*, Puerto Rico, 1994.

[6] A. Bestavros, S. Braoudakis, Value-cognizant Speculative Concurrency Control. *In 21st VLDB Conference*, Zurich, Switzerland, 1995

[7] J. R. Haritsa , M. J. Carey, and M. Linvy. On being optimistic about real-time Constraints. *In Proc. of the 9th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, Pages 331-343, 1990

[8] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, Experimental evaluation of real-time optimistic concurrency control schemes. *In Proc. of the 17th VLDB Conf., pp 35–46*, San Mateo, Calif., Sept. 1991. Morgan Kaufmann.

[9] J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of real-time transaction processing. In *Proc. of the 10th Real-Time Systems Symp.*, pp 144–153, Los Alamitos, Calif., Dec. 1989. IEEE Computer Society Press.

[10] J. Haubert, L. Amanton and B. Sadeg. Un protocole spéculatif pour le contrôle de concurrence et l'ordonnancement des transactions temps réel, *Proc. of the 12th Int. Conf. on Real-Time and Embedded Systems (RTS)*, pages 205-221, Paris, France (2004).

[11] Z. Mammeri, Expression et dérivation des contraintes temporelles dans les applications temps réel. *Journal Européen des Systèmes Automatisés (APII-JESA)*, Vol. 32, no. 5-6, p. 609–644, 1998.
[12] D. Menasce and T. Nakanishi. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information Systems*, 7(1):13–27, 1982.
[13] K. Ramamritham, Real-Time Databases. *Journal of Distributed and Parallel Databases*, Vol. 1, no. 2, p. 199–226, 1993.