Performance Evaluation of Two-Shadow Speculative Concurrency Control

Azer Bestavros (best@cs.bu.edu)

Spyridon Braoudakis (sb@cs.bu.edu)

Euthimios Panagos (thimios@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215

February 5, 1993

Abstract

Speculative Concurrency Control (SCC) [Best92a] is a new concurrency control approach especially suited for real-time database applications. It relies on the use of redundancy to ensure that serializable schedules are discovered and adopted as early as possible, thus increasing the likelihood of the timely commitment of transactions with strict timing constraints. In [Best92b], SCC-nS, a generic algorithm that characterizes a family of SCC-based algorithms was described, and its correctness established by showing that it only admits serializable histories. In this paper, we evaluate the performance of the Two-Shadow SCC algorithm (SCC-2S), a member of the SCC-nS family, which is notable for its minimal use of redundancy. In particular, we show that SCC-2S (as a representative of SCC-based algorithms) provides significant performance gains over the widely used Optimistic Concurrency Control with Broadcast Commit (OCC-BC), under a variety of operating conditions and workloads.

1 Introduction

Traditional concurrency control algorithms can be broadly classified as either *pessimistic* or *optimistic*. Pessimistic Concurrency Control (PCC) algorithms [Eswa76, Gray76] avoid any concurrent execution of transactions as soon as *potential* conflicts between these transactions are detected. Alternately, Optimistic Concurrency Control (OCC) algorithms [Boks87, Kung81] allow such transactions to proceed at the risk of having to restart them in case these suspected conflicts *materialize*.

For real-time database applications, where transactions execute under strict timing constraints, maximum concurrency (or throughput) ceases to be an expressive measure of performance. Rather, the number of transactions completed before their set deadlines becomes the decisive performance measure [Buch89]. Most real-time concurrency control schemes considered in the literature [Abbo88, Agra87, Stan88, Huan89, Sing88, Sha88, Sha91] are based on Two-Phase Locking (2PL), which is a PCC strategy. Despite its widespread use in commercial systems, 2PL has some properties such as the possibility of deadlocks and long and unpredictable blocking times that damage its appeal for real-time environments, where the primary performance criterion is meeting time constraints and not just preserving consistency requirements. Over the last few years, several alternatives to 2PL for RTDBMS have been proposed and explored [Kort90, Hari90a, Huan91, Kim91, Lin90, Son92]. Among these alternatives, the Optimistic Concurrency Control algorithm with Broadcast Commit (OCC-BC) [Mena82, Robi82] has been singled out as an attractive protocol for RTDBMS [Hari90b]. In this paper, OCC-BC is used as a representative of OCC-based algorithms.

In a recent study [Best92a], Bestavros proposed a categorically different approach to concurrency control for RTDBMS. His approach relies on the use of redundant computation to start on alternative schedules, once conflicts that threaten the consistency of the database are detected. These alternative schedules are adopted *only if* the suspected inconsistencies materialize; otherwise, they are abandoned. Due to its nature, this approach has been termed *Speculative Concurrency Control* (SCC). In [Best92b], SCC-nS, a generic algorithm that characterizes a family of such SCC-based algorithms was described, and its correctness established by showing that it only admits serializable histories. A particular member of this family, namely SCC-2S, which is notable for its minimal use of redundancy, is used in this paper as a representative of SCC-based algorithms.

The remainder of this paper is organized as follows. In section 2, we review the basic idea behind the SCC-based approach; SCC-2S, a simple yet powerful SCC-based algorithm, is overviewed. In section 3, we describe a detailed model of the client-server RTDBMS used in our simulations. In section 4, our experimental results are presented and discussed in detail. Finally, in section 5, we summarize our findings and conclude with future research directions.

2 Two-Shadow Speculative Concurrency Control

Various concurrency control algorithms differ basically in the time when conflicts are detected, and in the way they are resolved. The PCC and OCC alternatives represent the two extremes in terms of data conflict detection and conflict resolution. PCC locking protocols detect conflicts as soon as they occur and resolve them using blocking, whereas OCC protocols detect conflicts at transaction commit time and resolve them using restarts.

SCC algorithms combine the advantages of both PCC and OCC algorithms, while avoiding their disadvantages. 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. On the other hand, SCC resembles OCC in that it allows conflicting transactions to proceed concurrently, thus avoiding unnecessary delays that may jeopardize their timely commitment.

In the remainder of this section, we overview a simple yet powerful SCC-based algorithm, which can be thought of as a special case of the SCC-based algorithms described in [Best92a, Best92b, Best93]. The algorithm – called Two-Shadow SCC (SCC-2S) – allows a maximum of two shadows per uncommitted transaction to exist in the system at any point in time: a *primary* shadow and a *standby* shadow.

Let T_i be any uncommitted transaction in the system. The primary shadow for T_i runs under the optimistic assumption that it will be the first (among all the other transactions with which T_i conflicts) to commit. Therefore, it executes without incurring any blocking delays. The standby shadow for T_i, on the contrary, is subject to blocking and restarts. It is kept ready to replace the primary shadow, should such a replacement be necessary. The standby shadow runs under the pessimistic assumption that it will be the last (among all the other transactions with which T_i conflicts) to commit.

The SCC-2S algorithm resembles the Optimistic Concurrency Control with Broadcast Commit (OCC-BC) algorithm in that primary shadows of transactions continue to execute either until they validate and commit or until they are aborted (by a validating transaction). The difference, however, is that SCC-2S keeps a *standby* shadow for each executing transaction to be used if that transaction must abort. The standby shadow is basically a replica of the primary shadow, except that it is blocked at the *earliest* point where a Read-Write conflict is detected between the transaction it represents and any other uncommitted transaction in the system. Should this conflict materialize into a consistency threat, the standby shadow is *promoted* to become the primary shadow, and execution is *resumed* (instead of being *restarted* as would be the case with OCC-BC) from the point where the potential conflict was discovered.

To illustrate how SCC-2S works, consider the schedule shown in figure 1. Both transactions T_1 and T_2 start with one primary shadow, namely T_1^0 and T_2^0. When T_2^0 attempts to read object x, a potential conflict is detected. Instead of pessimistically blocking T_2 – like PCC blocking-based protocols – and instead of optimistically ignoring the potential conflict – like OCC restart-based protocols – our suggested SCC-2S approach would make a copy, or *shadow*, of the reader transaction (T_2 in this example). Thus, a backup shadow, T_2^1, is created.[1] The primary shadows T_1^0 and T_2^0 execute without interruption, whereas T_2^1 blocks. Later, if T_1^0 successfully validates and commits on behalf of transaction T_1, the primary shadow T_2^0 is aborted and replaced by T_2^1, which resumes its execution, hopefully committing before its set deadline.

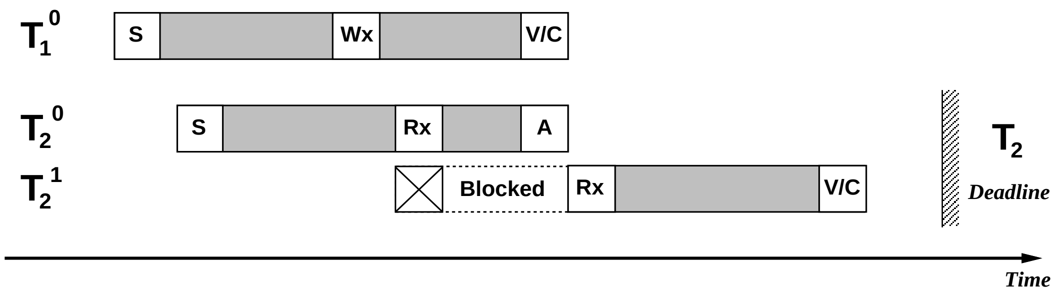


Figure 1: Schedule with a standby shadow promotion.

It is possible that multiple conflicts develop between executing transactions. Figure 2 illustrates the behavior of SCC-2S when a second conflict develops between T_2 and another transaction T_3. In particular, the primary shadow T_3^0 of T_3 attempts to write an object y that both shadows T_2^0 and T_2^1 had previously read. In this case, T_2^0 proceeds without any interruption, whereas T_2^1 is restarted and blocked as it attempts to read y. Should T_2^0 be aborted as a result of its conflict with T_3,[2] T_2^1 is promoted to become the primary shadow and is, thus, allowed to resume.

The SCC-2S algorithm allows at most two shadows for the same transaction to coexist at any given time. It is possible, however, that more that two shadows will be needed over a stretch of time. Figure 3 illustrates such a situation. In particular, after T_2^1 is promoted to become the primary shadow for T_2, a standby shadow T_2^2 is forked off to account for the read-write conflict between T_2^1 and T_1.

[1]This can be easily done by forking off a process from T_2^0.

[2]Or as a result of its conflict with T_1 (as was the case in figure 1).

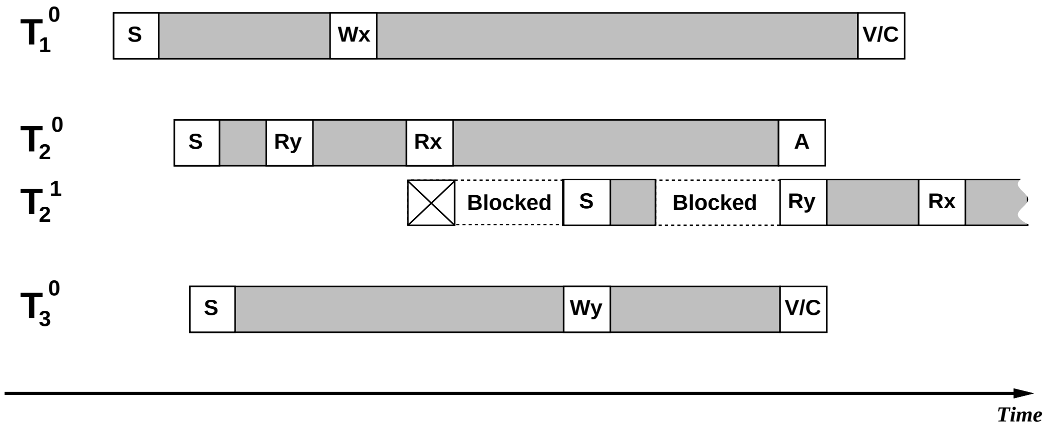


Figure 2: Schedule with a standby shadow restart and promotion.

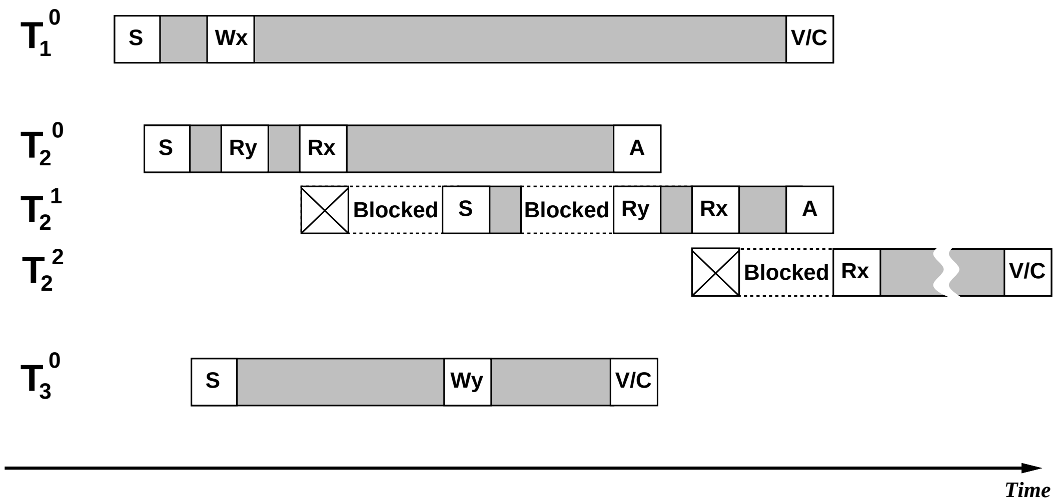


Figure 3: Schedule with two standby shadows.

3 A Client-Server Real-Time DBMS Model

Detailed experiments were carried on a simulated RTDBMS with two representative algorithms: OCC-BC and SCC-2S. Being interested in measuring the overhead imposed on the system by the implementation of each algorithm, we built our model to closely resemble a real system. In particular, the server's Transaction and Buffer Manager constitute partial implementations, whereas the Disk Manager is simulated. For the same reason actual, rather than simulated time, is measured. This includes the communication delays caused by the messages exchanged between the server and the clients. The organization of the model is depicted in Figure 4.

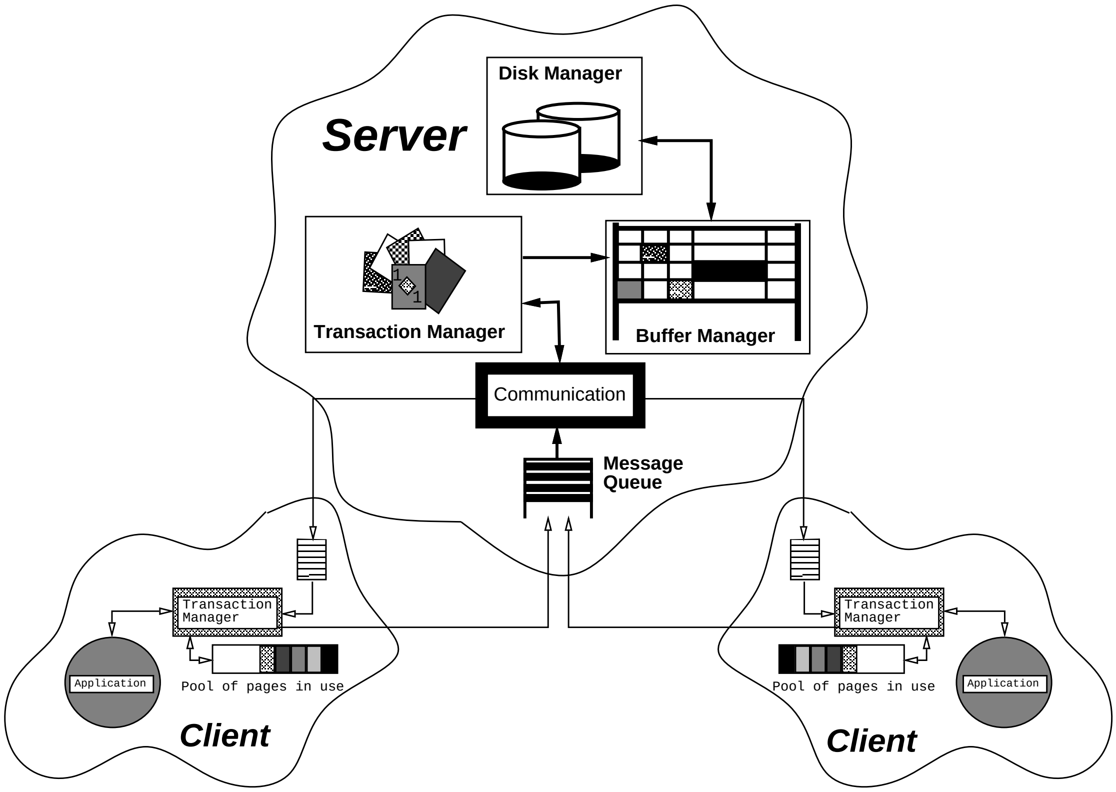


Figure 4: The system consists of one server process and a number of client applications.

The database is modeled as a collection of pages stored on a number of disks. The server is a shared memory multiprocessor which communicates with the transactions by exchanging messages. The Transaction Manager is responsible for keeping track of the pages used by the transactions running on the system. The Buffer Manager is responsible for providing the pages requested by the transactions, as well as storing into the buffer pool the dirty pages received by a committed transaction. The Least Recently Used (LRU) policy is employed for page replacement.

The transaction arrival rate follows a Poisson distribution with each transaction having an associated deadline time. Each transaction consists of a number of read and write operations. Each write operation is being preceded by a corresponding read operation on the same data object. The local transaction managers keep track of the pages accessed by their transactions, as well as their access modes. If a page is not present into the client's local Pool, it is requested from the server. This can cause up to two I/O operations on the server. During commit time, all updated pages are sent to the server. For each such updated page at most one I/O operation is performed.

3.1 Workload Model

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

The `DBSize` parameter determines the number of pages in the database. The number of pages accessed by a transaction is given by the `TRANSize` parameter. Page requests are generated from a uniform distribution spanning the entire database. The `WProb` parameter specifies the probability that a page which is already read will also be updated. The `SRatio` parameter provides the deadline slack factor in our simulations. By changing its value we can smoothly vary the tightness of transaction deadlines. The value of `SRatio` ranges from

Parameter	Meaning	Setting
`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
`SRatio`	Slack Ratio used in deadline formula	1.5
`RTime`	Average time to read a page	3 msec
`WTime`	Average time to update part of a page	15 msec

Table 1: The Workload Parameters

zero to infinity, with zero meaning that transactions have no laxity. The `RTime` and `Wtime` parameters are set to the average time that a transaction needs to read and update a page present in its client's local Pool, respectively.

In addition, we denote by R_{size}, and W_{size} the number of pages that a transaction reads, and writes, respectively. The average times needed to read and write a page are denoted by AVG_{read}, and AVG_{write}, respectively. T_{start} is the set-up time needed to start a transaction, and AVG_{end} is the time needed to commit a transaction. The following formula for the average execution time T_{avg} of a transaction can then be obtained:

$$T_{avg} = R_{size} * AVG_{read} + W_{size} * AVG_{write} + T_{start} + AVG_{end}$$

Knowing the average execution time for a transaction of a given size, T, we can calculate the deadline assigned to a transaction based on its Slack Ratio `SRatio` as follows:

$$D_T = T_{avg} + T_{avg} * \texttt{SRatio}$$

4 Performance Evaluation

In this section, a comparative evaluation of the performance of SCC-2S (as a representative of SCC-based algorithms) and OCC-BC (as a representative of OCC-based algorithms) in RTDBMS is presented. First, we describe the performance measures and list the parameter settings used in our baseline model. Next, we discuss our results and conclusions regarding the impact of data contention, resource contention, deadline tightness, deadline policies, and various loading conditions.

4.1 Performance Measures

Two primary performance metrics used in this paper are the number of transactions that miss their deadlines, *Missed Deadlines*, and the average time by which late transactions miss their deadlines, *Average Tardiness*. A transaction that commits within its deadline has a tardiness of zero. A transaction that completes after its deadline has a tardiness of $C_T - D_T$, where C_T and D_T are the transaction's completion time and deadline time, respectively.

Previous studies have argued that improving *both* of the aforementioned metrics is difficult [Hari90b]. Our simulations have shown that by adopting a superior concurrency control algorithm (SCC-2S in this case), *both* metrics can, indeed, be improved.

Our experiments assume that transaction deadlines are soft. This entails that late transactions (those missing their deadlines) must complete – nevertheless – with the minimum possible delay. Even though transaction response time was not explicitly measured in our simulations, the Average Tardiness metric can be used as an approximation. In particular, by reducing the `SRatio` value to 0, it can be shown that the transaction's Average Tardiness and Response Times are related. This observation coupled with our soft deadline assumption allow our simulations to be useful in the evaluation of SCC-2S for conventional DBMS.

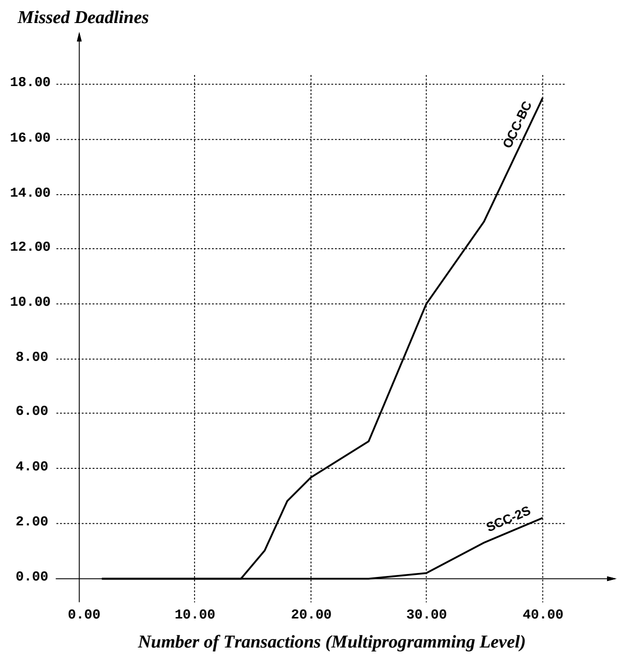
The simulations also generated a host of other statistical information, including CPU and disk utilizations, number of transaction restarts, average wasted computations, … *etc.* . These secondary measures (although not presented in this paper for space limitations) help explain the behavior of the algorithms under various loading conditions.

4.2 Parameter Settings and the Baseline Model

We started our experiments by first developing a baseline model around which we conducted further experiments, varying a few parameters at a time. Table 1 lists, the values assigned to the workload parameters in our baseline model. The database consisted of 1,000 pages from which each transaction accessed 20 pages randomly. The probability of a page been updated was set at 25%. These parameter settings are comparable to those used in similar studies [Hari90b].

Figures 5-a and 5-b depict the average number of transactions that missed their deadlines, and the extra time needed by late transactions – those missing their deadlines – to complete their operations, respectively. The performance of both algorithms is identical when the number of transactions in the system is small. But, as the multiprogramming level in the system increases, the superiority of the SCC-2S becomes evident. Not only do transactions running under the SCC-2S algorithm make most of their deadlines, but also the amount of time by which late transactions miss their deadlines is considerably smaller.

The reason that SCC-2S outperforms OCC-BC can be attributed to the fact that SCC-2S manages to preserve a large portion of the computation performed by each individual transaction. More precisely, when a transaction – say T – has to be aborted because of a conflict with another committing transaction, it does not have to restart from the very beginning, as it does under the OCC-BC algorithm. This means that some of the pages that were read or updated by transaction T will not need to be read or written again. This property of SCC-2S is especially advantageous when the number of data conflicts in the system is high.



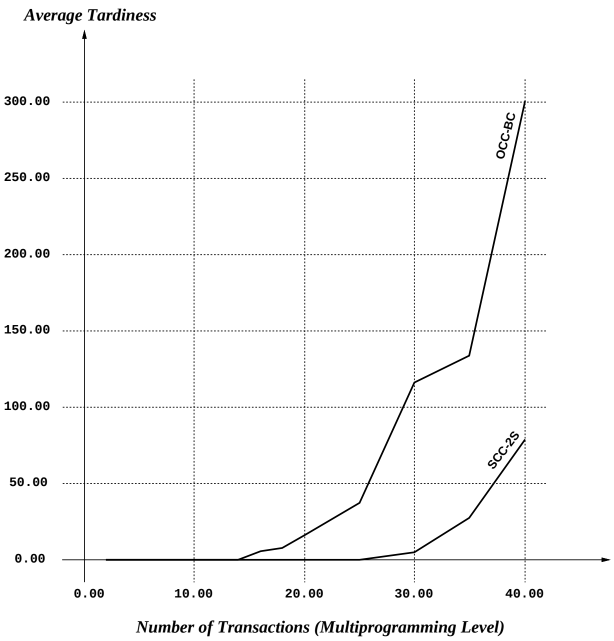


Figure 5: OCC-BC vs SCC-2S. Baseline Model (a) Missed Deadlines (b) Average Tardiness

However, the performance gained by using SCC-2S does not come for free. The cost incurred to set-up standby shadows is translated to extra *control* messages that have to be communicated with the server. Our simulations confirmed this fact. A 15%-increase in the average number of messages exchanged with the server was observed for our baseline model. But, as figures 5-a and 5-b demonstrate, these extra control messages pay off in the long run. In particular, the cost of these messages (in SCC-2S) is considerably lower than the cost incurred from transaction restarts (in OCC-BC).

It is worthwhile to mention that we reached the same conclusions presented above (viz-a-viz the number of Missed Deadlines, Average Tardiness, and Overhead Messages) when we experimented with a database residing in the main memory of the server's machine.

4.3 Deadline Tightness

In the next set of experiments we examined the effect of deadline tightness on the relative performance of the two algorithms. For this reason we varied the Slack Ratio while keeping all the other parameters the same as those of the baseline model. We present here two experiments for Slack Ratios of 0.7 and 2.0, respectively. The corresponding graphs are shown in figure 6 and figure 7, respectively.

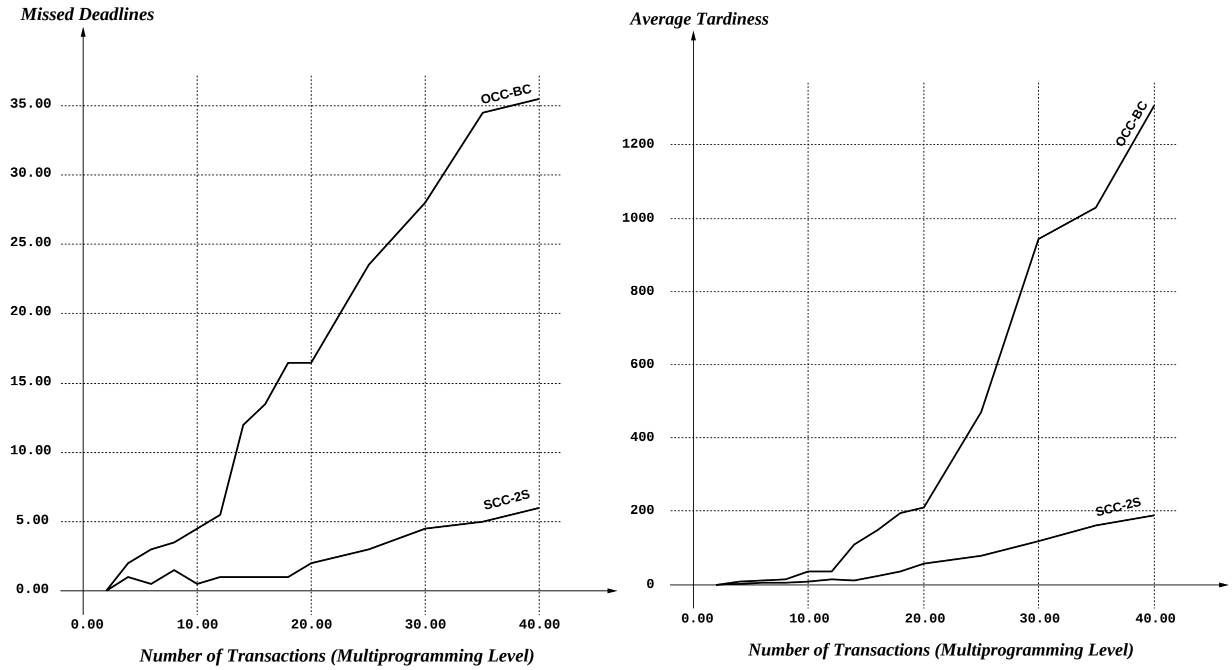


Figure 6: Slack Ratio of 0.7 (a) Missed Deadlines (b) Average Tardiness

At high Slack Ratios, both algorithms miss very few deadlines – with SCC-2S performing consistently better in all multiprogramming levels. However, as the Slack Ratio value decreases, and the system operates under very tight deadlines, the performance of the OCC-BC algorithm degrades rapidly, while the SCC-2S algorithm remains quite stable. Analogous results have been observed for Average Tardiness, with the gap between the two algorithms being even bigger.

4.4 Data Contention

To further demonstrate the superiority of SCC-2S, we experimented with a number of different data contention levels by varying the write probability, `WProb`, parameter. The `SRatio` factor was fixed to 1.5 for all the measurements taken. Figure 8-a depicts the number of transactions that missed their deadlines when the database consisted of 1000 pages and each transaction updated half of the pages it accessed (`DBSize` = 1000 and `WProb` = 50%). As we can see, the OCC-BC algorithm missed almost 50% of its deadlines, whereas its SCC-2S counterpart missed only around 10%. The results obtained with a `DBSize` of 500 pages, and a `WProb` of 50% are even more compelling. As figure 8-b suggests OCC-BC missed almost 70% of its deadlines. On the other hand, the SCC-2S appears more stable since only about 12% of the transactions in the system missed their deadlines.

4.5 Firm Deadlines

All of the previous experiments assumed a *soft* deadline policy, where all transactions have to be run to completion. In this section we look into the impact of having a *firm* deadline policy, whereby late transactions are immediately discarded from the system. Here the number of Missed Deadlines is sufficient to compare OCC-BC and SCC-2S under the firm deadline assumption. Both algorithms behaved considerably better than before; however, their *relative* performance was similar to that seen in the previous experiments. This improved behavior is explained if we consider that by discarding the transactions that already missed their deadlines, more resources are made available for the remaining transactions in the system. This, also, has a positive effect on the system load as well as the degree of data contention exhibited in the system.

We have also studied the effects of changes in write page probabilities, database sizes, and transaction sizes.[3] These experiments reinforced the aforementioned conclusions, especially under conditions of heavy loading, high data contention, and tight deadlines.

[3] These results are not fully discussed due to space limitations.

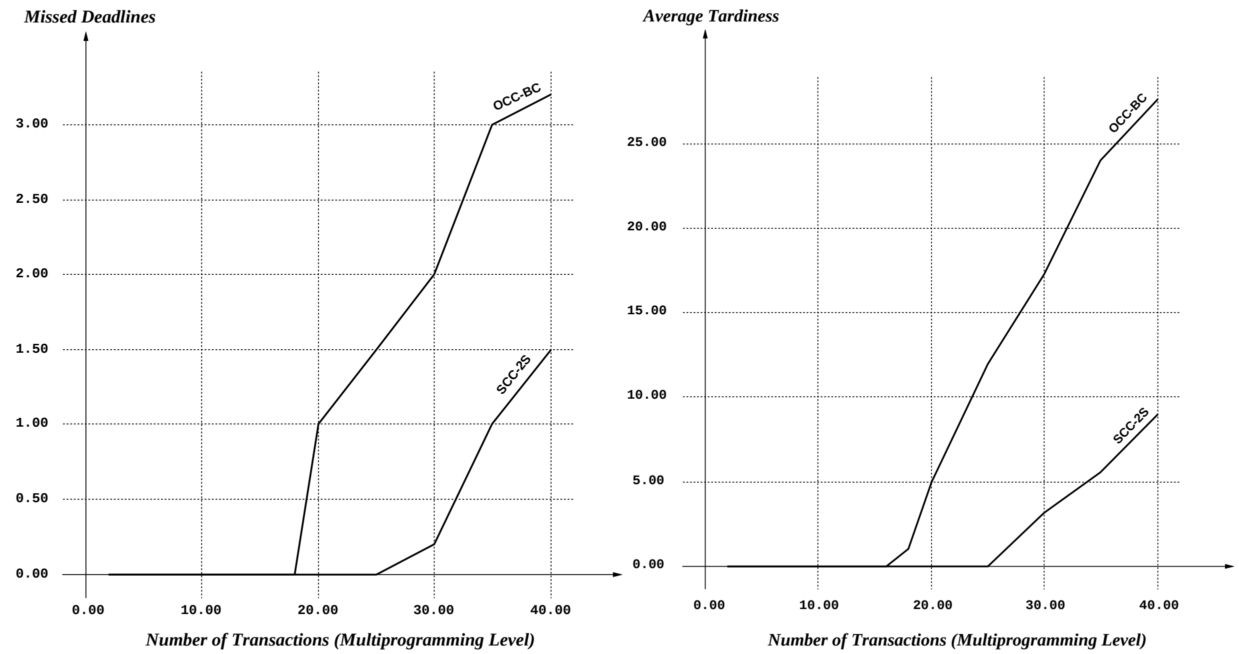


Figure 7: Slack Ratio of 2.0 (a) Missed Deadlines (b) Average Tardiness

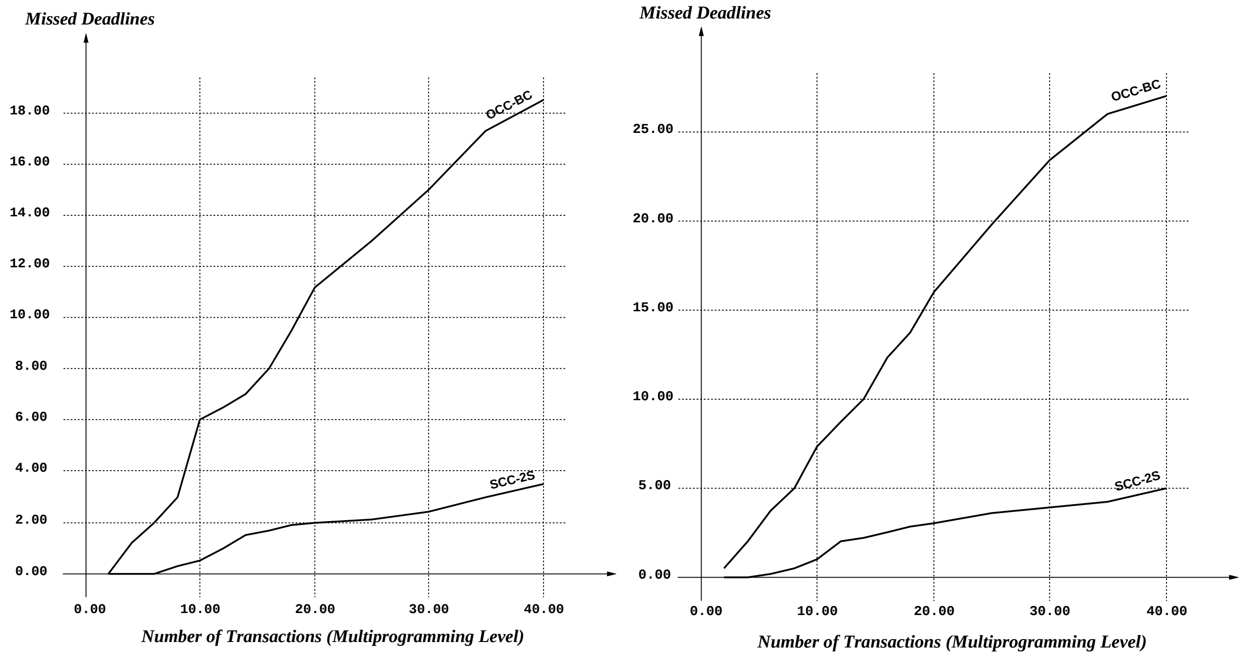


Figure 8: Miss Ratio for `WProb` of 50% (a) `DBSize` = 1000 (b) `DBSize` = 500

5 Conclusions

In this paper, we have presented a quantitative study of the relative performance of speculative and optimistic concurrency control techniques in the context of a distributed client-server real-time database system. The performance metrics used here, Missed Deadlines and Average Tardiness, are different from those used in a conventional DBMS, where response time and throughput are the main performance criteria.

In [Best92a], it was argued that SCC-based algorithms are better suited for RTDBMS. SCC relies on redundancy to ensure that serializable schedules are discovered and adopted as early as possible, thus increasing the likelihood of the timely commitment of transactions with strict timing constraints. Using SCC, several *shadow* transactions execute on behalf of a given uncommitted transaction so as to protect against the hazards of *blockages* and *restarts*, which are characteristics of PCC-based and OCC-based algorithms, respectively. To study the effect of these factors, extensive experiments were performed for two representative algorithms: OCC with Broadcast Commit (OCC-BC) and Two-Shadow SCC (SCC-2S). Our experiments indicate that SCC-2S offers a significant performance improvement over OCC-BC for a wide range of system loads. Therefore, from a performance standpoint, we argue that SCC-based protocols appear generally better suited than OCC-based protocols for RTDBMS.

In our simulations, we used conventional (not real-time) transaction and disk scheduling. Moreover, both the OCC-BC and SCC-2S protocols do not make use of transaction priorities in resolving data conflicts. Therefore, the results we obtained can be generalized to compare the two alternatives in the context of traditional DBMS. For RTDBMS, incorporating real-time transaction and disk scheduling will further improve the performance of SCC-2S. Also, accounting for transactions' priorities is likely to decrease the number of missed deadlines in the system [Hari90b]. In [Best92c] we investigate this problem in the context of SCC-based algorithms.

References

[Abbo88] Robert Abbott and Hector Garcia-Molina. "Scheduling real-time transactions: A performance evaluation." In *Prooceedings of the 14th International Conference on Very Large Data Bases*, Los Angeles, Ca, 1988.

[Agra87] R. Agrawal, M. Carey, and M. Linvy. "Concurency control performance modeling: Alternatives and implications." *ACM Transaction on Database Systems*, 12(4), December 1987.

[Best92a] Azer Bestavros. "Speculative concurrency control: A position statement." Technical Report TR-92-016, Computer Science Department, Boston University, Boston, MA, July 1992. Submitted for publication.

[Best92b] Azer Bestavros and Spyridon Braoudakis. "A family of speculative concurrency control algorithms." Technical Report TR-92-017, Computer Science Department, Boston University, Boston, MA, July 1992. Also submitted for publication to SIGMOD'93.

[Best92c] Azer Bestavros and Spyridon Braoudakis. "Speculative concurrency control algorithms for real-time databases: An alternative expression of transaction priority." Technical Report (In progress), Computer Science Department, Boston University, Boston, MA, September 1992.

[Best93] Azer Bestavros. "Speculative concurrency control for real-time databases." Technical Report TR-93-002, Computer Science Department, Boston University, Boston, MA, July 1993. Submitted for publication.

[Boks87] C. Boksenbaum, M. Cart, J. Ferrié, and J. Francois. "Concurrent certifications by intervals of timestamps in distributed database systems." *IEEE Transactions on Software Engineering*, pages 409–419, April 1987.

[Buch89] A. P. Buchmann, D. C. McCarthy, M. Hsu, and U. Dayal. "Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency controls." In *Proceedings of the 5th International Conference on Data Engineering*, Los Angeles, California, February 1989.

[Eswa76] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. "The notions of consistency and predicate locks in a database system." *Communications of the ACM*, 19(11):624–633, November 1976.

[Gray76] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. "Granularity of locks and degrees of consistensy in a shared data base." In G. M. Nijssen, editor, *Modeling in Data Base Management Systems*, pages 365–395. North-Holland, Amsterdam, The Netherlands, 1976.

[Hari90a] Jayant R. Haritsa, Michael J. Carey, and Miron Linvy. "Dynamic real-time optimistic concurrency control." In *Prooceedings of the 11th Real-Time Systems Symposium*, December 1990.

[Hari90b] Jayant R. Haritsa, Michael J. Carey, and Miron Linvy. “On being optimistic about real-time constraints.” In *Prooceedings of the 1990 ACM PODS Symposium*, April 1990.

[Huan89] 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*, December 1989.

[Huan91] Jiandong Huang, John A. Stankovic, and Don Towslwy Krithi Ramamritham. “Experimental evaluation of real-time optimistic concurrency control schemes.” In *Prooceedings of the 17th International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991.

[Kim91] Woosaeng Kim and Jaideep Srivastava. “Enhancing real-time dbms performance with multiversion data and priority based disk scheduling.” In *Prooceedings of the 12th Real-Time Systems Symposium*, December 1991.

[Kort90] Henry Korth. “Triggered real-time databases with consistency constraints.” In *Proceedings of the 16th International Conference on Very Large Data Bases*, Brisbane, Australia, 1990.

[Kung81] H. Kung and John Robinson. “On optimistic methods for concurrency control.” *ACM Transactions on Database Systems*, 6(2), June 1981.

[Lin90] Yi Lin and Sang Son. “Concurrency control in real-time databases by dynamic adjustment of serialization order.” In *Proceedings of the 11th Real-Time Systems Symposium*, December 1990.

[Mena82] D. Menasce and T. Nakanishi. “Optimistic versus pessimistic concurrency control mechanisms in database management systems.” *Information Systems*, 7(1), 1982.

[Robi82] John Robinson. *Design of Concurrency Controls for Transaction Processing Systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1982.

[Sha88] Lui Sha, R. Rajkumar, and J. Lehoczky. “Concurrency control for distributed real-time databases.” *ACM, SIGMOD Record*, 17(1):82–98, 1988.

[Sha91] Lui Sha, R. Rajkumar, Sang Son, and Chun-Hyon Chang. “A real-time locking protocol.” *IEEE Transactions on Computers*, 40(7):793–800, 1991.

[Sing88] Mukesh Singhal. “Issues and approaches to design real-time database systems.” *ACM, SIGMOD Record*, 17(1):19–33, 1988.

[Son92] S. Son, S. Park, and Y. Lin. “An integrated real-time locking protocol.” In *Prooceedings of the IEEE International Conference on Data Engineering*, Tempe, AZ, February 1992.

[Stan88] John Stankovic and Wei Zhao. “On real-time transactions.” *ACM, SIGMOD Record*, 17(1):4–18, 1988.