A New Approach to Real-Time Transaction Scheduling

Sang H. Son and Juhnyoung Lee

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903, USA
son@cs.virginia.edu, FAX: (804) 982-2214

Abstract

A real-time database system differs from a conventional database system because in addition to the consistency constraints of the database, timing constraints of individual transaction need to be satisfied. Various real-time transaction scheduling algorithms have been proposed which employ either a pessimistic or an optimistic approach to concurrency control. In this paper, we present new real-time transaction scheduling algorithms which employ a hybrid approach, i.e., a combination of both pessimistic and optimistic approaches. These protocols make use of a new conflict resolution scheme called dynamic adjustment of serialization order, which supports priority-driven scheduling, and avoids unnecessary aborts. Our experimental results indicate that hybrid protocols outperform other real-time concurrency control protocols in certain performance metrics.

Key words: real-time database, scheduling, concurrency control, deadline

1. Introduction

A *real-time database system* (RTDBS) differs from a conventional database system because in addition to the consistency constraints of the database, timing constraints of individual transaction need to be satisfied. In order to provide real-time response for queries and updates while maintaining the consistency of data, *real-time concurrency control* should involve efficient integration of the ideas from both database concurrency control and real-time scheduling. Various real-time concurrency control protocols have been proposed which employ either a pessimistic or an optimistic approach to concurrency control.

Most of the initial work in real-time concurrency control has been conducted on utilizing two-phase locking as the base of real-time concurrency control. The two-phase locking is termed as being *pessimistic*, because it, in anticipation of data conflicts, tends to delay the operations in order to avoid aborting them later. However, the very idea of delaying an operation is opposed to real-time systems. Besides, the degree of concurrency is low in the two-phase locking based algorithms because concurrent read and write locks on the same data object by several transactions are not possible. Furthermore, two-phase locking has some inherent problems such as the possibility of deadlocks and unpredictable blocking times, which are serious problems for real-time concurrency control.

An *optimistic* approach is a natural alternative. An optimistic scheduler aggressively schedules all operations, hoping that nothing will go wrong, such as a non-serializable execution. Later when the transaction is ready to commit, conflicts are checked for and a conflict resolution scheme is then applied to resolve the conflicts, if any. Ideally an optimistic approach has the properties of non-blocking and deadlock freedom, which make it suitable to real-time transaction processing. In addition, it has a potential for high degree of parallelism. However, the use of aborting for conflict resolution in optimistic schedulers results in a problem that transactions can end up aborting after having paid for most of the transaction's execution. This problem can particularly be serious for real-time transaction scheduling, where timing constraints of transactions should be satisfied. For conventional database systems, it has been shown that optimal performance can be achieved by combining blocking and aborting. We expect the same with RTDBS.

In this paper, we present two hybrid real-time concurrency control protocols which combine pessimistic and optimistic approaches to concurrency control in order to control blocking and aborting in a more effective manner. We compare the performance of these new protocols with that of other real-time concurrency control protocols proposed.

The first protocol is based on a priority-based locking mechanism to support real-time scheduling by adjusting the serialization order dynamically in favor of high priority transactions. This protocol uses the phase-dependent control of optimistic approach to support

This work was supported in part by ONR, by NOSC, and by IBM.

0-8186-2815-4/92 $3.00 © 1992 IEEE

dynamic adjustment of serialization order.

The second protocol is a combination of optimistic concurrency control and timesatmp ordering. This protocol also uses the phase-dependent control of optimistic approach. Furthermore, this protocol employs the notion of dynamic timestamp allocation and dynamic adjustment of serialization order using timestamp interval [Bok87], with which the ability of early detection and resolution of nonserializable execution is improved, and unnecessary aborts are avoided.

2. Related Work

A *real-time database system* (RTDBS) is a transaction processing system designed to handle workloads where transactions have completion deadlines. The objective of such system is to meet these deadlines. The real-time performance of an RTDBS depends on several factors such as the database system organization, the underlying processors and disk speeds. For a given system configuration, however, the primary determinants for the real-time performance are the policies used for scheduling transaction accesses to system resources, because these policies determine when service is provided to a transaction.

Recently, research on the scheduling problem in real-time database systems has been active [Abb88, Buc89, Coo91, Har90, Hua90, Hua91, Lin90, Sha91, Son90b]. As mentioned earlier, most of the current real-time concurrency control schemes are based on two-phase locking [Abb88, Hua90, Sha91]. Abbott and Garcia-Molina [Abb88] described a group of lock-based real-time concurrency control schemes for scheduling soft real-time transactions, and evaluated the performance of those protocols through simulation.

Some inherent problems of two-phase locking such as the possibility of deadlocks and unpredictable blocking times are serious for real-time transaction scheduling. In addition, a *priority inversion* can occur when a lower priority transaction blocks the execution of a higher priority transaction. Sha et al. [Sha91] proposed a conservative real-time concurrency control scheme called *priority ceiling*, which prevents deadlocks and transitive blocking.

Huang et al. [Hua90] developed and evaluated a group of real-time protocols for handling CPU scheduling, data conflict resolution, deadlock resolution, transaction wakeup, and transaction restart. Their study is based not on simulation but by actual implementation on a testbed system called RT-CARAT. They concluded that CPU scheduling protocols have a significant impact on the performance of RTDBS, that they dominate all other protocols, and that the overhead incurred in locking is non-negligible and cannot be ignored in real-time concurrency control analysis.

Recently, real-time concurrency control protocols based on optimistic method have been proposed and studied [Har90, Hua91, Lin90, Coo91]. Haritsa et al. [Har90] proposed a group of optimistic real-time concurrency control protocols and evaluated them on a simulation model. They have also conducted a study on the relative performance of locking-based protocols and optimistic protocols, and concluded that optimistic concurrency control protocols outperform two-phase locking-based protocols over a wide range of system utilization. Huang et al. [Hua91] also conducted a similar performance study of real-time optimistic concurrency control protocols, but on a testbed system, not through simulation. They examined the overall effects and the impact of the overheads involved in implementing real-time optimistic concurrency control on the testbed system. Their experimental results contrast with the results in [Har90], showing that optimistic concurrency control may not always outperform a two-phase locking-based protocol which aborts the lower priority transaction when a conflict occurs. They pointed out the fact that the physical implementation schemes have a significant impact on the performance of real-time optimistic concurrency control.

3. Hybrid Approach

Concurrency control protocols induce a serialization order among conflicting transactions. For a concurrency control protocol to accommodate the timing constraints of transactions, the serialization order it produces should reflect the priority of transactions. However, this is often hindered by the past execution history of transactions. A higher priority transaction may have no way to precede a lower priority transaction in the serialization order due to previous conflicts. For example, T_H and T_L are two transactions with T_H having a higher priority. If T_L writes a data object x before T_H read it, then the serialization order between T_H and T_L is determined as $T_L \rightarrow T_H$. T_H can never precede T_L in the serialization order as long as both reside in the execution history. Most of the current (real-time) concurrency control protocols resolve this conflict either by blocking T_H until T_L releases the writelock or by aborting T_L in favor of the higher priority transaction T_H. Blocking of a higher priority transaction due to a lower priority transaction is contrary to the requirement of real-time scheduling. Aborting is also not desirable because it degrades the system performance and may lead to violations of timing constraints. Furthermore, some aborts can be useless when the transaction which caused the abort is aborted due to another conflict. The objective of our hybrid approach is to avoid such unnecessary blocking and aborting.

3.1. Integrated Real-Time Locking Protocol

The first protocol, *Integrated Real-Time Locking* (IRTL), combines locking and optimistic concurrency control. By using a priority-dependent locking protocol, the serialization order of active transactions is adjusted dynamically, making it possible for transactions with higher priority to be executed first so that higher priority transactions are never blocked by uncommitted lower priority transactions, while lower priority transactions may not have to be aborted even in the face of a conflict. The adjustment of the serialization order can be considered as a mechanism to support real-time scheduling.

This protocol is an integrated protocol because it uses different solutions for read/write (rw) and write/write (ww) synchronization, and integrates the solutions to the two subproblems to yield a solution to the entire problem.

The protocol is similar to optimistic concurrency control in the sense that each transaction has three phases, but unlike the optimistic method, there is no validation phase. This protocol's three phases are read, wait, and write phases. The read phase is similar to that of optimistic concurrency control wherein a transaction reads from the database and writes to its local workspace. In this phase, however, conflicts are also resolved by using transaction priority. While other optimistic real-time concurrency control protocols resolve conflicts in the validation phase, this protocol resolves them in the read phase. In the wait phase, a transaction waits for its chance to commit. Finally, in the write phase, updates are made permanent to the database.

(1) Read phase

This is the normal execution of a transaction except that all writes are on private data copies in the local workspace of the transaction instead of on data objects in the database. Such write operations are called *prewrites*. The prewrites are useful when a transaction is aborted, in which case the data in the local workspace is simply discarded. No rollback is required.

In this phase read-prewrite and prewrite-read conflicts are resolved using a priority based locking protocol. A transaction must obtain the corresponding lock before it reads or prewrites. According to the priority locking protocol, higher priority transactions must complete before lower priority transactions. If a low priority transaction does complete before a high-priority transaction, it is required to wait until it is sure that its commitment will not lead to the higher priority transaction being aborted.

Suppose T_H and T_L are two active transactions and T_H has higher priority than T_L, there are four possible conflicts as follows.

(1) $r_{T_H}[x]$, $pw_{T_L}[x]$
The resulting serialization order is $T_H \rightarrow T_L$, hence satisfies the priority order, and does not need to adjust the serialization order.

(2) $pw_{T_H}[x]$, $r_{T_L}[x]$
Either of the two serialization orders can be induced with this conflict; $T_L \rightarrow T_H$ with immediate reading, and $T_H \rightarrow T_L$ with delayed reading. Certainly, the latter should be chosen for priority scheduling. The delayed reading in this protocol means blocking of $r_{T_L}[x]$ by the writelock of T_H on x.

(3) $r_{T_L}[x]$, $pw_{T_H}[x]$
If T_L is in read phase, abort T_L. If T_L is in its wait phase, avoid aborting T_L until T_H commits in the hope that T_L gets a chance to commit before T_H does. If T_H commits, T_L is aborted. But if T_H is aborted by some other conflicting transaction, then T_L is committed. With this policy, we can avoid unnecessary and useless aborts, while satisfying priority scheduling.

(4) $pw_{T_L}[x]$, $r_{T_H}[x]$
Either of the two serialization orders can be induced with this conflict; $T_H \rightarrow T_L$ with immediate reading, and $T_L \rightarrow T_H$ with delayed reading. If T_L is in its write phase, delaying T_H is the only choice. This blocking is not a serious problem for T_H because T_L is expected to finish writing x soon. T_H can read x as soon as T_L finishes writing x in the database, not necessarily after T_L completes the whole write phase. If T_L is in its read or wait phase, choose immediate reading.

As transactions are being executed and conflicting operations occur, all the information pertaining to the induced dependencies in the serialization order needs to be retained. In order to maintain this information, we associate the following with each transaction; two sets, *before-trans-set* and *after-trans-set*, and a count, *before-count* R. The before-trans-set (respectively, after-trans-set) of a transaction contains all the active lower priority transactions that must precede (respectively, follows) this transaction in the serialization order. The before-count of a transaction is the number of the higher priority transactions that precede this transaction in the serialization order. When a conflict occurs between two transactions, their dependency is determined and then the values of their before-trans-set, after-trans-set, and before-count are changed accordingly.

(2) Wait Phase

The wait phase allows a transaction to wait until it can commit. A transaction in the wait phase can commit if all transactions with higher priority that must precede it in the serialization order, are either committed or aborted. Since the before-count of a transaction is the number of such transactions, the transaction can commit only if its before-count becomes zero.

A transaction in the wait phase may be aborted due to two reasons; if a higher priority transaction requests a conflicting lock or if a higher priority transaction that must follow this transaction in the serialization order commits.

Once a transaction in its wait phase finds a chance to commit, then it commits and switches to its write phase and releases all readlocks. The transaction is assigned a final timestamp which is the absolute serialization order.

(3) Write Phase

In the write phase, the transaction is considered to be committed. All committed transactions are serialized by the final timestamp order. Updates are made permanent to the database while applying Thomas' Write Rule (TWR) for write-write conflicts [Ber87]. After each operation the corresponding writelock is released.

With the dynamic adjustment of serialization order of this protocol, priority order of transactions are retained as much as possible, which promises more timing constraints of real-time transactions to be satisfied. In addition unnecessary and useless aborts are avoided as much as possible. This protocol also takes advantage of the potential for high degree of parallelism of optimistic concurrency control, because concurrent read and write locks are possible. This protocol is proved to be deadlock-free [Lin90].

3.2. An Optimistic Method with Timestamp Intervals

The second protocol is a combination of optimistic concurrency control and *dynamic timestamp allocation* scheme [Bok87]. This protocol constructs the serialization order dynamically by using timestamp interval, associated with every active transaction. It updates the timestamp intervals of active transactions to adjust their serialization order.

This protocol enforces serializability by satisfying the following two conditions through every read, prewrite, and validation;

(C1) Each timestamp interval constructed when a transaction accesses a data object should preserve the order induced by the timestamps of all committed transactions which have also accessed that data object.

(C2) The order induced by final timestamp of a validating transaction should not destroy the serialization order constructed by the past execution, i.e., by committed transactions.

(C1) is used in the read phase of a transaction, when it reads or prewrites a data object. (C2) is for validation phase, when a final timestamp for the validating transaction is chosen. These provide a sufficient condition for serialization. While satisfying (C1) and (C2), the protocol also adjusts the serialization order in favor of priority order without violating data consistency.

Most timestamp-based concurrency control protocols use static timestamp allocation scheme, i.e., each transaction is assigned a timestamp value at its startup time, and a total ordering instead of a partial ordering is built up. This total ordering does not reflect any actual conflict. Hence, it is possible that a transaction is aborted even when it requests its first data access. Besides, the total ordering of all transactions is too restrictive, and degrades the degree of concurrency considerably. With dynamic timestamp allocation, serialization order among transactions are dynamically constructed on demand whenever actual conflicts are occurring. Only necessary partial ordering among transactions is constructed instead of a total ordering from the static timestamp allocation.

Furthermore, in this protocol, the use of timestamp interval instead of a single value for timestamp allows more flexibility to adjust serialization order. Subsequently, this protocol can provide a high degree of parallelism, and avoid many unnecessary aborts. As other optimistic protocols, this protocol divides the execution of a transaction into three phases: read, validation, and write. However, unlike other optimistic protocols, conflicts and nonserializable executions are detected and resolved during the read phase. This early resolution of nonserializable execution is another advantage of this protocol. The main weakness of this protocol is that priority inversion still can occur, because this protocol does not involve transaction priority for scheduling decision.

The algorithm of the protocol described so far is the basic form of the protocol, which does not use transaction priority in scheduling decision. Not only the protocol in its basic form has several advantages over other optimistic protocols, but also it can be improved in several ways by considering priority scheduling. One possibility is to manipulate the size of timestamp intervals of active transactions according to their priority. Because the size indicates the amount of possibility of restarting the transaction, a transaction with higher priority needs to have a larger timestamp interval than a transaction with lower priority. This strategy of manipulating the size of timestamp intervals depends on the policy of choosing final timestamp of the validating transaction from its timestamp interval. When choosing the final timestamp for a validating transaction, the protocol should check the priority of its conflicting transactions, and decide the timestamp in such a way that higher priority transactions are left with larger timestamp intervals. We have developed several extensions of the algorithm regarding this final timestamp selection. In addition, because this protocol is based on optimistic control using

forward validation mechanism, it can also be extended to support real-time scheduling by combining priority abort and priority sacrifice mechanisms with the validation mechanism, as in OPT-ABORT, OPT-SACRIFICE and OPT-WAIT protocols proposed in [Har90].

4. Performance Evaluation

We have conducted a comparative performance evaluation of various real-time concurrency control schemes, using a database prototyping tool [Son90]. We have examined one pessimistic protocol called *High Priority* (HP) protocol [Abb88], which is based on the two-phase locking, and employs priority abort policy for conflict resolution. We have included three optimistic protocols; OPT-CMT, OPT-SACRIFICE, and OPT-WAIT [Har90, Hua91]. Under OPT-CMT, the validating transaction always commit, while aborting all the conflicting active transactions. OPT-SACRIFICE uses priority abort policy for conflict resolution, while OPT-WAIT employs priority wait policy. As a hybrid protocol, we have also included IRTL. The primary metric used for performance evaluation is *percentage of missed deadlines*, which is the percentage of transactions that are not completed by their deadline. Furthermore, transactions are divided into ten priority groups according to their priorities. We have also employed deadline missing percentage with respect to each priority group, as one of our performance metrics. It identifies more precisely the discriminating power of the real-time concurrency control schemes in question.

Our experimental results show that over the entire operational range, optimistic schemes outperform the locking-based pessimistic protocol, HP. This result agrees with the results from [Har90]. The use of blocking in pessimistic control may incur more performance degradation than the use of aborting in optimistic control in real-time database systems. Our experimental results provide a solid ground for this intuitive reasoning for the use of aborting against blocking in real-time concurrency control.

Optimistic concurrency control protocols based on aborting such as OPT-CMT and OPT-SACRIFICE perform better than optimistic protocols based on blocking such as OPT-WAIT. Contrary to the previous result in [Har90], OPT-WAIT performs even worse than OPT-CMT which does not use transaction priority information. We may attribute the degraded performance of OPT-WAIT to its use of blocking, resulting in several potential problems. First, if a transaction finally commits after waiting for sometime, it causes all its conflicting transactions with lower priority to be restarted at a later point in time, hence decreasing the chance of these transactions meeting their deadlines. Second, while a validating transaction waits, new conflicts can occur and the number of conflicting transactions is increased, hence resulting in more restarts. Finally, a validating transaction with little slack time sometimes has to wait for a conflicting high priority transaction which is still in the early stage of its read phase, because the protocol blocks the validating transaction without considering the current stages of the conflicting transactions. This again will decrease the chance of meeting transaction deadlines.

Another observation is that OPT-SACRIFICE performs only slightly better than OPT-CMT. The benefit obtained by using priority information in OPT-SACRIFICE has been almost nullified by two potential problems of OPT-SACRIFICE; wasted sacrifice and mutual sacrifice [Har90].

Finally, the experimental results shows the hybrid protocol, IRTL performs competitively. In particular, under low data contention, it outperforms all of the other real-time concurrency control. The degraded performance of IRTL as data contention increases is primarily due to the overhead incurred in maintaining the information on the induced dependencies in the serialization order among transactions. We have to pay serialization order information retaining overhead to take advantage of an intelligent conflict resolution policy called dynamic adjustment of serialization order, which makes it possible to avoid unnecessary blocking and aborting. In terms of the discriminating power for different priority groups, IRTL and OPT-ABORT are good, and OPT-CMT is the worst.

Figure 1 shows the percentage of missed transaction deadlines for five protocols; HP, OPT-CMT, OPT-WAIT, OPT-SACRIFICE, and IRTL under different data contention levels by varying transaction interarrival time. Figure 2 shows the miss percentage for the same five schemes under different levels of data contention and system load by varying the transaction size, while fixing the values of other parameters. Figure 3 shows the percentage of missed deadlines with respect to priority group over four protocols: OPT-CMT, OPT-SACRIFICE, OPT-WAIT, and IRTL. The value of the mean interarrival time in Figure 3 is 50 msec, which represents a normal data contention.

REFERENCES

[Abb88] R. Abbott, H. Garcia-Molina, "Scheduling Real-time Transactions: A Performance Evaluation," *Proceedings of the 14th VLDB Conference*, 1988.

[Ber87] Bernstein, P. A., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, Mass., 1987.

[Bok87] Boksenbaum, C., M. Cart, J. Ferrie, and J. Pons, "Concurrent Certifications by Intervals

of Timestamps in Distributed Database Systems," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 4, April 1987.

[Buc89] Buchmann, A. et al., "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control," *5th Data Engineering Conference*, February 1989.

[Coo91] Cook, R. P., S. H. Son, H. Y. Oh, and J. Lee, "New Paradigms for Real-Time Database Systems," *8th IEEE workshop on Real-Time Operating Systems and Software (in Conjunction with) IFAC/IFIP Workshop on Real-Time Programming*, May 1991.

[Har90] J. R. Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," *IEEE Real-Time Systems Symposium*, Orlando, Florida, December 1990.

[Hua90] Huang, J., J. A. Stankovic, D. Towsley, and K. Ramamritham, "Real-Time Transaction Processing: Design, Implementation, and Performance Evaluation," Univ. of Massachusetts, COINS Technical Report 90-43, May, 1990.

[Hua91] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes," *VLDB Conference*, Barcelona, Spain, Sept. 1991.

[Lin90] Y. Lin and S. H. Son, "Concurrency Control in Real-Time Database Systems by Dynamic Adjustment of Serialization Order," *IEEE Real-Time Systems Symposium*, Orlando, Florida, December 1990.

[Sha91] Sha, L., R. Rajkumar, S. Son and C. Chang, "A Real-Time Locking Protocol," *IEEE Trans. on Computers*, Vol. 6, No. 7, July 1991.

[Son90] S. H. Son, "An Environment for Prototyping Real-Time Distributed Databases," *International Conference on Systems Integration*, Morristown, New Jersey, April 1990, pp 358-367.

[Son90b] Son, S. H., and J. Lee, "Scheduling Real-Time Transactions in Distributed Database Systems," *7th IEEE Workshop on Real-Time Operating Systems and Software*, Charlottesville, Virginia, May 1990.

[Yu90] Yu, P., and D. Dias, "Concurrency Control Using Locking with Deferred Blocking," *6th Intl. Conf. Data Engineering*, Los Angels, Feb. 1990, pp. 30-36.

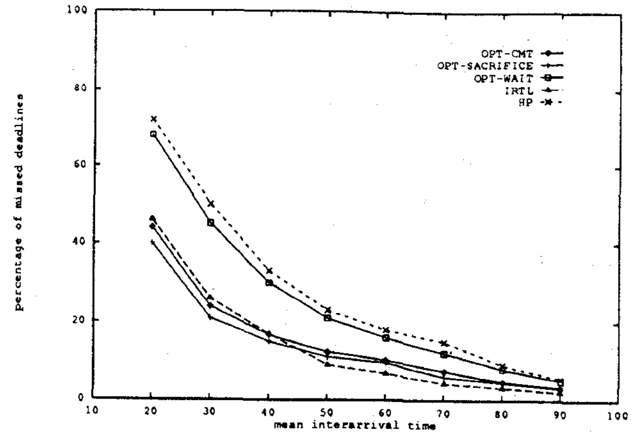

Figure 1. Mean interarrival time

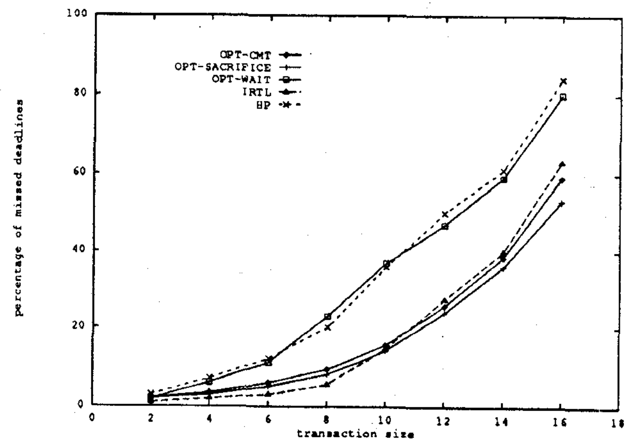

Figure 2. Transaction size

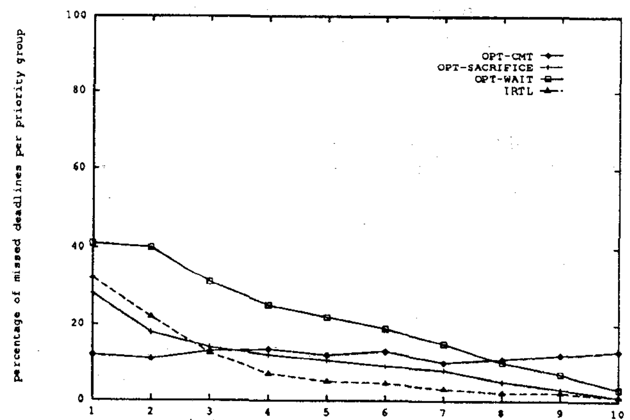

Figure 3. Miss percentage per priority group