Impact of Priority Assignment on Optimistic Concurrency Control in Distributed Real-time Databases

Kam-yiu Lam, Victor C.S. Lee and Sheung-lun Hung
Department of Computer Science
City University of Hong Kong
83 Tat Chee Avenue, Kowloon
HONG KONG

Benjamin C.M. Kao
Department of Computer Science
The University of Hong Kong
Pokfulam Road
HONG KONG

Abstract

In the studies of real-time concurrency control protocols (RT-CCPs), it is always assumed that earliest deadline first (EDF) is employed as the CPU scheduling algorithm. However, using purely (ultimate) deadline for priority assignment may not be suitable for distributed real-time database systems (DRTDBS) in which there exist different kinds of transactions, such as global and local transactions. In order to improve the performance, different priority assignment heuristics have to be used. In this paper, we have investigated the performance of different priority assignment heuristics for sub-transactions in DRTDBS with optimistic concurrency control (OCC) protocol. It is found that the heuristics, which are suitable for distributed real-time systems, are not suitable for DRTDBS. We find that our proposed heuristic, which considers both deadline constraint and data contention, can give the best performance.

1 Introduction

The research on *real-time database systems (RTDBS)* has received much attention in recent years. RTDBS are generally defined as the database systems in which the transactions have constraints on their completion times (usually are expressed as their deadlines) [11, 13]. The performance and the correctness of RTDBS are highly dependent on how well these deadlines are met. If the database in a RTDBS is partitioned in different sites, which are connected by communication links and logically related, it is called a *distributed real-time database system (DRTDBS)* [7]. It is more difficult to meet the transaction deadlines in DRTDBS. A number of factors can introduce unpredictability in transaction response times and the deadline constraints of the transactions are always very different.

In recent years, a lot of work has been devoted to the study of concurrency control protocols for RTDBS and DRTDBS [13]. The objectives are to design an algorithm which can minimize the number of deadline missing transactions and at the same time can maintain the database consistency. The protocols are called *real-time concurrency control protocols (RT-CCPs)*. In RT-CCPs, higher priority transactions are given preferences in using the data items. Most of the proposed RT-CCPs are based on locking and optimistic concurrency control (OCC) [2]. One well-known real-time locking protocol is High Priority Two Phase Locking (H2PL) [1] in which lock conflicts are resolved by restarting lower priority transactions. However, recent work has found that OCC is more suitable than locking for RTDBS [4].

The performance of RT-CCPs can be seriously affected by priority assignment methods as they determine which transaction should go first and which transaction should be blocked or restarted in resolving data conflicts [3, 5, 9, 12]. In the study of RT-CCPs, most of the work assumes that the importance and the characteristics of the transactions are similar. They use Earliest deadline first (EDF) for CPU scheduling and the priorities of the transactions are defined by their deadlines. Although EDF has been shown to be optimal and widely used in task scheduling in real-time systems [6], using deadlines as priorities may not be very suitable for distributed real-time systems and DRTDBS as the deadline constraints of the transactions or the tasks in the systems can be very different [3,5]. In [5], the problems in distributed real-time system have been discussed and new methods have been suggested. However, their studies have not included the impact of data synchronization on the performance of different priority assignment heuristics.

In DRTDBS, it is common to find transactions with different characteristics and deadline constraints in a

0-8186-7626-4/96 $5.00 © 1996 IEEE

system [5, 9]. For some transactions, a number of sub-transactions have to be created as their required data items are distributed in different sites. They are called *global transactions*. The others may only require to access the data items located in their site of origination (where the transactions are initiated). They are called *local transactions*. The deadline constraints of global transactions can be very different from local transactions. Although the deadlines of global transactions are usually longer than that of local transactions, the total time required to process a global transaction is also much longer than a local transaction. The use of deadlines as transaction priorities will make the system bias to local transactions. New methods have to be designed for priority assignment of sub-transactions of global transactions so that the probability of meeting the deadline constraints of the global transactions can be increased.

In [9], the heuristics suggested in [5] are re-examined in DRTDBS with H2PL as the concurrency control protocol. Surprisingly, these heuristics which have been found suitable for traditional distributed real-time systems are not suitable for DRTDBS. Different priority assignment methods can have a very significant impact on the performance of H2PL in DRTDBS. They affect the probability of lock conflicts and the probability of priority inversion. If a poorly designed priority assignment method is used, the systems will be suffered from the problem of cyclic transaction restarts and may result in deadlocks.

As OCC protocols have been found more suitable for RTDBS than locking protocols and it uses a completely different method to detect and resolve data conflicts, it is interesting to investigate the impact of different priority assignment methods on its performance and the relationship between them. The rest of this paper is organized as follows. In Section 2, we review the benefit of using OCC protocols for RTDBS and how they can be used for DRTDBS. In Section 3 we discuss a number of priority assignment heuristics. Section 4 describes our DRTDBS model and the workload model. The results of the simulation experiments comparing the various assignment heuristics is in Section 5. Finally, we conclude the paper in Section 6.

2 Optimistic Concurrency Control

In OCC protocols, the execution of a transaction is divided into three phases: (1) the read phase, (2) the validation phase, and (3) the write phase. During the read phase, the operations of a transaction or a sub-transaction will be processed one by one. The processing of an operation requires the access of data items in the database. The data items are read into the main memory. Computations based on the values of these data items are performed. If the operation is a write operation, new values are computed. They are not written into the database immediately. When all operations of a transaction have been processed, the transaction enters the validation phase in which the conflicts with other transactions which are in their read phase will be checked. If there are data conflicts, based on the conflict resolution method used, either the validating transaction or the other conflicting transactions will be restarted. Finally, if the validating transaction is not selected to restart, it enters the write phase in which updated data items are written back to the database from its private workspace.

If OCC protocols are extended to DRTDBS, two additional issues have to be catered: validation in a distributed environment and atomic commitment of transactions. Validation in DRTDBS is much more complex than that in a single-site RTDBS. In [8], a circular validation method based on locking is suggested. In the method, a lock table is defined in each site for the data items in that site. When a transaction wants to access a data item, it will set a lock in the lock table. The purpose of this lock is to indicate which transactions are accessing the data item. All the locks are compatible. In order to prevent distributed deadlock, the sites in the system are ordered. Validation of a transaction is started at the site with the highest order. In the validation at a site, the lock table will be examined. When validation at all the sites have been done, the transaction enters the write phase in which atomic commitment will be performed in addition to the permanent update of the database.

3 Priority Assignment Heuristics

In this section, several priority assignment heuristics will be introduced. They are divided into two groups. The first group considers only transaction deadlines. The second group considers both deadlines and the effect of data contention.

3.1 Deadline Based Heuristics

It is assumed that a global transaction T consists of m sub-transactions, T_1, T_2, ... , T_m, to be executed in series. The first $i-1$ sub-transactions are completed and sub-transaction T_i *is* ready for execution. Thus, a priority (i.e., a deadline) has to assign to T_i.

(1) Ultimate Deadline (UD)

The simplest way to assign a deadline to a sub-transaction is to adopt the deadline of its transaction. The first heuristic is called *Ultimate Deadline (UD)* in which the deadline of a sub-transaction is set to be the deadline of its transaction :

$$dl(T_i) = dl(T)$$

where $dl(X)$ is the deadline of X which is a transaction or a sub-transaction.

This is the priority assignment method used in most studies on RT-CCPs [1, 7]. The problem of UD is that it does not consider the amount of time that has to be reserved for the execution of the following sub-transactions ($T_{i+1}, \ldots, T_m$) of the transaction. It gives the scheduler incorrect information about how much time sub-transaction T_i can be delayed in its execution without causing the transaction T to miss its deadline.

(2) Effective Deadline (ED)

The second strategy corrects this misinformation by computing the *Effective Deadline (ED)* of the sub-transaction T_i. Under ED, the deadline of the sub-transaction T_i is the ultimate deadline minus the total predicted execution time of the sub-transactions of T following T_i. That is,

$$dl(T_i) = dl(T) - \sum_{j=i+1}^{m} pex(T_j)$$

where $pex(X)$ is the expected execution time of X.

The problem of UD and ED is that they allocate all the remaining slack of the global transaction to the current executing sub-transaction. Subsequently, the following sub-transactions ($T_{i+1}, \ldots, T_m$) may not have sufficient slack for their executions.

(3) Equal Slack (EQS)

A fair heuristic should distribute the slack among the sub-transactions. There are two slack distribution schemes. The first scheme is called *Equal Slack (EQS)* in which the slack is evenly distributed among the remaining sub-transactions:

$$dl(T_i) = ar(T_i) + pex(T_i) + [dl(T) - ar(T_i) - \sum_{j=i}^{m} pex(T_j)] / (m - i + 1)$$

where $ar(X)$ is the arrival time of X. It is assumed that a transaction or a sub-transaction will be ready for execution when it is arrived. The third term on the right hand side of the equation calculates how much slack should be distributed to T_i.

(4) Equal Flexibility (EQF)

The second scheme of distributing transaction slack is called *Equal Flexibility (EQF)* in which the distribution of the slack to the sub-transactions is proportional to their predicted execution time. The "flexibility" of a transaction T is defined as the ratio of the amount of slack of X to the amount of execution time of T:

$$dl(T_i) = ar(T_i) + pex(T_i) + [dl(T) - ar(T_i) - \sum_{j=i}^{m} pex(T_j)] \times pex(T_i) / \sum_{j=i}^{m} pex(T_j)$$

In EQS and EQF, the deadline assignment is dynamic which means that it is determined at run time just before sub-transaction T_i is submitted for execution. The total slack being distributed (the term in the square brackets) is the amount of slack global transaction T has with respect to the current time.

In [5], it has been shown that EQS and EQF perform much better than UD and ED in a distributed real-time system in terms of meeting global task deadlines. The reason is that by assigning to sub-task deadlines that can faithfully represent their degrees of urgency, EQS and EQF successfully monitor the progress of sub-tasks. This in turn avoids unwise delay to certain sub-tasks (especially those that are the first couple of stages of some global tasks) which are mistaken to have large amount of phantom slack. A higher percentage of task deadlines can thus be met.

Although EQS and EQF enjoy good performance in distributed real-time systems [5], they suffer major setbacks in DRTDBS using H2PL in [9]. The biggest problem in this kind of system is that EQS and EQF do not click with the traditional real-time concurrency control protocols. For example, while a transaction T is waiting, its slack decreases with time. Consequently, according to EQS and EQF, the priorities of T's sub-transactions will become higher relative to the sub-transactions of the executing transaction (lets say T'). The scheduler is thus likely to swing the CPU to a waiting transaction (T) whenever a sub-transaction (of T') is done. This interleaving, although ensures that transactions are progressing at pace, vastly increases the probability of data conflict as more unfinished transactions are holding locks at the same time.

3.2 Data Conflict Based Heuristics

In order to counteract the effect of intensifying data contention brought along by EQS and EQF, we need to inject transaction data requirements into the assignment of sub-transaction priorities. Here, we introduce three new heuristics.

(5) Static Equal Slack (SEQS)

To make the priority of the sub-transactions less dynamic, we can use a static method to distribute the slack of a transaction to its sub-transactions. In SEQS, the deadlines of all the sub-transactions are assigned once and for all when the global transaction arrives.

$$dl(T_i) = ar(T) + \sum_{k=1}^{i} pex(T_k) + [dl(T) - ar(T) - \sum_{j=1}^{m} pex(T_j)] \times \frac{i}{m}$$

In EQS, if the priority of a sub-transaction, $T_{1,i}$ is smaller than another sub-transaction, $T_{2,j}$, the priority of the sub-transactions following T_{2j} will become higher as the slack of T_2 becomes smaller while T_{2j} is waiting for scheduling. However, in SEQS, the priority of a sub-transaction will not be increased due to the waiting of the sub-transactions before it in its parent transaction as its priority is using the slack when its parent transaction arrives at the system. Thus, the degree of interleaving in transaction execution is smaller in SEQS as compared with EQS.

(6) Number of Data Items (NL)

Two important factors affecting the probability of data conflict in OCC protocols is the number of data items accessing by different transactions and the duration of using the data items. One way to reduce the data conflict probability is to give higher priority to the transactions which are accessing more data items. In *Number of Data Items (NL)*, the priority of a sub-transaction T_i is assigned according to the number of data items accessing by its parent transaction T (i.e., all data items used by $T_1, T_2, \ldots, T_i$ count towards the priority):

$$p(T_i) = \textit{number of data items accessing by } T$$

By assigning the highest priority to the transaction which is accessing the largest number of data items, the transaction can complete faster. This greatly reduces the probability of data conflicts and the number of transaction restarts. The priority of a sub-transaction is following the priority of its parent transaction.

Strategy NL focuses on reducing data contention while UD, ED, EQS and EQF focus on determining the milestones (sub-deadlines) monitoring the progress of transactions.

(7) Mixed Method:

We inject the idea of NL to the deadline-cognizant heuristics (in section 3.1): sub-transaction priorities can be assigned based on a function which includes both transactions' real-time constraints and the number of data items accessing by the transaction. We call this approach the *Mixed Method (MM)*:

$$dl(T_i) = ar(T) + \sum_{k=1}^{i} pex(T_k) + [dl(T) - ar(T) - \sum_{j=1}^{m} pex(T_j)] \times lock\_factor(T)$$

where $lock\_factor(T)$ = 1 − (number of data items accessing by T) / (total number of data items to be accessed by T)

The idea of MM is to artificially advance the deadline of a global transaction T (for scheduling purpose only) according to the number of data items it is accessing. The larger the number of data items T accessing, the smaller is the lock factor, and the earlier is T's artificial deadline (i.e., a higher priority). By raising the priority of a transaction which is accessing more data items, it is hope that the transaction can complete earlier and the degree of data contention in the system can be reduced. The priority of a sub-transaction is following the priority of its parent transaction. MM thus considers both the deadline requirement of the transactions as well as the data contention issue.

4 The Model

In this section, the DRTDBS model and the workload model, which are used to study the impact of different priority assignment heuristics on the performance of OCC in DRTDBS, are described. In the OCC protocol, the broadcast commit method [4] is used for solving data conflicts.

4.1 Distributed Real-time Database Model

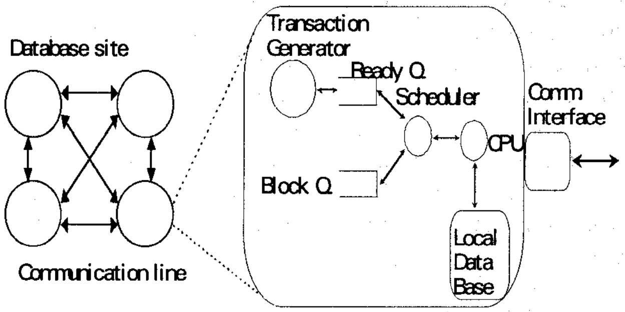

Figure 1: The DRTDBS model.

At each site, the transaction generator generates transactions independently according to Poisson distribution. Two types of transactions are considered in the model: global and local. While a local transaction only accesses local data items, a global transaction consists of a series of sub-transactions. If a sub-transaction requests a remote data item (a data item located other than its site of origination), it will be transmitted to a remote site through the communication network, and be processed there.

The processing of operations in both local transactions and sub-transactions is similar. It requires the use of the CPU and the access of the data items in the database. For simplicity, it is assumed that both local transactions and sub-transactions have similar CPU and data requirements. In particular, they access the same number of data items, and the execution times of the database operation are the same. Since main memory database systems can better support real-time applications, it is assumed that the databases are residing in the main memory. With the use of main memory database, the impact of different I/O scheduling on the system performance can be eliminated.

At each site, transactions and sub-transactions are scheduled to the CPU by the scheduler based on their priorities. Transactions and sub-transactions that are ready for execution are enqueued in the ready queue according to their priorities. In the OCC protocol, when a transaction accesses a data item in the read phase, the data item will be marked by setting a lock in the lock table to indicate that it is being used by the transaction. When all the operations of a transaction have been processed, the transaction enters the validation phase. Circular validation is started at the site with the highest site order. Data conflicts are checked by looking at the lock table in each site. In the model, we use the broadcast commit method for conflict resolution in which any conflicting transaction with the validating transaction will be restarted [4]. After the completion of the validation phase, the transaction enters the write phase in which two phase commit protocol [2] and permanent updates of the write operations will be performed. If all the sub-transactions (for global transactions only) are about to commit, the parent transaction will decide to commit. After all the sub-transactions and the parent transaction have committed, the transaction is completed.

We assume that the transactions are associated with *firm* deadlines [1]. Before a transaction is allocated the CPU, the scheduler checks its deadline. If it has already missed the deadline, the transaction is aborted immediately.

4.2 Workload Model and Measures

The arrival rates of global and local transactions in a site are λ_{global} and λ_{local} respectively. Each global transaction consists of m sub-transactions. We use the same model for both local transaction and sub-transaction, which requires N_{oper} number of database operations. Each operation involves locking of a data item (which takes T_{lock} amount of time) and processing of the data (which takes $T_{process}$ amount of time). Therefore, the total processing time for a local or sub-transaction is $(T_{lock} + T_{process}) \times N_{oper}$, and for a global transaction, m times that amount. The total system load is thus $(\lambda_{local} + \lambda_{global} \times m) \times N_{oper} \times (T_{lock} + T_{process})$ of which a fraction of $\lambda_{local} / (\lambda_{local} + \lambda_{global} \times m)$ is contributed by local transactions. We denote this latter fraction, *frac_local*. In addition, a sub-transaction may access data items in a remote site, in which case, a communication delay of T_{comm} amount of time is incurred before its execution.

The deadline of a local transaction, X_{local}, is generated according to the following formula:

$$\text{Deadline} = ar(X_{local}) + (T_{lock} + T_{process}) \times N_{oper} \times (1 + SF)$$

where SF is the slack factor which is a random variable uniformly chosen from a range (see Table 1 below).

For a global transactions, X_{global}, the deadline formula is modified to include the network delay:

$$\text{Deadline} = ar(X_{global}) + ((T_{lock} + T_{process}) \times N_{oper} \times m + T_{comm} \times N_{transit}) \times (1 + SF)$$

where $N_{transit}$ is the number of transit across the network required to access all the remote data.

In our model, a small database is used to create a high data contention environment. This helps us in understanding the interaction between the priority assignment strategies and the concurrency control protocols. The small database also allows us to study the effect of hot-spots, in which a small part of the database is accessed frequently by most of the transactions. Table 1 summarizes the model parameters and their baseline values.

Parameter	**Baseline Value**
CPU Scheduling	EDF
Concurrency Control	OCC
Database size / site	200 data items
Number of database sites	4
T_{lock}	2 msec
$T_{process}$	34 msec
T_{comm}	100 msec
N_{oper}	1 for both local and sub-transactions
m	4
SF	[1.0, 5.5] for system with short globals / [2.5, 13.75] for system with long globals
frac_local	0.75

Table 1: Baseline setting

The primary performance measure is the percentage of missed deadlines (or miss rate, MR) which is defined as the fraction of deadline missing transactions over the total number of transactions generated. As we divide the transactions into locals and globals, in the simulation, we collect statistics of the two types of transactions individually. We use MR_{global} (MR_{local}) to denote the fraction of global (local) transactions that missed their deadlines. For example, $MR_{global} = 0.1$ means that every one out of ten global transactions are tardy (the deadline is missed).

We also measure the restart rates for both local and global transactions. They are defined as the number of restarts of local transactions (or global transactions) over the total number of local transactions (or global transactions) completed before their deadlines.

5 Performance Results

In this section we summarize the results of our simulation experiments comparing the performance of the heuristics mentioned in Section 3. From the study, we observed that the performance of ED is similar to that of

UD and that of EQF is similar to that of EQS under most of the system configurations. In order to make the performance graphs more legible, in the following discussion, we do not show the performance of ED and EQF.

The simulator is built using OPNET [10] which is a proprietary graphical simulation package. Each simulation experiment (generating one data point) consists of 4 simulation runs, each lasting 300 simulation time units (around 10,000 transactions per run, many more for high load experiments). The 95% confidence interval is $\pm$ 0.5 percentage point for the missed deadlines figures shown in later sections.

Figure 2 shows the result in which MR_{global} and MR_{local} under the five heuristics are plotted against λ_{global} under the baseline setting. From the figure we see that when the loading is light, the miss rate of local transactions is higher than that of globals. Given a low data contention environment and a relatively tight slack, local transactions may not have enough slack time to complete before the deadlines. On the other hand, global transactions have much more slack time if they can commit in the first execution cycle without being restarted. However, when the loading is increased and resulted in a relatively high data contention environment, global transactions suffer a much higher miss rate than locals do if no remedial measures are taken such that long global transactions are given certain preference.

From the figure, we see that slack distribution heuristics (EQS and SEQS) improve the performance of the global transactions slightly compared with UD. The bad performance of UD is due to the fact that assigning the same ultimate deadline of a global transaction to all of its sub-transactions fails to capture the urgency of each sub-transaction. For instance, the first couple of sub-transactions of a global transaction is delayed extensively by the scheduler because of their erroneous slack. High MR_{global} thus ensues. For EQS and SEQS, distributing the slack among the sub-transactions allows the sub-transactions to proceed at a similar pace as local transactions. However, this pace-keeping feature is followed by the increase in data conflict. As can be seen in Figure 3, both EQS and SEQS cause more restarts than UD does. This is because these heuristics keep global transactions progressing *in pace*. If conflicting transactions are allowed to proceed together, it is likely that the losers (the restarted transactions) are close to finishing, and thus have already consumed much system resources. The cost of restarting them will be very high. Also, the committing transaction may have suffered extensive delay (due to the presence of its competitors) and misses its deadlines. Thus, it may result in a lose-lose situation. As a result, only a slight improvement can be observed. On the other hand, the price for saving global transactions is a higher local transaction miss rate.

Although MM and NL also pay the same price, they perform much better. To improve on meeting global transaction deadlines, MM and NL hoist the sub-transactions' priorities and expedite their executions. As shown in Figure 2, the results are a nice balance between MR_{global} and MR_{local}. They save a tremendous number of global transaction deadlines without losing too many local ones. For MM, the miss rate of the global transactions is almost 1/2 that of UD. Even better is NL, the miss rate of the global transactions is magically kept at a very low level.

From Figure 3, we see that the restart rates are low (less than 18%). To study the system behavior under high data contention situation, we increase the global transaction size to 12 sub-transactions. More data items are thus requested by global transactions, creating more severe data conflict. The transaction miss rates when long globals are present are shown in Figure 4. We see that the limited benefit offered by EQS and SEQS vanishes and the performance gets worse in this scenario, indicating that the adverse effect of data contention elevated by EQS and SEQS is taking its toll on system resources. Only a few global transactions make it to the validation phase before being restarted by the *shorter* local transactions. Also, from the experiment data, we observe that by forcing despondent global transactions to proceed along with locals, EQS and SEQS waste system resources as well as intensify data conflict. This argument is supported by Figure 5, which shows that the system suffers from a significant restart rate. For example, under both EQS and SEQS, for $\lambda_{global} > 0.3$, on average each transaction is restarted more than once.

For EQS and SEQS, losing their performance edge to UD due to long global transactions is in sharp contrast to its behavior under a data-contention-free environment. To make this fate reversal more dramatic, we reduce the fraction of local transactions contributed to the system. Figures 6 and 7 show the miss rates and restart rates when *frac_local* is reduced from 0.75 to 0.25. From the figures, we see that the performance of EQS and SEQS are much worse than UD. More globals (a smaller *frac_local*), therefore, gives higher chances of data conflicts and greater waste of system resources due to transaction restarts.

On the other hand, the other two heuristics which take data requirements into account give consistent and better performance in these harsh conditions. From Figures 4 and 6, we see that NL again gives the lowest MR_{global} among the heuristics. The reason is that global transactions require more data items than locals do. Under NL, the more data items a transaction is accessing, the higher is its priority. Global transactions therefore enjoy higher priorities under NL and miss fewer deadlines. This improvement on MR_{global} , however, is offset by a significant increase in MR_{local}. The offset is even larger when *frac_local* is reduced to 0.25. This poor performance suggests that considering only transaction data requirements alone is far from adequate. A good strategy needs to consider transaction timing requirements as well.

From Figures 4 and 6, we see that MM gives a smaller MR_{global} than UD does. Although this improvement is not as great as that provided by NL, the penalty for having a lower MR_{global} using MM is light: relatively fewer local transactions missed deadlines. Comparing with EQS and SEQS, MM misses significantly fewer global transaction deadlines. The major difference is that MM considers not only the real-time constraints but also transaction data requirements. By giving high priorities to transactions that are accessing more data items, MM helps these transactions to breeze through their executions and reduces undesirable restarts. This observation is supported by Figures 5 and 7, in which the restart rates for MM are seen to be lower than that of EQS and SEQS.

Thus, MM has the best overall performance over a wide spectrum of system characteristics. The advantage of MM lies in its ability to cope with the two conflicting factors: transaction timing and data requirements, which exert opposing demands on the priority assignment strategy.

6 Conclusions

The performance of real-time concurrency control protocols is heavily affected by the method used in assigning the priorities of the transactions. In this study the application of various sub-transaction priority assignment heuristics on DRTDBS using optimistic concurrency control (OCC) protocol is examined. We have found that the purely deadline-driven approaches, namely UD, ED, EQS, and EQF do not interact well with the concurrency control protocol. Although EQS and EQF perform well when there are relatively few and short global transactions, they fail miserably when the data contention is high due to severe data conflict and transaction restarts.

To reduce data contention, heuristics that consider transaction data requirements are applied. Our results show that NL, which gives higher priorities to transactions that access more data items, reduces global transaction miss rate significantly. This gain, however, is obtained at the price of missing large number of local transaction deadlines. Heuristic MM, which considers both transaction real-time constraints and the impact of data contention, gives the best overall performance. In fact, MM outperforms the other heuristics under different loading conditions. It exhibits the pace-keeping property of the deadline-driven approaches and takes care of the data-contention consideration. The results also show that MM is a well balanced priority assignment strategy in terms of meeting both local and global transaction deadlines.

References

[1] Abbott, R., and Garcia-Molina, H., "Scheduling Real-time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems,* vol. 17, no. 3, pp. 513-60, 1992.

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

[3] Y.W. Chen and Le Gruenwald, "Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real-time Database Systems," *Information Systems,* vol. 21, no. 1, pp. 103-124, 1996.

[4] Haritsa, J. R., Livny, M., Carey, M. J., "On Being Optimistic about Real-Time Constraints," *Proceedings of the 9th ACM Symposium on Principles of Database Systems*, 1990.

[5] Kao, B., and Garcia-Molina, H., "Deadline Assignment in a Distributed Soft Real-Time System," *Proceedings of 13th International Conference on Distributed Computing Systems*, pp. 428-37, 1993.

[6] Liu, C.L. and Layland, J.L., "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[7] Lam, Kam-yiu and Hung, Sheung-lun, "Concurrency Control for Time-constrained Transactions in Distributed Databases Systems," *The Computer Journal*, vol. 38, no. 9, 1995.

[8] Lam, Kwok-wa, Lee, Victor C.S., Lam, Kam-yiu and Hung, Sheung-lun, "Distributed Real-time Optimistic Concurrency Control Protocol," *Proceedings of 4th International Workshop on Parallel and Distributed Real-time Systems,* pp. 122-125, 1996.

[9] Lee, Victor C. S., Lam, Kam-yiu, Kao, Benjamin C. M., Lam, Kwok-wa, and Hung, Sheung-lun, "Priority Assignment for Sub-transaction in Distributed Real-Time Databases," *Proceedings of 1st Int. Workshop on Real-Time Databases: Issues and Applications (RTDB '96)*, pp. 101-106, California, March, 1996.

[10] OPNET Modeling Manual, Release 2.5, MIL 3, Inc., Washington, DC, 1996.

[11] Ozsoyoglu, G. and Snodgrass, R.T., "Temporal and Real-Time Databases: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, no. 4, pp. 513-532, 1995.

[12] Sivasankaran, R. M., Stankovic, J. A., Ramamritham, K., Towsley D. and Purimetla, B., "Priority Assignment in Real-Time Active Databases," *Journal of VLDB*, vol. 5, no. 1, pp. 19-34, 1996.

[13] Yu, P. S., Wu, K. L., Lin, K. J., and Son, S. H., "On Real-Time Databases: Concurrency Control and Scheduling," *Proceedings of IEEE*, vol. 82, no. 1, pp. 140-57, 1994.

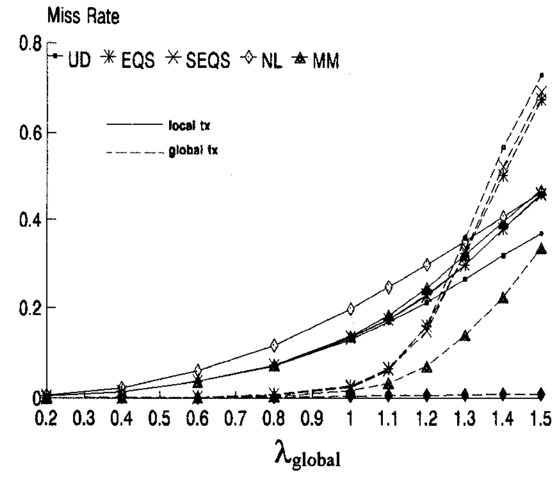

Figure 2: Transaction miss rates

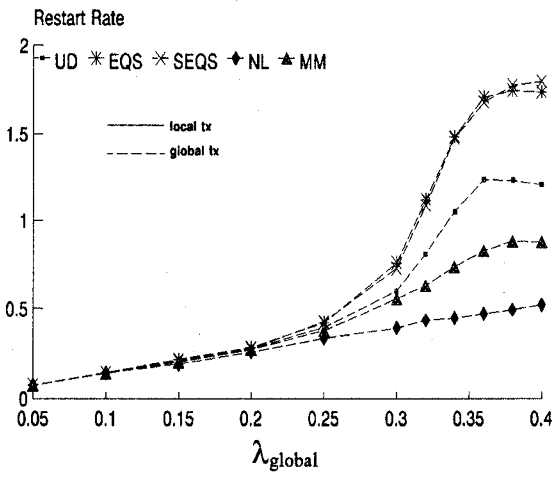

Figure 5: Restart rates (m=12)

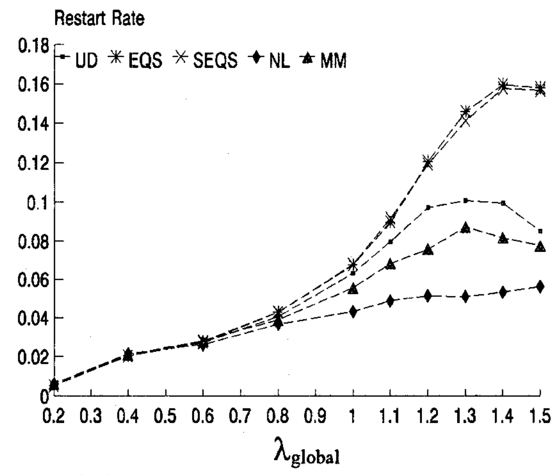

Figure 3: Restart rates

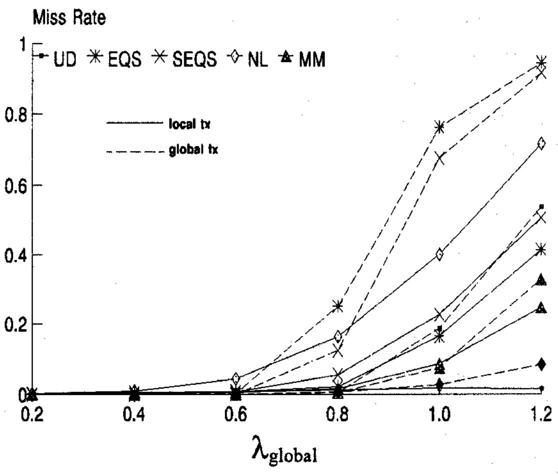

Figure 6: Transaction miss rates (m=12, frac_local=0.25)

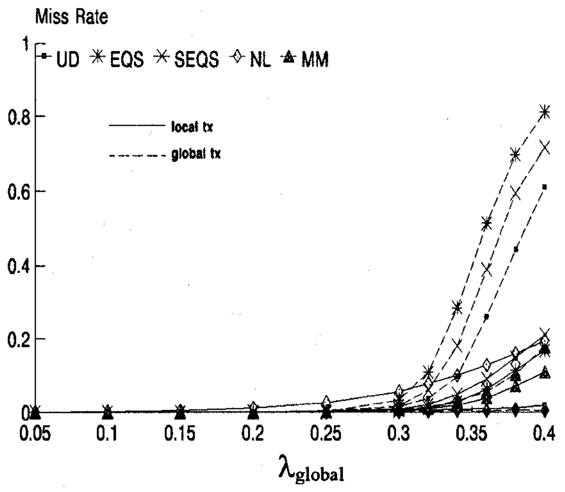

Figure 4: Transaction miss rates (m=12)

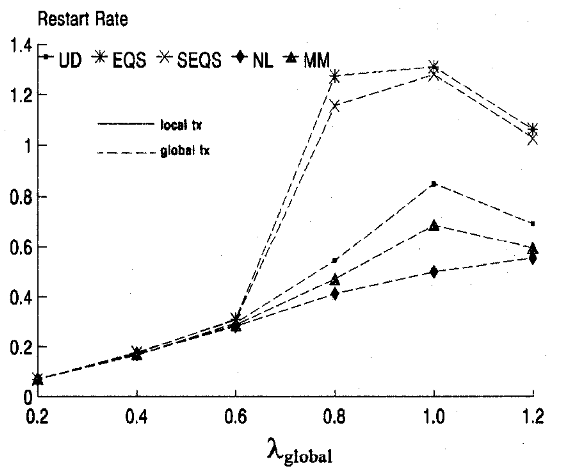

Figure 7: Restart rates (m=12, frac_local=0.25)