Priority Assignment for Sub-transaction in Distributed Real-time Databases

Victor C. S. Lee*, Kam-yiu Lam*, Benjamin C. M. Kao**, Kwok-wa Lam* and Sheung-lun Hung*
*Department of Computer Science, City University of Hong Kong
83 Tat Chee Avenue, Kowloon, Hong Kong
**Department of Computer Science, The University of Hong Kong
Pokfulam Road, Hong Kong

Abstract

Recent studies on deadline assignment to sub-tasks in distributed real-time systems have suggested different heuristics for priority assignment to improve the system performance [6,10]. These heuristics only consider the real-time constraints of the tasks and may not be suitable for distributed real-time database systems (DRTDBS). In this paper, we examine the performance of these heuristics for DRTDBS and suggest better alternatives. Our performance results show that many factors, such as data conflict resolution, transaction aborts and restarts, that are unique to a database system in fact have significant impact on the performance of the heuristics for sub-transaction priority assignment. One of our proposed heuristics, which considers both real-time constraints of the transactions and the impact on data contention, gives the best performance.

1 Introduction

Transactions in a distributed real-time database system (DRTDBS) have constraints on their completion times which are usually expressed as their deadlines [9]. It is vital to the correctness of the system to complete the transactions before their deadlines [3]. In DRTDBS, the processing of a transaction is much more complex than that in single-site RTDBS. It usually has to create a number of sub-transactions to access data objects in different sites [7,13]. The system performance is heavily dependent on the local scheduling of the sub-transactions in different sites [6,10].

In order to maintain the database consistency and to ensure failure atomicity of transactions [2], a real-time concurrency control protocol and an atomic commitment protocol have to be adopted in the systems. In recent years, a number of works are devoted to the study of real-time concurrency control protocols [1,3,4,5,9,14], and to reducing the delay for atomic commitment [11]. The problem of concurrency control in DRTDBS is that the principles and strategies used in the conventional concurrency control protocols for data scheduling are not compatible with those used in the well-known real-time CPU scheduling algorithms. Unbound blocking and priority inversion may occur as a result [7,13]. Different methods for conflict resolution are suggested to resolve this problem [9,14] and to make the scheduling of these protocols as consistent as possible with the real-time CPU scheduling algorithm adopted.

In real-time systems, task deadlines can be ensured by the use of a priority cognitive CPU scheduling algorithm [8,12] and by specially designed hardware resources. One of the more popular CPU scheduling algorithms is earliest deadline first (EDF) in which the transaction with the closest deadline is assigned the highest priority [8]. However, in DRTDBS, the complexity of transactional and system requirements make EDF ineffective. One basic problem is determining the sub-transactions priorities. (E.g., should they use the same priority as their parent transaction?) In recent studies on priority assignment for sub-tasks in distributed real-time systems, it has been found that the system performance in terms of meeting task deadlines can be improved by assigning appropriate priorities to the sub-tasks of a task [6]. However, a DRTDBS has the additional complexity due to the requirements of transactional support protocols such as locking and commit protocols. These protocols interfere with CPU scheduling and in many cases render the priority assignment heuristics ineffective. In this paper, we look at the sub-transaction deadline assignment problem in a DRTDBS. Our goal is to answer the following important questions:

(1) Should we assign the same priority to a sub-transaction for both CPU scheduling and data conflict resolution?
(2) One common method to resolve data conflict is to restart a lower priority transaction in favor of a higher priority one. Restarting a transaction requires the

release of locks from the lower priority transaction (for two phase locking). Which priority should be used for the lock releasing operations?

(3) Which priority should be assigned to a committing transaction? A transaction is committing if it is in its commitment phase and the atomic commitment protocol is being done.

In this study, we use a simulation model to compare the relative performance of the heuristics suggested in [6] and other heuristics suggested in this paper when applied to a DRTDBS. We also look at the impact of various important factors that are unique to a DRTDBS on the system performance. In the model, High Priority Two Phase Locking [1] is adopted for concurrency control and the conventional two phase commitment (2PC) [2] is used for atomic commitment owing to their popularity and simplicity. The rest of the paper is organized as follows. Section 2 reviews the priority assignment heuristics for sub-transactions and suggest new alternatives. Section 3 introduces the DRTDBS model used in our study. Section 4 gives the performance study of the heuristics for sub-transaction priority assignment. The conclusions of the paper and suggestions for future works are in Section 5.

2 Sub-transaction Priority Assignment Heuristics

We consider seven priority assignment strategies divided into two groups: One group considers only the deadlines of transactions. The second group considers also the effect of data contention.

2.1 Based on Real-time Requirements

In [6], different heuristics have been suggested for priority assignment for the sub-tasks based on the real-time requirements of the tasks. These include the ultimate deadlines of their tasks and their slack times. In these heuristics, it is assumed that the expected execution times of the sub-tasks are known and EDF is used for local CPU scheduling. Here, let us briefly summarize these heuristics. In the following description, we assume that a distributed transaction T consists of m sub-transactions, $T_1, T_2, \ldots, T_m$, to be executed in series. We also denote the deadline of a (sub-) transaction X by $dl(X)$. The four heuristics suggested in [6] are:

(1) Ultimate deadline (UD);
(2) Effective deadline (ED);
(3) Equal slack (EQS); and
(4) Equal flexibility (EQF).

The most intuitive way to determine the priorities of the sub-transactions of a transaction is to follow the priority of their parent transaction. Thus, under UD, a sub-transaction T_i is assigned the same priority (deadline) as of its parent transaction. That is,

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

The problem of UD is that it does not consider the execution time of a transaction. A transaction with farthest deadline is given the lowest priority even if it does not have any slack. Under ED, the deadline of a 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 $pex(T_i)$ is the predicted execution time of sub-transaction T_i.

The problem with UD and ED is that they allocate all the current slack time to the current executing sub-transaction. The sub-transactions in the later stages of the transaction may find that they do not have sufficient slack time for their executions. In EQS and EQF, the total slack time of a transaction is shared by all the sub-transactions. Under EQS, the slack is evenly distributed to the remaining sub-transactions. Thus,

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

where $ar(T_i)$ is the arrival time of sub-transaction T_i.

Under EQF, the slack is distributed to the remaining sub-transactions proportional to their remaining predicted execution time. That is,

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

2.2 Considering the Impact on Data Contention

The biggest problem of EQF and EQS is that they do not consider their impact on data contention which can seriously affect the system performance. For example, in EQS and EQF, while a transaction T is waiting, its slack decreases with time. Consequently, 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.

In order to reduce the degree of data contention, we consider alternative priority assignment strategies that avoid intensifying data contention:

(1) Number of Locks held (NL)
(2) Static EQS (SEQS)
(3) Mixed methods (MM)

In NL, the priority of a sub-transaction (*Ti*) is based on the number of locks being held by its parent transaction (*T*). I.e.,

priority of Ti = number of locks being held by T

By giving the highest priority to the transaction which holds the largest number of locks, the transaction can complete faster and release the locks earlier. This reduces the probability of lock conflict.

As we have demonstrated, the use of EQS and EQF in dynamically assign sub-transaction priorities increases the degree of data contention. This problem can be partially solved if sub-transaction priorities are computed once and for all at start time. We call this strategy Static EQS:

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

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. We can inject the idea of NL to the other four heuristics; Sub-transaction priorities can be based on a function which includes both transactions' real-time constraints and the number of locks being held. We call this approach the Mixed Method (MM).

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

where *n* is the number of locks holding by *T*. So, higher priority will be given to the transactions with more locks.

3 Distributed Real-time Database Model

Our distributed real-time database (DRDTBS) model consists of a number of inter-connected sites. It is assumed that the sites are fully connected to avoid complicating our analysis due to the different network configurations. Each site is a local database system which consists of a transaction generator, a scheduler, a CPU, a ready queue, a local database, a communication interface, and a block queue as shown in Figure 1.

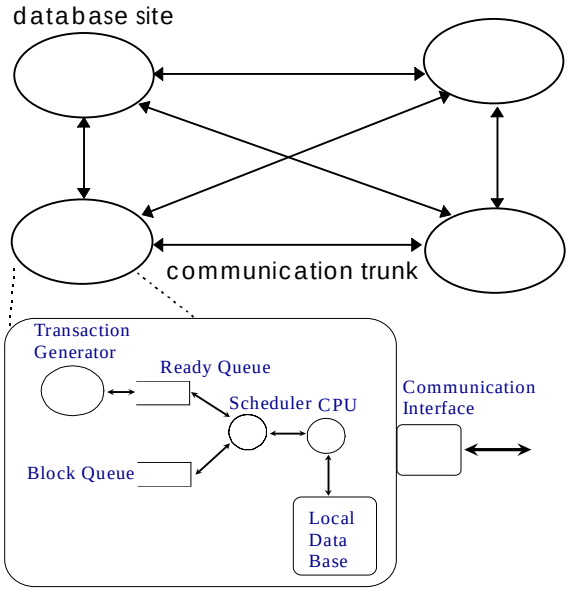

Figure 1: Topology of the Experimental Network and the Site Model

The transaction generator is responsible for the creation of transactions with inter-arrival time following the exponential distribution and is independent of the generators at other sites.

Two types of transactions are defined in the model, global and local. Local transactions only access data objects in its originating site. They create no sub-transactions. Global transactions, on the other hand, consist of a series of sub-transactions. A global sub-transaction consists of one or more operations. The objective is to determine the priorities (deadlines) of the sub-transactions so that the percentage of missed deadlines is kept as low as possible.

The processing of the operations for both local transactions and sub-transactions of global transactions are the same. It involves CPU computations and database accesses. In order to eliminate the impact of disk scheduling to the system performance, we assume that the database is main memory resident [1]. For global transactions, if a sub-transaction requests a remote object, the sub-transaction will be transmitted to, and processed by the remote site through the communication network. To simplify the model, we assume that each sub-transaction has one operation and each operation only accesses one data object.

At each site, the CPU is scheduled using *preemptive* EDF [8]. Outstanding operations are maintained in the ready queue in priority order.

The scheduler is also responsible for scheduling data objects to the operations. The database is partitioned across the different sites with the data objects in a site form a local database. Lock on a data object is residing at the same site as the data object. Before the access to a data object is granted, its lock has to be set in an appropriate mode. High Priority Two Phase Locking (H2PL) is used for concurrency control. That is, whenever a high priority transaction requests a lock which is being held by a low priority transaction, the low priority lock holder will be forced to release the lock and will be restarted. In order to ensure the failure atomicity of transactions, if any one of the sub-transactions has to be restarted or aborted, the parent transaction and all other sub-transactions of that parent have to be restarted or aborted as well. The high priority transaction will be granted the lock once the lock is released by the low priority transaction. If the lock requesting transaction has a lower priority, it will be put in the block queue until the lock is released or is changed to a non-conflicting mode.

The transactions are assumed to be firm real-time transactions [5]. Before they use the CPU or are restarted (due to data conflict), their deadlines are checked. If they have missed their deadlines, they will be aborted immediately, and the locks held by them are released at once.

After the completion of its last sub-transaction, a global transaction enters its commitment phase in which the two phase commit protocol is performed. If all sub-transactions are ready to commit, the parent transaction will decide to commit. After all the sub-transactions and the parent transaction have committed, the transaction is completed.

In order to focus our analysis on the impact of data contention on the sub-transaction deadline assignment strategies, and to prevent the variability of network delay to obscure our results, we model the network as a fully-connected point-to-point network. Each channel is modeled by a delay center. Therefore all kind of messages experience the same transmission delay.

4 Performance Experiments

4.1 Performance Parameters and Measures

The ultimate deadlines of the transactions (both global and local) are defined based on their expected execution times. The ultimate deadline of a global transaction is defined as:

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

The ultimate deadline of a local transaction is defined as:

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

where

T_{gen} : current time of transaction generated;
T_{lock} : time required to lock a data object;
$T_{process}$: time to process a data object;
T_{comm} : a constant time estimated for a transaction going from one site to another;
N_{oper} : number of operations in a transaction;
$N_{transit}$: number of transition across the network required to access all data objects in different sites;
SF : slack factor (uniformly distributed).

We also define *frac_local* to be the fraction of load contributed by local transactions. That is,

$$frac\_local = \frac{k \times \frac{\lambda_{local}}{\mu_{local}}}{m \times \frac{\lambda_{global}}{\mu_{subtransaction}} + k \times \frac{\lambda_{local}}{\mu_{local}}}$$

where,

k : the mean length of the local transactions defined in number of operations;
μ_{local} : the service rate of the local transactions;
$\mu_{subtransaction}$: the service rate of the sub-transactions;
λ_{local} : the arrival rate of the local transactions and
λ_{global} : the arrival rate of the global transactions;

In this study, the execution times of both local transaction and global sub-transaction are the same. Therefore, a *frac_local* of 1/2 means that there are the same number of local transactions and global sub-transactions. However, since a global transaction consists of *m* sub-transactions, local transaction arrival rate is *m* times that of global transactions when $k = 1$.

The most important performance measure used is the miss ratio MR which is defined as follows.

$$MR = N_{md} / (N_{com} + N_{md})$$

where

N_{md} : number of transactions missed the deadline;
N_{com} : number of transactions committed.

The other measure used is rollback frequency which is defined as:

$$\text{Rollback Frequency} = N_{rb} / N_{gen}$$

where

N_{rb}: number of rollbacks occurred;
N_{gen}: number of transactions generated.

Rollback Frequency indicates the cost to commit a transaction and the degree of data contention in the systems.

In our simulation model, a small database is used to create a high data contention environment so that the impact of data conflict on the system performance is more significant. Table 1 summarizes the model parameters and their baseline values.

CPU Scheduling Algorithm	Earliest Deadline First
Concurrency Control	H2PL
Database size / site	200 data objects
T_{lock}	2 msec
$T_{process}$	34 msec
T_{comm}	100 msec
N_{oper}	1 operation for local transaction 4 operations for a global transaction
SF	0.5 to 2.75 (uniformly distributed) for distributed real-time system 2.5 to 13.75 (uniformly distributed) for DRTDBS

Table 1: Model Parameters

4.2 Performance Results

In this section we report the results and findings of our simulation experiments. Before comparing the heuristics' performance, lets us discuss how data conflict is resolved, and what priority should be given to lock releasing operations. It has been found that if we use the priorities of the sub-transactions for data conflict resolution (the H2PL protocol), the performance of the system is very poor. It is because cyclic transaction restart and deadlock may occur as the relative priorities of the sub-transactions of different transactions can be very different. So, in the following experiments, we use the ultimate deadlines for data conflict resolution.

When a transaction is aborted, the locks it holds should be released. In our experiment, we assign the highest priority to rollback operations to reduce transaction blocking time due to data conflict.

In the remaining part of this section, we will discuss the performance of the heuristics mentioned in section 2 in more detail. Figure 2 shows the performance of the four heuristics: UD, ED, EQS, and EQF applied to a distributed real-time system in which the problem of data conflict are not handled. Transactions are granted a requested lock even if it is in a conflicting mode against another transaction. We modeled our experiment as closely as that reported in [6]. This experiment can therefore be used to validate the simulator and to confirm the results found in [6]. In our figures, solid lines and dashed lines represent the missed ratio of local transactions and global transactions respectively. The dotted lines represent the rollback frequency of global transactions. From Figure 2, we can see that the performance of the heuristics on global transactions follows the same order as that in [6]. That is, EQF and EQS perform similarly, they are better than ED which in turn, is better than UD. For local transactions, only a small increase in the miss ratio is found across the heuristics. The difference in the absolute values of the results (as compared with that in [6]) may be attributable to the database operations (such as 2PL) that are absent in a distributed system. As a whole, the performance result is consistent with that in [6].

Figure 3 shows the performance of the heuristics when applied to a DRTDBS when data conflict is resolved by H2PL. We see that the performance is similar to that shown in Figure 2. This can be explained by considering the rollback frequency (Figure 4). From Figure 4, we see that the rollback frequencies stay at a relatively low level (< 8%) for all heuristics. This is because in the baseline experiment, a global transaction only consists of four sub-transactions (N_{oper} = 4). Data conflicts are thus infrequent. Without data contention, the behavior of a DRTDBS resembles a distributed system with resource contention only.

To investigate the performance of the heuristics when applied to a DRTDBS with high data contention, long global transactions (N_{oper} = 12) are introduced. Figure 5 shows the results obtained. Since the size of the global transactions is increased, the mean global transaction arrival rate has to be reduced to maintain a similar loading. Now, the rollback frequencies are increased to large values as shown in figure 6. The results show that the four heuristics mentioned in [6] become very ineffective. To further illustrate the impact of data contention on the heuristics' performance, we reduce the fraction of local transactions (*frac_local*) from 0.75 to 0.25. In [6], it is reported that reducing *frac_local* reduces the performance differences among the four heuristics, but their relative order stays the same (I.e., EQF and EQS are better than UD and ED). In our experiment (Figures 7 and 8), surprisingly, EQS and EQF perform worse than UD and ED. The performance degradation of EQS and EQF is due to the increased number of long global transactions, which intensifies data conflict.

As a result, we can see that the differences between a DRTDBS and a distributed real-time system have major impacts on the effectively of the four heuristics. The point is that in a DRTDBS, an aggressive transaction which can get much of the resource does not mean that it can meet the deadline. However, the aggressiveness of this transaction may already harmfully block other transactions in due course and lead to an increase in MR.

Taking data contention into consideration, the other three heuristics mentioned: NL, SEQS, and MM have the potential of better performance. In Figure 9, using UD as a base case for comparison, we find that NL reduces the missed rate of global transactions pretty effectively. However, the side effect is a much higher local transaction missed rate. The reason for a smaller global transaction missed rate is that once a global transaction get more than one lock, its priority will be higher than all local transactions (which only request one lock).

For SEQS, the performance is even worse than that of UD. The problem is that it does not reduce data contention (comparing Figures 8 and 10) from EQS, and it does not consider enough real-time constraints of the transactions. The best performance can be found with MM in which both real-time constraints of the transactions and the impact on data contention are considered. The Rollback Frequency of the heuristics is shown in Figure 10. Consistent with our expectation, the amount of data contention is highest with SEQS and lowest with MM.

5 Conclusions and Future works

In this study, the application of various sub-transaction deadline assignment heuristics on a DRTDBS [6] is examined. When data contention is low, the performance of these heuristics is consistent with that obtained from a distributed real-time system. However, these heuristics become ineffective when data contention is non-trivial. This occurs when the fraction of global transactions increases or when the size of global transactions becomes large. A prima facie reason is the non-rewarding impact of dynamically changing or increasing the priority of a sub-transaction in the course of execution. This increases the degree of data contention and thus affects the system performance. To reduce data contention, new heuristics are suggested. Our performance results indicate that heuristic MM, which considers both transaction real-time constraints and the impact on the degree of data contention gives the best overall performance.

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] Bestavros, A. “Advances in Real-Time Database Systems Research”, *ACM Sigmod Record*, vol. 24, no. 1, 1995.

[4] Haritsa, J. R., Carey, M. J., and Livny, M., "Data Access Scheduling in Firm Real-Time Database Systems," *Journal of Real-Time Systems*, vol. 4, no. 3, pp. 203-42, 1992.

[5] Huang, J., Stankovic, J., Ramamritham, K. and Towsley, D., “Priority Inheritance in Soft Real-time Databases” *Journal of Real-Time Systems*, vol. 4, no. 3, pp. 243-268, 1992.

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

[7] Lam, Kam-yiu, “Cocnurrency Control in Distributed Real-time Database Systems” Ph.D. Thesis, Department of Computer Science, City University of Hong Kong, Hong Kong, 1994.

[8] 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.

[9] 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.

[10] Purimetla, B., Sivasankaran, R. M., Stankovic, J. A., Ramamritham, K., Towsley, D., "Priority Assignment in Real-Time Active Databases," *Proceedings of 3rd International Conference on Parallel & Distributed Information Systems*, pp.176-84, 1994.

[11] Son, S. H., and Kouloumbis. "A Token-Based Sychronization Scheme Using Epsilon-Serializability and its Performance for Real-Time Distributed Database Systems," *Proceedings of 3rd International Symposium on Advanced Applications*, 1993.

[12] Stankovic, J., Spuri, M and Natale, M.D., “Implications of Classical Scheduling Results for Real-Time Systems”, *IEEE Computer*, vol. 28, no.6., pp. 16-25, 1995.

[13] Ulusoy, O. “Processing of Real-time Transactions in a Replicated Database Systems”, Journal of Distributed and Parallel Databases, vol. 2, no. 4, pp. 405-436 1994.

[14] 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.

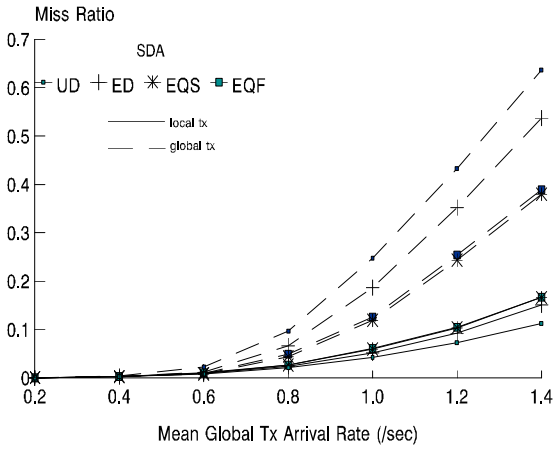

Figure 2: Distributed Real-time System

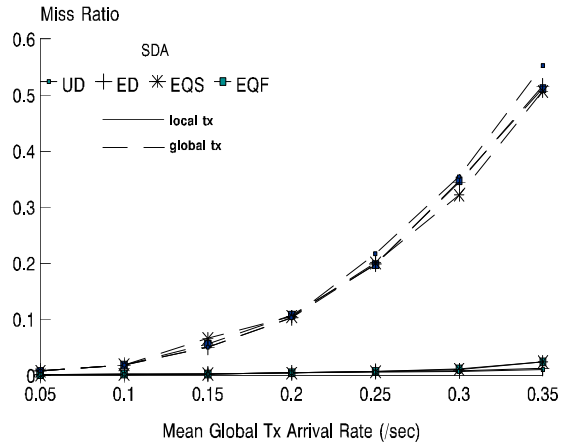

Figure 5: DRTDBS with Tx Size=12 (Miss Ratio)

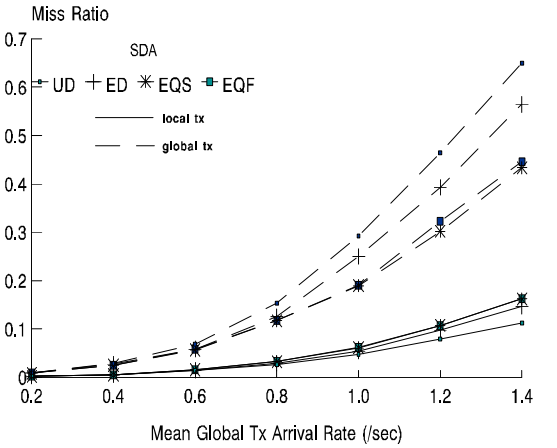

Figure 3: DRTDBS (Miss Ratio)

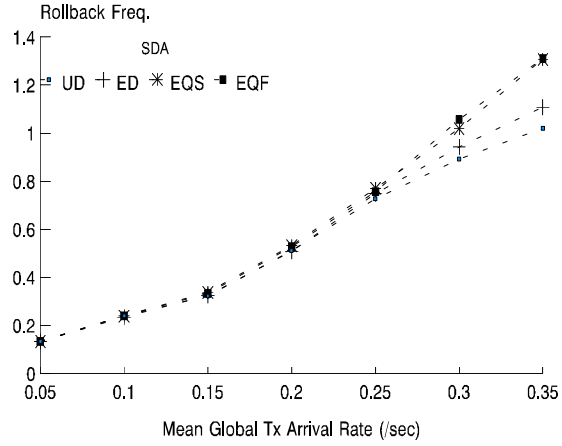

Figure 6: DRTDBS with Tx Size=12 (RB Freq.)

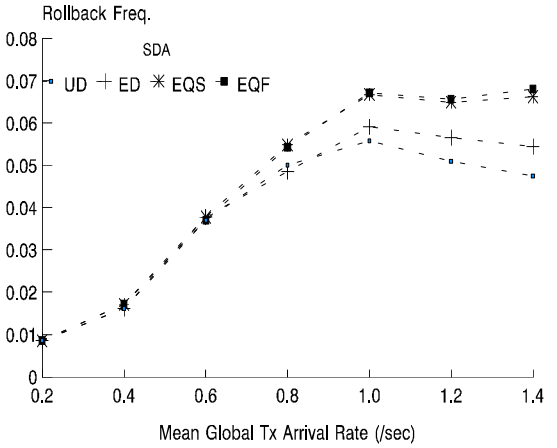

Figure 4: DRTDBS (Rollback Frequency)

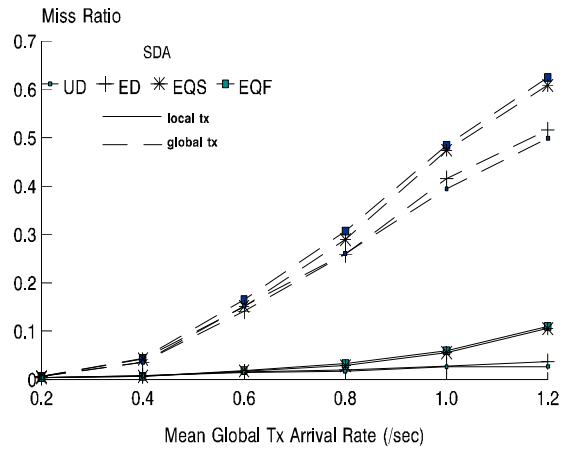

Figure 7: DRTDBS with *frac_local*=0.25 (Miss Ratio)

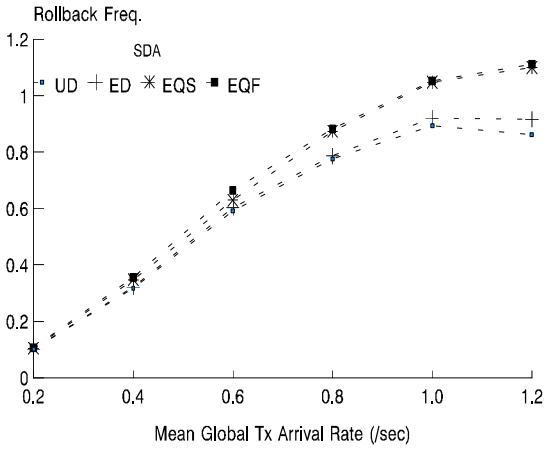

Figure 8: DRTDBS with *frac_local*=0.25 (Rollback Frequency)

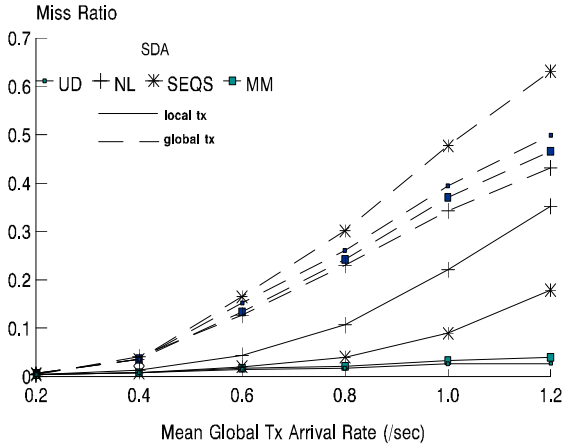

Figure 9: New Heuristics (Miss Ratio)

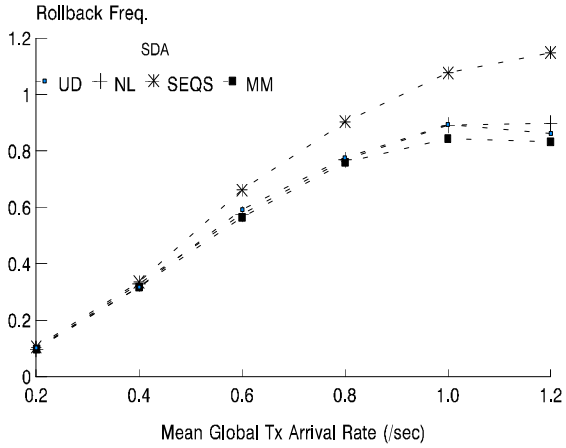

Figure 10: New Heuristics (Rollback Frequency)