No. 900

Transaction Scheduling with Temporal Data in Real-Time Database Systems

by

Yongbing ZHANG

February 2001

Transaction Scheduling with Temporal Data in Real-Time Database Systems

Yongbing ZHANG
Institute of Policy and Planning Sciences, University of Tsukuba,
Tsukuba Science City, Ibaraki 305-8573, Japan

Abstract

In a real-time database system, timing constraints are associated with transactions and data accessed by transactions may be valid for only specific time intervals. A correct transaction processing means that the transaction completes its processing before its timing constraints and uses data that are both absolutely and relatively timing-consistent. We define the two distinct performance indeces as *timing constraint* and *data consistent constraint*. It is ideal that an algorithm both improve the two indeces at the same time, but they may affect each other. In this paper, we explore that whether we can improve them simultaneously and, if not, the trade-off.

The simulation results show that by taking advantage of the data temporal consistency constraints in transaction scheduling improve significantly the system performance (miss percentage and temporal inconsistency percentage). The results also show that some system parameters, such as the period of a transaction and the *relative validity interval* of a data set, have strong effects on the system performance.

1 Introduction

In a real-time system, a result must not only be functionally correct but also be delivered by a deadline. A result produced too late becomes less useful or useless and, in some cases, may even cause severe damage to property or life. Applications in the latter case

are called *hard* real-time applications, while applications in the former case are called *soft* or *firm* real-time applications. Typically, the constraints of a real-time task include its ready time and deadline, as well as the data temporal consistency. In a real-time database system, a transaction begins its processing after its ready time and must complete before its deadline. Data objects accessed by a transaction reflects the state of the real world and continuously changes as the state of the real world changes. A transaction therefore must read sufficiently current data in order to deliver the correct results and the data objects accessed by the transaction should be sampled relatively close to each other. Data temporal consistency, defined in terms of the *absolute* and *relative consistency*, is concerned with the time characteristics of the data objects involved in the computations. A temporal data is *absolutely valid* only during a given interval and becomes *absolutely inconsistent* after its *absolute validity deadline*. A set of temporal data objects sampled within a given interval, that is, its *relative validity interval* is *relatively valid*; otherwise the date set becomes *relatively inconsistent*.

Even though there are many studies focusing on real-time database systems, most of them took only the timing constraints of transactions into consideration in transaction scheduling [8, 1, 4, 5, 6, 10, 9, 14, 17]. Recently, some authors studied the problems of data temporal consistency [12, 13, 7, 15, 16, 2]. The pessimistic and optimistic concurrency control algorithms were examined in maintaining data consistency for a hard real-time system in [13]. The absolute consistency constraint was taken into account in transaction scheduling for a firm real-time system in [15].

In this paper, a hard real-time system model similar to that in [13] was studied. A type of priority-driven scheduling algorithms, known as earliest-deadline-first (EDF) algorithm [8], with an optimistic concurrency control algorithm was examined. The EDF algorithm assigns the priorities to transactions according to their deadlines in execution. The original EDF algorithm was firstly extended to take the absolute consistency constraint of data objects into consideration in transaction scheduling. The absolute validity deadline of a data object read by a transaction is viewed as the deadline of the transaction

as well as the timing deadline of the transaction. Then, a waiting scheme was proposed for maintaining the relatively temporal consistency. When the read set of a transaction is relatively inconsistent then the transaction tries to postpone its execution and wait for the update of data objects in its read set. Simulations were carried out to evaluate the performance of these scheduling algorithms. The performance metrics are the percentage of temporally inconsistent (absolutely or relatively inconsistent) transactions and the percentage of transactions missing their deadlines (miss percentage for short).

2 Real-Time Database System Model

2.1 Real-Time Data Object Model

In the model under consideration, the system consists of a set of data objects representing the state of the real-world. There are two types of data objects in the system: *continuous* and *discrete*. Continuous data objects are related to real-world objects continuously changing with time. Discrete data objects are static in the sense that their values do not become obsolete as time passes. Depending on how their values are acquired, continuous data objects can be further classified to *image objects* and *derived objects*. Image objects are models of real-world objects and they are sampled periodically by sensors. An instant at which the value of a real-world object is sampled is called a *sampling time*. The sampled value of a real-world object at a sampling time is written to an image object stored in the system. Each image object x is stamped with the sampling time of the corresponding real-world object X. The value of a derived object is computed from the values of a set of image objects and/or other objects.

An image data object associates with a *time stamp* and an absolute validity interval. The time stamp shows when the value of the object was obtained. The absolute validity interval shows the length of time during which the value of the object is considered to be valid. The value of an image object k achieves absolutely temporal consistency only when $t_{now} - t_k \leq avi_k$, where t_{now} is the current time, t_k is the time stamp of object k, and avi_k is the absolute validity interval of object k. The absolute validity deadline of a data object

is used to show the time after which the value of the object becomes invalid. The absolute validity deadline of data object k, dd_k, is given by $dd_k = t_k + avi_k$. In addition to the time stamp and the absolute validity interval, a derived object associates with a relative validity interval showing the allowable dispersion of the time stamps between data objects in the set of data objects used to derive the new object. For such a derived object k and a set of data objects S_k used to derive object k, S_k achieves relatively temporal consistency when the time stamp difference between two data objects i and j in S_k is not greater than the relative validity interval rvi_k; that is $|t_i - t_j| \leq rvi_k, i, j \in S_k$.

The absolute and relative validity intervals reflect the temporal consistency requirements of an application. They show how current and close in time the data objects must be for the results of computations based on them to be considered correct. A continuous data object is in a correct state as long as the value of the object satisfies both the absolutely and the relatively temporal consistency constraints as well as the given integrity constraints, while a discrete data object is in a correct state if and only if the value of the object is logically consistent. A set of data objects is said to be temporally inconsistent if they are either absolutely or relatively inconsistent.

A continuous data object is assumed to have multiple versions and may have multiple valid versions simultaneously [13]. A new version comes into existence when a new value is written. This version is dated with a time stamp showing when it is obtained. The versions of an image object are called *image versions*, whereas the versions of a derived object are called *derived versions*. Each version of a data object has its own absolute validity deadline. In the model under consideration, a real-time database R consists of the following data objects: a set of image objects $X = \{x_1, x_2, ..., x_M\}$, a set of derived objects $Y = \{y_1, y_2, ..., y_N\}$, and a set of discrete data objects $Z = \{z_1, z_2, ..., z_Q\}$. A set of data objects which is used to compute the value of a derived object y is denoted as $S_y = \{s_1, s_2, ..., s_i, ..., s_R\}, s_i \in X \cup Y \cup Z, 1 \leq i \leq R$.

2.2 Real-Time Transaction Model

Transactions are assumed to be *periodic*; that is, a sequence of transactions are carried out at regular intervals [13]. In each period, a transaction can be started after the ready time and should be completed before its deadline. It is assumed that the ready time of a transaction is the beginning of each period and the deadline of a transaction is the beginning of the next period. A transaction is stamped with a time stamp showing its startup time. During its execution, a transaction reads and/or writes the same set of data objects as the transactions in other periods. A transaction τ can be characterized by five parameters $a_\tau, p_\tau, c_\tau, RS_\tau$, and WS_τ, where a_τ is the arrival (or release) time, p_τ is the period, and c_τ is the execution time. RS_τ and WS_τ are the *read set* and *write set* of the transaction, respectively; that is, they are the data objects the transaction may read and write in every period.

Transactions are classified as being *read-only*, *write-only*, and *update*. A write-only transaction models the periodic sampling of the reading of a sensor and updating of the sensor values. It does not read any data object and periodically writes a sampled value of a real-world object to the corresponding image in the system (i.e., $WS_\tau \subset X, RS_\tau = 0$). An update transaction reads a set of data objects, computes and writes to derived objects (i.e., $RS_\tau \subset X \cup Y \cup Z, WS_\tau \subset Y$). An update transaction never writes to any image object and a write-only transaction never writes to any derived object. It is assumed that an update transaction writes to only one derived object. The read set of an update transaction τ, RS_τ, therefore corresponds to the data set used to compute the corresponding derived object y, S_y. A read-only transaction retrieves the values of a set of data objects but does not write to any data object (i.e., $RS_\tau \subset X \cup Y \cup Z, WS_\tau = 0$).

3 Concurrency Control and Transaction Scheduling

The concurrency control algorithm examined here is an optimistic concurrency control algorithm. As in [13], the algorithm supports multiversion data based on the well-known version pool algorithm [3] and maintains *weak consistency*: Update transactions are guar-

anteed to be executed in a serializable manner, but read-only transactions see a consistent view of the database and can read any versions of data objects with a time stamp less than or equal to its startup time stamp. It is assumed that a read-only transaction reads the most current (committed) version that has a time stamp less than or equal to the startup time stamp of the transaction.

A write-only transaction does not read any data objects. Their write sets are disjoint from each other and from the write set of any update transaction and, therefore, they have no conflicts with any other transactions. When a write-only transaction writes an image in each period, it creates a new version of the image. Read-only transactions read the most recent versions of data objects and do not write to any data objects and, therefore, conflicts with update transactions are eliminated. Update transactions, on the other hand, may have read/write conflicts with each other.

With the optimistic concurrency control under consideration, an update transaction has three phases to commit: a *read phase*, a *validation phase*, and a *write phase*. In each period, an update transaction τ reads the most recent versions of the data in its read set without locking the data. Transaction τ creates a new version, in its own workplace, of each object it will write later. When transaction τ is ready to commit, it enters the validation phase. The validation test check if any data object written by τ has been read by any other update transaction σ that is currently in its read phase, that is, whether WS_τ and RS_σ overlap ($WS_\tau \cap RS_\sigma \neq 0$). Any conflicting update transaction (such as σ) found is immediately aborted and restarted. Transaction τ then enters its write phase. In the write phase, the new version of each object in the local workplace of transaction τ becomes permanent in the system.

Two well-known scheduling algorithms, rate-monotonic (RM) and first-deadline-first (EDF), are used as a baseline and for comparison purpose. In RM, higher priorities are assigned to transactions with shorter periods and the priorities are determined before execution. In EDF, on the other hand, the priorities are determined during execution according to the timing deadlines of transactions and higher priorities are assigned to trans-

actions with earlier deadlines [8]. The RM and EDF algorithms, however, ignore the data temporal consistency in transaction scheduling. Two extensions of the EDF algorithm, earliest-data-deadline-first (EDDF) and earliest-data-deadline-first-wait (EDDF-W) algorithms, which are cognizant of data temporal consistency requirements are examined.

The EDDF algorithm was proposed in [16] for a firm real-time database system and in this paper is extended for the hard read-time model. In EDDF, in addition to the timing deadline of a transaction, the absolute validity deadline of a data object in the read set of the transaction is also viewed as the deadline of the transaction. The deadline of transaction τ is therefore determined by $\min(dd_k, d_\tau)$ where $k \in RS_\tau$ and d_τ is the timing deadline of transaction τ. A higher priority is assigned to a transactions with a smaller $\min(dd_k, d_\tau)$.

The EDDF-W algorithm proposed in this paper attempts to reduce the percentage of relatively temporal inconsistency by using a simple waiting mechanism. When a transaction find that its read set is relatively inconsistent it tries to postpone its execution and wait for the update of the data object with the oldest time stamp in its read set. It checks whether the waiting still meets its timing deadline and whether the update makes its read set satisfy the relative consistency constraints. It will postpone its execution if it is true, otherwise it marks itself as relatively inconsistent and continues its execution. Two cases under the mechanism for updating the data object with the oldest time stamp in the read set of transaction τ are shown in Figs. 1 and 2. There is a transaction τ which has two data objects in its read set, s_i and s_j. In Fig. 1, transaction τ finds that the difference of the time stamps of s_i and s_j is greater than the relative validity interval; that is, $t_2 - t_1 > rvi_\tau$, and that after the update of s_j the difference becomes not greater than the relative consistency interval; that is, $t_3 - t_2 \leq rvi_\tau$. It therefore decides to wait for the update of s_j. Transaction s_j inherits the priority of transaction τ and is executed immediately [1] . Transaction s_j triggers the execution of transaction τ once it completes its execution. Fig. 2 shows the case where transaction τ has to wait for the update of s_j

[1] For simplicity, s_j is used to denote both a data object and the transaction writing to the data object s_j

until the next coming period for s_j.

4 Simulation Results

This section presents the assumptions made in the experiments and the results obtained in simulation. The transactions are periodic and their periods were chosen, as in [13], over a uniformly distributed interval $[1, P_{ratio}] * P_{base}$, where P_{base} is the baseline period and was set to 100 time units, and P_{ratio} is the ratio of the longest period to the baseline period and was set from 2 to 50. There were 10 update transactions and 10 write-only transactions in simulation and all transactions were started in phase; that is, the first periods of all the transactions begin at the same time. The write-only transactions are believed to need very little processor attention and therefore was set to one time unit and was assigned to the highest priority. It was assumed that update transactions read a set of data objects and write to only one derived object, and conflict with each other. It is assumed that the most recent versions of data objects are used for read-only transactions. A version selection scheme may be used to provide a proper version for a read-only transaction so that the resultant dispersions read by the transaction should be better than the results here indicate. The absolute validity interval of a data object written by transaction τ, avi_τ, was set to $2p_\tau$ and the relative validity interval for a derived object written by transaction σ, rvi_σ, was set to $2 \max_i(p_i)$ where $i \in RS_\sigma$.

The utilization of the update transactions was determined as in [13] using the following three distributions: EQ where the utilization of every update utilization is U/m (U is the total utilization of the update transactions and m is the number of update transactions); LH where a transaction i with a longer period is assigned to a higher utilization determined by $Up_i / \sum_{j=1}^{m} p_j$; and SH where a transaction i with a shorter period is assigned to a higher utilization determined by $(G - p_i) / \sum_{j=1}^{m} p_j, G = 2 \sum_{j=1}^{m} p_j / m$. The total number of transactions in the simulation is denoted by N. The number of inconsistent transactions that read either absolutely or relatively inconsistent data and the number of transactions that missed their deadlines are denoted by N_{in} and N_{miss}, respectively. The inconsis-

tency percentage and the miss percentage are therefore equal to N_{in}/N and N_{miss}/N, respectively.

Figs. 3–6 show the performance of the scheduling algorithms when the utilization distribution is LH. The results show that the inconsistency percentage and the miss percentage are improved significantly under EDDF and EDDF-W. From Figs. 3 and 5, it is observed that when the period ratio becomes larger the breakdown utilization [11], from which point the inconsistency occurs, becomes smaller. When the period ratio, P_{ratio}, is 50, as shown in Fig. 5 the breakdown utilization of EDDF and EDDF-W is close to that of EDF, even though the inconsistency percentages of EDDF and EDDF-W increase slowly afterward. It is also observed that the performance of EDDF is quite close to that of EDDF-W. Because the relative validity interval, rvi_τ, was set to be large enough (equal to $2\max_i(p_i)$, as described before) and therefore there were seldom cases happened where the relative consistency constraints were violated.

Figs. 7 and 8 show the effects of read-only transactions on the performance of EDDF-W when P_{ratio} is 50: read-only transactions are 0, 20 and 50% of the sum of read-only and update transactions. The performance of EDDF is quite close to that of EDDF-W and therefore was not shown here. The periods of read-only transactions were chosen randomly. In such a situation, a read-only transactions may preempt an update transaction or be preempted by an update transaction, but no update transaction will be aborted even when it is preempted by a read-only transaction. As shown in Fig. 7, the breakdown utilization becomes much larger as the percentage of read-only transactions increases.

Figs. 9 and 10 show the performance comparison of the EQ, LH, and SH distributions under EDDF-W when $P_{ratio} = 50$. Other parameters were fixed as in Figs. 3–6. The results obtained here are similar to those obtained in [12]. The SH distribution provides the smallest inconsistency percentage over a wide range of utilization, but when the utilization is close to 1 the inconsistency percentage becomes sharply large. At high utilization, the execution times of short period transactions are so long that they can hardly meet their deadlines, resulting in high inconsistency percentage. When the distribution is LH, longer

period transactions with longer execution times are likely to be preempted many times before they finish, resulting in violating the data consistency and deadline constraints.

Figs. 11–14 show the performance of EDDF-W in comparison with EDDF for various period ratios (from 50 to 2) and various relative validity intervals (p_τ, $2p_\tau$, and $\max_i(p_i)$ where $i \in RS_\tau$). The performance metric used here is the relative inconsistency percentage, that is, the ratio of the number of relatively inconsistent transactions to the number of the total transactions. It is observed that the period ratio and the relative validity interval have strong effects on the relative inconsistency percentage. When the period ratio P_{ratio} becomes large or when the rvi_τ becomes small the relative inconsistency percentage increases. It is observed that when the utilization is not high (below 0.6) the EDDF-W improves the relative inconsistency percentage over EDDF in all cases. When P_{ratio} is 2, on the other hand, EDDF-W behaves worse than EDDF if the utilization becomes large (above 0.6).

An attempting at improving the relative inconsistency may malign other performance metrics (the absolute inconsistency percentage or the miss percentage). The relationship of these performance metrics was examined as shown in Figs. 15–18 when P_{ratio} is 10. It is observed that EDDF-W behaves better than EDDF when rvi_τ is relatively large (equal to $\max_i(p_i)$ where $i \in RS_i$). Both the absolute and the relative inconsistency percentages are improved while the miss percentage remains almost unchanged. It is observed, on the other hand, that when rvi_τ becomes small (equal to $2p_\tau$), the absolute inconsistency percentage and the miss percentage were sacrificed for the improvements of the relative inconsistency percentage. It is observed that a shorter relative validity interval leads to a higher absolute inconsistency percentage. When rvi_τ is small, a transaction often finds that its read set is relatively inconsistent and tries to wait for the update of its read set. The waiting may leave the system resource idle. Additionally, if a transaction with higher priority comes into existence during the update process, the waiting transaction may be preempted, resulting in missing its deadline.

4 Conclusion

In this paper, transaction scheduling algorithms, EDDF and EDDF-W, that take the absolute and the relative consistency of temporal data objects were examined using simulation. To maintain the absolute consistency, the absolute validity deadline of a data object read by a transaction is also viewed as the deadline of the transaction. To maintain the relative consistency, a simple waiting scheme is employed where a relatively inconsistent transaction tries to postpone its execution and wait for the update of data objects in its read set.

The simulation results show that both EDDF and EDDF-W significantly improve the inconsistency percentage and the miss deadline percentage over either RM or EDF. The comparison of EDDF and EDDF-W show that when the relative validity interval is large (e.g., equal to $\max_i(p_i)$ where $i \in RS_i$) EDDF-W improves the inconsistency percentage over EDDF while the miss percentage remains almost unchanged. When the relative validity interval becomes small (e.g., equal to or less than $2p_\tau$), on the other hand, the improvement on the relative inconsistency percentage may sacrifice the absolute inconsistency percentage and the miss percentage. It therefore needs to take a trade-off into account between the performance metrics under such a situation. The relative inconsistency percentage can also be improved by selecting appropriate versions of data objects. Such issues will be studied in the future work. The results obtained in this study may give a guideline for how to maintain the temporal consistency and which system parameters play main roles in maintaining temporal consistency in a real-time database system.

References

[1] R.K. Abbott and H. Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Trans. Batabase Sys.*, 17(3):513–560, September 1992.

[2] N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. Data consistency in

hard real-time systems. Technical Report YCS-93-203, University of York, 1993.

[3] A. Chan and R. Gray. Implementing distributed read-only transactions. *IEEE Trans. Softw. Eng.*, SE-11(2):205–212, 1985.

[4] A. Datta, S. Mukherjee, P. Konana, I.R. Viguier, and A. Bajaj. Multiclass transaction scheduling and overload management in firm real-time database systems. *J. Information Systems*, 21(1):29–54, 1996.

[5] J.R. Haritsa, M.. Carey, and M. Livny. Data access scheduling in firm real-time database systems. *J. Real-Time Systems*, 4:203–241, 1992.

[6] J.R. Haritsa, M. Livny, and M.J. Carey. Earliest deadline scheduling for real-time database systems. In *Proc. IEEE Real-Time Systems Symp.*, pages 232–242, 1991.

[7] Y.K. Kim and S.H. Son. Supporting predictability in real-time database systems. In *Proc. IEEE Real-Time Technology and Applications Symp.*, pages –, 1996.

[8] C.L. Liu and J.W. Layland. Scdeduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[9] H.H. Pang, M. Livny, and M.J. Carey. Transaction scheduling in multiclass real-time database systems. In *Proc. IEEE Real-Time Systems Symp.*, pages 23–34, 1992.

[10] R. Sivasankaran, J. Stankovic, D. Towsley, B. Purimetla, and K. Ramaritham. Priority assignment in real-time active databases. *VLDB Journal*, pages 19–34, January 1996.

[11] X. Song and J. Liu. Performance of multiversion concurrency control algorithms in maintaining temporal consistency. In *Proc. 14th Annual Int. Computer Software and Applications Conf.*, pages 132–139, 1990.

[12] X. Song and J. Liu. How well can data temporal consistency be maintained. In *Proc. 1992 IEEE Symp. Computer-Aided Control System Design*, pages 275–284, 1992.

[13] X. Song and J. Liu. Maintaining temporal consistency: Pessistic vs. optimistic concurrency control. *IEEE Trans. Knowledge Data Eng.*, 7(5):786–796, October 1995.

[14] S. Thomas, S. Seshadri, and J.R. Haritsa. Integrating standard transactions in firm real-time database systems. *J. Information Systems*, 21(1):3–28, 1996.

[15] M. Xiong, K. Ramamritham, J. Stankovic, D. Towsley, and R. Sivasankaran. Maintaining temporal consistency: Issues and algorithms. In *Proc. 1st Int. Workshop on Real-Time Databases*, pages –, 1996.

[16] M. Xiong, R. Sivasankaran, J. Stankovic, K. Ramamritham, and D. Towsley. Scheduling transactions with temporal constraints: Exploiting data semantics. In *Proc. 17th IEEE Real-Time Systems Symposium*, pages 240–251, 1996.

[17] P.S. Yu, K.L. Wu K.J. Lin, and S.H. Son. On real-time databases: Concurrency control and scheduling. *Proc. of the IEEE*, 82(1):140–157, January 1994.

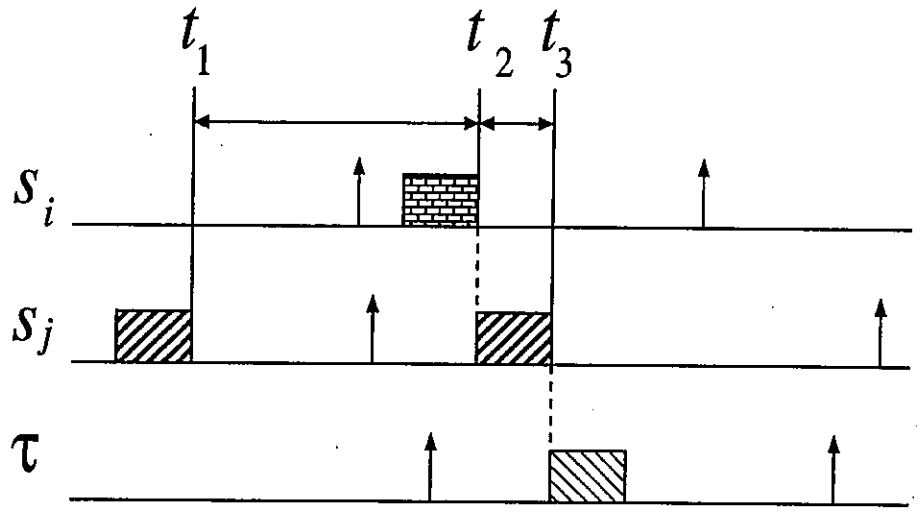

Figure 1: Update of s_j if transaction s_j has not been executed when transaction τ arrives.

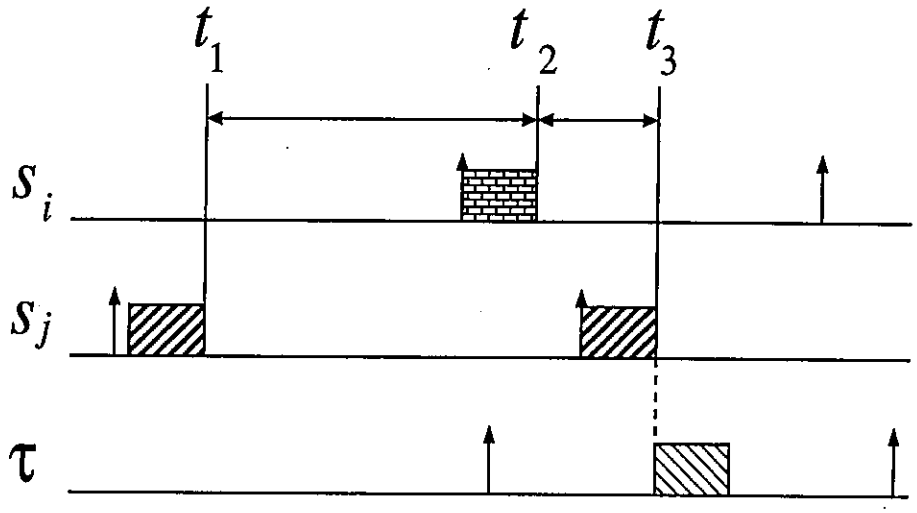

Figure 2: Update of s_j when transaction τ has to wait until the next coming period of s_j.

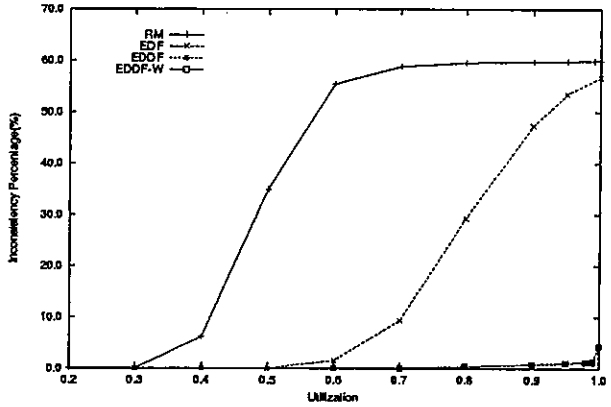

Figure 3: Inconsistency percentage (UtilDist: LH, $r = 0$, $P_{ratio} = 10$)

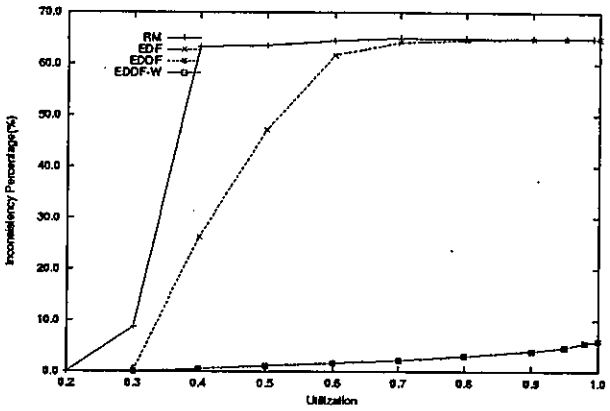

Figure 5: Inconsistency percentage (UtilDist: LH, $r = 0$, $P_{ratio} = 50$)

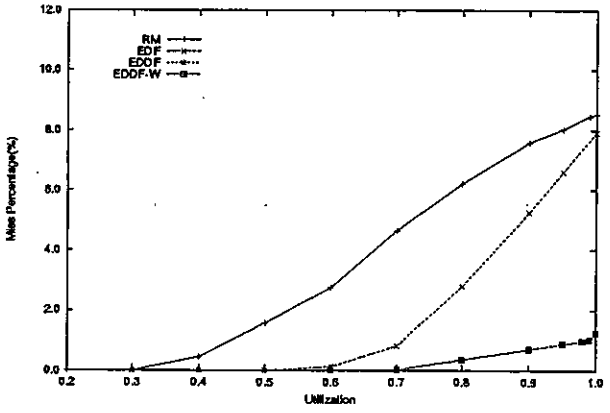

Figure 4: Miss percentage (UtilDist: LH, $r = 0$, $P_{ratio} = 10$)

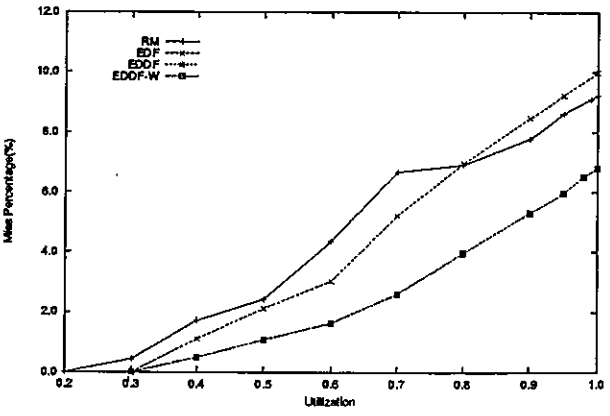

Figure 6: Miss percentage (UtilDist: LH, $r = 0$, $P_{ratio} = 50$)

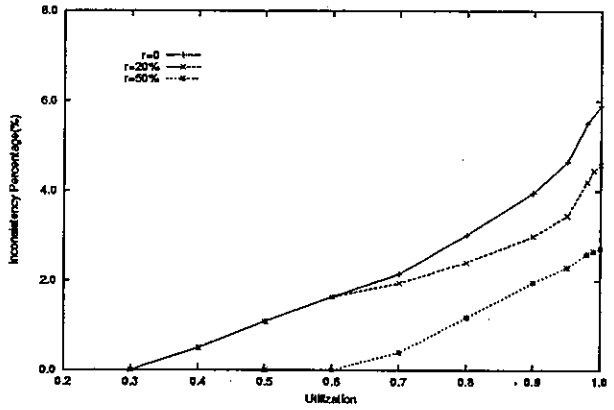

Figure 7: Effects of read-only transaction percentage on inconsistency percentage under EDDF-W (UtilDist: LH, $P_{ratio} = 50$)

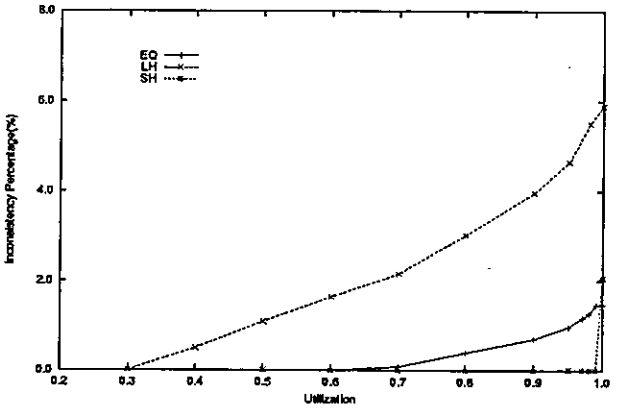

Figure 9: Inconsistency percentage for various utilization distributions under EDDF-W ($r = 0$, $P_{ratio} = 50$)

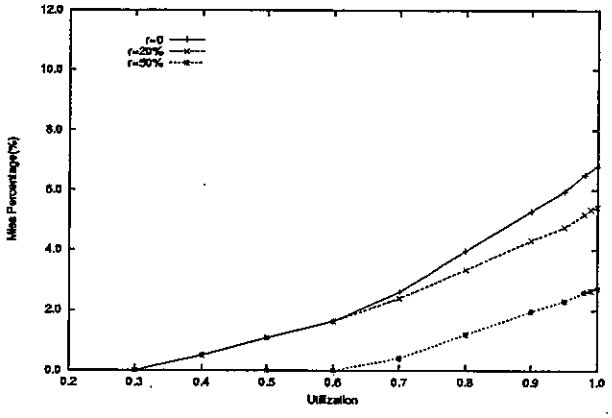

Figure 8: Effects of read-only transaction percentage on miss percentage under EDDF-W (UtilDist: LH, $P_{ratio} = 50$)

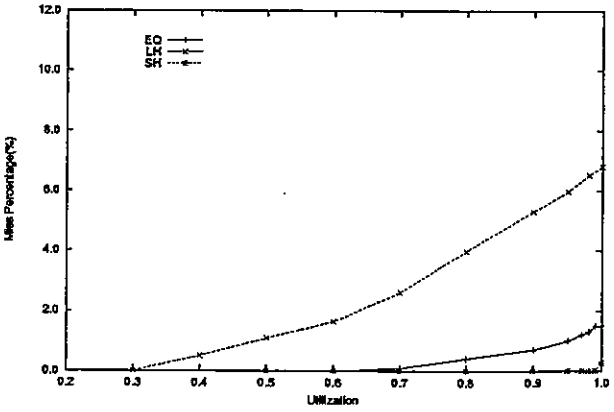

Figure 10: Miss percentage for various utilization distributions under EDDF-W ($r = 0$, P_{ratio}=50)

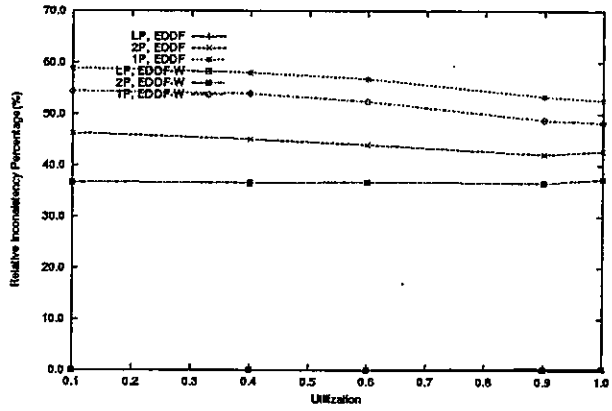

Figure 11: Relative inconsistency percentage under EDDF-W (UtilDist: LH, $r = 0$, $P_{ratio} = 50$)

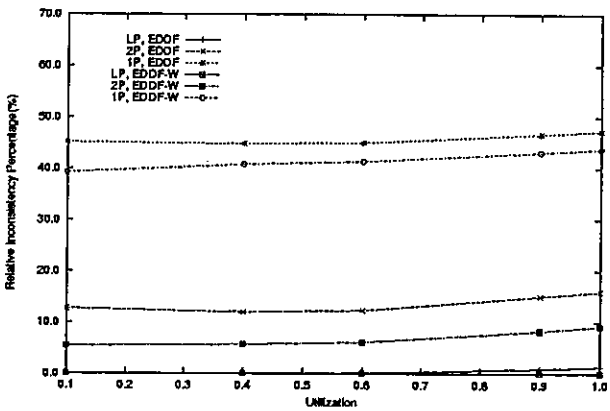

Figure 13: Relative inconsistency percentage under EDDF-W (UtilDist: LH, $r = 0$, $P_{ratio} = 5$)

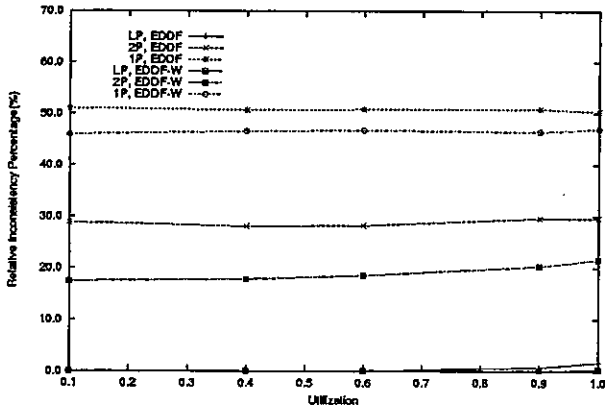

Figure 12: Relative inconsistency percentage under EDDF-W (UtilDist: LH, $r = 0$, $P_{ratio} = 10$)

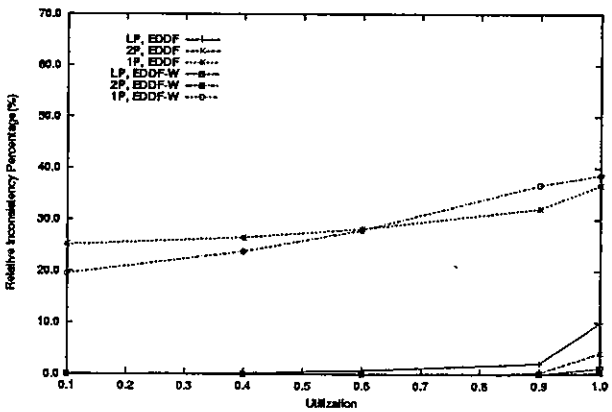

Figure 14: Relative inconsistency percentage under EDDF-W (UtilDist: LH, $r = 0$, $P_{ratio} = 2$)