Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp1097-1104)

Combining scheduling and concurrency control of Real-time transactions within the MOA Architecture

LEILA BACCOUCHE, SAMI LIMAM
RIADI laboratory
INSAT, National Institute of Applied Science and Technology
C.U. Nord, B.P 676, Tunis Cedex 1080
TUNISIA

Abstract: - This paper presents a multi-class scheduling algorithm for real-time transactions that combine scheduling and concurrency control. It is integrated to MOA, a multi-class overload architecture. The architecture consists of a real-time control layer which provides sophisticated admission control, scheduling and overload management. The proposed algorithm, DBP_CC, allows service differentiation and takes into account resource availability when extracting a transaction. There is no need to execute a concurrency control algorithm afterwards. Simulation results show that DBP_CC can achieve a significant performance even in overload situations.

Key words: - Real-time database systems, Service differentiation, Transaction scheduling, Concurrency control.

1 Motivation

Many current soft real-time applications and systems such as e-commerce applications, stock trading, internet bids, media servers, manipulate extensive amount of data under time constraints. Such applications need RTDBMS (Real-Time Database Management Systems) with time-cognizant protocols for concurrency control, commit processing, transaction scheduling, I/O, etc.

Often, real-time database applications need service differentiation. In real-time environments where transactions are ordered for execution based on their deadline, it can be useful to provide the application developer a way to correctly point out, how much it is critical to the system that some transactions meet their deadline. Recently, we observed a tendency towards RTDBMS that allow developers to specify the importance of transactions using a time attribute (by specifying a deadline) and using other attributes like a priority or an importance. Such RTDBMS need service differentiation oriented scheduling protocols.

To address this need, RTDBMS should support multi-class models and have specific scheduling algorithms. In [4], we have introduced a novel real-time transaction control layer called MOA (multi-class overload architecture), in the database system architecture. MOA consists of a set of modules which act together in order to guarantee predictability, overload resolution and service differentiation.

Within MOA, we propose an adaptation of DBP [7] (a multi-class scheduling algorithm used to serve streams in networks) to real-time transactions and we combine it with a concurrency control algorithm 2PL (two phase locking) [9]. The rest of this paper is organized as follows. The related work in real-time transaction scheduling is presented in next section. The MOA architecture and the main modules are described in Section 3. Section 4 presents the proposed scheduling algorithm. Performance evaluation results are presented in Section 5. Section 6 concludes this paper.

2 Related work

Transactions have been classified by Ramamritham in [14] as hard, firm and soft transactions. Real-time database transactions are usually in either firm deadline or soft deadline class. In this paper, we restrict our attention to real-time database systems that execute *firm deadline* transactions. A firm transaction that misses its deadline adds no value to the system. Thus it is discarded as soon as its deadline is missed.

Many efforts have been made in the design of real-time concurrency and commit protocols (used to guarantee the isolation and atomicity properties). Real-time concurrency control protocols may be optimistic or pessimistic. Optimistic protocols such as SCC [5], [6], Wait-50 [8] detect conflicts at transaction commit time and resolve those using rollbacks, while pessimistic protocols such as 2PL-HP (a real-time adaptation of 2PL) [9], avoid conflicts by applying resource blocking. Scheduling policies such as EDF [12], EDF-CR [1], [2] are optimal to schedule transactions but within a single queue. EDF performance decrease seriously in overload situations [1].

Recently, kang and al. have introduced the feedback control scheduling architecture which consists of admission control, performance measures and a quality of service controller. The target performance is achieved by dynamically adapting the system behavior based on the current performance error measured by the monitor [10]. Although their model is a multi-class one, the scheduling algorithm executes all transactions of high importance before passing to another queue.

When the server is overloaded, it is impossible for any schedule to meet every deadline. That's why the (m,k)-firm model was proposed by Hamdaoui [7]. The guarantee (m,k)-firm specifies the level of the temporal guarantee offered to real time applications tolerating the loss of some processes of tasks or some messages. The principle of the (m,k)-firm model is to guarantee that m tasks respect their deadlines among k consecutive tasks. In a multi-class model, each class may have its own (m_i,k_i) parameters. (m,k)-firm scheduling has an impact on the reduction of the system overload as it doesn't try to guarantee the respect of the deadlines of the totality of tasks but only of a proportion of them.

Suitable approaches to real-time systems that can tolerate occasional deadlines miss, fall into two categories: static and dynamic. In the static algorithms, the priority is determined off line while using a stationary parameter, for example the ratio of success m/k (examples of algorithms are (m,k)-WFQ [11], Enhanced Fixed Priority[15]).

Koubâa and Song present in [11] an algorithm which consists in integrating the temporal constraints (m,k)-firm to the process of scheduling of WFQ. The source marks m critical packets among all k consecutive packets and the rests being optional. The scheduler stamps the packet by its outgoing virtual time then the server selects the packet having the smallest outgoing virtual time among all present critical packets at the head of their queues. If no critical packet exists, the choice is made among the optional packets.

Another approach is the introduction by Ramanathan in [15] of the EFP, Enhanced fixed priority algorithm. The basic idea of this algorithm is to classify the instances of a task as either mandatory or optional. The mandatory instances are assigned a higher priority than optional instances. The classification of instances as mandatory or optional is based on the values m_i and k_i.

The static algorithms can't be applied to transactions scheduling because they don't allow the calculus of the priority during the execution. In contrast with packets, transactions can have resources conflicts; therefore we need a concurrency control (CC) scheme to solve these conflicts during execution.

The resolution of the CC scheme is dependent on the availability of the resources needed by the transaction to extract. Often the priority needs to be adjusted to take this last in consideration.

With the dynamic algorithms, the priority is determined according to the state of the system. Most famous algorithms are DBP (Distance Based Priority) [7], Matrix-DBP [13] and DWCS (Dynamic Window-Constrained Scheduling) [16].

The DBP algorithm is one of the famous dynamic algorithms applied to the (m,k) firm model. In [7], it was applied to periodic and aperiodic streams organized in multiple queues with different (m,k) requirements. DBP uses the history of the execution called k-sequence, to determine the queue which is going to miss its (m,k) requirements. The k-sequence is a sequence of k bits (1 indicates the respect of deadline and 0 the opposite). It is updated after the stream is served. The selected queue is considered of high priority and the stream at the head of the queue is extracted and served.

In [13], Poggi and al. introduce the Matrix-DBP algorithm which is an enhancement of DBP for periodic streams. In [16], West and al. present the DWCS, Dynamic Window-Constrained Scheduling which attempts to guarantee that no more than x out of y deadlines are missed for consecutive packets in real-time multimedia streams.

All these approaches are dedicated to the Network domain. Usually packets have the same length whereas transactions have individual execution times. Moreover, often packets don't have a deadline. Among dynamic algorithms, DBP is the one that seems interesting to adapt; indeed it is dynamic and accepts both periodic and aperiodic streams.

3 MOA: the Multi-class Overload Architecture

MOA is a real-time layer with suitable real-time protocols which allow:

- The concurrent execution of multi-class transactions,
- The support of various dispatching algorithms adapted to the multi-class transaction model,
- Reduced miss deadline, function of transaction classes,
- Overload situations detection and resolution.

Its main components are a transaction controller and a transaction scheduler. The transaction controller (TC) controls the transaction admission and resolves overload situations.

The transaction scheduler (TS) provides scheduling and dispatching algorithms which are specially designed for the multi-class transaction model. Our research addresses additional aspects in relation to the multi-class nature of the transactions (high, medium and low

importance) requiring more sophisticated admission control and rejection algorithms. Figure 1 presents the proposed architecture.

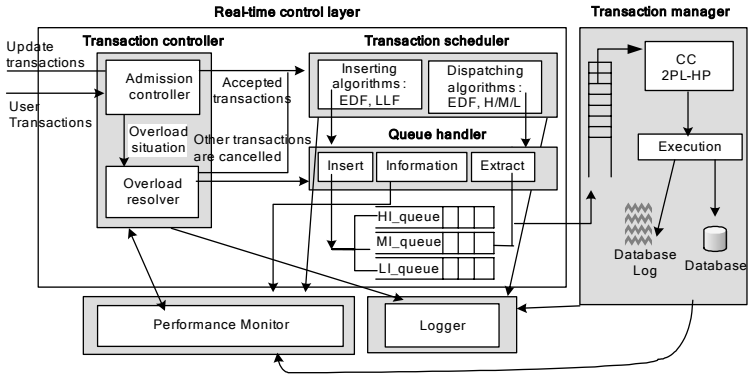

Fig. 1 The MOA Architecture

3.1 Transaction model

A transaction τ_i is characterized by the following attributes:

r_i – the ready time, when the transaction arrives to the system.

d_i – the deadline, it indicates the requirement to complete the transaction before the instant d_i.

we_i – the worst case execution time. The execution time of a transaction is data dependent.

re_i – the remaining execution time. A transaction is executed during a quantum; re_i represents the remaining execution time.

st_i – the slack time of τ_i. It represents the maximum amount of time the transaction can be delayed and still satisfy its deadline. d_i, st_i and r_i are related by : $st_i = d_i - r_i - we_i$. Initially the slack time is computed using we_i. This attribute is dynamic and at time t, $st_i = d_i - t - re_i$.

Transactions can be periodic or aperiodic. Periodic transactions are those which need to update data frequently. Let p_i be the invocation period. Usually $p_i = d_i$. Aperiodic transactions are those whose arrival to the system is unknown.

imp_i – importance of τ_i. The importance of a transaction τ_i indicates how much it is critical to the system that the transaction meets its deadline. We adopt three levels of importance: high, medium and low. The importance attribute is given by the application developer.

A high importance (Hi) means that the transaction is very important. A medium importance (Mi) means that the transaction should satisfy its deadline but there will be no problem if it is missed. A low importance (Li) is the default importance and means that a deadline miss for this transaction is not so important. The Li queue may also contain non-real-time transactions. We assume that some deadlines miss are inevitable due to unpredictable workloads.

In a multi-class environment, a politic that makes the differentiation of service must be capable to define a strategy to choose the queue of which will be made the extraction. Under pretext that the HI queue is the most important one, the algorithm shouldn't execute all transactions of this queue before passing to another one, but should achieve a specific minimum level of service for the queues.

3.2 MOA modules

3.2.1 The transaction controller

The transaction controller consists of an admission controller and a resolver. The admission controller receives submitted transactions and executes an admission test. The test is based on the current system state and the transaction attributes and is successful if the new transaction can meet its deadline without any risk that the previously accepted transactions miss their deadline. Accepted transactions are transmitted to the scheduler. Both the admission controller and the resolver are designed as components and implemented by aspects. An aspect can be seen like a non-functional aspect of a RTDBMS and can be disabled without disturbing the system functionality. Indeed, some scheduling algorithms apply an admission test but others don't. The DBP_CC algorithm doesn't apply a test upon the transaction arrival.

3.2.2 The transaction scheduler

The transaction scheduler consists of inserting and dispatching algorithms. Transactions accepted by the admission controller are sent to the inserting module. This module inserts them in the associated queues according to their priorities. Inserting algorithms are the most famous dynamic real-time scheduling algorithms: EDF and LLF [12]. EDF sorts queues by an increasing value of the deadlines.

The scheduler supports many dispatching algorithms: some of them are single class algorithms such as EDF, LLF.

The MOA architecture supports both single and multiple queue models. In that case, we apply sophisticated dispatching algorithms which consider additional aspects in relation with the multi-class transaction model.

The first one, H/M/L has been introduced in [3]. It is a parametrable priority based scheduling algorithm, which extracts H percent of the number of high importance transactions, M percent of the number of medium importance transactions and L percent of the low importance transactions that are ready (i.e. their ready time is reached). A threshold is defined for the execution of low importance transactions.

In this paper we present a novel scheduling algorithm DBP_CC, adapted to the multi-class model, which apply service differentiation and concurrency control before extracting a transaction. DBP_CC offers quality of service by allowing the programmer to specify different success ratios for the queues.

3.2.3 The transaction manager

The transaction manager can be compared to a conventional database engine. Transactions extracted by the dispatching algorithm are inserted in a queue called TM_queue which is used by the transaction manager to execute transactions.

The transaction manager can execute many transactions in parallel. We define C as the system capacity to execute transactions in parallel. For each transaction extracted from TM_queue, the resource set is checked and once all the required resources obtained, the transaction is executed during the necessary quanta. When a transaction resumes execution, if C is not reached, the transaction manager extracts the transaction at the head of the TM_queue and so on.

The concurrency control algorithm applied by the transaction manager is 2PL-HP (Two phase locking high priority) which is free from "priority inversion".

3.2.4 Additional modules: The queue handler, the logger and the performance monitor

MOA uses three additional modules. The queue handler provides the basic infrastructure for the queues handling and the maintenance of information on the queues.

The logger represents the journalizing module of the real-time database system. It receives the most relevant information to preserve from the various modules and it registers them in a log.

The performance monitor measures the state of the system in terms of deadlines miss and utilization ratios. Examples of useful statistics are system utilization, arrival rates and miss ratios.

4 The proposed algorithm

This section presents the DBP_CC algorithm. We first present the application model and then describe the algorithm.

4.1 Application model

In a RTDBMS, we distinguish mainly two types of transactions: update transactions and user transactions. Update Transactions regularly update the data gathered near sensors. These transactions are carried out periodically to refresh the value of the real time data. User transactions carry out operations of read/write on non real-time data and read operations on real time data.

Within the MOA architecture, we have chosen to organize the transactions by the following way:

- The queue Hi will contain the update transactions since these are the more critical in the system.
- The queue Mi will contain the user transactions judged important.
- The Li queue will contain the user transactions judged of less importance.

Let's recall that user transactions are marked of medium or low importance by the application developer. In addition, he has to specify the values of the parameters (m,k). More m is close to k more the queue has priority. The ratio m/k of the Hi queue must be distinctly superior to the one of the Mi queue in order to give more priority to the update transactions. For the present model, since the high queue contains periodic update transactions, m is equal to k.

The queue handler supports both single and multiple queue models. Presently, the multiple queue model consists of 3 queues but one can add as much queues as he wants. Therefore, the administrator can specify *n* queues for user transactions, each one having its own (m_i,k_i) requirements.

4.2 Building the k-sequence with DBP

The basic idea of DBP is very simple: more a queue is close to the dynamic failure, more it has priority. A queue is in dynamic failure if there are less than m transactions that respect their deadlines among the k last transactions.

It supposes that we save information concerning the respect of deadlines in a structure named k-sequence. The k-sequence of a stream represents the history of execution of the k last transactions. It is a sequence of k bits (1 indicates the respect of deadline and 0 the opposite).

In figure 2, the system is in dynamic failure until 2 transactions among 3 consecutive are executed. After the execution of the first transaction, the k-sequence is 001, if the next transaction is executed with success the k-sequence will be 011 and if it is missed the k-sequence will be 010. In case of success, the system leaves the dynamic failure state.

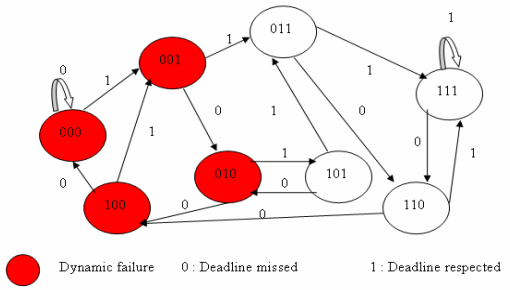

Fig. 2 Example of k-sequence

Given the k-sequence of a set of transactions, the distance that separates the present state to a dynamic failure state is equal to the number of' 0' that it will be necessary to add to the k-sequence so that it is in dynamic failure. With DBP, if ever two transactions have the same distance, it is the EDF policy that is applied. The priority (distance) is computed by DBP in the following way:

$$\Pr iority\_DBP = k - l(m,s) + 1 \qquad (1)$$

Where *l(m,s)* is the position leaving from the right of the m^{th} success (1) and *s* the state of the queue. In our model, each queue has its own (m_i,k_i) constraints and its own k-sequence. Each queue having the weakest priority is the closest one to dynamic failure (0 being the top priority: dynamic failure). The k-sequence is updated after the complete execution of the transaction.

4.3 DBP_CC outline

All (m,k) firm algorithms presented in the section 2 are specific to scheduling within networks. The model to which they are applied consists of periodic and/or aperiodic streams with packets of same length. When adapting DBP to schedule transactions, the resolution of the concurrency control scheme, often interfere with transactions scheduling. The significant contribution of this work is the adaptation of DBP to DBP_CC, which combines both scheduling and conflict detection and resolution so that the transaction manager doesn't need to execute a concurrency control algorithm such as 2PL-HP.

For each queue of the model, DBP_CC updates its k-sequence. A transaction is extracted if the transaction manager hasn't reached its maximal capacity C. Using the k-sequences, DBP_CC computes the priority of each queue and determines $Queue_{priority}$ the closest one to a dynamic failure state. Before extracting the transaction at the head, a conflict detection test is executed. This test scans a resource table which indicates for each resource locked, the mode (shared or exclusive) and the transaction holding the lock (figure 3). A conflict occurs when at least one resource needed by the requester transaction is locked.

If the transaction at the head has a conflict with one transaction that has already began its execution, we are tugged between (1) pass to the following transaction in the same queue and (2) change queue. Indeed, the transaction at the head in another queue may have no conflicts with the transactions in TM_queue. Our primary goal is to avoid a dynamic failure state so if the transaction at the head can't be executed, the algorithm passes to the next transaction in the same queue.

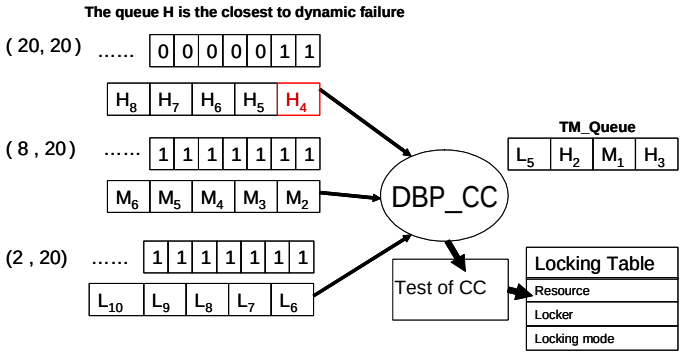

Fig. 3 Execution of DBP_CC

4.3.1 Guarantying transaction serializability

The isolation property (part of the ACID properties) is seriously compromised when there are concurrent transactions. Transactions are serializable when the effect on the database is the same whether the transactions are executed in serial order or in an interleaved schedule. The two phase locking algorithm is the common mechanism applied by commercial databases to guaranty transaction serializability. 2PL applies a lock acquisition phase followed by a lock release phase. The DBP_CC algorithm enforces serializability because it respects the principle of the two phases, hence once the release phase starts, the transaction can't acquire new locks.

4.3.2 Priority inversion

Priority inversion occurs when a transaction of high importance has to wait that a lower importance transaction releases locks put on resources needed by the high importance transaction.

DBP_CC is free from priority inversion. Each time a resource conflict is detected, before considering another transaction, DBP_CC compares the importance of the requester and holder transactions. The following table is used by DBP_CC to make decision.

		Requester (transaction to extract from $Queue_{priority}$)		
		Hi	**Mi**	**Li**
Holder	**Hi**	nothing	nothing	nothing
	Mi	Restart Holder	nothing	nothing
	Li	Restart holder	Restart holder	nothing

Table 1 Conflict resolution strategies

Table 1 shows that the algorithm interrupts a transaction holding locks only if the requester transaction has higher importance. The algorithm browses TM_queue and requisitions resources to less importance transactions. We settle the problem of priority inversion, but by aborting transactions of lower

importance (the k-sequence is updated with value 0) and we risk the dynamic failure for the other queues.

4.3.3 Guarantying the requirements of update transactions

Update transactions have the higher importance in our model and have to meet their deadline. A way to achieve this goal is to execute all transactions of Hi_queue before passing to another one. The DBP_CC algorithm applies (m,k) firm scheduling and it may happen that a Mi or Li transaction increases the response time of a Hi transaction. To achieve the success requirements of update transactions, when the closest queue to dynamic failure state, $queue_{priority}$ is the Mi_queue or the Li_queue, the algorithm checks if the slack time of the head transaction in the Hi_queue is sufficient to execute the selected transaction and the transactions in TM_queue.

Let $\tau_{extract}$ be the transaction at the head of the $queue_{priority}$ and τ_H the transaction at the head of Hi queue. If $\tau_{extract} \neq \tau_H$, $\tau_{extract}$ is extracted only if

$$st_H > we_{extract} + \sum_{1}^{C} re_i \ \forall \tau_i \in \text{TM\_queue} \quad (2)$$

5 Performance evaluation

We have simulated a main memory database, this way transaction execution involves only cpu and no I/O and we can estimate the execution time. The decreasing main memory cost, allows very large databases to remain in memory and main memory databases are suitable to real-time applications.

The design of MOA architecture has been carried out with UML 2.0 and the unified process 2TUP. We have developed a java simulator named RTDS, based on this architecture. The present version is 1.01.

5.1 Simulation model

In our simulation, we execute transactions workloads consisting of update transactions and user transactions.

Update transactions constitute a periodic workload of 40% of processor time. User transactions follow an arrival rate (poisson distribution) that vary from 1 to 50 transactions per second in increments of 10, which represents a medium to heavy loaded system. The execution time and the deadline of a transaction follow a uniform distribution. The deadline is calculated by this formula: $d_i = r_i + we_i * (1 + sf_i)$. The slack time ($sf_i * we_i$) represents the time during which the transaction can be delayed without missing its deadline. The slack factor is uniformly distributed between 3 and 5. The estimated execution time of an update transaction and a user transaction is uniformly distributed in a range (30ms, 70ms) and in a range (30ms, 150 ms) respectively. For all the experiments, the system enters an overload state when the number of transactions reaches 20 (processor utilization exceeds 1)

The number of resources in the database is 100. Each transaction needs a set of resources uniformly distributed in a range (1, 3). The resource set of a user transaction follow a Bernoulli distribution with a write probability of 25 percent. The following table summarizes the parameters and their baseline values.

Parameters	Baseline values
Number of resources (which represents the database size)	100
CPU time for update transactions	30 to 70 ms
CPU time for user transactions	30 to 150 ms
Slack factor for update transactions	3 to 5
Slack factor for user transactions	3 to 5
Mean transaction arrival rate per second	1 to 50 Transactions Per Second
Processor utilization for periodic transactions	40 %
Probability of write operation for user transactions	0.25

Table 2 System parameters

5.2 Experiment 1 : comparison between DBP and DBP_CC

The first experiment compares the performance of DBP and DBP_CC. DBP calculates the priority of each queue and extracts the transaction at the head of the queue with the lowest priority. DBP_CC performs a conflict detection test before extracting the transaction. In case of conflict, it applies resolution strategies or extracts the following transaction. We define GMR (global miss ratio) as the total number of transactions that miss their deadline compared to the total number of transactions.

$$GMR = \frac{\#\text{ of missed transactions}}{\text{total }\#\text{ of transactions}} \quad (3)$$

For the first step of this experiment, (m_H,k_H) is (20,20), (m_M,k_M) is (8,20) and (m_L,K_L) is (2,20) which represents requirements of 100% success for update transactions, 40 % for medium importance transactions and 10% for Low importance transactions. As can be observed in figure 4, the performance of DBP_CC is better than DBP. Indeed DBP_CC executes two tests before extracting a transaction: one to detect a possible resource conflict and one to check if there is enough time to guarantee that the transaction of high importance at the head meets its deadline.

The difference between the GMR is not very significant because this is an average ratio. It is necessary to examine the per-class miss ratios in order to highlight the superiority of DBP_CC.

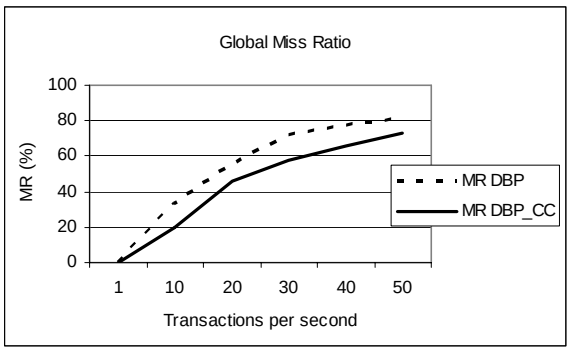

Fig. 4 GMR for DBP and DBP_CC

5.3 Experiment 2 : Per-class Miss ratio

As can be observed in figure 4, for an average number of transactions per second that varies between 20 and 50 GMR is high. In this experiment, we measure the per-class miss ratios for DBP and DBP_CC. We define MR_Hi as:

$$MR\_Hi = \frac{\#\text{of missed deadlines for Hi transactions}}{\text{total \# of Hi transactions}} \quad (4)$$

Mr_Mi et MR_Low are calculated according to the same principle. On figure 5 which describes the per-class miss ratios of DBP, we observe for an average of 20 transactions per second, that Mr_Hi, MR_Mi and MR_Li are respectively 16%, 64% and 68%; What gives us success rates respectively of 84 %, 36 % and 32 %. When the system is overloaded, with a number of 40 transactions per second, the success rates of the various queues are respectively 56%, 27% and 10%. We notice that DBP deviates from the specified rates of 100%, 40% and 10% especially for transactions of high importance.

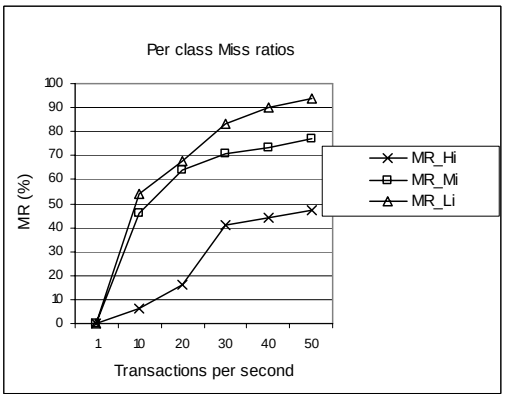

Fig. 5 Per-class miss ratios for DBP

According to figure 6, DBP_CC provides for an average of 20 transactions per second, miss ratios for the queues Hi, Mi and Li which are respectively 2%, 53% and 68%. Success rates are 98%, 47% and 32 %. For an average of 40 transactions per second, these success rates varies little and are respectively 98%, 44% and 14%. We notice that DBP_CC has success rates which are close to the values (m,k) specified for each queue.

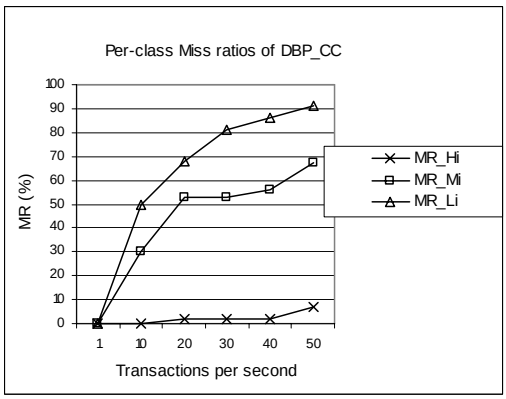

Fig. 6 Per-class miss ratio for DBP_CC with requirements of 100%, 40% and 10%.

Figure 7 shows the per-class miss ratios with requirements of 100%, 70% and 10% (the Mi queue has (m_M,k_M) set to (14, 20)). The algorithm extracts more Mi transactions, consequently the miss ratios of high importance transactions is slightly degraded. That results in data updated with a little delay but no serious consequences since the environment is a soft real-time system that tolerates some deadlines miss.

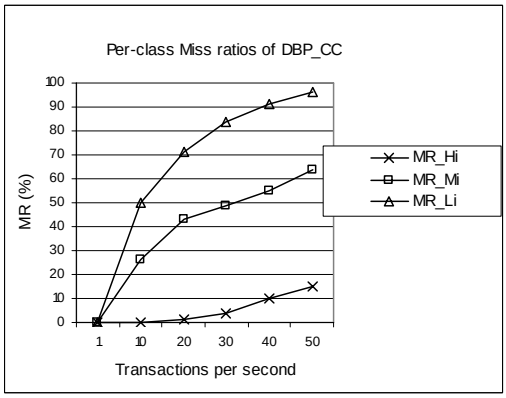

Fig. 7 Per-class miss ratio for DBP_CC with requirements of 100%, 70% and 10%.

6 Conclusion

When scheduling transactions in a real-time database management system, the concurrency control resolution often interfere with the decisions taken by the scheduler, what makes transactions guaranteed by the scheduling algorithm miss their deadline. Usually RTDBMS apply a concurrency control algorithm and schedule transactions with an increasing value of their deadline. In a multi-class model where transactions have different success

requirements, a suitable scheduling algorithm is necessary.

In this paper we have considered DBP, a scheduling algorithm for (m,k) firm streams and we have adapted it to execute transactions and resolve resource conflicts. The proposed DBP_CC algorithm allows service differentiation and is free of priority inversion. We have simulated and compared the algorithms. Results show that DBP_CC always outperforms DBP and respects the specified (m,k).

Other simulation experiments are being conducted to study the relation between the (m,k) of the different queues. Based on the (m,k) firm requirements of the Hi queue, the system will be able to certify if the others (m,k) firm constraints can be met or if they need to be changed. An analytic study of the proposed algorithm based on the response time is also being done. Our goal is to reject transactions upon their arrival if the system can't guarantee their deadline. In that case, the admission controller which is implemented as an aspect in the MOA architecture will be associated to the execution of the DBP_CC algorithm.

References:

[1] R. Abbot, and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation", In Proceedings of the 14th International Conference on Very Large Database Systems, Los Angeles, California, USA, Aug. 1988.

[2] R. Abbot, and H. Garcia-Molina, "Scheduling Real-Time Transactions with Disk Resident Data", In *Proceedings of the 15th International Conference on Very Large Database systems*, pages 385–396, Amsterdam, The Netherlands. Aug. 1989.

[3] L. Baccouche, "Scheduling multi-class real-time transactions: a performance evaluation", In *Proceedings of the 5th International conference on databases and datamining DBDM 2005*, pages 249-252, turkey, June 2005.

[4] L. Baccouche, "An overview of MOA, a multi-class overload architecture for real-time database systems: framework and algorithms", In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-06)*, March 2006, Dubai/Sharjah, UAE.

[5] A. Bestavros, and S. Braoudakis, "Timeliness via speculation for real-time databases", In *Proceedings 14th IEEE RTS Symposium (RTSS'96)*, San Juan, Puerto Rico, 1996.

[6] S. Braoudakis, "Concurrency control protocols for real-time databases", *PhD dissertation*, Computer Science Department, Boston University, USA, 1995.

[7] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m,k)-firm deadlines," *IEEE Transactions on Computers*, April1995.

[8] J.R Haritsa, M. J. Carey, and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control", In *Proceedings of the Real-Time Systems Symposium*, pages 94-103, Lake Buena vista, Florida, USA, Dec. 1990.

[9] J.R Haritsa, M. J. Carey, and M. Livny, "Data access scheduling in firm real-time database systems", In Journal of RTS, vol.4, n° 3, pages 203-241, 1992.

[10] K.-D. Kang, S. Son, and J. Stankovic, "Service Differentiation in Real-Time Main Memory Databases", In *5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (IEEE ISORC'02)*, Washington D.C.

[11] A. Koubaa Y-Q. SONG, J-P THOMESSE, " Integrating (m,k)-Firm Real-Time Guarantees in the Internet QoS Model", *LNCS 4th IFIP Networking Conference Networking '2004*, Athens (Greece) 9-14 May 2004

[12] C. L. Liu, and J. Layland, "Scheduling algorithms for multiprogramming in hard real-time environments", In *Journal of the ACM,* 20(1): pages 46-61, Jan. 1973.

[13] E.-M. Poggi, Y.-Q. Song, A. Koubaa, Z. Wang, "Matrix-DBP For (m, k)-firm Real-Time Guarantee", In *Proceedings of Real Time Systems Conference RTS'2003*, pages 457-482, Paris (France), 1-3 April 2003.

[14] K. Ramamritham, "Real-time databases", *Distributed and Parallel Databases (Invited Paper)*, vol. n°(2), pages 199–226, April 1993.

[15] P. Ramanathan, "Overload management in Real-Time control applications using (m, k)-firm guarantee". *IEEE Transactions on Parallel and Distributed Systems*, 10(6) pages 549–559, Jun 1999.

[16] R. West, Y. Zhang, K. Schwan, and C. Poellabauer, "Dynamic window-constrained scheduling of real-time streams in media servers," *IEEE Transactions on Computers*, vol. 53, pp. 744–759, June 2004.