A Dynamic (m,k)-firm Constraint to Avoid Dynamic Failures in a Real-Time DBMS

Sami Limam, Leila Baccouche
Riadi-GDL Laboratory
Insat, National Institute of Applied Science And Technology
C.U. Nord, B.P 676, Tunis Cedex 1080, Tunisia
Limamsami@yahoo.fr
Leila.Baccouche@free.fr

Henda Ben Ghezala
Riadi-GDL Laboratory
ENSI, National School of Computer Science
University of la Manouba, Tunisia
hhbg.hhbg@gnet.tn

***Abstract*—Current applications such as stock markets, e-business, multimedia and telecommunications require further real-time services. Such systems deal with large quantities of data, and transactions have temporal constraints. Among these applications, some of them must face unpredictable workloads. The aim of our work is to maintain a robust RTDBS behavior and to decrease the number of transactions which miss their deadline.In this paper, we propose an approach based on the (m,k)-firm model that allows to control the RTDBS behavior, in particular during the instability phases, in order to improve the quality of service (QoS) of applications. Multi-class transactions are scheduled by DBP_CC and we have to adapt QoS level for each class to the load of the system.**

***Keywords*-(m,k)-firm; real time scheduling; Imprecise Computation.**

I. Introduction

Recently, the demand for real-time services has increased considerably in many applications such as multimedia applications (video conferencing, video on demand services,Web services and e-business,) . In addition, these applications reflect an important requirement in data management. Therefore, real-time database systems (RTDBS) become the support to the implementation of these applications. In these applications, it is important to obtain complete and accurate results before a certain deadline, while using fresh data. However, as the users requests remain unforeseeable, the RTDBS can be overloaded leading to the system inability to meet deadlines and to control the data freshness.

The real-time systems distinguish generally between firm and hard real time tasks. In firm systems, Authors suggested relaxing the real time constraints of the tasks such as using the (m,k)-firm model [8], [5], [11] or imprecise computations [13]. The notion of (m, k)-firm constraints was introduced in particular for periodic tasks [9]. The (m,k)-firm model tolerate some deadline misses according to the (m,k)-firm constraints.

In this paper, we propose such an architecture which allows a differentiation of services between transaction classes. To this end, we exploit the (m,k)-firm transactions model and we propose a scheduling algorithm based on this model, which guarantees a certain QoS for each transaction class. However, we notice that if the system is overloaded, the (m,k)-firm constraints are not respected. Thus we must adapt the (m,k)-firm constraints to system load and introduce imprecision in order to respect these constraints.

In Section 2, we present the related work in management of QoS in RTDBS. In Section 3, we present the architecture that we propose to manage transaction classes: we present our transaction's model, our queue's model. Then we formulate the problem. In Section 4, we present our solution to manage transactions classes in overload situations. Section 4 is devoted to the presentation of the simulations we conducted to validate our model, as well as to the comments on the results obtained. The simulation platform used is called RTDS (Real-Time Database Simulator) that we have developed in our laboratory. The simulations results show the effectiveness of the proposed architecture and its ability to support overload situations and to guarantee QoS for each transaction class. Finally, we conclude the paper.

II. Related Work

As one of the first efforts to address QoS management in RTDBs, Kang et al. [10] developed a novel QoS management architecture called QMF. In QMF, a formal control theoretic approach is applied to compute the required workload adjustment considering the miss ratio error, i.e., the difference between the desired miss ratio and the currently measured miss ratio. Feedback control is used since it is very effective to support the desired performance. To the feedback control loop, Amirijoo et al. [1] added the use Imprecision in order to manage the QoS. Imprecise computation techniques [13] provide means for achieving graceful degradation during transient overloads by trading off resource needs for the quality of a requested service. Imprecise computation and feedback control scheduling have been used for QoS management of RTDBs. In this approach the notion of imprecise computation is applied on transactions as well as data, i.e., data objects are allowed to deviate, to a certain degree, from their corresponding values in the external environment. Many other works trade with QoS by using imprecision. Bouazizi et al. [6] proposed an approach based on Feedback Control Scheduling (FCS) that allows to control the RTDBS behavior, in particular during the instability phases, in order to improve the quality of service (QoS) of applications. They proposed a technique which allows to minimize the number of conflicts by exploiting multi-versions data, while taking into account the database size.

III. System architecture

A. Transaction's Model

In a RTDBS, we distinguish two types of transactions:

- Update Transactions: they are periodic and have to refresh regularly the database by updating data reported by sensors.
- User transactions: they read/write non real-time data and/or only read real-time data. They are generally non periodic. As their arrival in the system is unpredictable, they may cause overload situations. That is why imprecise transactions models are often used.

In this paper, we assume that transactions have firm deadlines, i.e., a transaction which misses its deadline becomes useless and is aborted. In addition, we exploit the imprecision model of transactions in overload situations [13]. A transaction consists in a mandatory part and an optional part. The mandatory part must be executed before the transaction deadline. Optional part is composed of sub-transactions which are executed if there remains enough time before deadline. More the number of optional sub-transactions executed is great, better is the result, i.e., the quality of service of the result is enhanced.

We base our work on (m,k)-firm model. A transaction is divided into k sub-transactions. To each sub-transaction is assigned a weight according to the criticality of the data it accesses. The m (less than k) sub-transactions having the higher weight are labeled mandatory; the others are optional. So, the transaction is composed of m mandatory sub-transactions and (k-m) optional sub-transactions. Like in Milestone model, the execution of the optional sub-transactions will make the result more precise. The transaction is successfully executed if at least the m mandatory sub-transactions are executed before deadline.

B. Queue's Model

We adopt a multi queues model with a single server. Indeed, in our model of transactions, we have different types of transactions, executed by only one processor. We begin to determine the number of queues needed by the architecture. We have defined a parameter, called Importance which will differentiate the types of transactions. We define three levels of importance:

- Update transactions.
- High importance user transactions that perform write operations.
- Low importance user transactions that perform read operations.

In our queue model, we have as many queues as levels of importance. So, we can differentiate service between transaction classes.Also, as user and distributed transactions consist of a set of mandatory sub-transaction and several optional sub-transactions, we have mandatory parts and optional parts, in each queue. As transaction classes need to be separated to allow service differentiation, each queue is divided into two sub-queues: one for the mandatory parts and one for the optional parts (cf. Fig. 1).

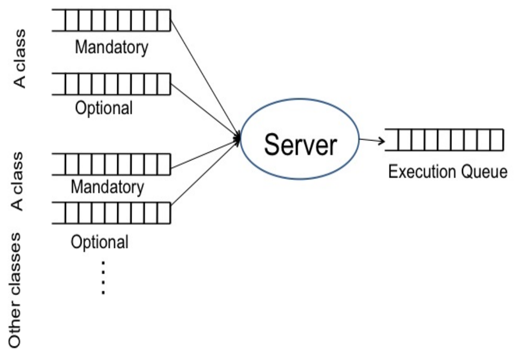

Figure 1. Queue model

C. Serving queues

In a real-time system, attempting to guarantee the respect of all transaction deadlines often leads to overload of the system. Then, to avoid this situation where the system becomes unstable, its better to guarantee a certain level of quality of service instead of guaranteeing all the deadlines. This is the idea behind (m,k)-firm model. The principle of (m,k)-firm model is to guarantee that m tasks meet their deadlines among k consecutive tasks. Therefore, we tolerate that (k-m) tasks miss their deadlines, among any sequence of k tasks.

Our problem is to schedule transactions inserted in different queues. We thus choose DBP algorithm [9], [11] already proposed to schedule network packets on several queues. This algorithm is adapted to the real-time context and allows to guarantee QoS. However, it does not include an access control to data, i.e., a concurrency controller.

For this purpose, we proposed DBP_CC algorithm (Distance Based Priority and Concurrency Control) [3], which is an adaptation of DBP algorithm [9](Distance Based Priority) to the RTDBS context.

DBP is a dynamic algorithm issued from the (m,k)-firm model. The algorithm we propose provides several levels of QoS described by various couples (m,k) specified by the database administrator for each class of tasks.

DBP uses the history of the execution to compute the DBP priority of each queue and determine the queue which is going to miss its (m,k)-firm requirements and be in dynamic failure state (less than m tasks among k consecutive tasks respect their deadlines). The selected queue is considered of high priority DBP and the task at the head of the queue is extracted and served.

DBP saves the history of execution in a structure named k-sequence which is a sequence of k bits updated after task execution (1 indicates the respect of deadline and 0 the opposite). The priority (distance) is computed by DBP in the following way:

$$Priority_{DBP} = k - l(m, s) + 1$$

where l(m,s) is the position leaving from the right of the m^{th} success (1) in the k_sequence, s (the state of the queue). If there are less than m byte 1 in the k-sequence s then l(m,s) = k+1. Each queue has its own (m,k)-firm constraint and its own k-sequence. Before extracting the transaction at the head, a conflict detection and resolution test is executed.

D. Problem Formulation

This policy which we adopt has numerous advantages:

- It allows a differentiation between transactions (We so obtain five classes and a queue is suited to every class of transactions).
- It allows a differentiation of service between classes. This is guaranteed by the attribution of a (m, k)-firm constraint specified by the administrator of the database and who fix the desired quality level. Besides, the algorithm DBP_CC aims to respect for this constraint by extracting a transaction from the closest queue to the state of dynamic failure. In addition, a differentiation is made during the execution step as the system assigns more time to the execution of the most important classes.
- We can introduce imprecision easily. With this model we can go of the most precise case (no tolerance to the imprecision: constraints (k, k)-firm for the diverse queues and the constraint (1,1)-firm for the transactions) in the most imprecise case. The degree of imprecision is controlled and it has two types:
  - Imprecision of queues: controlled by an (m, k)-firm constraint for each queue and which gives an idea of the desired level of QoS of the queue: m transactions have to succeed for any window of k successive transactions and (k-m) transactions at most can fail.
  - Imprecision of transactions: controlled by a single (m, k)-firm constraint for the user transactions as only them are decomposable.

Naturally, it also presents inconveniences such as:

- In front of an overload, all the queues sink in dynamic failure. In that case, the algorithm DBP_CC applies the EDF policy which is inappropriate in the case of overload.
- The imprecision is introduced with only the (m,k)-firm model. We can add to our model other techniques of imprecision to decrease the probability of dynamic failure (as the notion of epsilon-data or delta-deadline[7]).

IV. Solution

To improve this policy, it is necessary to address both quoted problems:

A. Dynamic Failure

As we consider an opened system, the overload is a state which can arise. In contrast to a closed system where all the transactions are known (their resources also), in an opened system we have transactions from the users that are unpredictable. When the system becomes overloaded, the system is unable to execute all the transactions before their deadlines. If the overload is important, the (m, k)-firm constraint is violated because we have less than m transactions which respect their deadlines. So, the queue is in a state of dynamic failure and if we have several queues in our queue's model, all will meet in this state. According to DBP_CC, if all the queues are in a state of dynamic failure, the EDF policy is applied (badly suitable to the excess loads) and we notice a decrease of the performances.

To address this problem, we have to avoid the state of dynamic failure or to minimize it. So the solution of this problem is quite clear: the priorities of queues have not to be equal to 0.

The DBP priority of a queue is according to the position of the m^{th} 1 (success of the transaction). Thus more m is big, smaller will be the priority and we approach the case of the dynamic failure (priority equal to 0). This is confirmed by experiments when we choose (m, k)-firm constraints where m approaches of k. In such systems, if some transactions miss their deadlines, the queue is in dynamic failure. In this situation, the system is forced to extract transactions of this queue and a transaction has to succeed to go out of the state of dynamic failure. In case we have flexible constraints, the system is less often in state of dynamic failure. In view of these observations and their confirmation by simulations, we intend to deploy a policy which decreases the probability of dynamic failure. This is possible by the following two actions:

- Apply as departure a flexible (m, k)-firm constraints (where m goes away from k).
- Decrease the probability of dynamic failure for the diverse queues. This can be made only on acting on the DBP priorities (take away these priorities of 0 has for consequence to decrease the probability of dynamic failure)

B. How to avoid Dynamic Failure

To decrease the probability to be in dynamic failure means to avoid having a priority equal to zero. The solution thus is to analyze the formulation of the priority DBP_CC and to ensue from it the solution. As we know, the priority expression is as follow:

$$Priority_{DBP} = k - l(m, s) + 1$$

We have the following parameters in this formulation:

- The parameters of the constraint (m, k)-firm: m and k.
- The history execution of the k last transactions: the k-sequence.

We can act only on the (m, k)-firm constraint as we cannot touch the execution history of the queue. Indeed, when the system is overloaded, to require less allows improvement of the performances. Thus, if the priority of a queue aims towards zero, we can decrease the value of m. Consequently l (m, s) decrease and then the DBP_CC priority increase. So, to take away the priority of zero it is enough to decrease the level of wished QoS (m). However it is necessary to clear up when and how?

- When: indeed on step when this priority becomes equal to zero because there it is too late. It is necessary to specify a threshold S below which we suppose that the system aims towards the failure. This presents another advantage as the system becomes preventive to the overload state.
- How: by specifying a law of regression of m. Indeed, m has to oscillate between its original value and a minimal value.

C. Expressing dynamically the (m,k)-firm constraint

The policy is to decrease the value of m when we notice that the queue approaches the state of dynamic failure. The detection of the approach of a state of dynamic failure is noticed by fixing a threshold for every queue below which we envisage the regression of m. However, we cannot decrease m until it reach its minimal value of 1 because in that case all the queues will have the same constraint (the same importance). Thus, it is necessary to fix for every queue a minimal value of m (by operating a differentiation of service so that $m_{minH} > m_{minM} > m_{minL}$).

By drawing up the curve of regression of m between its original value and its minimal value, we notice that m follows an exponential function between the priority 0 and the fixed threshold. Except this interval m takes its original value.

So, we obtain the following mathematical formulation:

$$m = m_{min} + (c \times Priority^{\Omega}) \quad if\ 0 \leq Priority \leq Threshold$$

$$m = m_{original} \quad if \quad Priority \geq Threshold$$

By setting (priority = Threshold), we can fix the values of c and Ω and we obtain:

$$c \times Threshold^{\Omega} = m_{original} - m_{min}$$

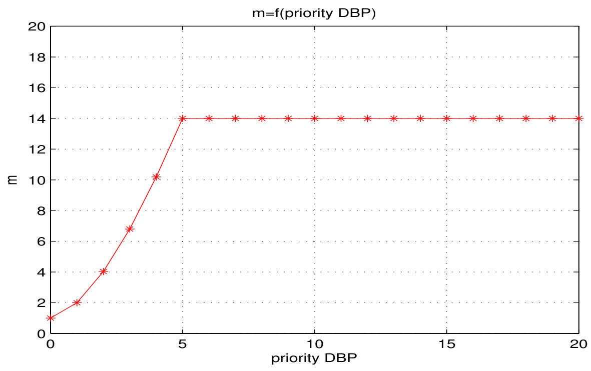

Figure 2. Evolution of m according to the DBP priority

D. More imprecision

To introduce more imprecision during the phases of overload, we can resort much techniques which introduce imprecision at the level of data and transactions. The imprecision is introduced on two levels: data and Transactions. We deal with Quality of Data (QoD) and Quality of Transactions(QoT). The QoD can be translated by the notion of epsilon-data or Data-Error where the data becomes valid on an interval not on a given value. It can also be translated in the notion of multi-versions data. The QoT is translated by a partial execution of the transaction or its replacement by the other one less expensive at CPU time. So the main advantage of our approach is that it combines QoD and QoT. It integrates two notions :

- the notion of ϵ - data
- the notion of Δ-deadline

1) ϵ-data: Under epsilon-data, the isolation is not maximal as a transaction can read a data which is being updated by another transaction. Thus, a transaction T which wishes to acquire a lock in a conflicting mode (W/R or W/W) can obtain this lock provided that the generated imprecision does not overtake epsilon.

2) Δ-deadline: Under delta-deadline, we can relax the deadline of a transaction and so the new deadline is equal to the former one to whom we add Δ.

V. Simulations and Results

A. Context

Simulation of the protocols developed for RTDBS requires a platform which respects transactions ACID (Atomicity, Consistency, Isolation, Durability) properties and which offers adequate mechanisms to the support of time constraints. Most research teams on RTDBSs developed their own simulator : Beehive [14], DeeDS [2], J-Radex [12]. In our research team, we have developed a simulator, called RTDS [4], which is described briefly, bellow.

B. RTDS simulator

RTDS [4] is a discrete-event simulator written in Java, designed by our research team to simulate real-time database behavior. It is especially devoted to test and to compare concurrency control and scheduling algorithms that are proposed in a RTDBS.

C. Simulation parameters

Parameter	Value
Number of ressources	100
Number of temporal ressources	20
Number of non real-time resources	80
Load of update transactions	40
Execution time of periodic transactions	10 to 20 ms
Execution time of a user transaction	70 to 100 ms
Number of mandatory sub-transactions in an user transaction	1
Number of optional sub-transactions	1 to 4
Number of resources used per sub-transaction	1

Table I
SIMULATION PARAMETERS

D. Simulation results

We carried out four experiments:

1) In the first simulation, all transactions are inserted in the same queue. So, in our Queue's Model we have one single queue. Then we apply EDF (Earliest Deadline First) to schedule this queue.
2) In the second simulation, we show the impact of the system load on the system performances. In the simulation, we have five queues in our Queue's Model, each queue holding a class of transactions (mandatory and optional parts of a transaction are put in different queues).
3) In the third experiment, the (m,k)-firm constraint becomes dynamic and it is expressed formally according to the established equation. We show that the system prevents the state of overload by relaxing the (m,k)-firm constraints of queues which approach the dynamic failure. This relaxation has for consequence improvement of results.
4) In the last experiment, we measure the impact of the introduction of imprecision on the performances of the system.

1) Simulation 1: Effect of Scheduling policy and Queue's Model: In this simulation, our queue's model includes a single queue that will contain the different classes of transactions. We use EDF to schedule the queue and the queue is ordered according to the deadline (the transaction in the top is that having the nearest deadline).

In Figure 3, we notice:

- EDF is not suitable for transient overload: When the load of the system increases, we have transactions with nearest deadlines trying to execute and they are unable to meet their deadlines. In addition, the time wasted to execute failed transactions has an impact on the respect of deadlines of waiting transactions.
- Inserting transactions on a single queue leads to a poor differentiation of service as the criteria that matters is the deadline not the importance of the transaction.

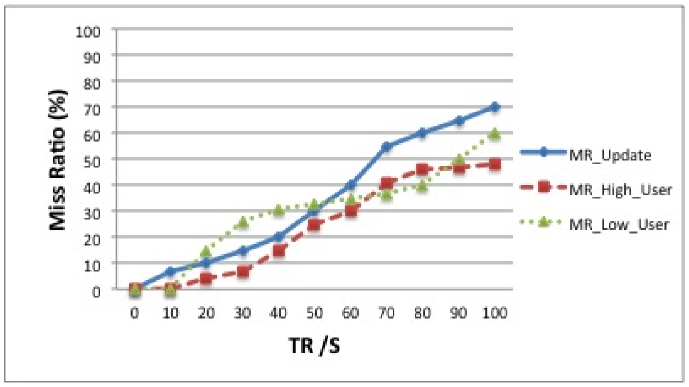

Figure 3. Transaction Miss Ratio (MR) when using EDF, with one single queue

2) Simulation 2: Variation of the system load: Figure 4 traces the variation of the miss ratio according to the system load. We vary the system from under loaded state (10 transactions per second) to an overload state (arrival rate of 40 transactions per second). In under loaded state, all transactions meet their deadlines, whereas in overload state several transactions miss their deadlines. We notice that the system becomes overloaded at the arrival rate of 40 transactions per second. We also note that when we use the (m, k)-firm model, we can differentiate service between different classes of transactions, thus, transactions' miss ratio of the prioritized class is lower than other classes.

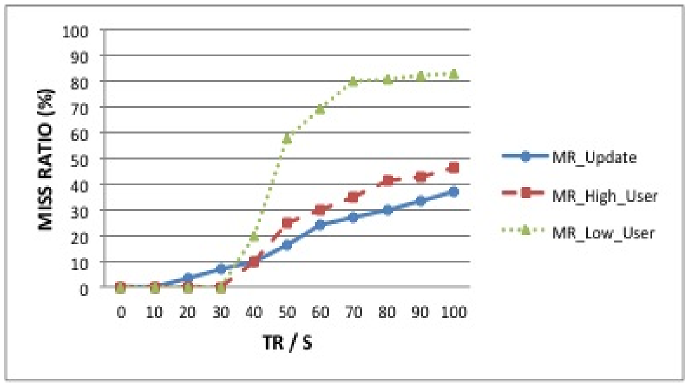

Figure 4. Transaction Miss Ratio (MR) when using (m,k)-firm model, with five queues in the system

In Figure 4, MR_Update, MR_High_User and MR_Low_User have the following meaning respectively: miss ratio of update transactions, miss ratio of high

	m	K	m_{min}	Threshold	c	Ω
Update	18	20	10	2	6	1
High Mandatory	14	20	6	5	1.2	1
High Optional	7	20	2	1	5	1
Low Mandatory	4	20	1	1	3	1
Low Optional	1	20	1	1	0	0

Table II
QUEUES PARAMETERS

importance user transactions and miss ratio of low importance user transactions. In the simulation, we have three classes of transactions dispatched in five queues: Update, High_Mandatory, High_Optional, Low_Mandatory and Low_Optional with respectively the following (m,k)-firm constraints: (18,20), (14,20) ,(7,20), (4,20) and (1,20).

3) Simulation 3: A dynamic (m,k)-firm constraint: In the second experiment, m follows the mathematical function established previously and we have the parameters expressed in Table II.

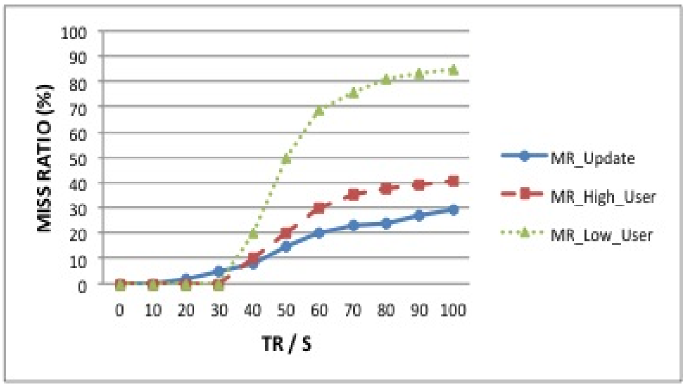

Figure 5. Transaction Miss Ratio when we apply dynamic (m,k)-firm constraints

In the Figure 5, we notice that usage of dynamic (m,k)-firm constraints allows to improve the results as they decrease the probability of dynamic failure. We notice that the performances in this simulation are better than those of the first simulation. However, the performances with flexible constraints are better as the flexible constraints correspond to the minimal values required for m (less probability of dynamic failure). Applying the minimal constraints present however two anomalies:

- The performances do not correspond to the wished levels of QoS: the performances are widely superior to the constraints even when the system is overloaded or not.
- The distance between the constraints is small and so we have a bad differentiation of service.

To address the noticed problems, we suggest the use of imprecise actions, to specify hard constraints and lead the system to respect them.

4) Simulation 4: Impact of imprecision.: In the last simulation, we apply dynamic constraints and we use imprecise actions. Indeed, if the DBP priority becomes lower than the threshold specified for the queue, we decrease the value of m and we apply two imprecise actions:

- We omit to execute any transaction which wishes to update a data if the new value is meanwhile [real value - ϵ , real value + ϵ] and we consider them as successful. This action (allows us to spare CPU time whom we can assign to the execution of the other transactions.
- We relax the deadlines of all the transactions. So, the system can respect the new ones. The performances obtained in Figure 6 confirm that the combination of (m, k)-firm constraints and imprecision allow us to obtain the best performances while respecting the levels of quality of service specified.

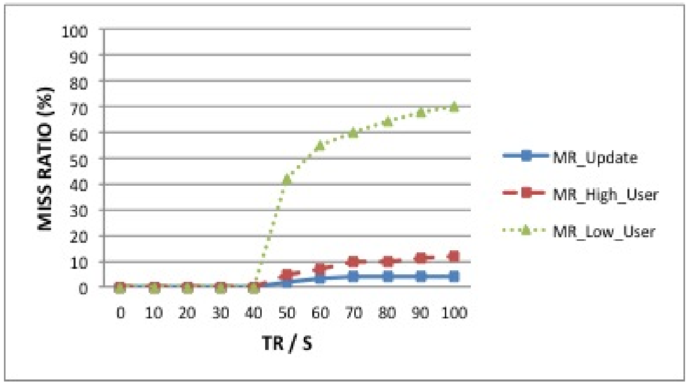

Figure 6. Transaction Miss Ratio when we apply dynamic (m,k)-firm constraints and we use imprecision

VI. CONCLUSION AND FUTURE WORK

In RTDBS, execution of transactions before their deadlines often leads to overload situations, during which the system performances are degraded. Based on (m,k)-firm model, we proposed, in this paper, an approach to control this degradation for each class of transactions. Results of the simulations we have carried out have shown that when using dynamic constraints and imprecision, we are able to differentiate service in a RTDBS according to the transaction classes. A transaction consists of a mandatory part and several optional parts. We insert each transaction class into two queues: one containing the mandatory part and the other contains optional parts. Then, we choose an (m,k)-firm constraint for each queue and we serve queues by applying the DBP_CC algorithm. Finally, when a queue approaches a dynamic failure, we decrease formally the value of m and we use imprecise actions in order to respect levels of QoS of each queue. In the near future, we plan to deploy and adapt this approach to a distributed context. Also, we project to use the (m,k)-firm model for load balancing or data replication.

REFERENCES

[1] M. Amirijoo, J. Hansson and S. Son. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. IEEE Trans. Computers, 55(3), 304-319, 2006.

[2] S.F. Andler, J. Hansson, J. Eriksson, J. Mellin, M. Berndtsson and B. Eftring. DeeDS Towards a Distributed and Active Real-Time Database System. ACM SIGMOD Record, 25(1), pp. 38-40, 1996.

[3] L. Baccouche and S. Limam. (m,k)-firm scheduling of real-time transactions with concurrency control. In Proceedings of the second international conference on systems, ICONS'07, Guadeloupe, French Caribbean, pp. 14-21, April 2007.

[4] L. Baccouche and S. Limam. RTDS: A component and Aspect-based Real-Time Database System Simulator. In Proceedings of ESM08, Le Havre, October 2008, pp. 551-555.

[5] G. Bernat, A. Burns and A. Llamos. Weakly Hard Real-Time Systems. IEEE Transactions on Computers, pp. 308-321, 1999.

[6] E. Bouazizi, C. Duvallet and Bruno Sadeg. Using Feedback Control Scheduling and Data Versions to enhance Quality of Data in RTDBSs. Proceedings of International Computer System and Information Technology (IEEE ICSIT'2005), Alger, Algeria, 322-327, 2005.

[7] S. Bouzefrane, J.P.Etienne, C. Kaiser. Handling Overload and Data-Relaxation Control in Distributed Real-Time Database Systems. I. J. Comput. Appl. 15(3),pp. 187-200 (2008).

[8] C. Evequoz. Guaranteeing Optional Task Completions on (m,k)-Firm Real-Time Systems. Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology.CIT '10. pp. 1772–1779.2010.

[9] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. IEEE Transactions on Computers, 44(12), pp. 1443-1451 (1995).

[10] K. Kang, S. Son and J. Stankovic Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. IEEE Trans. Knowl. Data Eng, 16(10), pp. 1200-1216, 2004.

[11] A. Koubaa and Y-Q. Song. Graceful Degradation of Loss-Tolerant QoS using (m,k)-Firm Constraints in Guaranteed Rate. Networks Computer Communications Journal, (Elsevier), Vol. 28/12, pp. 1393-1409, July 2005.

[12] J. Haubert, B. Sadeg, L. Amanton. and R. Coma. J-RADEx : un simulateur de SGBDTR convivial. Journal of Information Science for Decision Making, pp. 64-71, 2004.

[13] J.W.S. Liu, W.K. Shih, K.J. Lin, R. Bettati and J.Y. Chung. Imprecise Computations, Proc. IEEE, 82(1), pp. 83-94 (1994).

[14] J. Stankovic, S. Son and J. Liebeherr. BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications. Chapter in Real-Time Databases and Information Systems (Kluwer Academic Publishers, 1997), pp. 409-422.