Improvement of QoD and QoS in RTDBSs*

Emna Bouazizi, Claude Duvallet, Bruno Sadeg
LIH, Université du Havre, 25 rue Philippe Lebon
BP 540, F-76058 LE HAVRE Cedex
{Emna.Bouazizi,Claude.Duvallet,Bruno.Sadeg}@univ-lehavre.fr

Abstract

In current research toward the design of more powerful behavior of RTDBS under unpredictable workloads, different research groups focus their work on QoS (Quality of Service) guarantee. Their research is often based on feedback control real-time scheduling theory. However, due to the high service demand, and even with guarantee, some transactions may miss their deadlines. In this paper, we propose a technique which allows to execute transactions on time using fresh and precise data while taking into account the global size of the database. We have extended the feedback-based miss ratio control by both using multi-versions data, and proposing a data management policy combining (1) limitation of the versions number and (2) dynamic adjustment of this limit according to a maximum database size parameter. Simulation results have shown that the proposed approach successfully provides tight miss ratio guarantees and hight quality of data freshness.

1 Introduction

In previous years, a lot of work has been done on RTDBS [12][13], which are systems that are designed to manage applications where it is desirable to execute transactions timely using fresh and precise data [1]. Since the workload in this systems is unpredictable, the system may become quickly overloaded, leading to the decrease of the well-known RTDBS performance criterion (the number of transactions that complete before their deadlines).

To support these applications, some techniques based on Quality of Service (QoS) guarantee have been proposed to control the transient overshoot. They are often based on feedback control real-time scheduling theory [10][11]. Up to now, the major drawback is that in case of conflicts between transactions, some transactions are blocked, or aborted and restarted. This may lead to the transactions miss deadlines. To address this problem, in this paper, we have extended the feedback-based miss ratio control by using a multi-versions data architecture. This limits data access conflicts between transactions, enhancing then the concurrency and limits the deadline miss ratio.

The main objective of our approach is to maximize the number of transactions which meet their deadlines. In addition, our work aims to support a certain freshness for the data accessed by time-constrained transactions under a condition: the fixed maximum size of the RTDB. To this purpose, we merge two previously approaches proposed in [6] and [7]. In the new mixed approach, the number of versions is dynamically adjusted, but does not have to exceed a threshold which consists in a maximum data versions number, and also does not have to exceed a fixed threshold which consists in the maximum database size.

The remaining of the paper is organized as follows. Section 2 describes the real-time database model. In Section 3, our proposed model is described. Some simulation results are given in Section 4. In Section 5, we conclude the paper and give some perspectives.

2 Real-Time Database model

We consider firm RTDBS model, in which late transactions are aborted because they are useless after their deadline, and we consider a main memory database model.

This work on QoS guarantees is guided by the following premises:

1. Transactions are executed according to their priority and they are classified into

*Real-Time Database Systems

two categories: update transactions and user transactions (see section 2.2).

2. We keep different versions for each data item. These versions are dynamically adjusted by verifying the data freshness and considering the Data Error (DE). Data Error is computed by comparing the data version stored in the database with the corresponding value of the data in the real world. DE must respect an upper bound given by the Maximum Data Error (MDE) [2]. In our real-time database, validity intervals are used to maintain the temporal consistency between the real world and the sensor data stored in the database [12]. A data version d_i is considered temporally inconsistent (not fresh) or stale if the current time is later than the timestamp of d_i followed by the absolute validity interval of d_i (denoted AVI_i), i.e $CurrentTime > Timestamp_i + AVI_i$ (see section 2.3).

2.1 Data model

Data objects are classified into either real-time or non real-time data. A non real-time data is a classical data found in conventional databases, whereas a real-time data has a validity interval beyond which it becomes useless. These data may change continuously to reflect the real world state (for example, the current temperature value). Each real-time data has a timestamp indicating the last observation of the real world state. In our model, we consider only real-time data.

Many versions of a real-time data item may be stored in the database and the number of versions considered may be either fixed or dynamically adjusted. To store a version, data freshness and the MDE parameters are taken into account.

2.2 Transaction model

Transactions are classified into two classes: update transactions and user transactions. Update transactions are used to update the values of real-time data in order to reflect the state of real world. Update transactions are executed periodically and have only to write sensor data. User transactions, representing user requests, arrive aperiodically and may read real-time data, and read or write non real-time data.

2.3 Performance metrics

Three main performance metrics are considered: $MissRatio$ (MR), $DataFreshness$ (DF), and $DataError$ (DE) [3].

1. $MissRatio$: the transactions miss ratio is defined as follows:

$$MR = 100 \times \frac{\#Late}{\#Terminated}(\%) \qquad (1)$$

where $\#Late$ denotes the number of transactions that have missed their deadline, and $\#Terminated$ is the number of terminated transactions.

2. $DataFreshness$: in RTDBS, data can become outdated. To measure the freshness of a data item d_i in a RTDB, the notion of absolute validity interval (AVI) is used. A data version is related to a timestamp indicating the latest observation of this data item in the real world. d_i is considered temporally consistent (or fresh) if $(CurrentTime - Timestamp(d_i) \leqslant AVI(d_i))$. The database freshness can also be measured. It represents the ratio between fresh data and all the data in the database.

3. $DataError$: it represents the deviation between the current data value (CV(d_i)) and the updated value (UV(d_i)) when $d_i \neq 0$. The upper bound of the error is given by the maximum data error (denoted MDE). DE of data version d_i is defined as:

$$DE_i = 100 \times \frac{CV(d_i) - UV(d_i)}{CV(d_i)}(\%) \qquad (2)$$

We note that the quality of data (QoD) depends on its freshness and on DE, whereas the quality of transaction (QoT) depends on the Miss Ratio.

3 Multi-Versions Data-Feedback Control Scheduling Architecture

3.1 Introduction

It is well known that feedback control is a very effective in management of QoS in RTDBS, under unpredictable workloads [4]. The goal is to control the system performances, defined by a set of controlled variables in order

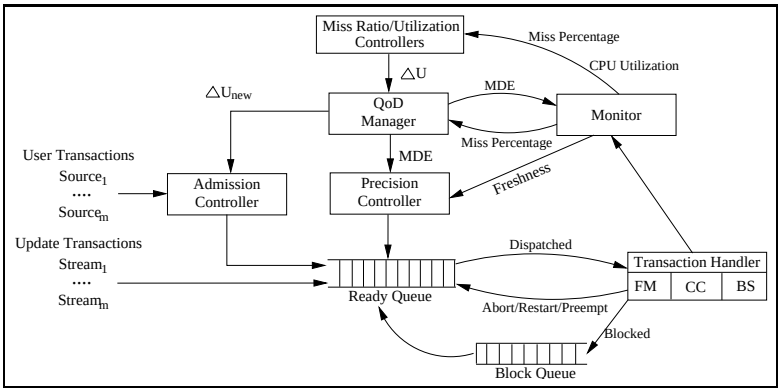


Figure 1. A Feedback Control Scheduling Architecture (FCSA) [2].

to satisfy a given QoS specification. The general outline of the feedback control scheduling architecture is given in Figure 1. A RTDBS consists of several components. In our study, the component we are interested in are: the admission controller, the ready queue, the block queue, and the transaction handler.

For the QoS management, a monitor, a miss ratio controller, an utilization controller and a QoD manager are added to the system in order to adjust its performances and to control the information flows. An Admission Controller (AC) is used to avoid system overload by reject some user transactions. Transactions handler provides a platform for managing transactions. It consists of a Concurrency Controller (CC), a Freshness Manager (FM) and a basic scheduler (BS). Transactions are scheduled by a Basic Scheduler in the ready queue using, for example, the EDF scheduling policy [9]. The FM checks the freshness before a transaction accesses a data item. It blocks a user transaction if the target data item is stale. Based on the two phase locking protocol, the CC ensures the concurrent transactions serializability. In case of conflict between transactions, when a higher priority transaction uses the data item, transactions with lower priority will be blocked. At each sampling period, the monitor samples the system performance data from the transaction manager and sends them to the controller. The miss ratio and utilization controller generates signals based on the sampled miss ratio and utilization data.

Feedback control has been proven to be very effective in supporting a required performance specification [3].

We have based our work on [2][5]. We have extended the FCS architecture by exploiting several versions of real-time data, and then proposed the Multi-Versions Data-Feedback Control Scheduling (MVD-FCS) approach [6][7]. In this section, we present a new approach which enhances MVD-FCSA depicted in Figure 2 where the solid arrows represent the transaction flows and the dotted arrows represent the real-time data flows.

3.2 Motivation for MVD-FCSA

In RTDBS, an update transaction always writes a real-time data item while a user transaction reads real-time data items. In general, only update transactions modify the real-time data. Most conflict cases come from incompatible access patterns when an update transaction wants to modify a data item that is accessed by user transactions. One of these transactions must be aborted and restarted according to the used concurrency control protocol. Furthermore, when the accessed data item are stale, the FM blocks the user transaction. This increases the risk that transactions miss their deadline. MVD notion is used to alleviate this risk. When an update transaction wants to modify a real-time data, a new data version is created. We consider all data values that correspond to different versions of the same data item.

To summarize, in the FCSA, two actions are considered important for improving the MVD-FCSA service:

- In case of conflict between transactions, when a higher priority transaction uses the data item, transactions with lower priority will be blocked.
- FM blocks user transactions if the acceded data are stale.

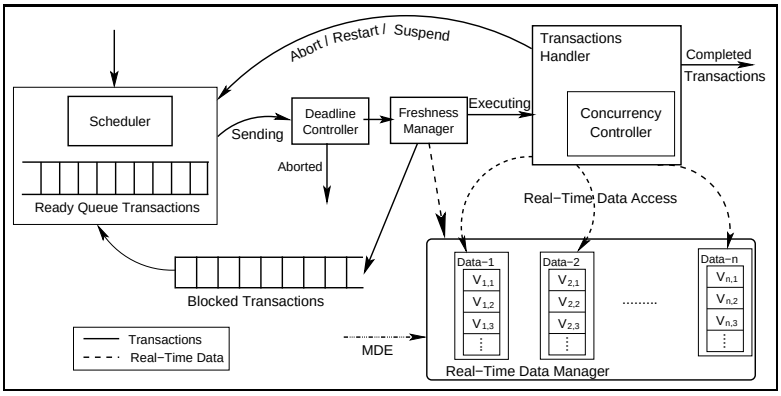


Figure 2. Multi-Version Data - Feedback Control Scheduling Architecture.

To enhance this protocol and to minimize the transaction miss ratio, we have proposed the MVD-FCSA, which consists of the creation of data versions as soon as conflicts (read-write) occur between transactions. This approach limits the data access conflicts between transactions, and then enhances the concurrency. When an update transaction wants to modify a real-time data, a new data version is created.

3.3 MVD-FCSA components

The majority of MVD-FCSA components exist in the classical feedback control scheduling architecture [2], but they are adapted as follows.

3.3.1 Real-time transactions Scheduler

Transactions are scheduled according to their priority. The priority of a transaction depends on both its deadline and its type (update or user transaction). Hence, we merge EDF policy with respect to transaction type and priority. A lower priority transaction can be scheduled if there are no ready transactions with higher priority to schedule.

3.3.2 Deadline Controller

To control transaction validity [9][8], the Deadline Controller (denoted DC) uses three controlled variables: transaction deadline, current time and minimal execution time. If the current time is greater than transaction deadline, transaction will be aborted. Otherwise, DC makes a second test, where a transaction is accepted only if the sum of the minimal execution time and the current time is lower than transaction deadline. Otherwise the transaction will be aborted. If the two verifications steps succeed, then the transaction is transferred to the data Freshness Manager (FM).

3.3.3 Freshness Manager

The Freshness Manager checks if transactions issued from the deadline controller access fresh data. Freshness Manager (FM) is used to provide better QoS in RTDBS where several transactions access to the same fresh data. It checks the freshness of acceded data just before a transaction commits. This way, the data accessed by committed transactions are always fresh at commit time. If the accessed data is fresh, transactions can be executed and then sent to the transactions handler. Otherwise, if the accessed data item is currently stale or if it's validity is estimated to be expired before the deadline of the transaction, then FM blocks the user transaction. The blocked transaction will be transferred from the blocked queue to the ready queue as soon as the corresponding update has committed. In our architecture, the FM checks the freshness of accessed data, i.e., AVI is not reached and data remains fresh until the end of the transaction execution. Therefore, we verify that AVI of accessed data is greater than the transaction deadline. To this purpose, we use MVD. So, the freshness condition must be considered by checking the freshness of the most recent data version that is not write-locked by other transaction(s).

3.3.4 Real-Time Data Manager

The main objective of this component is to guarantee the data freshness and to enhance

the deadline miss ratio even in the presence of conflicts and unpredictable workloads.

To achieve this goals, in [7] we have used a MVD with a fixed number of data versions. This number is fixed in advance by the DBA according to QoS requirement level, and it is the same for each real-time data. In [6], we have enhanced this approach by allowing the dynamic adjustment of the version number. For each data, we have a version queue. The queue is continuously updated in order to limit the number of data versions by supressing/adding versions, based on both the data freshness and MDE criterion. The size of each version queue, denoted SVQ, is dynamically adjusted according to the following formula:

$$SVQ_{i,j} = \lfloor \frac{AVI_j}{Period_i} \rfloor \quad (3)$$

where $Period_i$ is the periodicity of $transaction_i$, and AVI_j is the absolute validity interval of d_j.

RTDBS usually monitor the current real-world state using periodic updates. In this paper, we investigate several important problems to guarantee the desirable quality of real-time data services in terms of timeliness, freshness, precision, the decreasing of transaction miss deadline and the database size constraint. In a recent paper [6], we have shown that MVD-FCSA is a good solution to alleviate the risk that transactions miss their deadlines compared to the classical feedback control scheduling approaches. However, in [6] the proposed approach does not take into account the database size.

In this paper, we have extended the last approach (MVD-FCSA with dynamically adjusted number of data versions) by taking into account the database size constraint. We merge the two approaches described in [7] and [6], i.e. the number of data versions is dynamically adjusted and does not have to exceed the fixed threshold representing the number of data versions, and we have considered in the same time a threshold representing the database size. In this new approach, a data item will be accessed only if its version number is lower than the maximum database size. This way, RTDB size constraints are respected. The respect of the threshold of the RTDB size is a practical factor for RTDBS specification.

3.3.5 Concurrency Control with MVD

One of the most important issues in the design of RTDBS is the concurrency control component. Its objectives are (i) to control the interaction between concurrently transactions and (ii) to maintain the database consistency. In this paper, we focus on the interaction between update and user transactions.

The database consistency can be maintained using concurrency control protocols. We use 2PL-HP (Two Phase Locking-High Priority Protocol) where if a higher priority transaction accesses a data item, then other transactions (with lower priority) will be blocked. Otherwise, the transaction is aborted and restarted. Consequently, the 2PL-HP may increase the execution time of transactions. This leads the transaction to miss their deadlines. To address this problem, i.e. to alleviate this risk, we propose (1) the MVD technique that allows user transactions to access a less recent data version when update transaction writes a new data version, and (2) an adapted 2PL-HP when the maximum number of versions is reached. The priority is applied on transactions group that accessed to the same data version [6]. The priority of transactions group corresponds to the highest transaction priority among all transactions in this group.

4 Simulations and results

4.1 Simulations

We have studied and evaluated the behavior of an RTDBS according to a set of performance metrics. The performance evaluation is undertaken by a set of simulation experiments, where a set of parameters have been varied. Table 1 summarizes simulations parameters.

The simulated workload consists of update and user transactions. Update transactions occupy approximately 50% of the workload. The period of update transaction ($Period_i$) is uniformly distributed and estimated execution time is given by: $ExecutionTime_i = NbOfOperation_i \times OpExecTime$, where $NbOfOperation_i$ and the $OpExecTime$ represent respectively the number of operation in the transaction T_i and the execution time of an operation.

The model consists of eight components. We have used two transaction generators, an update transaction generator (UpdateTransGen) which generates update transactions and a user transaction generator (UserTransGen) which generates user transactions. The workload model characterizes transaction in terms of

Parameter	Meaning	Value
$NbOfOperations$	Number of operations in an user transaction	[1, 5]
$OpExecTime$	Execution time of an operation	1s
$Period_i$	Periodicity of update transaction	[1000ms, 5000ms]

Table 1. Parameters of Simulation.

the number of read/write operations. Update transaction can only write one data. Transactions are scheduled by a scheduler (Scheduler) in ready queue according to their priority. The priority assignment formula is given by P(T_i)= 1/deadline (T_i). Deadline controller (DC) uses three controlled variables: transaction deadline (deadline), current time (StartTime) and minimal execution time (ExecutionTime). The deadline formula is calculated as follows: $deadline(T_i) = StartTime + ExecutionTime \times (1 + SlakTime)$ where $SlakTime$ is a constant that provides control over tightness/slackness of transaction deadlines. To check the freshness of accessed data, the freshness manager use the AVI parameter. Transactions handler controls the execution of transactions. Concurrency controller use the adapted 2PL-HP protocol to control the interaction between transactions [6]. The real-time data management (RTDM) is the most important component of our model. In the sets of experiments, we have varied the database size and the maximum number of data versions for each data item. The database size represents the number of versions.

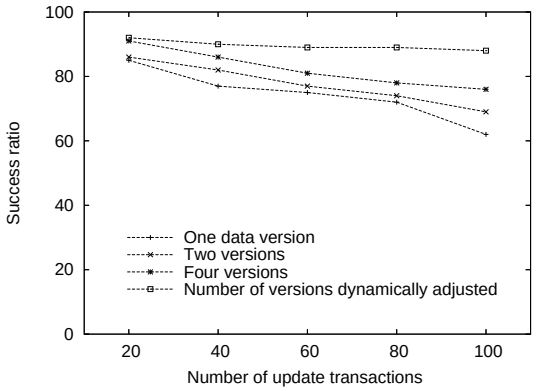
4.2 Results

Since in our approach we have only extend the transactions flows of the classical FCSA, the performance metric in our experiments is the success ratio. The graphical results show the miss ratio of transactions when using MVD-FCSA. We have evaluated the behavior of the system by varying a set of parameters:

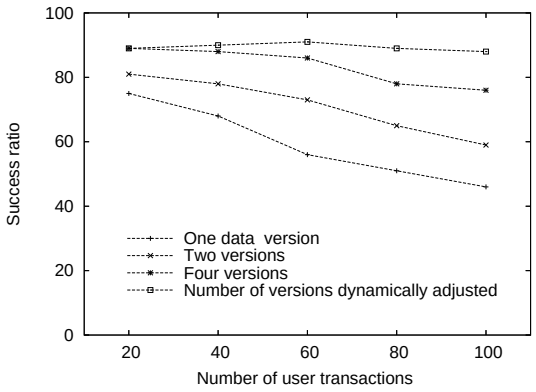
1. The threshold of data versions number
2. The threshold of database size
3. The number of transactions

4.2.1 Experiment 1: Results of MVD-FCSA

As shown in Figure 3, when we use the classical FCSA (with one version), the MVD-FCSA with two versions and the MVD-FCSA with four versions, the resulting success ratio increases according to the increase of the number of versions.



(a) For update transactions



(b) For user transactions

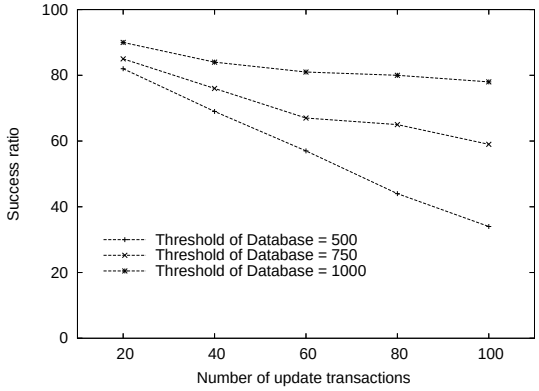
Figure 3. Simulation results of the MVD-FCSA.

Compared to the effect of using MVD-FCSA with fixed number of versions, using MVD-FCSA with dynamically adjusted number of data versions shows a relatively high success ratio, as shown in Figure 3.

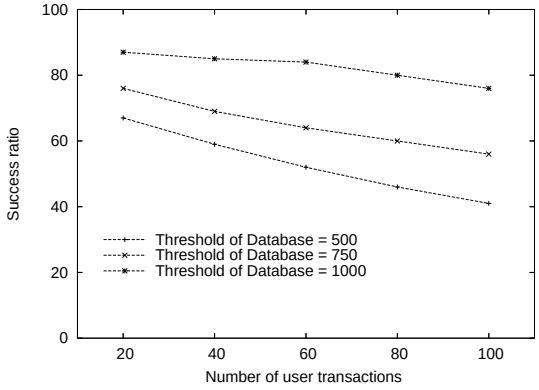
4.2.2 Experiment 2: Varying the threshold of database size using the mixed approach of MVD-FCSA

We use our mixed approach (dynamic adjustment of data versions with maximum fixed number). In Figure 4, we have fixed a threshold of data versions number (equal to 4 versions) and we have varied the database size (500, 750, 1000). In Figure 5, we have also varied the database size, while the threshold of data versions number is equal to 6.

Figures 4 et 5 show the effect of varying the database size. The resulting success ratio increases according to the increase of the database size.

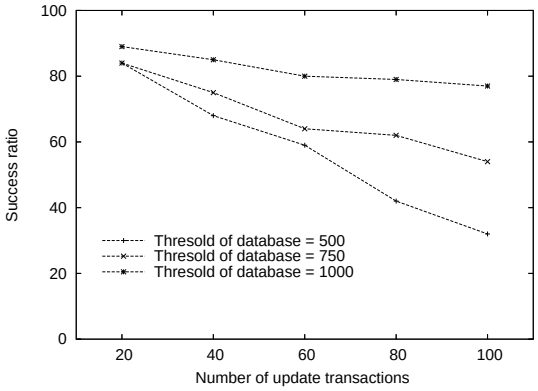


(a) Update transactions

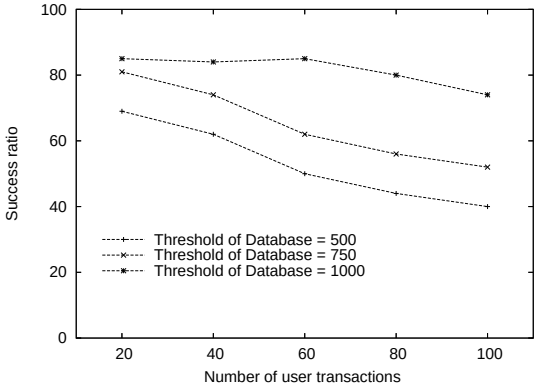


(b) For user transactions

Figure 4. Simulation results when using the mixed approach of MVD-FCSA (maximum number of versions = 4) and varying the threshold of database size.



(a) For update transactions



(b) For user transactions

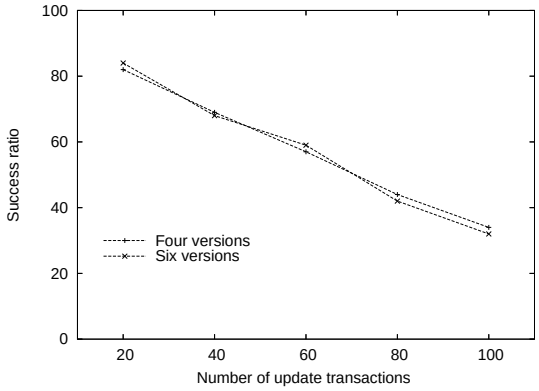
Figure 5. Simulation results when using the mixed approach of MVD-FCSA (maximum number of versions = 6) and varying the threshold of database size.

4.2.3 Experiment 3: Varying the threshold of data versions number

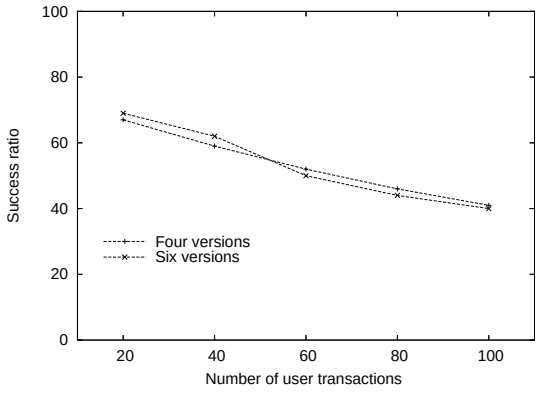
We also use the mixed approach of MVD-FCSA. We have fixed the database size and we have varied the threshold of data versions number. Compared to the effect of using a maximum of six versions, the use of four versions shows a relatively high success ratio, as shown in Figures 6 and 7.

4.3 Summary of results and discussions

We have compared the system performances, in terms of miss ratio, by varying the

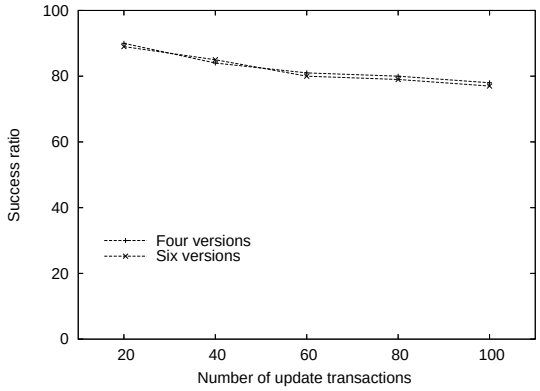


(a) For update transactions

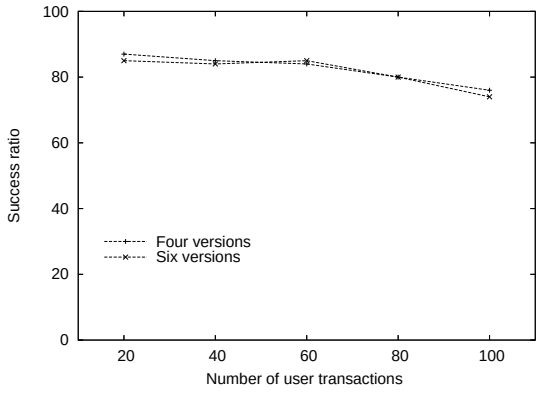


(b) For user transactions

Figure 6. Simulation results of using the mixed approach of MVD-FCSA: varying the number of versions and the threshold of database size 500.



(a) For update transactions



(b) For user transactions

Figure 7. Simulation results of using the mixed approach of MVD-FCSA: varying the number of versions and the threshold of database size 1000.

database size and by varying the maximum number of data versions. All experiments simulation show that:

1. MVD-FCSA minimizes transactions miss deadline (compared to the classical FCSA).

2. The success ratio increases according to the decrease of the number of versions.

3. The success ratio increases according to the increase of the database size.

We can also compare the system performances, in terms of miss ratio, by using the transactions type (user or update). In comparison with the curves show in Figure 3(a), the curves in Figure 3(b) are closer. This is because, in our approach, the update transactions have generally, the higher priority. That's why, in Figure 3, the increase of the versions number is the most significant and influential criterion on success ratio.

5 Conclusion and future work

In this paper, we have presented the multi-versions data-feedback control scheduling architecture for quality of service management. We have used multi-versions data with dynamically adjusted number of versions while taking into account the RTDB size constraint. This improvement consists in minimizing the number of conflicts by minimizing the number of

aborted transactions when using an adapted 2PL-HP concurrency control protocol.

Simulation results show that MVD-FCSA with dynamically adjusted number of data versions may be applied efficiently in RTDBS, i.e more transactions meet their deadlines. We note that the respect of the threshold of the RTDB size might be a practical factor for RTDBS specification.

We plan to extend this work in several ways. We will take into account the data *importance*. Indeed, in case of a small threshold of the RTDB size, all data beyond the threshold value are not accessed whatever their importance. The importance of the data item may be modeled by assigning to each data item a weight according to its importance. Further, we also plan to extend our work to manage derived data and to consider other aspects to study different components of the feedback control scheduling architecture for quality of service management in RTDBS. Among them, we will deal with imprecise computing [2], applied to video contents.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. *International Journal of Distributed and Parallel Databases*, 1(2), 1988.

[2] M. Amirijoo, J. Hansson, and S. H. Son. Algorithms for Managing Real-time Data Services Using Imprecise Computation. In *Proceedings of International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, Taiwan, 2003.

[3] M. Amirijoo, J. Hansson, and S. H. Son. Error-Driven QoS Management in Imprecise Real-Time Databases. In *Proceedings of* 15^{th} *Euromicro Conference on Real-Time Systems (ECRTS)*, Portugal, 2003.

[4] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Imprecise Real-Time Databases. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*, Hong Kong, 2003.

[5] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. *IEEE Transaction Knowledge and Data Engineering*, 55(3):304–319, March 2006.

[6] E. Bouazizi, C. Duvallet, and B. Sadeg. Management of QoS and Data Freshness in RTDBSs using Feedback Control Scheduling and Data Versions. In 8^{th} *IEEE International Symposium on Object-oriented Real-time distributed Computing (IEEE-ISORC'05)*, Washington, 2005.

[7] E. Bouazizi, C. Duvallet, and B. Sadeg. Using Feedback Control Scheduling and Data Versions to enhance Quality of Data in RTDBSs. In *IEEE International Computer System and Information Technology (ICSIT'2005)*, Alger, Algerie, 2005.

[8] C.S. Date. *An Introduction to Database Systems*. Addison-Wesley, 1985.

[9] C. Liu and J. Leyland. Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[10] C. Lu. *Feedback Control Real-Time Scheduling*. PhD thesis, University of Virginia, May 2001.

[11] C. Lu, J.A. Stankovich, G. Tao, and S.H. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Real-Time Systems*, 23(1/2):85–126, 2002.

[12] K. Ramamritham. Real-Time Databases. *Journal of Distributed and Parallel Databases*, 1(2):199–226, 1993.

[13] K. Ramamritham, S.H. Son, and L.C. DiPippo. Real-Time Databases and Data Services. *Real-Time Systems*, 28:179–215, 2004.