Une nouvelle approche pour la gestion de la QdS[1] dans les SGBD temps réel

Emna Bouazizi, Claude Duvallet et Bruno Sadeg

LITIS, UFR des Sciences et Techniques,
25 rue Philippe Lebon, BP 540
76 058 Le Havre Cedex, FRANCE.
{Emna.Bouazizi,Claude.Duvallet,Bruno.Sadeg}@univ-lehavre.fr

RÉSUMÉ. *Ces dernières années, les besoins en termes de données et de services temps réel se sont beaucoup accrus dans un grand nombre d'applications. Traditionnellement, ces applications sont gérées par des systèmes temps réel, bien adaptés pour la prise en compte des contraintes temporelles. Cependant, ils ne sont pas satisfaisants pour la gestion efficace de grands volumes de données. Les systèmes de type SGBDTR[2] ont été conçus pour gérer de grandes quantités de données tout en respectant les contraintes temps réel des applications. Mais le système doit souvent faire face à des charges d'utilisation transitoires, à cause de l'imprévisibilité d'accès aux données de la base. L'objet de nos travaux est de maintenir le comportement du SGBDTR dans un état stable et de diminuer le nombre de transactions qui ratent leur échéance. Ce qui risque de se produire en particulier lorsqu'elles doivent être redémarrées suite à un conflit d'accès aux données. Dans cet article, nous présentons nos travaux qui s'appuient sur des modèles de SGBDTR utilisant une boucle de rétroaction. Cette boucle permet de contrôler le comportement du SGBDTR, notamment durant les phases d'instabilité, afin d'améliorer la qualité de service (QdS) des applications. Nous proposons de réduire, voire de supprimer, les conflits d'accès aux données temps réel en créant plusieurs versions de chaque donnée lorsqu'un conflit d'accès survient. Nous explorons pour cela plusieurs approches comme la variation de la taille maximale de la base de données et du nombre de versions pour une donnée.*

ABSTRACT. *These recent years, a lot of applications are becoming increasingly sophisticated in their needs for real-time data and real-time services. Traditionally, these applications are managed by real-time systems, which are well-adapted to manage temporal constraints. But, real-time systems are less efficient when they have to manage large amount of data. Real-Time Database Systems (RTDBSs) are systems designed to deal with great quantities of data while respecting the applications real-time 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. This may often occur particularly when transactions have to be aborted and restarted in case of data access conflict. In this paper, we propose 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. We propose a technique which allows to minimize the number of conflicts by exploiting multi-versions data, while taking into account the database size.*

MOTS-CLÉS : *SGBD temps réel, Transactions temps réel, Qualité de Service, Données Multi-Versions*

KEYWORDS: *RTDBS, Real-Time Transactions, Quality of Service, Multi-Versions Data*

1. Qualité de service.
2. Système de gestion de base de données temps réel.

1. Introduction

De nombreuses applications ont besoin d'utiliser des quantités importantes de données temps réel. Il s'agit par exemple des applications utilisées pour le contrôle des usines chimiques ou des centrales nucléaires et des applications de commerce électronique. Ces applications doivent à la fois fournir des résultats respectant des échéances, mais aussi manipuler des données temps réel (données sensorielles de base) ou calculées à partir d'autres données (données dérivées) [DUV 99] [RAM 93] [RAM 04]. Dans ce type d'applications, les transactions soumises par les utilisateurs arrivent à des fréquences variables. Lorsque la fréquence augmente de façon considérable, l'équilibre du SGBDTR est mis en péril. Durant ces périodes de surcharge, le SGBDTR va potentiellement manquer de ressources et les transactions temps réel vont alors manquer leur échéance en plus grand nombre. Pour gérer ces phases d'instabilité du système (phase de surcharge ou phase de sous utilisation), des travaux basés sur une approche QdS (Qualité de Service) [KAN 02a] [AMI 03c] tentent de rendre les SGBDTR plus robustes. Ces travaux s'appuient sur des techniques de contrôle avec rétroaction (par exemple FCSA[1] [LU 01]) et autorisent la manipulation de résultats imprécis [LIU 91] [AMI 03a]. Dans les SGBDTR, le non respect des échéances des transactions est souvent dû à des conflits d'accès aux données qui entraînent l'abandon puis le redémarrage tardif des transactions les moins prioritaires.

Dans ce travail, notre objectif est de réduire de façon substantielle le nombre de conflits entre les transactions pour l'accès aux données temps réel. Notre approche, appelée "MVD-FCSA" (Multi-Versions Data-FCSA) consiste à créer plusieurs versions des données temps réel pour éviter les conflits d'accès en mode lecture/écriture ou écriture/lecture. Nous considérons trois politiques de gestion de données. Dans la première politique, nous avons utilisé le même nombre fixe de versions pour chaque donnée, alors que dans la deuxième politique, le système est basé sur un ajustement dynamique du nombre de versions pour chaque donnée. Dans la troisième politique, nous combinons les deux précédentes politiques (1) en limitant le nombre de versions et (2) en ajustant dynamiquement cette limite selon un seuil qui fixe la taille maximale de la base de données. Les résultats des simulations que nous avons effectuées ont confirmé l'apport de l'utilisation des données multi-versions par rapport à l'architecture classique du contrôle par rétroaction et ont permis d'établir des comparaisons entre les trois politiques de l'approche MVD-FCSA pour mettre en évidence l'importance du facteur *taille de la base de données*.

Cet article est structuré de la manière suivante. Dans la section 2, nous présentons le modèle de SGBDTR basé sur le principe de rétroaction. Dans la section 3, nous présentons l'architecture que nous proposons et qui est basée sur des données multi-versions. La section 4 est consacrée à la présentation de simulations et des résultats obtenus. Enfin, nous concluons sur l'apport de notre travail et sur les perspectives que nous envisageons.

2. Le modèle de SGBD temps réel basé sur la rétroaction

Dans une application temps réel reposant sur l'utilisation d'un SGBDTR, il n'est pas toujours possible de prévoir les arrivées des transactions *utilisateur*. En conséquence, des phases d'instabilité (phase de surcharge ou phase de sous utilisation) peuvent se produire. Pour gérer ces phases d'instabilité du système, des travaux basés sur une approche QdS [KAN 02a] [AMI 03a] tentent de rendre les SGBDTR plus robustes. Ces travaux s'appuient sur une technique de contrôle avec rétroaction [LU 01].

Dans cette section, nous allons décrire ces travaux [BOU 05a] [BOU 05b], puis nous présentons le modèle de SGBDTR que nous considérons, ainsi que l'approche que nous proposons.

2.1. *Le modèle de données*

Les valeurs des données temps réel dans la base doivent représenter l'état du monde réel. Ces données doivent donc être mises à jour régulièrement afin qu'elles reflètent le plus fidèlement possible l'état du monde réel. Chaque donnée possède une durée de validité pendant laquelle elle peut être utilisée de manière

1. Feedback Control Scheduling Architecture.

efficace [RAM 04]. Cette durée de validité est matérialisée par un paramètre, appelé AVI (intervalle de validité absolue).

Dans [AMI 03a] et [AMI 03c], Amirijoo et al. ont introduit la notion de qualité des données (QdD) qui permet de considérer qu'une donnée stockée dans la base peut présenter un certain écart par rapport à sa valeur dans le monde réel. On appelle cet écart "erreur sur la donnée" (notée DE[2]) et il est calculé en faisant la différence entre la valeur de la donnée dans la base et sa valeur dans le monde réel. Cet écart possède un seuil d'erreur maximum (MDE[3]) qui permet de déterminer si la transaction qui souhaite mettre à jour une donnée temps réel peut être écartée (si $DE \leq MDE$) ou pas (si $DE > MDE$). Ce travail est effectué par le contrôleur de précision (cf. Figure 1).

2.2. *Le modèle de transactions*

Les transactions temps réel considérées dans nos travaux possèdent des échéances de type firm (échéances strictes non critiques [DUV 99]). C'est à dire que lorsqu'elles dépassent leur échéance, elles deviennent inutiles pour le système et sont abandonnées. Nous considérons deux types de transactions temps réel :

– Les transactions de mise à jour : elles ont pour tâche de mettre à jour régulièrement les données de base. Ces transactions sont exécutées périodiquement pour rafraîchir la base de données.

– Les transactions *utilisateur* : elles effectuent des opérations de lecture/écriture de données non temps réel et/ou des lectures de données temps réel. L'imprévisibilité de leur arrivée dans le système et la charge qu'elles suscitent rend adéquate l'utilisation d'une architecture basée sur la rétroaction pour gérer les variations importantes de charge [LU 01].

Dans cet article, nous considérons qu'une transaction de mise à jour crée une nouvelle version d'une donnée à chaque fois qu'elle effectue une écriture dans la base de données. Les transactions *utilisateur* accèdent alors à la donnée la plus récente qui n'a pas été verrouillée en écriture (voir paragraphe 3.2.4).

2.3. *Les critères de performance*

Trois principales mesures de performance sont considérées dans les travaux relatifs au modèle de SGBDTR basé sur le principe de rétroaction [AMI 03b] [AMI 03c] : $MissRatio$ (MR), $DataFreshness$ (DF) et $DataError$ (DE).

1) $MissRatio$: ce paramètre est défini comme suit :

$$MR = 100 \times \frac{\#Tardy}{\#Submitted}(\%) \quad (1)$$

où $\#Tardy$ représente le nombre de transactions qui ratent leur échéance, et $\#Submitted$ est le nombre total des transactions.

2) $DataFreshness$: dans une base de données temps réel, une donnée peut devenir obsolète (non fraîche) avec le passage du temps. Pour mesurer la fraîcheur d'une donnée d_i, on utilise son paramètre AVI. Une version de donnée possède une estampille qui indique la valeur de la dernière observation de la donnée dans le monde réel. La donnée d_i est considérée comme fraîche si : $TempsCourant - Estampille(d_i) \leqslant AVI(d_i)$. La fraîcheur de la base de données peut ainsi être mesurée. Elle représente le rapport entre les données fraîches et l'ensemble des données de la base.

3) $DataError$: représente l'écart entre la valeur courante de la donnée (*CurrentValue*(d_i)) et sa valeur mise à jour (*UpdateValue*(d_i)). La limite supérieure de l'erreur est représentée par le paramètre MDE, erreur maximale sur la donnée (voir paragraphe 2.1). Le paramètre DE d'une version de donnée d_i est défini comme :

$$DE_i = 100 \times |\frac{CurrentValue(d_i) - UpdateValue(d_i)}{CurrentValue(d_i)}|(\%) \quad (2)$$

2. Data Error.
3. Maximum Data Error.

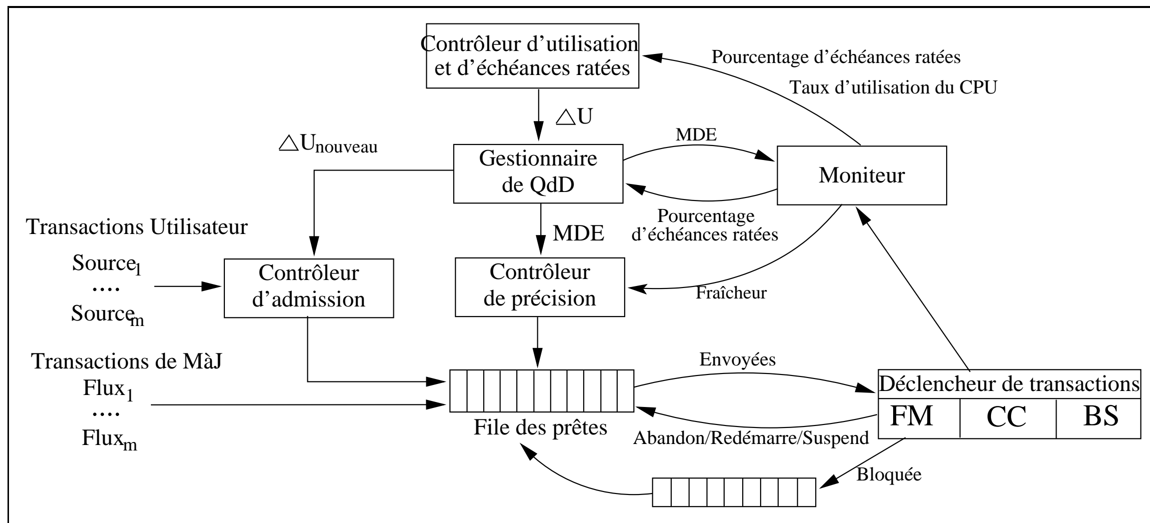


Figure 1. *Architecture de SGBDTR basée sur la rétroaction.*

Notons que la qualité de donnée (QdD) dépend de sa fraîcheur et de la valeur du paramètre DE.

2.4. *L'architecture d'ordonnancement contrôlé par rétroaction*

La figure 1 illustre le modèle général basé sur une architecture de contrôle par rétroaction et utilisé pour la gestion de la qualité de service [KAN 02b] [AMI 03c]. Nous allons donner une brève description de chacun des composants du modèle.

2.4.1. *Le contrôleur d'admission*

La tâche du contrôleur d'admission est de sélectionner les transactions *utilisateur* qui seront acceptées ou pas dans le système. Il effectue ce contrôle en fonction de la charge d'utilisation calculée et des paramètres de qualité de service spécifiés par le DBA[4]. Son fonctionnement est contrôlé par la boucle de rétroaction qui lui fournit ses paramètres.

2.4.2. *Le déclencheur de transactions*

Les transactions qui sont admises dans le système sont placées dans une file d'attente avant d'être envoyées au déclencheur de transactions, qui a pour fonction de gérer l'exécution des transactions. Il dispose de plusieurs modules complémentaires :

– Un gestionnaire de fraîcheur : il vérifie la fraîcheur des données qui vont être accédées par une transaction. Si les données sont obsolètes alors la transaction est mise en attente dans une file.

– Un contrôleur de concurrence : il est chargé de gérer les conflits d'accès aux données qui surviennent entre les transactions. Dans la plupart de ces travaux [KAN 02b] [AMI 03a], il s'agit du protocole 2PL-HP (Two Phase Locking High Priority) [ABB 88].

– Un ordonnanceur de base : il s'agit bien souvent d'EDF (Earliest Deadline First) [LIU 73] [BUT 97], dont le principe est de donner la plus haute priorité à la transaction qui possède l'échéance la plus proche.

2.4.3. *Le moniteur*

Il permet de mesurer les performances du système en inspectant l'exécution des transactions (nombre de transactions terminées, abandonnées, qui ont ratées leur échéance, ...). Les valeurs ainsi mesurées font

4. Administrateur de la base de données.

partie de la boucle de contrôle par rétroaction qui contribue à stabiliser le système et permettent d'alimenter le contrôleur d'utilisation et d'échéances ratées.

2.4.4. *Le contrôleur d'utilisation et d'échéances ratées*

Ce composant permet de réajuster les paramètres de QdS en fonction des valeurs déterminées par le moniteur et des paramètres de référence fixés par le DBA. Les valeurs ainsi obtenues sont transmises au gestionnaire de qualité des données.

2.4.5. *Le gestionnaire de qualité des données*

Il permet de réajuster la valeur du paramètre MDE (Maximum Data Error) qui constitue le paramètre de QdD. La valeur de MDE est calculée en fonction de l'utilisation du système. Ce paramètre est ensuite fourni au contrôleur de précision qui écarte les transactions de mise à jour lorsque les données à mettre à jour sont suffisamment représentatives du monde réel en considérant la valeur de MDE. Le gain d'utilisation ainsi obtenu est considéré, puis l'effort supplémentaire à fournir pour stabiliser le système est demandé au contrôleur d'admission (envoi de $\Delta\, U_{new}$ au contrôleur d'admission).

2.5. *La boucle de contrôle par rétroaction*

La boucle de rétroaction a pour tâche de stabiliser le système durant les phases de surcharge. Pour cela, elle s'appuie sur le principe d'observation puis d'auto-adaptation. L'auto-adaptation a lieu tout au long du fonctionnement du système car les demandes des utilisateurs sont imprévisibles et la charge doit être ajustée en permanence. L'observation consiste à prendre en compte l'état de fonctionnement du système et à déterminer s'il correspond aux paramètres de qualité de service initialement spécifiés. Cette observation se fait via le moniteur. À partir de l'observation effectuée, le système adapte ses paramètres, via le contrôleur de qualité de service, afin d'augmenter ou de diminuer le nombre de transactions acceptées dans le système. Le fonctionnement de la boucle de rétroaction doit faire tendre le système vers un état stable représenté par une valeur de référence fixée par le DBA (par exemple, il peut s'agir d'un taux d'utilisation de 80%).

3. MVD-FCSA : une architecture d'ordonnancement contrôlé par rétroaction pour les données multi-versions

3.1. *Motivation*

Dans les SGBDTR, on utilise des transactions qui permettent d'effectuer des actions plus ou moins complexes basées sur des opérations de lecture et d'écriture dans une base de données. Ces données peuvent être de type temps réel ou non. Lorsqu'une donnée est accédée par plusieurs transactions dans des modes incompatibles[5], il se produit un conflit d'accès. Pour résoudre ce conflit, on peut selon les cas mettre en attente ou abandonner des transactions. La suspension ou l'abandon puis le redémarrage de certaines transactions augmentent de façon considérable le risque qu'elles ratent leur échéance. C'est pourquoi nous proposons une nouvelle approche pour réduire le nombre de conflits entre les transactions et réduire ainsi le nombre de transactions qui ratent leur échéance. Nos travaux concernent l'amélioration de l'architecture de contrôle des ordonnancements temps réel avec rétroaction dans une approche basée sur la spécification de la qualité de service [KAN 02b] [AMI 03c] [AMI 03a].

3.2. *Les composants de MVD-FCSA*

Comme indiqué sur la figure 2, l'architecture MVD-FCSA est composée des éléments suivants : un contrôleur d'échéances, un gestionnaire de fraîcheur, un contrôleur de concurrence et un gestionnaire de données temps réel.

5. Lecture/Écriture, Écriture/Lecture, Écriture/Écriture.

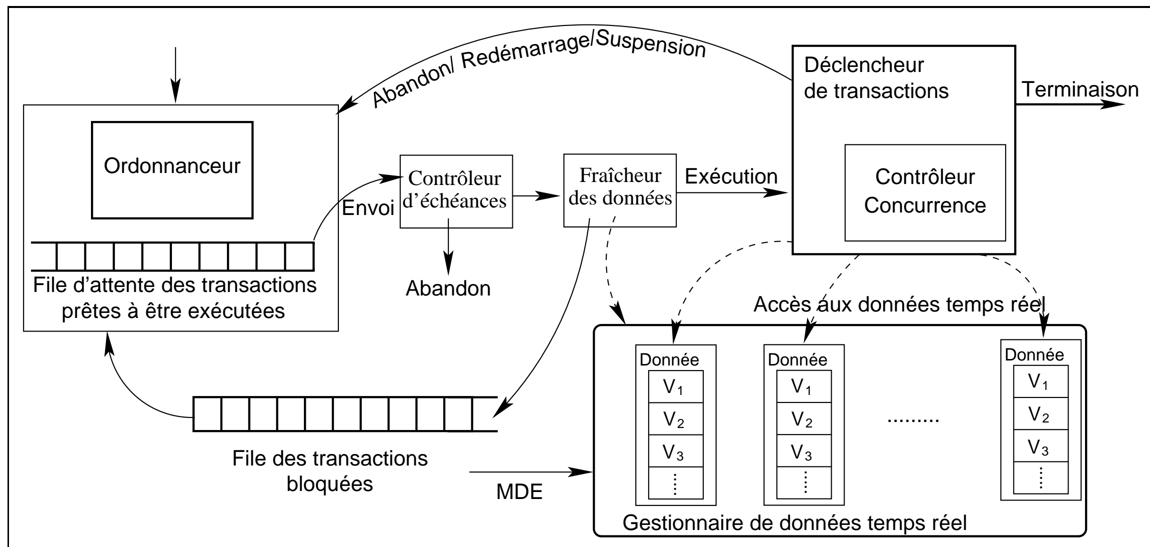


Figure 2. *Gestion de données multi-versions dans une architecture d'ordonnancement temps réel par rétroaction.*

3.2.1. *Le contrôleur d'échéances*

Il cumule la durée minimale d'exécution d'une transaction avec la date courante. Il vérifie ensuite que cette valeur cumulée est inférieure à l'échéance de la transaction. Si cette valeur est supérieure à l'échéance alors la transaction ne pourra pas se terminer et valider avant son échéance, et elle est écartée. Lorsque la transaction a passé cette vérification avec succès, elle est transmise au gestionnaire de fraîcheur des données.

3.2.2. *Le gestionnaire de fraîcheur*

Le gestionnaire de fraîcheur, d'une part, vérifie que les données sensorielles (temps réel) sont fraîches et qu'elles le resteront jusqu'à la fin de l'exécution de la transaction qui souhaite y accéder. Pour cela, il vérifie que la borne supérieure de l'intervalle de validité des données accédées est supérieure à l'échéance de la transaction. D'autre part, la vérification est effectuée par rapport à la valeur la plus fraîche de la donnée qui n'est pas verrouillée en écriture par une autre transaction. Rappelons qu'il existe dans le système plusieurs versions pour chaque donnée sensorielle.

3.2.3. *Le contrôleur de concurrence*

Dans cet article, nous ne considérons que les données temps réel. La plupart des conflits proviennent du fait qu'une transaction veuille mettre à jour une donnée qui est accédée par une autre transaction (*utilisateur*) en mode lecture et qui n'a pas encore été validée. Dans le protocole 2PL-HP, on considère que si la transaction qui lit la donnée est plus prioritaire alors la transaction de mise à jour est bloquée. Dans le cas contraire, la transaction qui lit la donnée est abandonnée puis redémarrée. Dans les deux cas, la durée d'exécution de la transaction est rallongée et donc les risques qu'elle rate son échéance sont augmentés. Pour tenter de diminuer ce risque, nous utilisons la technique de MVD qui permet à une transaction *utilisateur* d'accéder à une ancienne version de la donnée quand la donnée est verrouillée en écriture : nous considérons que la transaction de mise à jour crée une nouvelle version de la donnée. Ainsi la transaction "de lecture" peut continuer à considérer une *ancienne* version de la donnée pendant que la transaction de mise à jour écrit une nouvelle valeur.

Lorsque le nombre maximal de versions d'une même donnée est atteint et qu'une transaction veut écrire une nouvelle version de cette donnée, on met en œuvre un protocole de contrôle de concurrence. Ce protocole doit tenir compte de l'ensemble des transactions qui accèdent aux différentes versions des données et de la transaction qui veut mettre à jour la donnée. Il faut noter qu'il est possible qu'une donnée soit accédée en lecture par plusieurs transactions. La transaction, qui entre en conflit sur la donnée, va donc devoir consi-

dérer chacun des groupes de transactions accédant aux différentes versions de la donnée. Nous appliquons alors le protocole 2PL-HP-MVD qui est une adaptation du 2PL-HP pour les données multi-versions (cf. Algorithme 1).

3.2.4. *Le gestionnaire de données temps réel*

Les données temps réel sont mises à jour par des transactions de mise à jour. Ces données sont utilisées en lecture par des transactions *utilisateur*. On considère que seules les transactions de mise à jour accèdent aux données temps réel en mode écriture. Lorsqu'une transaction de mise à jour souhaite écrire une donnée temps réel, une nouvelle version est créée.

La gestion des données doit être faite, d'une part de façon à garantir la fraîcheur des données, et d'autre part de façon à respecter le seuil d'erreur toléré sur la donnée (MDE) pour être en accord avec l'architecture du contrôle d'ordonnancement par rétroaction. Lorsqu'une version d'une donnée ne respecte pas l'un de ces deux paramètres de qualité des données, elle est supprimée de la base de données.

Afin d'éviter le stockage de versions inutiles de données, celles qui ne seront pas accédées seront supprimées à condition qu'il ne s'agisse pas de l'unique version accessible. C'est à dire qu'il existe au moins une autre version de cette donnée respectant les paramètres de la qualité des données et qui n'est pas verrouillée en écriture.

– MVD avec un nombre fixe de versions : Il s'agit de limiter le nombre de versions d'une même donnée afin d'éviter une augmentation trop importante de la taille de la base de données. Lorsqu'une transaction libère une version ancienne (qui n'est pas la plus récente), plusieurs solutions sont considérées :

- Nous supprimons la version ainsi libérée, ce qui permet de libérer une place et éventuellement de pouvoir stocker une version plus récente de la donnée. On ne doit faire cela que si la version que l'on souhaite supprimer n'est pas accédée par une autre transaction.

- Nous gardons la donnée dans l'éventualité où la donnée la plus fraîche serait verrouillée par une transaction de mise à jour. On la supprimera dès qu'une donnée plus fraîche sera disponible et non verrouillée en écriture.

Lorsque le nombre maximal de versions d'une même donnée (fixé initialement par le concepteur de la base selon les paramètres de QdS exigés) est atteint et qu'une transaction veut écrire une nouvelle version de cette donnée, on met alors en œuvre le protocole de contrôle de concurrence 2PL-HP-MVD.

– MVD avec un ajustement dynamique du nombre de versions : Dans cette approche, il ne s'agit plus de fixer le nombre de versions pour chaque donnée. Ce nombre est plutôt ajusté dynamiquement. Le principe est le suivant : chaque donnée possède sa propre file de versions dont le nombre de versions est indépendant du nombre des versions des autres données. En effet, la file est continuellement mise à jour en supprimant/ajoutant des versions afin de limiter les nombres de versions en tenant compte du critère de fraîcheur et du paramètre MDE. La taille de la queue des versions, notée SVQ, est dynamiquement ajustée, et est définie comme suit :

$$SVQ = \lfloor \frac{AVI_j}{Period_i} \rfloor \qquad (3)$$

Où $Period_i$ est la période de la $transaction_i$, et AVI_j est l'intervalle de validité de la donnée d_j.

– Approche mixte : Nous prenons en considération plusieurs paramètres pour satisfaire la QdS souhaitée en termes de fraîcheur et de précision des données, augmentant ainsi le nombre de transactions qui réussissent avant échéance. Il s'agit de combiner les deux approches précédentes (MVD avec un nombre fixe de versions et MVD avec un ajustement dynamique du nombre de versions) et de prendre en compte la taille de la base de données. Le nombre de versions d'une donnée ne doit pas dépasser un seuil fixé par le DBA et représente le nombre de versions maximum autorisé. Dans cette approche, une donnée est accédée si et seulement si la taille cumulée de toutes les versions est inférieure à la taille maximale de la base de données. Cette approche garantit le respect de la contrainte "taille de la base de données".

Remarque : La priorité d'un groupe de transactions correspond à la priorité la plus élevée parmi celles de toutes les transactions du groupe.

Algorithme 1 Protocole 2PL-HP-MVD

```
Tr_maj  : Transaction de mise à jour (accès en écriture).
Tr_user : Transaction utilisateur (accès en lecture).
Gr(Tr_user) : Un groupe de transactions utilisateur accédant à une même version d'une donnée.
Gr_i(Tr_user) : ième groupe de transactions utilisateur.
nb_groupes : Nombre de groupes de transactions accédant chacun à une version de la donnée.
Debut
    Si Priorité (Tr_maj) < Priorité (Gr(Tr_user)) Alors
        Tr_maj bloquée en attente de libération d'une version
    Sinon
        Pour i allant de 1 à nb_groupes Faire
            Si Priorité (Tr_maj) > Priorité (Gr_i(Tr_user)) Alors
                Abandon et redémarrage de Gr_i(Tr_user)
            Finsi
        FinPour
    Finsi
Fin
```

4. Simulations et résultats

4.1. *Principe de simulation*

Pour étudier les bénéfices apportés par notre modèle MVD-FCSA, nous allons comparer ses performances par rapport au modèle FCSA classique. Ce dernier peut être vu comme un modèle MVD-FCSA particulier où le nombre de versions est égal à 1. Pour évaluer les performances de MVD-FCSA, nous avons réalisé une série d'expériences en faisant varier les valeurs de certains paramètres. Le tableau 1 récapitule ces valeurs. Dans notre modèle, le principe est le suivant : chaque transaction, quelque soit son type (*utilisateur* ou mise à jour), doit passer par une suite de tests depuis sa création jusqu'à son exécution. Les transactions de mise à jour représentent 50% de l'ensemble des transactions générées par le système.

Le modèle est composé de huit composants. Nous avons commencé par implémenter deux générateurs de transactions. L'arrivée des transactions *utilisateur* dans le système étant imprévisible, le générateur des transactions *utilisateur* utilise une distribution aléatoire pour générer les transactions. L'arrivée des transactions de mise à jour étant périodique, le générateur des transactions de mise à jour génère les transactions suivant un processus d'arrivée bien défini, qui respecte la périodicité des transactions.

Le temps d'exécution est estimé comme suit : ExecutionTime = NbOfOperations $\times$ OpExecTime, où NbOfOperations représente le nombre d'opérations dans une transaction T_i et OpExecTime représente respectivement le temps d'exécution d'une opération de lecture (OpLecExecTime) ou d'une opération d'écriture (OpEcrExecTime). L'ordonnanceur reçoit les transactions fournies par les deux générateurs et ordonnance les transactions reçues dans la file des transactions prêtes à être exécutées en fonction de leurs priorités, selon la formule : P(T_i) = 1 / échéance (T_i). Dès sa sortie de la file, la transaction va subir une suite de tests effectués par les autres éléments de l'architecture. Le contrôleur d'échéances (DC) utilise trois variables contrôlées : l'échéance de la transaction (deadline), le temps courant (StartTime) et le temps d'exécution minimal (ExecutionTime). L'échéance de la transaction est calculée comme suit : deadline (T_i) = StartTime + ExecutionTime $\times$ (1 + SlackTime), où SlackTime est une constante qui contrôle le rapport exigence/relâchement (tightness/slackness) des échéances des transactions.

Pour vérifier la fraîcheur de la donnée accédée, le gestionnaire de fraîcheur (FM) utilise le paramètre AVI. Le gestionnaire des transactions (TM) supervise l'exécution des transactions. Le contrôleur de concurrence (CC) utilise le protocole 2PL-HP-MVD pour gérer les interactions entre les transactions. Le gestionnaire des données temps réel (RTDM) est la composante la plus importante dans notre modèle. Dans la série d'expériences que nous avons réalisées, nous avons fait varier la taille de la base de données et le nombre maximum de versions pour chaque donnée. La taille de la base de données représente le nombre total des versions.

Paramètre	Signification	Valeur
NbOfOperations	Nombre d'opérations dans une transaction *utilisateur*	[1, 5]
OpLecExecTime	Temps d'exécution d'une opération de lecture	1s
OpEcrExecTime	Temps d'exécution d'une opération d'écriture	2s
$Period_i$	Périodicité d'une transaction de mise à jour	[1000ms, 5000ms]

Tableau 1. *Valeurs des paramètres de simulation.*

4.2. *Descriptif des modules du simulateur*

L'architecture globale du simulateur est illustrée par la figure 3. La conception de ce simulateur est basée sur l'architecture de MVD-FCSA présentée dans la section 3 (cf. Figure 2).

Le simulateur est composé des modules suivants :

1) Le générateur de transactions, composé de deux parties :

- le générateur des transactions *utilisateur*, qui génère les transactions selon une distribution aléatoire.

- le générateur des transactions de mise à jour, qui génère les transactions suivant un processus d'arrivée défini qui respecte la périodicité des transactions.

2) L'ordonnanceur des transactions : il détermine l'ordre dans lequel les transactions fournies par les générateurs doivent s'exécuter (en fonction de leurs priorités).

3) Le contrôleur d'échéances : il vérifie que les transactions vont pouvoir être exécutées avant leur échéance sinon elles sont immédiatement écartées (abandonnées).

4) Le gestionnaire de fraîcheur : il vérifie la fraîcheur des données qui vont être accédées par la transaction. Si les données sont fraîches, alors la transaction peut être exécutée et elle est envoyée au déclencheur de transactions. Sinon, elle est mise en attente dans une file des transactions bloquées. Elle est ensuite réinsérée, dès la mise à jour des données, dans la file des transactions prêtes à être exécutées.

5) Le déclencheur de transactions : il déclenche l'exécution des transactions.

6) Le contrôleur de concurrence : il s'agit du module responsable de la résolution des conflits d'accès aux données en fonction d'un protocole de contrôle de concurrence.

7) Le gestionnaire des données temps réel : il est chargé de créer les versions des données, et de mettre à jour la file des versions. La gestion des données doit être faite, d'une part de façon à garantir la fraîcheur des données et d'autre part de façon à respecter le seuil d'erreur toléré sur la donnée (MDE). Lorsqu'une version d'une donnée ne respecte pas l'un de ces deux paramètres de qualité des données, elle est supprimée de la base de données

8) La base de données : elle est construite par un générateur de données qui initialise la base en fonction des paramètres retenus. La validité (durée de validité) et la valeur (valeur courante de la donnée) sont les paramètres retenus pour la spécification des données.

4.3. *Résultats et commentaires*

Pour évaluer les performances de MVD-FCSA, nous avons réalisé une série d'expériences où le critère de performance considéré est le taux de succès des transactions. Les résultats des simulations sont représentés graphiquement pour montrer la variation du taux d'échéances ratées, en utilisant la technique de MVD-FCSA et en faisant varier les paramètres suivants :

1) Le seuil du nombre de versions.

2) La taille maximale de la base de données.

3) Le nombre de transactions.

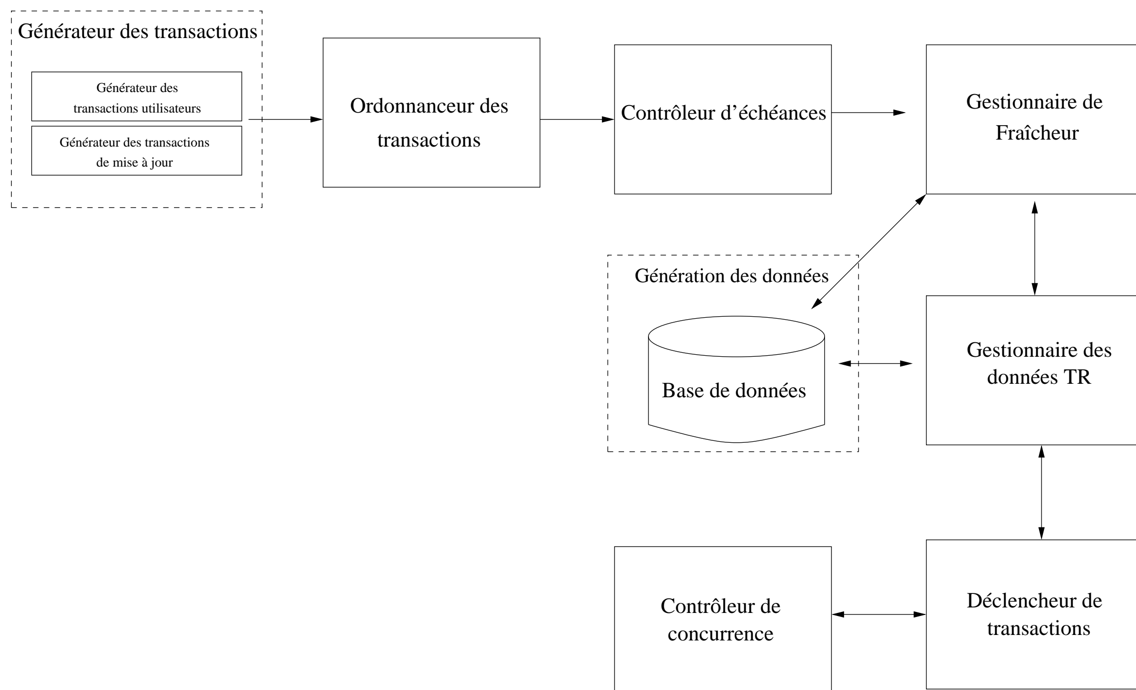
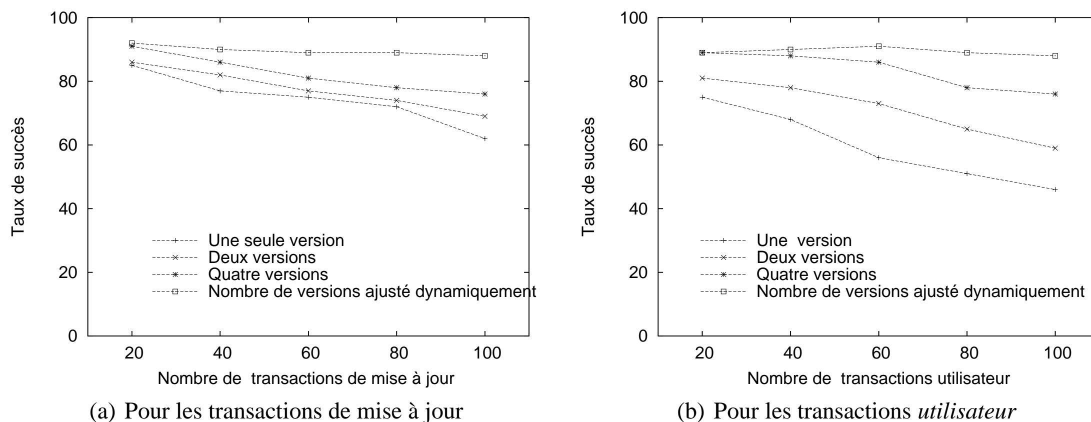


Figure 3. *Architecture globale du simulateur.*

4.3.1. *Expérience 1 : résultats utilisant l'architecture MVD-FCSA*

Comme montré dans la figure 4, nous constatons que le taux de succès observé croît avec l'augmentation du nombre de versions par donnée. Plus précisèment, l'utilisation de l'architecture FCSA utilisant quatre versions par donnée offre les meilleures performances et l'architecture FCSA classique (équivalent à une seule version par donnée), fournit les performances les moins bonnes. Cependant, ce résultat est à atténuer comme le montre l'expérience 3.



(a) Pour les transactions de mise à jour

(b) Pour les transactions *utilisateur*

Figure 4. *Résultats de simulation avec l'architecture MVD-FCSA.*

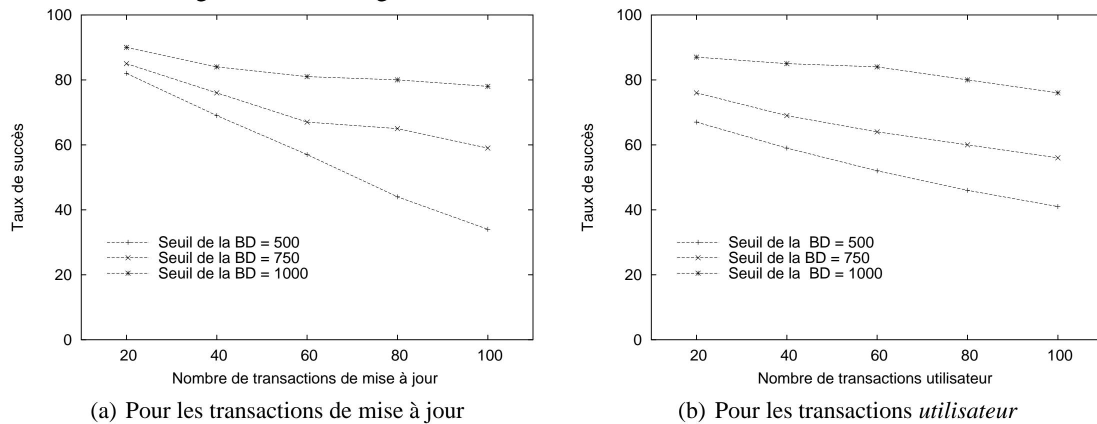
Nous constatons également que pour les deux types de transactions (de mise à jour et *utilisateur*), le taux de succès augmente considérablement en utilisant les données multi-versions par rapport à l'utilisation d'une version unique, comme dans [LU 01]. Par rapport à l'architecture MVD-FCSA avec un nombre fixe

de versions, MVD-FCSA avec un ajustement dynamique du nombre de versions montre un taux de succès relativement plus important (cf. Figure 4).

4.3.2. *Expérience 2 : variation du seuil de la taille de la base de données en utilisant l'approche mixte de MVD-FCSA*

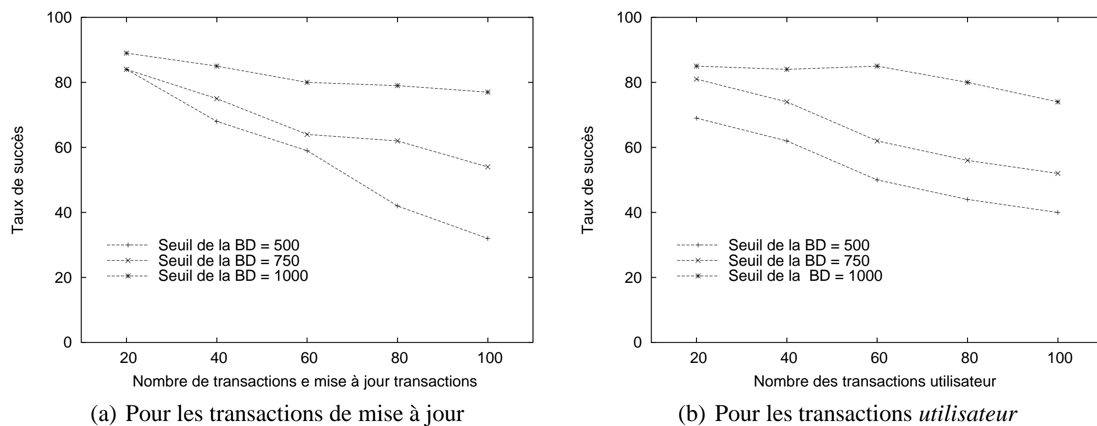
Nous considérons maintenant l'approche mixte (ajustement dynamique du nombre de versions borné par un nombre fixe de versions). Dans la figure 5, nous avons fixé le seuil du nombre de versions à 4 en faisant varier la taille de la base de données (500, 750 et 1000). Dans la figure 6, nous avons aussi fait varier la taille de la base de données en fixant le nombre de versions à 6.

Les figures 5 et 6 montrent l'effet de la variation de la taille de la base de données. Le taux de succès des transactions augmente avec l'augmentation de la taille de la base de données.



(a) Pour les transactions de mise à jour (b) Pour les transactions *utilisateur*

Figure 5. *Résultats des simulations en utilisant l'approche mixte de MVD-FCSA (nombre maximal de versions = 4) et variation du seuil de la taille de la base de données.*

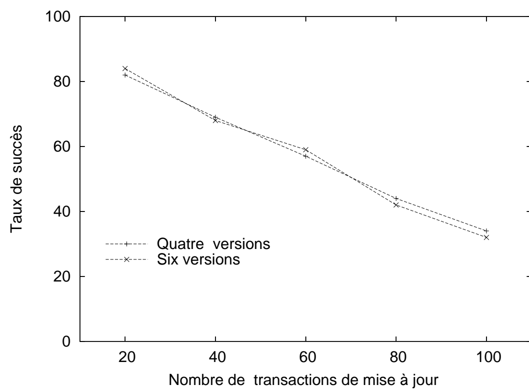


(a) Pour les transactions de mise à jour (b) Pour les transactions *utilisateur*

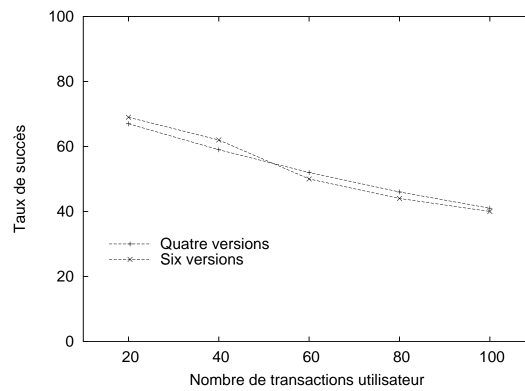
Figure 6. *Résultats des simulations en utilisant l'approche mixte de MVD-FCSA (nombre maximal de versions = 6) et variation de la taille maximale de la base de données.*

4.3.3. *Expérience 3 : variation du seuil du nombre de versions*

Dans cette expérience, nous avons aussi utilisé l'approche mixte de MVD-FCSA. Nous avons fixé la taille de la base de données tout en faisant varier le seuil du nombre de versions. Par rapport à l'utilisation d'un maximum de 6 versions, le taux de succès des transactions en utilisant 4 versions est légèrement plus élevé, comme montré dans les figures 7 et 8. Il ne suffit donc pas d'augmenter le nombre de versions indéfiniment pour améliorer les performances. Cela peut être expliqué par la gestion plus compliquée d'un nombre élevé de versions.

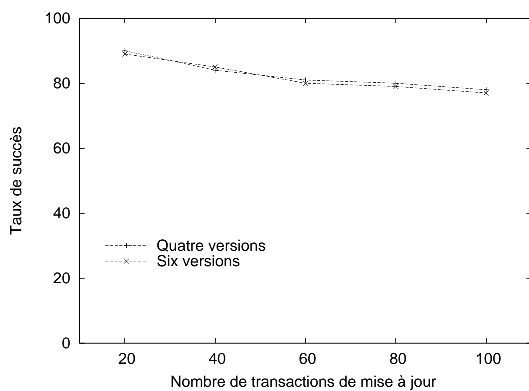


(a) Pour les transactions de mise à jour

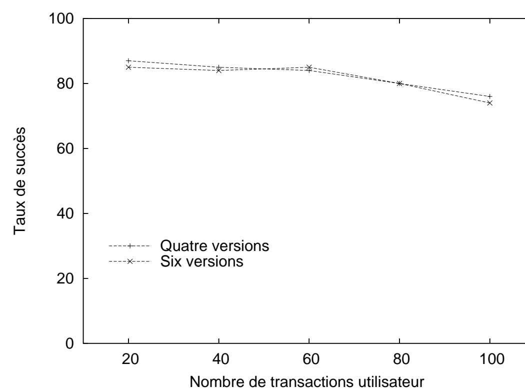


(b) Pour les transactions *utilisateur*

Figure 7. *Résultats des simulations utilisant l'approche mixte de MVD-FCSA une taille maximale de la base de données (max = 500).*



(a) Pour les transactions de mise à jour



(b) Pour les transactions *utilisateur*

Figure 8. *Résultats des simulations utilisant l'approche mixte de MVD-FCSA avec une taille maximale de la base de données (max = 1000).*

4.4. *Bilan de l'utilisation des données multi-versions dans l'architecture FCSA*

La nouvelle approche de l'architecture de contrôle par rétroaction en utilisant la notion de données multi-versions semble une solution prometteuse pour garantir une meilleure QdS des applications. En effet, par rapport à l'architecture classique de contrôle par rétroaction, elle permet :

– de limiter le nombre de conflits entre les transactions sur les données accédées,

– d'augmenter le nombre de transactions qui réussissent avant leur échéance et par conséquent d'obtenir de meilleures performances,

– d'exploiter de la meilleure façon possible la donnée pendant toute sa durée de validité,

– d'améliorer la qualité des données en n'utilisant que des données fraîches et précises (à la valeur du paramètre DE près),

– d'améliorer la qualité de service par la garantie d'une meilleure qualité de données et de transactions.

5. Conclusion et perspectives

Dans cet article, nous avons présenté notre modèle MVD-FCSA. Nous avons considéré trois politiques de gestion de données : (1) en utilisant un nombre fixe de versions, (2) en ajustant dynamiquement le nombre de versions et (3) en utilisant l'approche mixte, pour réduire le nombre de conflits dans le cadre

d'une approche basée sur la gestion de la qualité de service. Notre méthode repose sur l'exploitation et la sauvegarde de plusieurs versions d'une même donnée temps réel. Des simulations ont confirmé l'apport de l'utilisation des données multi-versions par rapport à l'architecture classique de FCSA et ont permis d'établir des comparaisons entre les trois politiques de l'approche MVD-FCSA pour mettre en évidence l'importance du facteur taille de la base de données.

À court terme, dans nos travaux futurs, nous étudierons la possibilité de la gestion des données dérivées et l'utilisation des transactions de mise à jour déclenchées. Une autre voie que nous envisageons d'explorer est l'adaptation de ces travaux à des systèmes multimédia tel que la vidéo à la demande.

6. Bibliographie

[ABB 88] ABBOTT R., GARCIA-MOLINA H., « Scheduling Real-Time Transactions : A Performance Evaluation », *International Journal of Distributed and Parallel Databases*, vol. 1, n° 2, 1988.

[AMI 03a] AMIRIJOO M., HANSSON J., SON S. H., « Algorithms for Managing Real-time Data Services Using Imprecise Computation », *Proceedings of International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, Taiwan, 2003.

[AMI 03b] AMIRIJOO M., HANSSON J., SON S. H., « Error-Driven QoS Management in Imprecise Real-Time Databases », *Proceedings of* 15th *Euromicro Conference on Real-Time Systems (ECRTS)*, Portugal, 2003.

[AMI 03c] AMIRIJOO M., HANSSON J., SON S. H., « Specification and Management of QoS in Imprecise Real-Time Databases », *Proceedings of International Database Engineering and Applications Symposium (IDEAS)*, Hong Kong, 2003.

[BOU 05a] BOUAZIZI E., DUVALLET C., SADEG B., « Using Feedback Control Scheduling and Data Versions to enhance Quality of Data in RTDBSs », *IEEE International Computer System and Information Technology (IEEE ICSIT'2005)*, Alger, Algerie, 2005.

[BOU 05b] BOUAZIZI E., SADEG B., DUVALLET C., « Ordonnancement contrôlé par rétroaction dans les SGBD temps réel », 13th *conference on Real-Time Systems (RTS)*, Paris, France, 2005.

[BUT 97] BUTTAZO G., *Hard Real-Time Computing Systems*, Kluwer Academic Publishers, 1997.

[DUV 99] DUVALLET C., MAMMERI Z., SADEG B., « Les SGBD Temps Réel », *Technique et Science Informatiques*, vol. 18, n° 5, 1999, p. 479-517.

[KAN 02a] KANG K., SON S., STANKOVIC J., « Service Differentiation in Real-Time Main Memory Databases », 5th *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'02)*, Washington D.C., April 29 - May 01 2002.

[KAN 02b] KANG K., SON S., STANKOVICH J., ABDELZAHER T., « A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases », *Proceedings of* 14th *Euromicro Conference on Real-Time Systems (ECRTS)*, June 19-21 2002.

[LIU 73] LIU C., LEYLAND J., « Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment », *Journal of the ACM*, vol. 20, n° 1, 1973, p. 46–61.

[LIU 91] LIU J., LIN K., SHIN W., YU A.-S., « Algorithms for scheduling imprecise computations », *IEEE Computer*, vol. 24, n° 5, 1991.

[LU 01] LU C., « Feedback Control Real-Time Scheduling », PhD thesis, University of Virginia, May 2001.

[RAM 93] RAMAMRITHAM K., « Real-Time Databases », *Journal of Distributed and Parallel Databases*, vol. 1, n° 2, 1993, p. 199–226.

[RAM 04] RAMAMRITHAM K., SON S., DIPIPPO L., « Real-Time Databases and Data Services », *Real-Time Systems*, vol. 28, 2004, p. 179-215, Kluwer Academic Publisher.