Ordonnancement contrôlé par rétroaction dans les SGBD temps réel

Emna Bouazizi, Bruno Sadeg et Claude Duvallet
LIH, UFR des Sciences et Techniques,
25 rue Philippe Lebon, BP 540
76 058 Le Havre Cedex, FRANCE.
{emna.bouazizi,bruno.sadeg,claude.duvallet}@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. La quantité de données temps réel acquises par des capteurs rend nécessaire l'utilisation de SGBDTR[1] *qui permettent de prendre en compte les contraintes temps réel des applications. Ces applications doivent aussi faire face à des charges d'utilisation imprévisibles. L'objet de nos travaux est de maintenir le comportement des SGBDTR dans un état stable et de diminuer le nombre de transactions qui ratent leur échéance (critère de correction généralement accepté dans les SGBDTR). De nombreuses transactions risquent de rater leur échéance lorsqu'elles doivent être redémarrées suite à un conflit d'accès aux données. Dans cet article, nous proposons de réduire, voire de supprimer les conflits d'accès aux données temps réel, en créant plusieurs versions de ces données lorsqu'un conflit de type écriture/lecture ou lecture/écriture survient. Nous utilisons également une boucle de rétroaction pour contrôler l'admission des transactions dans le système. Ce travail est mis en œuvre au moyen de simulations qui ont confirmé l'apport de l'utilisation des données multi-versions et du contrôle par rétroaction.*

Plan

Mots Clés

SGBD temps réel, contrôle de concurrence, données multi-versions, ordonnancement contrôlé par rétroaction.

[1] Système de Gestion de Bases de Données Temps Réel.

1 INTRODUCTION

De nombreuses applications temps réel nécessitent d'utiliser des quantités importantes de données temps réel. Il s'agit par exemple des applications utilisées pour le contrôle des usines chimiques, des centrales nucléaires, des applications de commerce électronique, etc. 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).
L'exploitation de ces données temps réel dans des applications est mieux prise en charge par les SGBD temps réel [Duvallet , 1999], [Ramamritham, 1993]. Ce type d'application doit très souvent faire face à des charges d'utilisation imprévisibles qui causent la surcharge du système. Durant ces périodes, les transactions temps réel chargées d'exploiter les données de l'application peuvent manquer leur échéance ou être amenées à utiliser des données obsolètes (non fraîches). C'est pourquoi plusieurs travaux utilisant une boucle de rétroaction pour l'adaptation des paramètres de qualité de service ont été menés [Kang , 2002], [Lu , 2002], [Amirijoo , 2003]. Dans ces travaux, l'objectif est de faire varier dynamiquement les paramètres de la qualité de service afin d'arriver à une stabilisation du comportement du SGBD temps réel et d'éviter ainsi la surcharge du système ou sa sous-utilisation (gaspillage des ressources).
Dans les SGBDTR, le non-respect des échéances des transactions est souvent dû à des conflits d'accès aux données qui entraînent le redémarrage des transactions les moins prioritaires [Ramamritham, 1993], [Ramamritham , 2004]. Dans ce travail, l'objectif est de réduire de façon substantielle le nombre de conflits entre les transactions pour l'accès aux données temps réel. Notre approche consiste à créer plusieurs versions des données temps réel pour éviter les conflits d'accès en mode lecture/écriture ou écriture/lecture.
Dans la section 2, nous présentons les travaux relatifs au modèle de SGBDTR basé sur le principe de rétroaction. Dans la section 3, nous donnons un bref aperçu de la problématique que nous avons traitée avant de présenter notre contribution. Dans la section 4, nous présentons l'architecture que nous proposons et qui est basée sur des données multi-versions. Dans la section 5, nous décrivons le fonctionnement de chaque élément de l'architecture. La section 6 est consacrée à la présentation de simulations et aux résultats obtenus. Enfin, nous concluons sur l'apport de notre travail et les perspectives que nous souhaitons lui donner.

2 MODÈLE

2.1 Présentation

Dans une application reposant sur l'utilisation de SGBDTR, des transactions en provenance des 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 un grand nombre de transactions temps réel vont alors manquer leur échéance. Des travaux basés sur une approche « Qualité de Service » (QdS) tentent de rendre les SGBDTR plus robustes face aux périodes d'instabilité (périodes de sous-utilisation et

périodes de surcharge) [Kang , 2002], [Ramamritham , 2004] [Amirijoo , 2003]. Ces travaux s'appuient sur des techniques de contrôle avec rétroaction[2] [Lu, 2001] et autorisent la manipulation de résultats imprécis [Liu , 1994]. Dans la suite de cette section, nous allons détailler ces travaux sur lesquels s'appuie notre proposition. Nous allons présenter plus particulièrement le modèle de SGBDTR que nous considérons.

2.2 Les données temps réel

Les données temps réel représentent la capture de l'état du monde réel. Par exemple, dans une centrale nucléaire, on peut trouver des capteurs de température qui sont utilisés pour contrôler le système et détecter les éventuelles anomalies (surchauffe du système ou autre). Dans les systèmes boursiers, des transactions mettent à jour les prix des actions à un instant donné. Ces données doivent être remises à jour régulièrement afin de refléter au plus près l'état du monde réel. Elles possèdent donc une durée de validité qui représente la période pendant laquelle elles peuvent être utilisées [Ramamritham , 2004].
Amirijoo et al. ont introduit la notion de Qualité des Données (QdD) [Amirijoo , 2003] 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[3]) et on le calcule en faisant la différence entre la donnée en base et la valeur du monde réel. Par exemple, on peut admettre que la donnée température lorsqu'elle vaut « 100°2 » en base est très proche de « 100°3 » qui est la donnée réelle et que par conséquent, il n'est pas nécessaire de mettre à jour cette donnée. Cet écart possède un seuil (MDE[4]) qui permet de déterminer si la transaction qui souhaite mettre à jour une donnée temps réel peut être écartée ($DE \geq MDE$) ou pas (DE < MDE). Ce travail est effectué par le contrôleur de précision (cf. figure 1).

2.3 Les transactions temps réel

Les transactions temps réel considérées dans nos travaux possèdent des échéances de type firm (échéances strictes non critiques [Duvallet , 1999]). Par conséquent, 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. La non-prévisibilité de leurs arrivées dans le système et de la charge qu'elles suscitent, rend adéquate l'utilisation d'une architecture basée sur le retour d'expérience (ou rétroaction) pour gérer les variations importantes de charge [Lu, 2001].

Dans cet article, nous considérons qu'une transaction de mise à jour crée une nouvelle version d'une donnée lorsqu'elle souhaite écrire dans la base de données. Les

[2] Feedback Control Scheduling (FCS).
[3] Data Error.
[4] Maximum Data Error.

transactions « utilisateur » accèdent alors à la donnée la plus récente qui n'a pas été verrouillée en écriture.

2.4 Architecture globale

Nous allons présenter en détail dans cette section les parties du modèle qui nous intéressent plus particulièrement. En reprenant la figure 1, nous considérons le modèle général et donnons une brève description de chacun des composants de ce modèle.
Le contrôleur d'admission a pour tâche de contrôler les transactions « utilisateur » qui sont acceptées 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[5]. Son fonctionnement est contrôlé par la boucle de rétroaction qui lui fournit ses paramètres d'ajustement.
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 leur exécution. 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 (non fraîches) 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 apparaissent entre les transactions. Dans la plupart des travaux [Kang , 2002], [Amirijoo , 2003], il s'agit de 2PL-HP[6] [Abbott , 1988] où la transaction de plus haute priorité est privilégiée.
- Un ordonnanceur de base : il s'agit bien souvent de Earliest Deadline First (EDF) [Liu , 1973] qui ordonnance les transactions en considérant que la transaction qui possède l'échéance la plus proche doit être exécutée en priorité.

Dans cette architecture, on peut modifier la politique de contrôle de concurrence ou modifier la politique d'ordonnancement en remplaçant simplement le module correspondant.
Un moniteur permet de mesurer les performances du système en inspectant l'exécution des transactions (nombre de transactions terminées, abandonnées, ou qui ont ratées leur échéance). Les valeurs ainsi mesurées permettent d'alimenter le contrôleur de QdS et font partie de la boucle de contrôle par rétroaction qui va contribuer à stabiliser le système.
Le contrôleur d'utilisation et d'échéances ratées (ou contrôleur de QdS) permet de réajuster les paramètres de QdS en fonction des valeurs déterminées par le moniteur et des paramètres de référence. Les valeurs ainsi obtenues sont transmises au contrôleur d'admission et au gestionnaire de qualité des données.
Le gestionnaire de qualité des données permet de réajuster la valeur du paramètre MDE qui constitue le paramètre de QdD. La valeur de MDE est calculée en fonction de l'utilisation du système. Le paramètre MDE 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).

[5] Administrateur de la base de données.
[6] Two Phase Locking-Hight Priority.

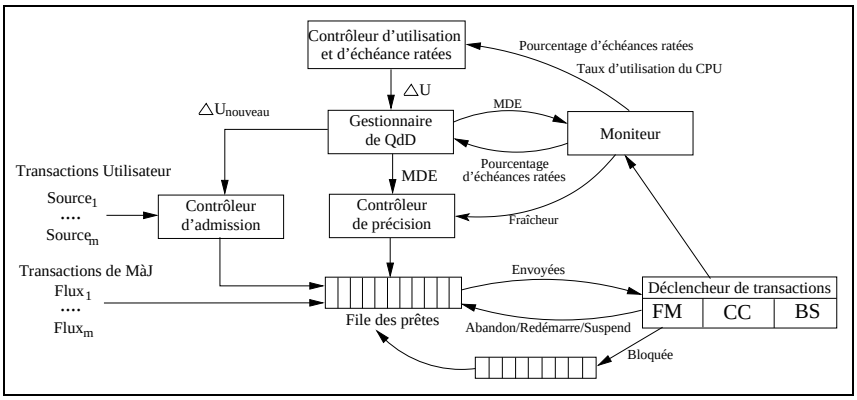

FIG. 1 – Architecture de SGBDTR basée sur la rétroaction.

2.5 Le 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 (taux d'utilisation et taux de transactions réussies) 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 les transactions acceptées dans le système. Cette adaptation va provoquer une modification du comportement du système et donc des valeurs observées par le moniteur. Le fonctionnement de cette boucle doit tendre vers une stabilité du système autour d'une valeur de référence, fixée par le DBA (par exemple, il peut s'agir d'un taux d'utilisation de 80%). Il s'agit donc de réduire l'oscillation du système autour de cette valeur de référence. En effet, si le nombre des demandes des utilisateurs observé par le moniteur dépasse la valeur de référence fixée par le DBA (80%), le système, via la boucle de rétroaction, diminue la valeur observée afin de l'adapter avec la valeur de référence.

3 PROBLÉMATIQUE

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 un mode incompatible[7], il se produit un conflit d'accès. Pour résoudre ce conflit, on peut selon les cas mettre en attente ou abandonner des transactions.

[7] Lecture/Écriture, Écriture/Lecture, Écriture/Écriture.

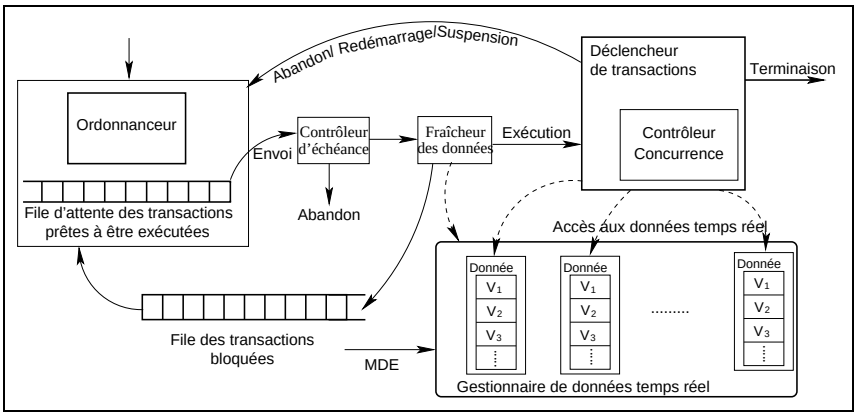

FIG. 2 – Gestion de données multi-versions dans une architecture d'ordonnancement temps réel par rétroaction.

Dans le cadre des applications temps réel, les données sensorielles (données temps réel acquises par des capteurs) sont mises à jour périodiquement par des transactions dédiées appelées « de mise à jour » et peuvent être utilisées par des transactions « utilisateur ».
Lorsqu'une transaction souhaite modifier une donnée déjà accédée en lecture par une autre transaction, on doit résoudre le problème de conflit d'accès au moyen d'un protocole de contrôle de concurrence. Il en est de même si une transaction essaie de lire une donnée verrouillée en écriture par une autre. Considérons alors ces deux cas :

- Dans le premier cas, on compare les priorités des deux transactions et on abandonne la transaction de plus faible priorité avant de la redémarrer. Il s'agit du protocole 2PL-HP [Abbott , 1988].
- Dans le second cas, on peut :
  - soit procéder de la même façon,
  - soit suspendre la transaction qui cherche à lire la donnée.

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 temps réel et réduire ainsi le nombre de transactions qui ratent leur échéance.

4 LE MODÈLE DES DONNÉES MULTI-VERSIONS

Nos travaux concernent l'amélioration de l'architecture de contrôle des ordonnancements temps réel avec rétroaction dans une approche basée sur la spécification de qualité de service dans les SGBDTR [Kang , 2002], [Amirijoo , 2003].
Pour la gestion de la QdS, comme montré dans la figure 1 et décrit dans la section 2, un moniteur, un contrôleur d'admission, un contrôleur d'utilisation et d'échéances ratées, un déclencheur de transactions, et un gestionnaire de qualité de données sont utilisés

pour ajuster les performances du système et contrôler la circulation des informations. Dans notre système, on étend l'architecture du contrôle d'ordonnancement par rétroaction en utilisant la notion de données multi-versions pour garantir la qualité des données (fraîcheur et précision), et pour réduire le nombre de conflits d'accès. Le taux de manquement des échéances pour les transactions est ainsi minimisé.
Dans notre architecture (cf. figure 2), nous avons modifié la partie concernant la gestion des transactions (file d'attente des transactions prêtes à être exécutées et déclencheur de transactions) et nous avons conservé la partie concernant la circulation des informations de contrôle (taux d'échéances ratées, taux d'utilisation, etc.). Notre architecture se compose d'une file d'attente, d'un ordonnanceur, d'un gestionnaire de données temps réel multi-versions, d'un vérificateur d'échéances, d'un gestionnaire de fraîcheur des données et d'un déclencheur de transactions.
Dans la file des transactions prêtes à être exécutées, les transactions sont ordonnancées suivant la proximité de leur échéance selon l'algorithme EDF [Liu , 1973]. Ensuite, ces transactions vont transiter par deux composants différents avant d'être exécutées :

- Un vérificateur d'échéances qui contrôle que les transactions vont pouvoir être exécutées avant leur échéance sinon elles sont immédiatement écartées (abandonnées).
- Un gestionnaire de fraîcheur qui 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 FONCTIONNEMENT

5.1 Vérificateur d'échéances

Il prend en compte la durée minimale d'exécution restante à laquelle il ajoute 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 être terminée et validée 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.

5.2 Gestionnaire de fraîcheur

D'une part, le gestionnaire de fraîcheur vérifie que les données sensorielles 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, il existe dans le système plusieurs versions pour chaque donnée sensorielle. La vérification est donc effectuée par rapport à la valeur la plus fraîche qui n'est pas verrouillée en écriture par une autre transaction.

5.3 Gestion des 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.

5.4 Accès concurrent aux données temps réel

Dans cet article, nous ne considérerons 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 dont la mise à jour n'a pas encore été validée. Dans le protocole 2PL-HP [Abbott , 1988], 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 d'une des deux transactions est allongée et donc les risques qu'elle rate son échéance sont augmentés. Pour tenter de diminuer ce risque, nous allons considérer que lors d'un tel conflit, nous allons ajouter une version supplémentaire de la donnée. Ainsi la transaction «de lecture» peut continuer de considérer une ancienne version de la donnée pendant que la transaction de mise à jour écrit une nouvelle valeur.
Il est cependant nécessaire 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 :

- On supprime 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.
- On garde 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 (fixé initialement par le concepteur) d'une même donnée est atteint et qu'une transaction veut écrire une nouvelle version de cette donnée, on met alors 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'une donnée peut-être accédée en lecture par plusieurs transactions. La transaction, qui entre en conflit sur la donnée, va donc devoir considérer chacun des groupes de transactions accédant aux différentes versions de la donnée. On applique alors le pro-

tocole 2PL-HP « adapté ». Nous présentons ici notre adaptation de ce protocole (cf. algorithme 1).

Algorithme 1 : Protocole 2PL-HP adapté pour les données multi-versions

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.
$Gr(Tr_{user})$: Ensemble des groupes de transactions utilisateur accédant aux différentes versions.
nb_groupe : nombre de groupes de transactions accédant chacun à une version de la donnée.

```
début
    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
```

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

5.5 Utilisation des données multi-versions

La nouvelle version de l'architecture de contrôle par rétroaction en utilisant la notion des données multi-versions est une solution pour garantir une meilleure QdS. 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 des transactions qui réussissent avant leur échéance et par conséquent obtenir une meilleure qualité de transactions,
- une meilleure exploitation de la donnée pendant toute sa durée de validité,
- d'augmenter la qualité des données en n'utilisant que des données fraîches et précises (à la valeur du paramètre DE près),
- d'augmenter la qualité de service par la garantie d'une meilleure qualité de données et de transactions.

6 SIMULATIONS ET RÉSULTATS

6.1 Principe

Dans notre modèle, le principe de base est le suivant : chaque transaction quelque soit son type doit passer par une suite de tests depuis sa création jusqu'à son exécution. Il s'agit de commencer par implémenter deux générateurs de transactions. Vu

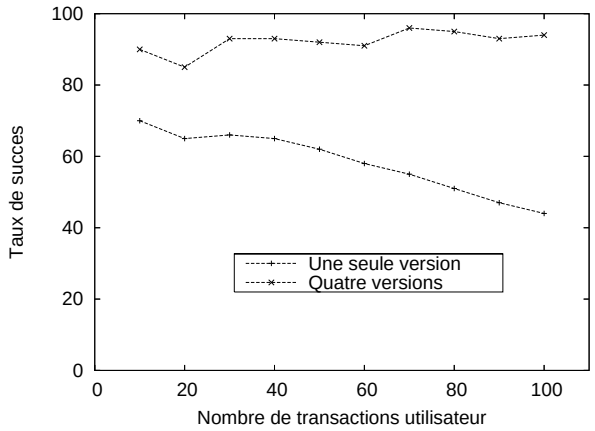

FIG. 3 – Influence du nombre de versions sur le taux de succès des transactions « utilisateur »

que l'arrivée des transactions « utilisateur » dans le système est imprévisible, le générateur des transactions « utilisateur » génère aléaroirement la distribution de ces transactions. L'arrivée des transactions de mise à jour étant périodique, le générateur des transactions de mise à jour génère ses transactions suivant un processus d'arrivée défini, qui respecte la périodicité des transactions. La composante qui reçoit les transactions distribuées par les deux générateurs est l'ordonnanceur qui va ordonnancer les transactions reçues dans la file des transactions prêtes à être exécutées en fonction de leurs priorités. Dés sa sortie de la file, la transaction va subir la suite des tests effectués par les autres éléments de l'architecture (le contrôleur d'échéance, le gestionnaire de fraîcheur et le contrôle de conccurence si nécessaire).
Pour faciliter le travail dans ce contexte objet, le langage de programmation choisi est Java (sous l'environnement Eclipse).

6.2 Résultats

Une exécution de notre simulateur a permis de confirmer l'apport de l'utilisation des données multi-versions. Les résultats de la simulation sont représentés graphiquement par deux figures. La figure 3 représente les résultats de l'exécution des transactions « utilisateur » et la figure 4 représente les résultats de l'exécution des transactions de mises à jour. Pour les deux types des transactions, le taux des transactions qui réussissent augmente considérablement en utilisant les données multi-versions par rapport à l'utilisation d'une version unique, comme dans [Lu, 2001].

7 CONCLUSION ET PERSPECTIVES

Dans cet article, nous avons présenté une nouvelle méthode pour réduire le nombre de conflits en lecture/écriture des transactions temps réel dans le cadre d'une approche

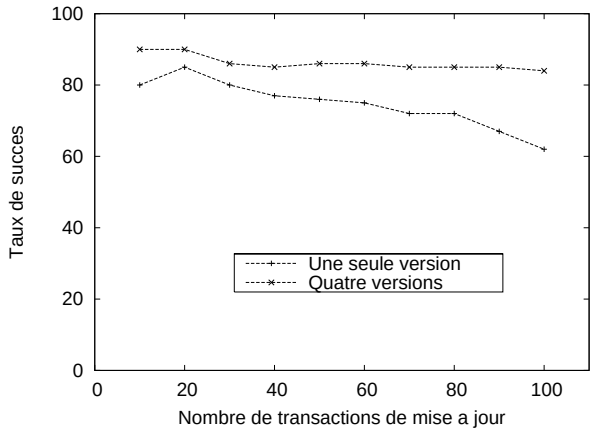

FIG. 4 – Influence du nombre de versions sur le taux de succès des transactions de mise à jour

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. Cette approche permet de maximiser le nombre de transactions qui se terminent avant leurs échéances. Cette approche est mise en œuvre au moyen de simulations qui ont confirmé l'apport de l'utilisation des données multi-versions.
Actuellement, le nombre maximum de versions pour une donnée est fixé par l'administrateur de la base de données et il est le même pour toutes les données et donc la taille maximale de la base est connue à l'avance. Á court terme, dans nos travaux futurs, nous étudierons la possibilité d'ajuster dynamiquement pour chaque donnée le nombre de versions. Cela permet de réduire d'avantage le nombre de conflits d'accès en mode lecture/écriture et de prendre en compte l'impact de la taille de la base de données. Le nombre de versions des données est ainsi limité et il est fonction du seuil global sur la base de données.
À moyen terme, un autre aspect à prendre en compte concerne la gestion des données dérivées qui n'a pas été étudié dans cet article. Une autre voie que nous explorerons est l'adaptation de ces travaux à des systèmes multimédia tels que les serveurs de Vidéo à la Demande (VoD) qui doivent faire face à des phases de surcharge lorsqu'un grand nombre d'utilisateurs se connectent pour demander la visualisation de films.

8 REMERCIEMENTS

Les auteurs tiennent à remercier le Ministère Français de la recherche pour sa contribution financière à la réalisation de cet article (ACI-JC 1055).

BIBLIOGRAPHIE

[Abbott , 1988] Abbott, R. and Garcia-Molina, H. (1988). Scheduling Real-Time Transactions : A Performance Evaluation. In Proceedings of the 14^{th} Very Large Data Bases Conference, pages 1–12.

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

[Duvallet , 1999] Duvallet, C., Mammeri, Z., and Sadeg, B. (1999). Les SGBD temps réel. Technique et Science Informatiques, 18(5) :479–517.

[Kang , 2002] Kang, K., Son, S., Stankovic, J., and Abdelzaher, T. (2002). A QoS-Sensitive Approach For Timeliness and Freshness Guarantees in Real-Time Databases. In Proceedings of the Euromicro Conference on Real-Time System.

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

[Liu , 1994] Liu, J., Shih, W.-K., and K.-J. Lin, R. Bettati, J.-Y. C. (1994). Imprecise Computations. In Proceedings of the IEEE, volume 82, pages 83–94.

[Lu, 2001] Lu, C. (2001). Feedback Control Real-Time Scheduling. PhD thesis, University of Virginia.

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

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

[Ramamritham , 2004] Ramamritham, K., Son, S., , and Dipipo, L. C. (2004). Real-Time Databases and Data Services. Real-Time Systems, 28 :179–215.