Utilisation des contraintes (m,k)-firm dans les SGBDTR manipulant des transactions périodiques de mise à jour.

Jérôme Haubert, Emna Bouazizi et Claude Duvallet
LIH, UFR des Sciences et Techniques,
25 rue Philippe Lebon, BP 540
76 058 Le Havre Cedex, FRANCE.
{jerome.haubert,emna.bouazizi,claude.duvallet}@univ-lehavre.fr

Résumé : Dans les systèmes temps réel manipulant des tâches périodiques, des travaux introduisant la notion de contraintes (m,k)-firm ont été effectués. Ces contraintes permettent de relaxer les contraintes temps réel des tâches en autorisant que certaines invocations d'une tâche soient perdues. Plus précisément, une tâche sous contraintes (m,k)-firm doit garantir que *m* invocations parmi *k* respectent leurs échéances. D'un autre côté, dans les SGBD, notamment temps réel, certaines transactions sont périodiques. Elles permettent, par exemple, de rafraîchir des données temporelles de la base à partir de capteurs. Ces transactions sont appelées "transactions de mise à jour" (*update transactions*) par opposition aux "transactions utilisateur" (*user transactions*) qui manipulent des données temporelles ou non et/ou des données dérivées. Dans ce papier, nous proposons une adaptation des contraintes (m,k)-firm aux transactions de mise à jour dans les SGBDTR. En se basant sur l'exploitation d'une architecture avec contrôle par rétroaction et sur les contraintes temps réel (m,k)-firm, nous proposons d'augmenter cette architecture afin de filtrer l'arrivée des transactions de mise à jour en tenant compte des variations de la charge du système.
Mots-clés : Systèmes d'information, ordonnancement, transactions de mise à jour, contraintes (m,k)-firm, SGBD temps réel.

1 INTRODUCTION

Les systèmes temps réel distinguent généralement les tâches temps réel souples et dures. Des auteurs ont proposé de relaxer les contraintes temps réel des tâches en distinguant, pour chaque tâche, une partie obligatoire et une partie optionnelle [Hamdaoui, 1995, Wang, 2002]. Ceci augmente la capacité d'une tâche à respecter ses contraintes temps réel puisque seule la partie obligatoire doit réellement les respecter. La notion de contraintes temps réel (m,k)-firm a notamment été introduite pour les tâches périodiques [Hamdaoui, 1995]. Cette approche se fonde sur la possibilité de perdre certaines invocations d'une tâche tout en conservant une certaine qualité de service globale. Par exemple, une interpolation des résultats peut permettre de récupérer les invocations perdues.
Pour gérer efficacement les tâches temps réel sous contraintes (m,k)-firm, de nouveaux algorithmes d'ordonnancement ont été proposés [Hamdaoui, 1995, Ramanathan, 1999, Wang, 2002]. Ces algorithmes sont basés sur des méthodes dynamiques ou statiques permettant de déterminer (en ligne ou hors ligne) les invocations des tâches qui peuvent être considérées comme optionnelles [Wang, 2002]. Ensuite, en fonction de la charge du système, on peut décider de ne pas exécuter les invocations optionnelles. De même, en cas de conflit (d'accès à un serveur par exemple), les invocations obligatoires seront plus prioritaires que les invocations optionnelles. Nous présentons plus en détails la gestion des tâches (m,k)-firm dans la section 2.
Les applications temps réel qui manipulent de grandes quantités de données, sont généralement gérer à l'aide des SGBD Temps Réel (SGBDTR). Les transactions qui s'exécutent dans un tel système sont soumises à des échéances [Ramamritham, 1993, Duvallet, 1999]. Une transaction temps réel est considérée comme correcte si elle se termine correctement avant son échéance.
Dans la littérature, il a été proposé des modèles de SGBDTR qui manipulent, d'une part, des transactions de mise à jour (périodiques) pour rafraîchir les données temporelles et, d'autre part, des transactions utilisateur qui manipulent généralement des données temps réel ou non et/ou des données dérivées [Amirijoo, 2003]. L'architecture proposée est composée de différents modules permettant de gérer efficacement l'exécution à la fois des transactions de mise à jour et des transactions utilisateur. Nous présentons plus précisément cette architecture dans le section 3.
Dans un grand nombre d'applications temps réel basées sur un SGBDTR, il n'est pas possible de prévoir les arrivées des transactions utilisateur et des phases d'instabilité peuvent se produire. Pour gérer ces phases d'instabilité du système (phase de surcharge ou phase de sous-utilisation), une méthode basée sur la rétroaction a été présentée [Lu, 2001, Amirijoo, 2003]. Elle permet d'influencer le contrôleur d'admission pour écarter des transactions utilisateur lorsque la charge du système augmente ou de laisser plus de transactions entrer dans le système en cas de sous-utilisation [Lu, 2001]. D'autre part, en fonction de la fraîcheur des données, certaines transactions de mise à jour peuvent également être omises (cf. section 3.2).
Dans cet article, nous nous intéressons aux transac-

tions de mise à jour et nous étudions l'application des contraintes (m,k)-firm à ces transactions. Nous étudions la façon dont la charge du système (en combinaison avec la fraîcheur des données) peut être utilisée pour influencer l'admission des transactions de mise à jour. Pour cela, nous rappelons tout d'abord l'utilisation des contraintes (m,k)-firm pour les tâches temps réel (cf. section 2). Ensuite, nous présentons le modèle de SGBDTR que nous utilisons et nous rappelons le principe de la rétroaction dans un tel système (cf. section 3). Enfin, basé sur ces deux notions, nous présentons notre contribution pour gérer efficacement les transactions de mise à jour (cf. section 4). Enfin, nous concluons cet article sur l'originalité de notre approche et quelques perspectives à notre travail.

2 CONTRAINTES (M,K)-FIRM DANS LES SYSTÈMES TEMPS RÉEL

La récurrence des tâches périodiques dans les systèmes temps réel permet d'ignorer certaines invocations (ou jobs) en utilisant les contraintes (m,k)-firm. Ces contraintes spécifient qu'au moins m jobs dans une fenêtre de k invocations consécutives ($0 \leq m \leq k$) doivent respecter leur échéance [Hamdaoui, 1995]. Autrement dit, parmi k invocations, m sont obligatoires et $k-m$ sont optionnelles. On peut remarquer que les tâches *firm* traditionnelles, c'est à dire strictes non critiques, sont un cas particuliers des tâches (m,k)-firm où $m = k$ ($\neq 0$). En effet, dans ce cas, toutes les invocations d'une même tâche doivent respecter leur échéance.

Bernat a montré par le biais d'un exemple, pourquoi il est préférable d'utiliser deux paramètres pour définir ce type de contraintes [Bernat, 1998]. En effet, il explique pourquoi le ratio de succès ne donne pas suffisamment d'informations sur les exécutions des tâches : un ratio de succès de 90% est équivalent à une instance perdue parmi 10 ou à 100 parmi 1000. Si les jobs qui manquent leur échéance sont consécutifs, alors le deuxième cas peut amener à des résultats très imprécis, voire erronés. On utilise alors deux paramètres : le premier indique le nombre d'instances qui doivent respecter leur échéance et le second limite la fréquence à laquelle les invocations peuvent échouer. Dans ce cas, une tâche ayant des contraintes (9,10)-firm est plus critique qu'une tâche sous contraintes (900,1000)-firm.

Une tâche sous contrainte (m,k)-firm peut se trouver dans deux états différents : un "état normal" et un "état d'échec dynamique". Dès que $(k-m+1)$ invocations d'une tâche ne peuvent respecter leur échéance, la tâche passe dans un état d'échec dynamique. Le but d'un système sous contraintes (m,k)-firm est donc de limiter le nombre de tâches qui passent dans un état d'échec dynamique.

De plus, il a été montré que la notion de contraintes (m,k)-firm est appropriée pour la gestion (la spécification) de la Qualité de Service (QdS) d'une application temps réel [Wang, 2002]. En effet, un système soumis à des contraintes (m,k)-firm fournit différents niveaux de QdS correspondant aux différentes valeurs comprises entre m et k ($0 \leq m \leq k$). En d'autres termes, la QdS augmente en fontion du nombre d'invocations optionnelles qui s'exécutent. D'autre part, il est clair qu'un système sous contraintes (m,k)-firm nécessite moins de ressources qu'un système classique puisque certaines invocations peuvent être écartées.

Pour gérer efficacement l'exécution des tâches (m,k)-firm, de nouveaux algorithmes d'ordonnancement ont été proposés [Hamdaoui, 1995, Ramanathan, 1999, Wang, 2002]. Ils se décomposent en deux grandes familles :

- Les algorithmes dynamiques : la priorité de chaque job est déterminée en fonction de l'état du système. Par exemple, le protocole DBP (*Distance Based Priority*) calcule la distance pour une tâche de tomber dans un état d'échec à partir de l'historique de la tâche [Hamdaoui, 1995]. La distance correspond au nombre de tâches qui peuvent être écartées tout en respectant les contraintes (m,k)-firm.
- Les algorithmes statiques : la priorité est déterminée hors ligne en utilisant un paramètre fixe, par exemple le ratio de succès m/k. Un exemple de protocole statique est ERM (*Enhanced Rate Monotonic*) qui détermine au préalable les invocations obligatoires et les invocations optionnelles [Ramanathan, 1999].

En bref, les algorithmes statiques fournissent une vision déterministe du système alors que les algorithmes dynamiques fournissent plutôt une vision probabiliste. En contrepartie, les algorithmes dynamiques tiennent compte des éventuels changements du système. Par ailleurs dans nos travaux, nous avons déjà démontré l'apport d'une approche basée sur l'utilisation de contraintes (m,k)-firm pour les transactions utilisateur dans les SGBDTR [Haubert, 2004].

3 MODÈLE DE SGBDTR BASÉ SUR LA RÉTROACTION

Dans une application reposant sur l'utilisation de SGBD temps réel, des transactions en provenance des utilisateurs arrivent à des fréquences variables. Lorsque la fréquence augmente de façon considérable, l'équilibre du SGBD temps réel est mis en péril. Durant ces périodes de surcharge, le SGBDTR va potentiellement disposer de ressources moins importantes et les transactions temps réel vont alors manquer leur échéance en plus grand nombre. Des travaux basés sur une approche Qualité de Service (QdS) [Kang, 2002, Amirijoo, 2003] tentent de rendre les SGBDTR plus robustes face aux périodes d'instabilité (périodes de sous-utilisation et périodes de surcharge). Ces travaux s'appuient sur des techniques de contrôle avec rétroaction[1] [Lu, 2001] et autorisent la manipulation de résultats imprécis [Liu, 1994]. Dans cette section, nous allons détailler ces travaux sur lesquels s'appuie notre proposition et plus particulièrement nous allons présenter le modèle de SGBDTR que nous considérons.

[1] Feedback Control Scheduling (FCS)

3.1 Les transactions temps réel

Les transactions temps réel utilisées possèdent des échéances de type *firm* (à échéances strictes non critique [Duvallet, 1999]). Par conséquent lorsqu'elles dépassent leur échéance, elles deviennent inutiles pour le système et sont abandonnées. Dans notre cas, 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 acquises auprès de capteurs. Ces transactions sont exécutées périodiquement pour rafraîchir la valeur des données temps réel.
- 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 allons surtout considérer les transactions de mise à jour pour contribuer à la stabilisation de la charge du système lorsque cela est nécessaire (phase de surcharge ou phase de sousutilisation).

3.2 Les données temps réel et la qualité des données

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). Ces données doivent être remises à jour régulièrement afin de refléter au plus près le monde réel. Elles possèdent donc une durée de validité qui représente la période pendant laquelle elles peuvent être utilisées.

Amirijoo et al. ont introduit une notion de Qualité des Données (QdD) [Amirijoo, 2003] qui permet de considérer qu'une donnée stockée dans la base peut posséder une certaine déviation par rapport à sa valeur dans le monde réel. On appelle cette déviation "erreur sur la donnée" (notée DE[2]) et on la 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 "37°2" en base est très proche de "37°3" qui est la donnée réelle et que, par conséquent, il n'est pas nécessaire de mettre à jour cette donnée. Cette déviation possède un seuil (MDE[3]) qui permet de déterminer si la transaction qui souhaite mettre à jour une donnée temps réel peut être écartée ($DE < MDE$) ou pas ($DE \geq MDE$). Ce travail est effectué par le *contrôleur de précision*.

3.3 Le modèle global

Nous allons présenter en détail dans cette section les parties du modèle qui nous intéressent plus particulièrement dans cet article. En reprenant la figure 1 [Amirijoo, 2003], nous allons considérer le modèle général et donner 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 les DBA[4]. Son fonctionnement est contrôlé par la boucle de rétroaction qui lui fournit ses paramètres de fonctionnement.

Les transactions qui sont admises dans le système sont placées dans une file d'attente avant d'être envoyées au *déclencheur de transactions*. Ce *déclencheur de transactions* possède 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 (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 du protocole 2PL-HP (*Two Phase Locking High Priority*) [Abbott, 1988].
- Un *ordonnanceur de base* : il s'agit bien souvent de EDF (*Earliest Deadline First*) [Buttazzo, 1997] qui ordonnance les transactions selon le principe que la transaction qui possède l'échéance la plus proche doit être exécutée en priorité.

Les différents modules du *déclencheur de transactions* peuvent être remplacés par des modules équivalents. C'est ainsi que l'on peut modifier la politique de contrôle de concurrence ou modifier la politique d'ordonnancement.

Un *moniteur* permet de mesurer les performances du système en inspectant l'exécution des transactions (quantité de transactions terminées, abandonnées, qui ont ratées leur échéance,...). Les valeurs ainsi mesurées permettent d'alimenter le *contrôleur de qualité de service* et font parties 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 qualité de service*) permet de réajuster les paramètres de QdS en fonction des valeurs déterminées par le *moniteur* et des paramètres de référence[5]. 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 considération de la valeur de MDE.

[2] Data Error
[3] Maximum Data Error
[4] DataBase Administrator
[5] fournit par le DBA

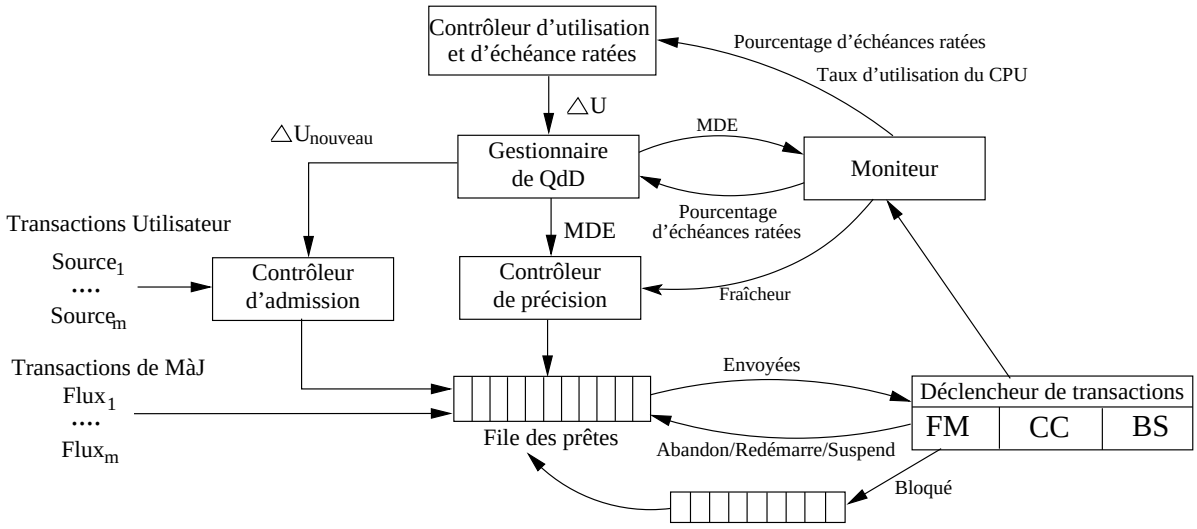

FIG. 1 – Architecture de SGBDTR basée sur la rétroaction.

3.4 La boucle de contrôle par rétroaction

La boucle de rétroaction possède pour tâche de stabiliser le système durant les phases d'instabilité. 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 les transactions acceptées dans le système. Cette adaptation va provoquer une modification du comportement du système et ainsi des valeurs observées par le *moniteur*.

Le fonctionnement de la boucle de rétroaction 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%). Le but est donc de réduire l'oscillation du système autour de cette valeur de référence.

4 TRANSACTIONS DE MISE À JOUR ET CONTRAINTES (M,K)-FIRM

Dans cette section, nous nous proposons d'appliquer les contraintes (m,k)-firm en combinaison avec la boucle de rétroaction pour pouvoir écarter certaines instances des transactions de mise à jour lorsque la charge du système varie.

4.1 Problématique

Le contrôle par rétroaction présenté dans la section précédente permet de filtrer les transactions utilisateur suivant la charge observée du système. Si la charge du système augmente, alors on écarte de plus en plus de transactions utilisateur pour rétablir cette charge. Pour les transactions de mise à jour, le contrôle s'effectue sur la fraîcheur des données. Autrement dit, on peut écarter des transactions si la valeur à mettre à jour est similaire à la précédente. Cela implique que, dans un premier temps, toutes les transactions de mise à jour sont acceptées par le système et insérées dans la file d'attente. Dans le cas où la charge du SGBDTR devient trop importante, on ajuste la valeur du paramètre MDE et on vérifie la fraîcheur des données à mettre à jour. En d'autres termes, toutes les transactions de mise à jour sont considérées comme étant de même niveau (de même priorité) ; seule la périodicité des mises à jour permet de spécifier la criticité des données temps réel.

Le contrôle d'admission des transactions utilisateur est plus "sévère" puisque dès leur arrivée dans le système, elles doivent répondre aux critères du contrôleur d'admission pour être insérées dans la file d'attente. Ainsi, on considère que les transactions de mise à jour sont plus prioritaires que les transactions utilisateur : il est difficilement concevable de laisser passer des transactions utilisateur alors que les données qu'elles doivent accéder ne sont pas fraîches. Cependant, nous pensons qu'un contrôle des transactions de mise à jour dès leur arrivée permet de rétablir plus rapidement la stabilité de la charge du système tout en garantissant une certaine QdS.

Dans la suite de ce article, nous nous intéressons au contrôle d'admission des transactions de mise à jour en utilisant les contraintes (m,k)-firm proposées pour la gestion des tâches dans les systèmes temps réel en combinaison avec la méthode de rétroaction.

4.2 Définition des transactions de mise à jour sous contraintes (m,k)-firm

Avant de montrer les modifications apportées au système pour intégrer le contrôle des transactions de mise à jour, nous allons définir le concept de "transactions de mise à jour (m,k)-firm". Le but de ce contrôle est de pou-

voir écarter des transactions de mise à jour avant même qu'elles n'entrent dans le système tout en garantissant qu'elles respectent certaines contraintes.

L'une des caractéristiques intéressantes des transactions de mise à jour est la périodicité. En effet, les contraintes (m,k)-firm sont basées sur la périodicité des tâches temps réel pour pouvoir écarter certaines invocations en limitant la fréquence de perte. Nous proposons d'appliquer les contraintes (m,k)-firm aux transactions de mise à jour de la façon suivante.

Chaque transaction (de mise à jour) se voit assigner deux paramètres m et k : le premier indique le nombre minimum d'instances qui doivent respecter leur échéance et le second limite la période durant laquelle des instances peuvent être perdues. Autrement dit, la définition des contraintes (m,k)-firm pour les transactions de mise à jour est identique à celle des contraintes (m,k)-firm pour les tâches dans les systèmes temps réel.

On attribue également un autre paramètre, m_c (m courant), permettant de balayer les différentes valeurs comprises entre m et k. Au début, la valeur de m_c est fixée à k. Lorsqu'une surcharge du système survient, on tente de diminuer la valeur du paramètre m_c tout en respectant la contrainte $m_c \geq m$. Inversement, si on constate une sous-utilisation du système, alors la valeur de m_c peut être incrémentée en respectant la contrainte $m_c \leq k$.

Il reste à déterminer dans quelle mesure l'invocation courante d'une transaction de mise à jour peut être écartée du système en respectant ses contraintes (m,k)-firm. Pour cela, on utilise une méthode similaire au protocole d'ordonnancement DBP proposé pour les tâches temps réel. En fait, on vérifie, à partir de l'historique de la transaction, si l'invocation courante est une invocation obligatoire ou optionnelle tout en respectant les contraintes (m_c,k)-firm de la transaction. Si l'invocation courante est optionnelle, alors elle peut être écartée du système.

Il est intéressant de noter que même une fois acceptées par le contrôleur (m,k)-firm, les transactions de mise à jour peuvent à nouveau être écartées par le *contrôleur de précision*.

4.3 Contraintes (m,k)-firm et criticité des transactions

Les contraintes (m,k)-firm permettent de gérer différents niveaux de criticité des transactions. En effet, plus l'écart entre m et k est important, plus la transaction peut écarter des instances et moins elle est critique. Dans la pratique, les valeurs lues par les capteurs ne sont pas toutes de la même importance. Par exemple, dans un système temps réel embarqué tel qu'un avion, les informations concernant la température des moteurs sont plus importantes que les informations concernant l'air conditionné de la cabine. Dans ce cas, pour les capteurs de la cabine, on permettra à plus d'instances de la transaction de mise à jour d'être perdues que pour la transaction qui gère la température des moteurs. On pourra, par exemple, attribuer des contraintes (9,10)-firm pour la température des moteurs et des contraintes (10,100)-firm pour la température de la cabine. Les contraintes (m,k)-firm devront donc être fixées par le concepteur de l'application pour chaque transaction de mise à jour en fonction de la criticité des données auxquelles elles accèdent.

4.4 Modifications apportées au modèle par rétroaction

Pour manipuler les transactions de mise à jour (m,k)-firm, il est nécessaire d'apporter quelques modifications au modèle général de SGBDTR présenté sur la figure 1. Il faut notamment ajouter le contrôleur d'admission des transactions de mise à jour basé sur les contraintes (m,k)-firm des transactions. Nous le nommerons "contrôleur (m,k)-firm".

Ce contrôleur nécessite des informations provenant d'autres composants pour gérer l'admission des transactions. En effet, on souhaite écarter plus ou moins les invocations des transactions lorsque la charge du système varie. C'est pourquoi, comme pour le contrôleur d'admission des transactions utilisateur, un paramètre provenant de la boucle de rétroaction lui est transmis (ΔU_{maj}). Cela nécessite de modifier l'algorithme employé au niveau du *gestionnaire de qualité des données* afin de répartir l'effort à fournir pour chacun des composants (*contrôleur d'admission*, *contrôleur (m,k)-firm*, *contrôleur de précision*).

La seconde information nécessaire au contrôleur (m,k)-firm est l'historique des transactions de mise à jour. Il permet en effet de savoir si une instance peut être rejetée tout en respectant les contraintes (m,k)-firm. Pour cela, le *déclencheur de transactions* reporte la réussite (ou non) des transactions de mise à jour au contrôleur (m,k)-firm.

Pour maintenir un historique qui reflète correctement le déroulement des transactions de mise à jour, il est nécessaire de considérer que les instances écartées par le contrôleur de précision soient des instances ayant respecté leur échéance. En effet, les opérations du *contrôleur de précision* ne doivent pas influencer le fonctionnement du *contrôleur (m,k)-firm*. Du point de vue du contrôleur (m,k)-firm, la transaction s'est exécutée correctement et elle a réécrit la même valeur dans la base de données.

La figure 2 illustre le modèle du SGBDTR basé sur la rétroaction et tenant compte des contraintes (m,k)-firm des transactions de mise à jour. Le contrôleur (m,k)-firm permet de gérer l'admission des transactions de mise à jour en fonction de la charge du système et de l'historique des transactions. Ainsi, le contrôleur (m,k)-firm permet de résoudre les phases d'instabilité du système en permettant à plus ou moins de transactions de mise à jour de s'exécuter à partir des renseignements fournis par la boucle de rétroaction.

5 CONCLUSION ET PERSPECTIVES

Dans cet article, nous avons mis en œuvre un contrôleur (m,k)-firm pour gérer l'admission des transactions de mise à jour dans une architecture de contrôle par rétroaction. Le contrôleur (m,k)-firm utilise la notion d'invocations obligatoires et optionnelles des transactions périodiques pour écarter certaines d'entre elles en fonction des oscillations de la charge du système. Ce

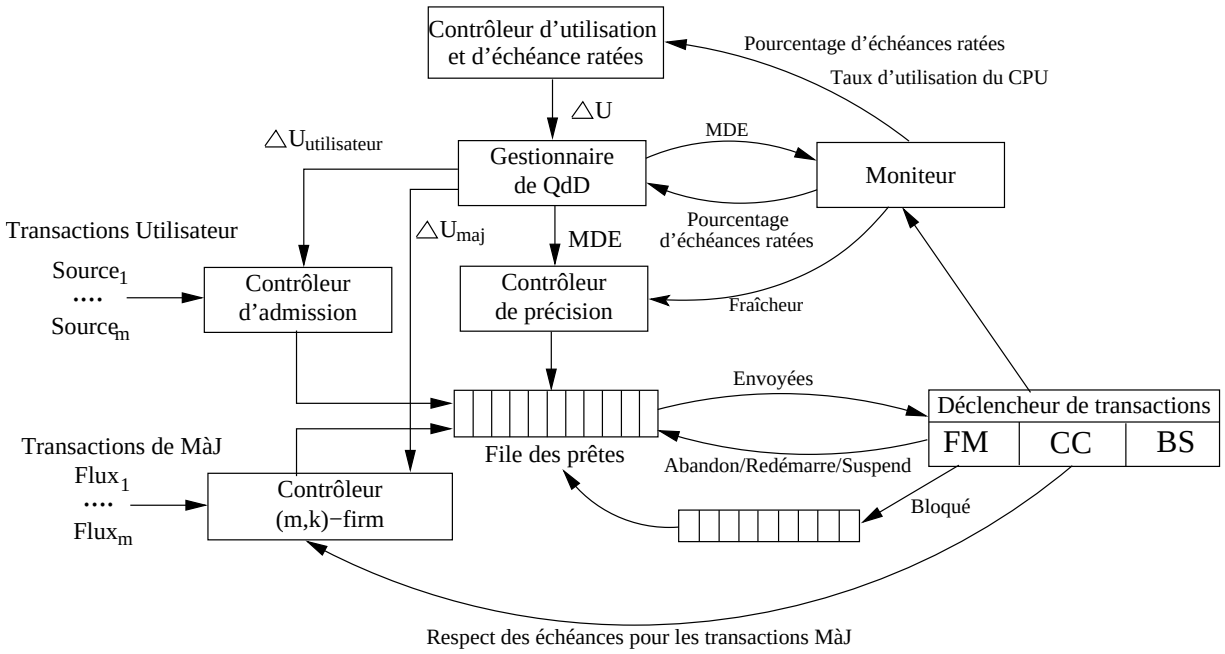

FIG. 2 – Application des contraintes (m,k)-firm aux transactions de mise à jour

contrôleur nous permet donc de spécifier une politique d'admission des transactions qui compense les variations de la charge du système (ΔU_{maj}) et ainsi de faire varier la qualité des données et plus généralement la qualité de service dans les SGBD temps réel.

Il s'agissait de démontrer que l'ajout du contrôleur (m,k)-firm permet de diminuer et de rétablir plus rapidement les phases d'instabilité du système (phases de sous utilisation, et phase de surcharge). Dans nos travaux immédiats, il est nécessaire de mettre en œuvre notre approche au sein du simulateur employé par Kang et al. [Kang, 2002] et par Amirijoo et al. [Amirijoo, 2003]. Cette implémentation est en cours de réalisation et permettra de démontrer l'apport de notre contribution dans la stabilisation des systèmes. Les mesures de performances reposeront sur le taux d'échecs[6] des transactions temps réel et une comparaison avec le système dépourvu de contrôleur (m,k)-firm.

Dans nos travaux futurs, nous continuerons d'étudier les possibilités d'améliorations de la robustesse des SGBDTR par une approche basée sur la spécification de la qualité de service. Notamment, nous allons étudier l'intégration des contraintes (m,k)-firm au niveau du contrôleur de concurrence pour tenir compte des parties obligatoires et optionnelles des transactions.

Une autre voie que nous explorons est l'adaptation de ses 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.

6 $\frac{NombreDeTransactionsQuiManquentLeurEch\acute{e}ance}{NombreTotalDeTransactions}$

6 REMERCIEMENTS

Les auteurs tiennent à remercier le Ministère Français de la Recherche et l'Université du Havre pour leur contribution financière à la réalisation de ce papier (ACI-JC 1055).

BIBLIOGRAPHIE

[Abbott, 1988] Abbott, R. and Garcia-Molina, H. "Scheduling Real-Time Transactions : A Performance Evaluation". Proc. of the Proc. of the 14th Int. Conf. on Very Large Data Bases Conference (VLDB), pages 1–12, Los Angeles, California (1988).

[Amirijoo, 2003] Amirijoo, M., Hansson, J., and Son, S. "Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation". Proc. of the Proc of the 9th Real-Time and Embedded Computing Systems and Applications (RTCSA) (2003).

[Bernat, 1998] Bernat, G. Specification and Analysis of Weakly Hard Real-Time Systems. PhD thesis, Dep. Ciènces Matemàtiques i Informàtica, Universitat de les Illes Balears (1998).

[Buttazzo, 1997] Buttazzo, G. "Hard Real-Time Computing Systems". Kluwer Academic Publishers (1997).

[Duvallet, 1999] Duvallet, C., Mammeri, Z., and Sadeg, B. "Les SGBD Temps Réel". Technique et Science Informatiques, Vol. 18, N°. 5, p. 479–517 (1999).

[Hamdaoui, 1995] Hamdaoui, M. and Ramanathan, P. "A Dynamic Priority Assignement Technique for Streams with (*m*,*k*)-Firm Deadlines". IEEE Transactions on Computers, Vol. 44, N°. 4, p. 1325–1337 (1995).

[Haubert, 2004] Haubert, J., Sadeg, B., and Amanton, L. "(m,k)-firm real-time distributed transactions". Proc. of

the 16th Euromicro Conference on Real-Time Systems (ECRTS) (2004).

[Kang, 2002] Kang, K., Son, S., Stankovic, J., and Abdelzaher, T. "A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases". Proc. of the Euromicro Conference on Real-Time Systems (2002).

[Liu, 1994] Liu, J., Shih, W.-K., Lin, K.-J., Bettati, R., and Chung, J.-Y. "Imprecise computations". Proceedings of the IEEE, Vol. 82, Nº. 1, p. 83–94 (1994).

[Lu, 2001] Lu, C. Feedback control real-time scheduling. PhD thesis, University of Virginia (2001).

[Ramamritham, 1993] Ramamritham, K. "Real-Time Databases". Journal of Distributed and Parallel Databases, Vol. 1, Nº. 2, p. 199–226 (1993).

[Ramanathan, 1999] Ramanathan, P. "Overload Management in Real-Time Control Applications Using (m,k)-firm Guarantee". IEEE Transaction on Parallel and Distributed Systems, Vol. 10, Nº. 6, p. 549–559 (1999).

[Wang, 2002] Wang, Z., Song, Y., Poggi, E., and Sun, Y. "Survey of Weakly-Hard Real-Time Scheduling Theory and Its Application". Proc. of the Proc. of the Int. Symp. on Distributed Computing and Applications to Business, Engineering and Science (DCABES) (2002).