Prise en compte des données dérivées temps réel dans une architecture de contrôle par rétroaction

Mounir Katet, Claude Duvallet, Emna Bouazizi, Bruno Sadeg
LITIS, UFR des Sciences et Techniques,
25 rue Philippe Lebon, BP 540,
F-76058 LE HAVRE CEDEX,
Courriel : {Claude.Duvallet,Emna.Bouazizi,Bruno.Sadeg}@univ-lehavre.fr

Résumé : Dans les bases de données temps réel, il existe des données, dites de base (ou sensorielle). Ces données sont issues de capteurs et reflètent l'état de l'environnement. Elles sont régulièrement mises à jour par des transactions de mise à jour. Ces mêmes données servent à calculer d'autres données, dites dérivées, qui sont donc temps réel. À chaque mise à jour d'une donnée sensorielle, les données dérivées dont elles dépendent doivent être mises à jour. Plus le nombre de données temps réel utilisées pour calculer une donnée dérivée est grand, plus les mises à jour de ces données risquent d'être nombreuses. Par conséquent, le nombre transactions de mise à jour des données dérivées peut surcharger rapidement le système. Dans cet article, nous proposons de définir une politique pour la mise à jour des données dérivées basée sur une approche permettant de minimiser la charge engendrée et de maximiser l'efficacité des mises à jour en garantissant la fraîcheur des données lorsqu'elles sont utilisées. Cette politique est mise en place dans une architecture à ordonnancement par rétroaction.

Mots-clés : architecture logicielle, bases de données temps réel, données dérivées temps réel, transactions de mise à jour.

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 utiliser 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 [Ramamritham, 1993] [Duvallet , 1999]. Il s'agit, en effet, de prendre en compte les contraintes temporelles des applications (transactions) tout en exploitant des données temporellement valides. La cohérence temporelle d'une base de données temps réel consiste à disposer de données dont l'échéance n'est pas dépassée. Cette échéance est déduite de l'estampille (date de dernière mise à jour) et de la durée de validité de la donnée. On peut considérer deux types de données temps réel :

– les données de base (sensorielles) : il s'agit généralement de données issues de capteurs qui permettent de rendre compte de l'état de l'environnement (exemple : température, pression, etc.).
– les données dérivées : elles sont calculées à partir d'une ou plusieurs données de base et permettent de déduire des informations sur l'environnement. On peut par exemple à partir de la distance parcourue par un véhicule et de la durée du parcours déduire sa vitesse et éventuellement sa nouvelle position.

Les applications temps réel doivent 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, 2001] [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). La rétroaction consiste à modifier les paramètres de fonctionnement du système en fonction des performances observées. Après ces modifications, le système continue son travail et une nouvelle observation est effectuée. Puis, de nouveau une modification des paramètres peut être effectuée. La boucle "observation - modification - fonctionnement" fonctionne indéfiniment et doit tendre vers la stabilité du système autour de paramètres de référence spécifiés par l'administrateur de la base de données. Dans les travaux réalisés jusqu'à maintenant, on ne considère que les données de bases (ou sensorielles).

Notre objectif est ici d'introduire l'utilisation des données dérivées dans des architectures avec rétroaction. Pour cela, nous nous appuyons sur les travaux effectués sur les données dérivées [Adelberg , 1996] et sur les travaux concernant les architectures basées sur une approche qualité de service [Amirijoo , 2006].

Dans la section 2, nous présentons les travaux effectués sur les données et sur la gestion de qualité de service (QdS) dans les SGBDTR. Puis, dans la section 3, nous

décrivons en détails la problématique de notre approche. Nous proposons ensuite une modification de l'architecture de contrôle par rétroaction pour la prise en compte des données dérivées (cf. section 4). Dans la section 5, nous discuterons de la validation de nos travaux. Enfin, nous conclurons sur un bilan de nos résultats et les perspectives que nous donnons à notre travail.

2 ÉTAT DE L'ART

2.1 La gestion de la QdS dans les SGBD temps réel

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'efficacité du SGBDTR est mise 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].

2.1.1 Les transactions temps réel et la qualité des transactions

Les transactions temps réel utilisées possèdent des échéances de type *firm* (à échéances strictes non critiques [Duvallet , 1999]). Par conséquent, lorsqu'elles dépassent leur échéance, elles deviennent inutiles pour le système et sont abandonnées.

Dans la plupart des travaux de ce type, deux catégories de transactions temps réel sont considérées :

- Les *transactions de mise à jour :* elles ont pour tâche de mettre à jour régulièrement les données acquises depuis les capteurs. Ces transactions sont exécutées périodiquement pour rafraîchir les valeurs des données temps réel.
- Les *transactions « utilisateur » :* elles effectuent des opérations de lecture/écriture des données non temps réel et/ou des lectures des 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 contrôle par rétroaction pour gérer les variations importantes de charge [Lu, 2001].

Dans les travaux d'Amirijoo [Amirijoo , 2003], la conception des transactions temps réel est basée sur des techniques de calcul imprécis [Liu , 1994]. Chaque transaction est composée d'une partie obligatoire et de plusieurs parties optionnelles qui sont exécutées suivant le temps disponible pour l'exécution de la transaction. Plus le nombre de parties optionnelles exécutées avant échéance est grand, meilleure est la qualité des transactions (QdT).

[1] Feedback Control Scheduling Architecture (FCSA).

2.1.2 Les données temps réel de base et la qualité des données

Les données temps réel de base 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 la notion de qualité des données (QdD) [Amirijoo , 2003] qui permet de considérer qu'une donnée stockée dans la base peut posséder un certain écart par rapport à sa valeur dans le monde réel.

On appelle cet écart "erreur sur la donnée" (noté DE[2]) 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 "37°2" dans la 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* [Kang , 2002].

2.1.3 Le modèle global

Dans cette section, nous allons présenter les différentes parties du modèle utilisées dans la gestion de la qualité de service basé sur une architecture de contrôle par rétroaction. La figure 1 donne une vision générale du modèle souvent utilisé dans ce type de travaux [Amirijoo , 2003]. Nous allons donner une brève description de chacun des composants de ce modèle.

La tâche du *contrôleur d'admission* est de contrôler les transactions « utilisateur » qui sont 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 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* qui gère l'exécution des transactions. Il dispose pour cela 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 apparaissent entre les transactions. Dans la plupart des tra-

[2] Data Error.

[3] Maximum Data Error.

[4] administrateur de la base de données

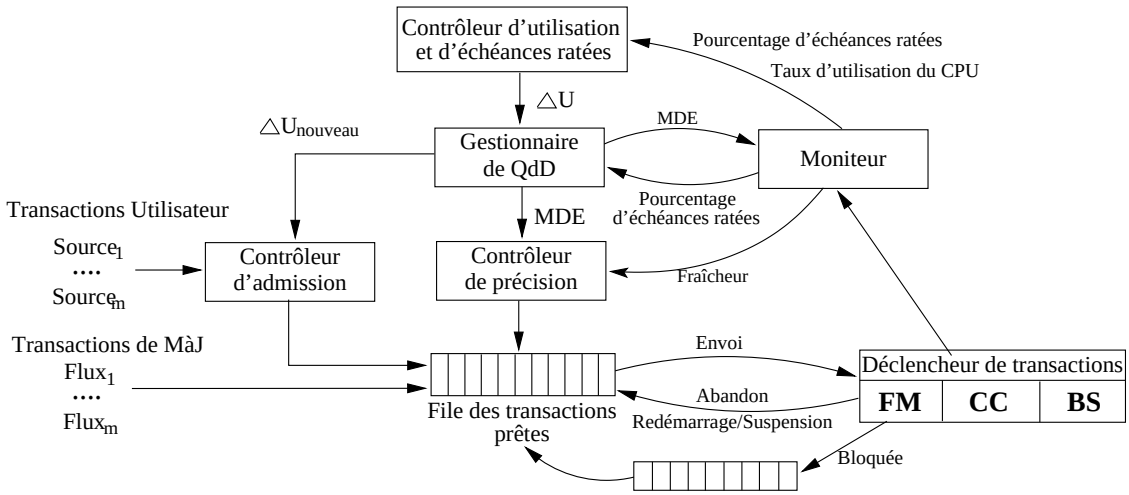

FIG. 1 – Architecture d'ordonnancement par rétroaction.

vaux [Kang , 2002] [Amirijoo , 2003], il s'agit du protocole 2PL-HP (*Two Phase Locking High Priority*) [Abbott , 1988] où une transaction de basse priorité est mise en attente en cas de conflits avec une transaction de plus haute priorité.

– Un *ordonnanceur de base* : il s'agit 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.

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 d'utilisation et d'échéances ratées* 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* 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. 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 considération de la valeur de MDE.

2.1.4 La boucle de contrôle par rétroaction

La boucle de rétroaction a 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 d'utilisation et d'échéances ratées*, afin d'augmenter ou de diminuer le nombre de transactions acceptées dans le système.

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%).

2.2 Les données dérivées temps réel

Les données dérivées sont des données qui sont calculées ou mises à jour à partir d'autres données de la base. La problématique essentielle de ces données est leur mise à jour par des transactions de mise à jour. En effet, chaque fois qu'une des données utilisées pour la mise à jour des données dérivées est modifiée alors il faut recalculer les données qui en dépendent.

Les données dérivées temps réel utilisent des données de base qui sont temps réel, c'est-à-dire qui possèdent un intervalle de validité pendant lequel elles peuvent être utilisées. Lorsque l'on calcule une donnée dérivée à partir de plusieurs données temps réel, cette donnée va elle-même devenir temps réel et son intervalle de validité sera l'intersection des intervalles de validité de chaque donnée temps réel utilisée.

2.3 Les différentes politiques de mise à jour des données dérivées

Pour mettre à jour les données dérivées, on peut utiliser différentes politiques de mises à jour en fonction des résultats attendus. Il est habituel de considérer les cinq politiques suivantes :

– la mise à jour périodique,
– la mise à jour déclenchée,

[5] fournis par le DBA.

– la mise à jour à la demande,
– la mise à jour à retard forcé,
– la mise à jour à délai forcé.

Nous allons maintenant détailler le principe de ces différentes politiques et en exprimer les avantages et les inconvénients.

2.3.1 Mise à jour périodique

Dans [Song , 1990] et [Ramamritham, 1993], les auteurs ont considéré que la base de données temps réel a besoin d'une mise à jour périodique. Ils ont donc mis en œuvre une politique de mise à jour périodique des données de base et des données dérivées. Cette politique se justifie complètement pour la mise à jour des données de base qui, la plupart du temps, sont des données sensorielles issues de capteurs et qui changent de façon continue. Pour les données dérivées, cette politique possède un inconvénient majeur : elle peut aboutir à disposer de données non fraîches à certaines périodes. En effet, les périodes de mise à jour des données de base ne sont pas obligatoirement toutes identiques et elles ne coïncident pas forcément avec la période de mise à jour des données dérivées. Une transaction temps réel qui cherche à accéder à une donnée dérivée qui est non fraîche doit donc attendre la prochaine période de mise à jour. Ceci allonge leur durée d'exécution et augmente le risque de rater leur échéance.

2.3.2 Mise à jour déclenchée

Dans [Adelberg , 1996], les auteurs considèrent que les données dérivées deviennent non fraîches chaque fois qu'une de leurs données de base est actualisée. Par conséquent, dans [Gustafsson , 2004], toutes les mises à jour des données dérivées sont déclenchées dès qu'une mise à jour de l'une des données de base est effectuée. Dans [Adelberg , 1996], les auteurs décrivent deux scénarios pouvant aboutir au déclenchement d'un très grand nombre de transactions de mise à jour des données dérivées :

1. Si une donnée de base est utilisée dans le calcul de nombreuses données dérivées, alors à chaque mise à jour de cette donnée, un grand nombre de transactions de mise à jour des données dérivées est déclenché.
2. Si les données de base utilisées pour le calcul d'une donnée dérivée sont nombreuses et mises à jour très régulièrement alors le nombre de déclenchements de la mise à jour de la donnée dérivée devient très important.

2.3.3 Mise à jour à la demande

Pour limiter le nombre de mises à jour des données, Ahmed et Vrbsky [Ahmed , 2000] proposent une autre politique qui représente un compromis entre la correction temporelle et la ponctualité des transactions (transaction timeliness). Dans leur modèle, quand une transaction « utilisateur » demande à accéder à une donnée non fraîche (obsolète), le système génère une transaction de mise à jour pour rafraîchir cette donnée (appelée mise à jour à la demande). La transaction de l'utilisateur attend la fin de la mise à jour de la donnée. Elle accède donc à la valeur la plus récente de la donnée et la cohérence temporelle est maintenue. L'inconvénient de la mise à jour à la demande réside dans le fait que la transaction « utilisateur » soit obligée d'attendre la mise à jour. Ceci augmente le risque que cette transaction rate son échéance.

2.3.4 Mise à jour à retard forcé

L'échéance d'une donnée est l'instant après lequel la donnée n'est plus temporellement correcte : elle devient obsolète. Pour maintenir la cohérence temporelle, chaque transaction qui accède à une donnée doit être validée avant que la donnée n'ait atteint son échéance, sinon la transaction sera abandonnée. Pour réduire le nombre de transactions qui ratent leurs échéances, dans [Xiong , 1997] les auteurs proposent la politique à retard forcé. Dans cette politique, la transaction qui accède à une donnée temps réel vérifie qu'elle pourra être validée avant la fin de la validité de la donnée. Si cela n'est pas possible, la transaction doit attendre jusqu'à ce que la donnée demandée soit mise à jour.

2.3.5 Mise à jour à délai forcé

Si une transaction de mise à jour est exécutée sur une donnée de base, alors toutes les données dérivées qui en dépendent doivent normalement être recalculées. Dans cette politique, afin de limiter le nombre de mises à jour des données dérivées, on bloque la mise à jour de la donnée dérivée pendant un certain temps après sa dernière mise à jour. Durant ce délai, même s'il y a des mises à jour pour les données de base, la donnée dérivée n'est plus recalculée jusqu'à la fin du blocage [Adelberg , 1996].

3 PROBLÉMATIQUE DES DONNÉES DÉRIVÉES TEMPS RÉEL

Pour le traitement et l'utilisation des données dérivées temps réel, on s'appuie sur les travaux effectués pour les données dérivées classiques. Mais, on doit cette fois tenir compte du fait que les données utilisées pour mettre à jour ces données dérivées sont des données temps réel. Par conséquent, elles possèdent une période de validité qui va servir au calcul de la période de validité de la donnée dérivée. Pour chaque donnée dérivée temps réel, on va donc construire un ensemble de validité relative qui tiendra compte de chaque période de validité des données utilisées.

De plus, on considère trois types de données dérivées temps réel en fonction des données utilisées pour calculer ces données :

1. les données dérivées temps réel calculées uniquement à partir de données temps réel de base (données sensorielles).
2. les données dérivées temps réel calculées à partir de données temps réel de base et des données dérivées temps réel. Il semble que l'utilisation de données dérivées temps réel pour calculer d'autres

données temps réel ne pose pas de problèmes supplémentaires par rapport au premier cas. Par conséquent, on pourrait regrouper ces deux cas.

3. les données dérivées temps réel calculées uniquement à partir de données temps réel de base et de données classiques (non temps réel).

La question essentielle est donc de concevoir une politique de gestion des transactions de mise à jour des données dérivées temps réel. Il faut faire varier la quantité de transactions de ce type qui s'exécutent en fonction de la charge du système.
Les questions posées sont celles concernant le déclenchement des mises à jour des données dérivées. La méthode la plus simple consiste à déclencher la transaction de mise à jour d'une donnée dérivée dès qu'une de ses données relatives (dont elle dépend pour son calcul) a été modifiée. Mais, imaginons que cette donnée dépende de nombreuses données, alors le nombre de mises à jour va être d'autant plus grand. Imaginons encore que de nombreuses données dérivées dépendent d'une même donnée de base, alors dès que cette donnée est mise à jour, tout un ensemble de transactions est déclenché pour mettre à jour les données dérivées.
De plus, la période de validité d'une donnée n'expire pas toujours en même temps que sa mise à jour comme on peut le voir dans des approches de type "more-less" [Ramamritham , 2004]. Par conséquent, la validité des données dérivées, qui sont calculées à partir de cette donnée, n'expire pas au moment de la mise à jour d'une de ses données relatives.

4 UNE ARCHITECTURE POUR LA PRISE EN COMPTE DES DONNÉES DÉRIVÉES TEMPS RÉEL

Pour prendre en considération les données dérivées temps réel dans les architectures à contrôle par rétroaction, nous considérons (1) la mise à jour de ces données et la définition d'une politique permettant de tenir compte des variations de charge et (2) les modifications nécessaires à faire dans l'architecture.

4.1 Une politique mixte de mise à jour des données dérivées

Vu l'insuffisance de chacune de ces politiques de mise à jour des données dérivées, nous proposons une politique qui garantit la fraîcheur des données dérivées. Par conséquent, un grand nombre des transactions « utilisateur » respectent leur échéance tout en tenant compte de l'état du système qui passe par des périodes de surcharge et par des périodes de sous-utilisation. Nous proposons d'utiliser une politique mixte basée sur :

– L'utilisation de la politique de mise à jour à la demande de données dérivées dans le cas où le système est surchargé. Cette politique permet de mettre à jour les données uniquement lorsqu'elles sont demandées par d'autres transactions en lecture et qu'elles ne sont plus suffisamment fraîches. Cette politique est mise en œuvre par le gestionnaire de fraîcheur chaque fois qu'une donnée dérivée qui doit être accédée en lecture n'est plus valide.
– L'utilisation de la politique de mise à jour déclenchée est effectuée pendant les périodes de sous-utilisation du système. Il s'agit de la politique utilisée par défaut pour la mise à jour des données dérivées. En effet, même si cette politique nécessite des ressources importantes, elle permet de garantir une très grande fraîcheur des données dérivées dans les périodes de sous-utilisation du système.
– L'utilisation de la politique de mise à jour à retard forcé est effectuée lorsque le système entre dans une période de surcharge. Le paramètre MVI (Maximum Validity Interval), qui reflète l'état de surcharge de système, est utilisé pour déterminer le retard maximum auquel la mise à jour de la donnée dérivée sera soumise.

Nous allons intégrer cette politique mixte de mise à jour des données dérivées dans l'architecture de contrôle d'ordonnancement avec rétroaction. Elle sera particulièrement mise en œuvre par le contrôleur d'admission des transactions de mise à jour des données dérivées.

4.2 L'architecture de gestion des données dérivées temps réel

Pour manipuler les transactions de mise à jour des données dérivées, il faut notamment ajouter à l'architecture de base (FCSA Classique, cf. figure 1) le contrôleur d'admission des transactions de mise à jour des données dérivées. Ce contrôleur nécessite des informations provenant d'autres composants pour gérer l'admission des transactions.

C'est pourquoi, comme pour le contrôleur d'admission, un paramètre provenant de la boucle de rétroaction lui est transmis (MVI). La valeur de MVI est calculée en fonction de l'utilisation du système. Cela nécessite de modifier l'algorithme employé au niveau du gestionnaire de qualité des données afin de répartir l'effort à fournir sur chacun des composants (contrôleur d'admission, contrôleur d'admission des données dérivées, contrôleur de précision) qui ont en charge les différents types de transactions présentes dans le système.

Lorsqu'une transaction « utilisateur » cherche à accéder à une donnée dérivée, le gestionnaire de fraîcheur présent au sein du déclencheur de transactions effectue une vérification sur la fraîcheur de la donnée. Si la donnée est obsolète (non valide) alors la transaction « utilisateur » est envoyée vers la file d'attente des transactions bloquées et parallèlement un ordre de mise à jour à la demande est envoyé au contrôleur d'admission des transactions de mise à jour des données dérivées. Cet ordre aura pour effet de ne pas écarter la transaction de mise à jour de la donnée dérivée correspondante, mais de l'exécuter directement, quelque soit la valeur de MVI.

La Figure 2 illustre le modèle de SGBD temps réel basé sur la rétroaction et tenant compte de la gestion des données dérivées. Le contrôleur d'admission des transactions de mise à jour des données dérivées permet de gérer l'admission de ces transactions en fonction de la charge du système et de l'état des données dérivées

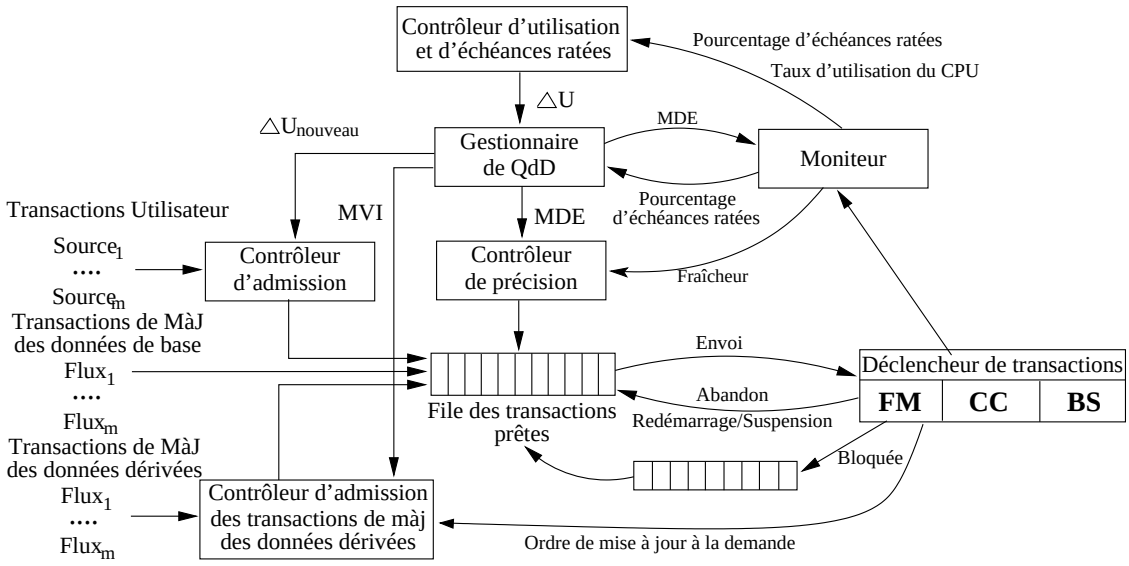

FIG. 2 – Architecture d'ordonnancement par rétroaction et prise en compte de données dérivées.

présentes dans le système. Ainsi, ce contrôleur permet de prendre en considération les phases d'instabilité du système en permettant à plus ou moins de transactions de s'exécuter à partir des renseignements fournis par la boucle de rétroaction. De même, ce contrôleur se base sur la valeur du paramètre MVI pour appliquer la politique mixte de mise à jour des données dérivées vue dans le paragraphe 2.3.

Le fonctionnement du contrôleur d'admission des transactions de mise à jour des données dérivées est basé sur l'exploitation de la politique mixte que nous avons évoquée dans le paragraphe 4.1 et sur l'utilisation du paramètre MVI. Le paramètre MVI représente le délai maximum qu'il faudra attendre pour la mise à jour d'une des données de base utilisées pour recalculer la donnée dérivée.

Pour une donnée dérivée, nous prenons en considération toutes les périodes des transactions servant à mettre à jour ses données de base. Ensuite, nous regardons la distance entre l'instant courant et la date de mise à jour (la plus proche) pour l'une de ses données de base. Si cette distance est inférieure à la valeur de MVI, nous attendons cette mise à jour (politique de retard forcé) sinon nous déclenchons la transaction de mise à jour de la donnée dérivée considérée. Afin de ne pas retarder trop longtemps la mise à jour de la donnée dérivée, nous accumulons les retards pour les comparer avec la valeur de MVI. Ce retard étant remis à zéro lors de la mise à jour de la donnée dérivée.

4.3 Bilan

Dans cette section, nous avons conçu et mis en œuvre un contrôleur d'admission des transactions de mise à jour des données dérivées pour gérer les transactions de mise à jour de ces données dérivées dans une architecture de contrôle par rétroaction. Ce contrôleur nous permet de spécifier une politique d'admission des transactions de mise à jour des données dérivées qui tient compte des variations de charge du système (MVI) et ainsi de faire varier la qualité des données et plus généralement la qualité de service dans les SGBD temps réel. Cette nouvelle architecture permet de prendre en considération la gestion des données dérivées et se base sur l'utilisation d'une politique mixte de mise à jour (déclenchée, à la demande et retard forcé).

5 DISCUSSION

Les travaux précédents sur les architectures de SGBDTR basées sur le contrôle par rétroaction n'ont pas pris en considération la gestion des données dérivées et leur influence sur la QdS dans le système pendant les périodes d'instabilité. Dans notre travail, nous avons tenu compte de ce type de données afin de partager la charge du système sur trois types de transactions (utilisateur, de mise à jour de données de base, de mise à jour de données dérivées). Une bonne qualité de données dérivées conduit à une bonne QdS du système.

Pour gérer les données dérivées en fonction de la charge du système, nous avons proposé une politique mixte de mise à jour des données dérivées qui prend en considération l'état de la charge du système. L'apport de cette politique mixte est qu'elle essaie d'utiliser, dans la plupart de temps, les données dérivées fraîches. Par conséquent, elle minimise le nombre de transactions demandeuses d'une donnée dérivée qui ratent leur échéance.

6 CONCLUSIONS ET PERSPECTIVES

Dans cet article, nous avons proposé une nouvelle architecture d'ordonnancement par rétroaction permettant la gestion de la qualité de service. L'apport de notre travail consiste à introduire une gestion spécifique des données dérivées temps réel. Nous avons, pour cela, (1) ajouté un

nouveau composant : le contrôleur d'admission des transactions de mise à jour des données dérivées temps réel et (2) défini une politique de mise à jour des données dérivées temps réel.

Plusieurs perspectives peuvent être données à ce travail. L'une d'entre elles consiste à appliquer ce travail dans d'autres architectures à ordonnancement par rétroaction [Bouazizi , 2006], car ici nous avons travaillé uniquement avec l'architecture de base. De plus, dans ce travail, nous n'avons pas tenu compte de la gestion des données dérivées qui dépendent d'autres données dérivées car cela pose une nouvelle problématique qui est celle de l'absence de période de mise à jour de certaines données relatives (celles dont dépendent les données dérivées et qui sont elles-même des données dérivées temps réel).

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.

[Adelberg , 1996] Adelberg, B., Kao, B., and Garcia-Molina, H. (1996). Database support for efficiently maintaining derived data. In Extending Database Technology, pages 223–240.

[Ahmed , 2000] Ahmed, Q. and Vrbsky, S. (2000). Triggered updates for temporal consistency in real-time databases. International Journal of Time-Critical Computing System, 19(3) :209–243.

[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'2003), Taiwan.

[Amirijoo , 2006] Amirijoo, M., Hansson, J., and Son, S. H. (2006). Specification and management of qos in real-time databases supporting imprecise computations. *IEEE Transactions* on *Computers*, 55(3).

[Bouazizi , 2006] Bouazizi, E., Duvallet, C., and Sadeg, B. (2006). Une nouvelle approche pour la gestion de la QdS dans les SGBD temps réel. In Proceedings of INFORSID'2006, Hammamet, Tunisie.

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

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

[Gustafsson , 2004] Gustafsson, T. and Hansson, J. (2004). Dynamic on-demand updating of data in real-time database systems. In Proceedings of ACM SAC 2004 - Symposium on Applied Computing - track on Embedded Systems : Applications, Solutions, and Techniques.

[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 , 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.

[Ramamritham, 1993] Ramamritham, K. (1993). Real-time databases. In Proceedings of the 14^{th} International VLDB Conference.

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

[Song , 1990] Song, X. and Liu, J. (1990). Performance of multiversion concurrency control algorithms in maintaining temporal consistency. In Proceedings of the Fourteenth Annual International Computer Software and Application Conference, pages 132–139.

[Xiong , 1997] Xiong, M., Sivasankaran, R., Stankovic, J., Ramamritham, K., and Towsley, D. (1997). Scheduling access to temporal data in real-time databases, pages 167–191. Kluwer Academic Publishers.