Prise en compte des données dérivées dans les SGBD temps réel

Claude Duvallet

LITIS, Faculté des Sciences et Techniques
25 rue Philippe Lebon, BP 540,
F-76058 LE HAVRE CEDEX
Courriel : Claude.Duvallet@univ-lehavre.fr

Résumé. Parmi les applications informatiques, nous considérons plus particulièrement celles qui manipulent des données en quantités importantes, ces données et les traitements qui les accompagnent pouvant être soumis à des contraintes temporelles. Depuis une trentaine d'années sont apparues les SGBD temps réel qui permettent de gérer des données temps réel tout en tenant compte des contraintes temporelles des transactions (traitements effectués sur les données temps réel). De nombreux travaux ont été effectués pour concevoir des architectures de SGBD temps réel qui soient robustes face aux périodes de surcharge dues à l'imprévisibilité des demandes utilisateurs. Néanmoins, dans ces travaux, il n'avait presque jamais été tenu compte des données dérivées temps réel. Après avoir proposé une première architecture, nous étendons nos travaux sur la gestion des données dérivées temps réel en tenant compte des données dérivées temps réel qui utilisent elles-mêmes d'autres données dérivées.

1 Introduction

De nombreuses applications temps réel nécessitent d'utiliser des quantités importantes de données temps réel. Ces applications doivent à la fois fournir des résultats respectant des échéances, mais aussi utiliser des données temps réel. L'exploitation des données temps réel dans des applications est mieux prise en charge par les SGBD temps réel [1] [2]. 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. 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.

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 [3] [4] [5]. 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, seul les données de bases (ou sensorielles) avaient été considérées.

Dans des travaux précédents, nous avons introduit l'utilisation des données dérivées dans des architectures avec rétroaction. Pour cela, nous nous sommes appuyés sur les travaux effectués sur les données dérivées [6] et sur les travaux concernant les architectures basées sur une approche qualité de service [7]. Dans ces travaux, nous n'avons pas considéré la gestion des données dérivées qui dépendent d'autres données dérivées ce qui est une limitation pour cette architecture. Dans cet article, notre objectif est donc d'étendre notre architecture afin de prendre en considérations les données dérivées qui sont calculées à partir d'autres données dérivées.

Dans la section 2, nous présentons les travaux effectués sur les données et sur la gestion de qualité de service (QdS) dans les SGBDTR. Nous revenons ensuite sur l'architecture de contrôle par rétroaction pour

la prise en compte des données dérivées (cf. section 3) que nous avions proposée. Puis, dans la section 4, nous décrivons en détails la problématique de notre approche. Puis, nous présentons notre proposition pour la gestion des données dérivées calculées à partir d'autres données dérivées (cf. section 5). Enfin, nous concluons sur un bilan de notre contribution 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) [3] [5] 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] [4] et autorisent la manipulation de résultats imprécis [8].

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 [2]). 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 [4].

Dans les travaux d'Amirijoo [5], la conception des transactions temps réel est basée sur des techniques de calcul imprécis [8]. 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).

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) [5] 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* [3].

Le modèle global : La figure 1 donne une vision générale du modèle souvent utilisé dans ce type de travaux [5]. Nous allons donner une brève description de chacun des composants de ce modèle.

[1] Feedback Control Scheduling Architecture (FCSA).
[2] Data Error.
[3] Maximum Data Error.

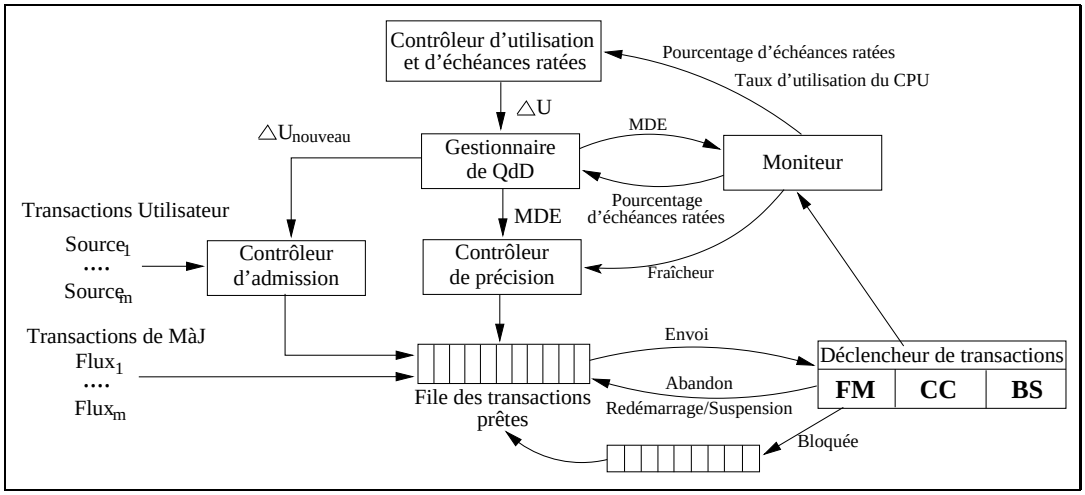


FIG.1. Architecture d'ordonnancement par rétroaction.

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 travaux [3] [5], il s'agit du protocole 2PL-HP (*Two Phase Locking High Priority*) [9] 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*) [10] 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.

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 initia-

[4] administrateur de la base de données
[5] fournis par le DBA.

lement 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.

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

Mise à jour périodique : Dans [11] et [1], 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 non fraîche doit donc attendre la prochaine période de mise à jour. Ceci allonge sa durée d'exécution et augmente le risque de rater son échéance.

Mise à jour déclenchée : Dans [6], les auteurs considèrent que les données dérivées deviennent non fraîches chaque fois qu'une de leurs données de base est actualisée. Par conséquent, dans [12], 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 [6], 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.

Mise à jour à la demande : Pour limiter le nombre de mises à jour des données, Ahmed et Vrbsky [13] 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 « utilisateur » attend la fin de la mise à jour de la donnée ce qui est son principal inconvénient. Ceci augmente le risque que cette transaction rate son échéance.

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 [14] 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.

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 [6].

3 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.

3.1 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 avons intégré 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.

3.2 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.

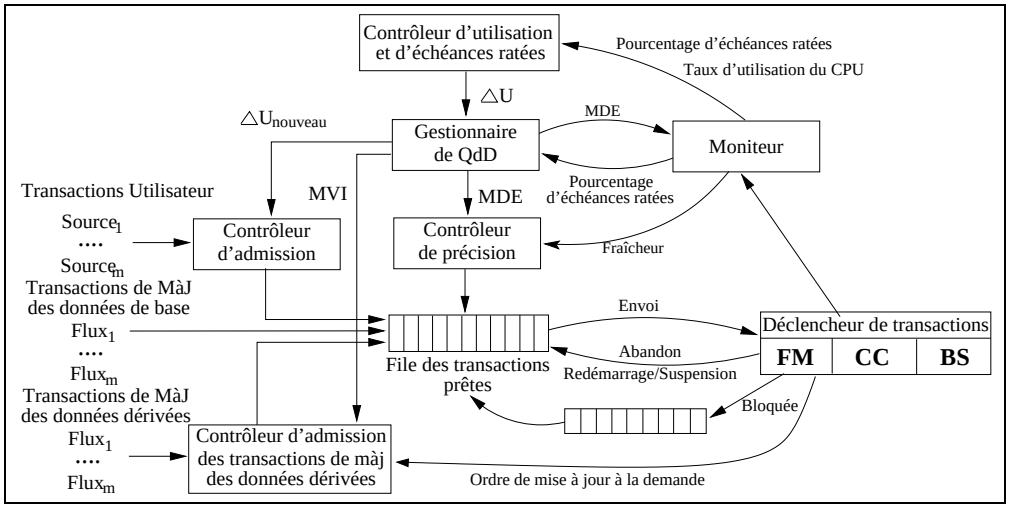


FIG.2. Architecture d'ordonnancement par rétroaction et prise en compte de données dérivées.

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 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 3.1 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.

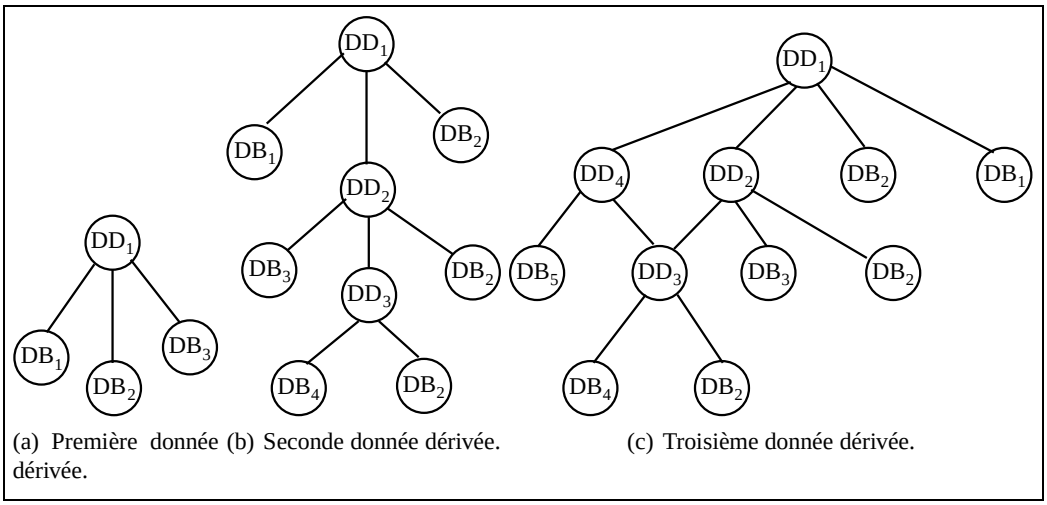
4 Problématique

Dans nos précédents travaux [15], nous avions pris en considération la gestion des données dérivées dans les architectures à contrôle d'ordonnancement par rétroaction. Les données dérivées dont des données qui sont calculées à partir d'une ou plusieurs données sensorielles (dite aussi données de base). Toutes ces données (de base ou dérivées) sont des données temps réel. La principale limitation de nos précédents travaux est l'absence de prise en compte des données dérivées qui sont calculées en utilisant elles-mêmes d'autres données dérivées.

Lorsqu'une donnée dérivée dépend uniquement de données de base pour sa mise à jour alors il suffit de déclencher sa mise à jour à chaque fois qu'une des données de base qu'elle utilise est mise à jour. Comme

nous l'avion vu précédemment dans nos travaux cela peut engendrer une charge de travail très importante pour le système car le nombre de transactions de mise à jour des données dérivées peut croître très rapidement. C'est pourquoi, nous avions proposé une nouvelle architecture de contrôle d'ordonnancement par rétroaction permettant de contrôler la charge engendrée par ces transactions et de l'équilibrer avec les autres transactions.

Maintenant, nous ajoutons un problème supplémentaire qui est la prise en considération des données dérivées qui dépendent pour leur calcul d'autres données dérivées. Nous allons voir en quoi, cela engendre de nouvelles difficultés.



(a) Première donnée dérivée. (b) Seconde donnée dérivée. (c) Troisième donnée dérivée.

FIG.3. Différentes données dérivées.

La figure 4 illustre les différents types de données dérivées qui peuvent exister et va nous permettre de comprendre les différents types de problèmes qui peuvent survenir. Pour la première donnée dérivée (cf. figure 3(a)), la problématique est simple car cette donnée dérivé n'utilise que des données de bases pour effectuer sa mise à jour. Dans nos précédents travaux, nous avions considéré que toutes les données dérivées étaient de ce type-là.

Pour la seconde donnée dérivée (cf. figure 3(b)), nous compliquons un peu la façon de calculer la nouvelle dérivée et nous considérons qu'une donnée dérivée peut être calculée à partir d'autres données dérivées. Cette donnée dérivée (DD_2) utilise les données de base DB_2 et DB_3 ainsi que la donnée dérivée DD_3. La donnée dérivée DD_3 utilise uniquement deux données de base DB_2 et DB_4. Lorsque la donnée de base DB_1 est mise à jour : que se produit-il ? Dans le premier cas (cf. figure 3(a)), il se produit le déclenchement d'une transaction de mise à jour de la donnée dérivée DD_1. Ce cas avait été traité dans nos précédents travaux [15]. Dans le second cas (cf. figure 3(b)), il se produit dans un premier temps le déclenchement de trois transactions de mise à jour pour les données dérivées DD_1, DD_2 et DD_3. Lorsque la donnée dérivée DD_3 est mise à jour, elle déclenche la mise à jour de la donnée dérivée DD_2. Lorsque la donnée dérivée DD_2 est mise à jour, elle déclenche de nouveau la mise à jour de la donnée dérivée DD_1.

Au final, pour la mise à jour d'une unique donnée de base, il y aura eu le déclenchement de 6 transactions de mise à jour des données dérivées : 3 mises à jour de la donnée dérivée DD_1, 2 mises à jour de la donnée dérivée DD_2, et une mise à jour de la donnée dérivée DD_1. Le nombre de transaction de mises à jour des données dérivées est donc relativement important.

La troisième donnée dérivée nous sert à illustrer le cas où dans un même arbre de dérivation servant à calculer une donnée dérivée DD_1, une autre donnée dérivée DD_3 peut-être utilisée deux fois. Lorsque l'une des données DB_2 et DB_4, cela déclenche la mise à jour de la donnée dérivée DD_3 qui à son tour déclenche la mise à jour des données dérivées DD_2 et DD_4. Ces deux dernières vont déclenchées deux fois la mise à jour de DD_1.

5 Une architecture pour la prise en compte des données dérivées utilisant d'autres données dérivées

Notre objectif est de réduire le nombre de transactions qui sont déclenchées lorsque des données dérivées utilisent elles-mêmes d'autres données dérivées tel qu'on a pu le voir sur figure 3(b). À cet effet, nous allons proposer un algorithme de contrôle d'admission des transactions de mise à jour des données qui viendra compléter l'architecture proposée dans nos précédents travaux.

5.1 Algorithme de contrôle d'admission

Pour mettre en œuvre cet algorithme nous allons ajouter un module qui se placera avant le contrôleur d'admission des transactions de mise à jour des données dérivées.

Notre solution est la suivante : lorsqu'une donnée de base (DB_i) est mise à jour, elle provoque le déclenchement d'un groupe de transactions de mise à jour des données dérivées qui l'utilisent. Ce groupe sera noté $Gr(Tr_{MAJ\_DD})$. Ces données dérivées ont pour particularité qu'elles utilisent toutes DB_i pour leur calcul.

Algorithme 1 : Algorithme de contrôle des transactions de mise à jour des données dérivées

```
Gr(Tr_MAJ_DD) : Un groupe de transactions de mise à jour des données dérivées.
début
    pour Tr_MAJ_DD ∈ Gr(Tr_MAJ_DD) : faire
        si Tr_MAJ_DD utilise une ou plusieurs données dérivées alors
            pour chaque donnée dérivée utilisée faire
                si Il existe une transaction de MAJ de cette donnée alors
                    Écarter la transaction Tr_MAJ_DD
                finsi
            finpour
        sinon
            Accepter la transaction Tr_MAJ_DD
        finsi
    finpour
fin
```

Maintenant, appliquons l'algorithme 1 à l'exemple de la figure 3(b) lorsque la donnée de base DB_2 est mise à jour. Nous avons constitution d'un groupe de transactions de mise à jour des données dérivées DD_1, DD_2, DD_3 : $Gr(Tr_{MAJ\_DD})$=$\{Tr_{DD_1}, Tr_{DD_2}, Tr_{DD_3}\}$. On a donc trois transactions de mise à jour des données dérivées dans notre ensemble, on applique notre algorithme :

1. La transaction Tr_{DD_1} utilise la donnée dérivée DD_2 et une transaction de mise à jour Tr_{DD_2} est présente dans $Gr(Tr_{MAJ\_DD})$ alors on écarte Tr_{DD_1}. Ce n'est pas grave car la mise à jour de DD_2 déclenchera une nouvelle transaction Tr_{DD_1}.
2. La transaction Tr_{DD_2} utilise la donnée dérivée DD_3 et une transaction de mise à jour Tr_{DD_3} est présente dans $Gr(Tr_{MAJ\_DD})$ alors on écarte Tr_{DD_2}. Ce n'est pas grave car la mise à jour de DD_3 déclenchera une nouvelle transaction Tr_{DD_2}.
3. La transaction Tr_{DD_3} utilise aucune donnée dérivée, la transaction est donc acceptée.

Après la mise à jour de la donnée DD_3, il y a déclenchement d'une transaction Tr_{DD_2} qui même si elle utilise une donnée dérivée DD_3 est acceptée car aucune transaction Tr_{DD_3} n'est présente. A l'issue de la mise à jour de la donnée DD_2, une transaction Tr_{DD_2} et on applique la même règle.

Maintenant, appliquons l'algorithme 1 à l'exemple de la figure 3(c) lorsque la donnée de base DB_2 est mise à jour. Nous avons constitution d'un groupe de transactions de mise à jour des données dérivées DD_1, DD_2, DD_3 : $Gr(Tr_{MAJ\_DD})$=$\{Tr_{DD_1}, Tr_{DD_2}, Tr_{DD_3}\}$. On a donc trois transactions de mise à jour des données dérivées dans notre ensemble, on applique notre algorithme :

1. La transaction Tr_{DD_1} utilise les données dérivées DD_2 et DD_4 et une transaction de mise à jour Tr_{DD_2} est présente dans $Gr(Tr_{MAJ\_DD})$ alors on écarte Tr_{DD_1}.

2. La transaction Tr_{DD_2} utilise la donnée dérivée DD_3 et une transaction de mise à jour Tr_{DD_3} est présente dans $Gr(Tr_{MAJ\_DD})$ alors on écarte Tr_{DD_2}. Ce n'est pas grave car la mise à jour de DD_3 déclenchera une nouvelle transaction Tr_{DD_2}.
3. La transaction Tr_{DD_3} utilise aucune donnée dérivée, la transaction est donc acceptée.

Lors de la mise à jour de la donnée dérivée DD_3, il y a constitution d'un nouveau groupe de transaction de mise à jour pour les données dérivée DD_2 et DD_1. Suivant notre algorithme ces deux données sont acceptées car elles n'utilisent pas de données dérivées dont les transactions sont présente dans le groupe.

La mise à jour des données DD_2 et DD_4 va déclencher la mise à jour de la donnée dérivée DD_1. Si nous n'y prenons pas garde deux transactions de mise à jour pour une même donnée dérivée vont être déclenchées. Il faut donc ajouter un test qui vérifie que la transaction n'a pas déjà été acceptée.

5.2 Nouvelle architecture d'ordonnancement par rétroaction pour la gestion des données dérivées

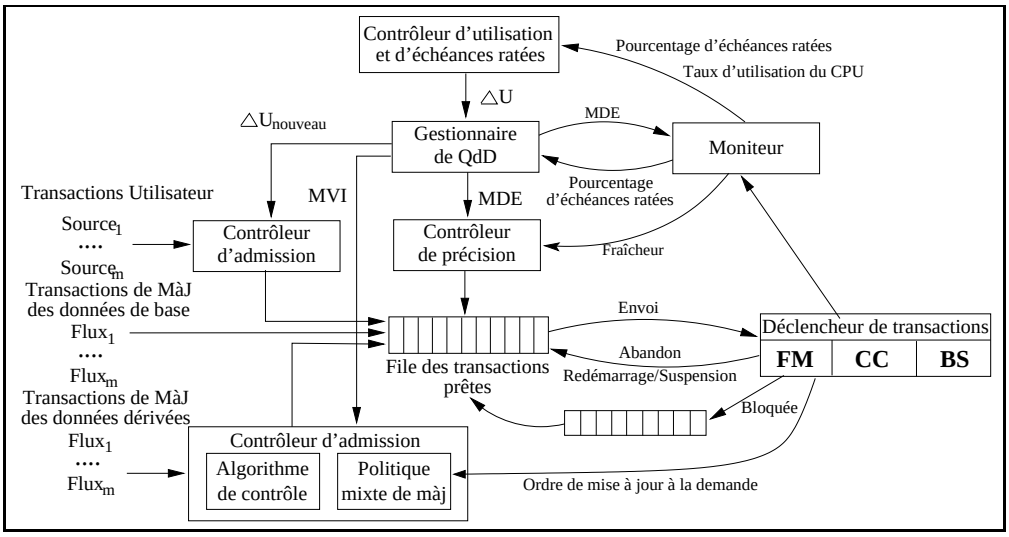


FIG.4. Architecture d'ordonnancement par rétroaction pour la prise en compte des données dérivées utilisant des données de base et d'autres données dérivées.

Pour mettre en œuvre notre nouvel algorithme de contrôle des transactions des transactions de mise à jour des données dérivées, nous avons du modifier l'architecture de contrôle d'ordonnancement par rétroaction pour la prise en compte des données dérivées. Comme nous pouvons le voir dans la figure 4, nous avons modifié le contrôleur d'admission des transactions de mise à jour des données dérivées.

Nous avons ajouté notre algorithme pour le contrôle d'admission des transactions et nous avons gardé la mise en œuvre de la politique mixte de mise à jour des données dérivées. L'algorithme agit sur les transactions dès leur entrée dans le contrôleur d'admission afin d'éviter la redondance des transactions. Dans la seconde partie du contrôleur d'admission, la politique mixte de mise à jour nous sert toujours à contrôler le nombre de transactions exécutées en fonction de la charge du système.

6 Conclusions et perspectives

Dans cet article, nous avons étendu l'architecture de gestion des données dérivées afin de pouvoir prendre en considération les données dérivées qui utilisent d'autres données dérivées et de limiter ainsi le nombre de transactions qui s'exécutent dans le système. Pour cela, nous avons proposé un algorithme permettant d'écarter certaines transactions qui auraient été initialement déclenchées plusieurs fois de façon inutile. Nous avons modifié l'architecture proposée dans [15].

Dans ces architectures qui ont pour objectif de stabiliser le comportement des SGBD temps réel même lors des périodes de surcharge, nous minimisons le nombre de transactions qui s'exécutent et nous maximisons leurs chances de respecter leurs échéances lorsqu'elles ont été acceptées.

À court terme, une évaluation des performances de cette architecture sera effectuée bien qu'elle soit difficilement comparable aux architectures de base qui n'avait jamais tenu compte de la gestion des données dérivées. Notre étude comparative se basera sur l'utilisation des approches basiques qui pourraient être utilisées par rapport à notre approche plus sophistiquée qui tient compte du respect de garantie de la qualité de service.

Plusieurs perspectives peuvent être données à nos travaux. L'une d'entre elles consiste à appliquer ce travail dans d'autres architectures à ordonnancement par rétroaction [16], car ici nous avons travaillé uniquement avec l'architecture de base.

Références

1. Ramamritham, K. : Real-time databases. Journal of Distributed and Parallel Databases **1**(2) (1993) 199–226
2. Duvallet, C., Mammeri, Z., Sadeg, B. : Les SGBD temps réel. Technique et Science Informatiques **18**(5) (1999) 479–517
3. Kang, K., Son, S., Stankovic, J., Abdelzaher, T. : A QoS-Sensitive Approach For Timeliness and Freshness Guarantees in Real-Time Databases. In : Proc. of the Euromicro Conf. on Real-Time System. (2002) 203–212
4. Lu, C. : Feedback Control Real-Time Scheduling. PhD thesis, University of Virginia (2001)
5. Amirijoo, M., Hansson, J., Son, S. : Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation. In : Proc. of the Int. Conf. on Real-Time and Embedded Computing Systems and Applications, Taiwan (2003) 136–157
6. Adelberg, B., Kao, B., Garcia-Molina, H. : Database support for efficiently maintaining derived data. In : Extending Database Technology. (1996) 223–240
7. Amirijoo, M., Hansson, J., Son, S.H. : Specification and management of qos in real-time databases supporting imprecise computations. *IEEE Transactions* on *Computers* **55**(3) (2006) 304–319
8. Liu, J., Shih, W.K., K.-J. Lin, R. Bettati, J.Y.C. : Imprecise Computations. In : Proceedings of the IEEE. Volume 82. (1994) 83–94
9. Abbott, R., Garcia-Molina, H. : Scheduling Real-Time Transactions : A Performance Evaluation. In : Proceedings of the 14^{th} Very Large Data Bases Conference. (1988) 1–12
10. Buttazzo, G. : Hard Real-Time Computing Systems. Kluwer Academic Publishers (1997)
11. Song, X., Liu, J. : Performance of multiversion concurrency control algorithms in maintaining temporal consistency. In : Proceedings of the Fourteenth Annual International Computer Software and Application Conference. (1990) 132–139
12. Gustafsson, T., Hansson, J. : 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. (2004) 846–853
13. Ahmed, Q., Vrbsky, S. : Triggered updates for temporal consistency in real-time databases. International Journal of Time-Critical Computing System **19**(3) (2000) 209–243
14. Xiong, M., Sivasankaran, R., Stankovic, J., Ramamritham, K., Towsley, D. In : Scheduling access to temporal data in real-time databases. Kluwer Academic Publishers (1997) 167–191
15. Katet, M., Duvallet, C., Bouazizi, E., Sadeg, B. : Prise en compte des données dérivées temps réel dans une architecture de contrôle par rétroaction. In : Proceedings of MCSEAI'2006. (2006) 114–119
16. Bouazizi, E., Duvallet, C., Sadeg, B. : Une nouvelle approche pour la gestion de la QdS dans les SGBD temps réel. In : Proceedings of INFORSID'2006, Hammameth, Tunisie (2006) 547–559