# *EXTENSION D'UN PROTOCOLE DE CONTROLE DE CONCURRENCE DES TRANSACTIONS TEMPS REEL*[1]

**Jérôme Haubert,**
Etudiant Doctorant en Informatique
Jerome.Haubert@univ-lehavre.fr, +33 (0)2 32 74 43 84

**Bruno Sadeg, Laurent Amanton,**
Maîtres de conférences en Informatique
Bruno.Sadeg@univ-lehavre.fr, +33 (0)2 32 74 44 05
Laurent.amanton@univ-lehavre.fr, +33 (0)2 32 74 43 19

**Adresse Professionnelle**
LIH, UFR des Sciences et Techniques ★ Université du Havre
★ 25 rue Philippe Lebon ★ BP 540 ★ F-76 058 Le Havre Cedex

**Résumé :** Le protocole SCC (*Speculative Concurrency Control*) est l'un des premiers protocoles de contrôle de concurrence des transactions temps réel. Il utilise la duplication des transactions et les estampilles des transactions pour résoudre les problèmes de conflits. Néanmoins, des problèmes subsistent dans son utilisation. Nous proposons une nouvelle extension de ce protocole pour résoudre ces problèmes et permettre à davantage de transactions de respecter leur échéance.

**Abstract:** The SCC protocol (*Speculative Concurrency Control*) is one of the first concurrency control protocol especially designed for real-time transactions. It is based on the transactions duplication and the use of the transactions timestamp to solve conflicts between transactions. Nevertheless, some problems remain in its principle. We propose a new extension of this protocol to solve these problems and to allow more transactions to meet their deadline.

**Mots clés :** Contrôle de concurrence temps réel, duplication, transactions fantômes, ordonnancement temps réel.

**Key words :** Real-time concurrency control, transaction duplication, shadow transaction, real-time scheduling.

[1]Ce travail est subventionné par le Ministère Français de la Recherche dans le cadre d'une ACI-JC (n°1055)

# Extension d'un protocole de contrôle de concurrence des transactions temps réel

Le protocole SCC (*Speculative Concurrency Control*) est l'un des premiers protocoles de contrôle de concurrence des transactions temps réel. Il utilise la duplication des transactions et les estampilles des transactions pour résoudre les problèmes de conflits. Néanmoins, des problèmes subsistent dans son utilisation. Nous proposons une nouvelle extension de ce protocole pour résoudre ces problèmes et permettre à davantage de transactions de respecter leur échéance.

## 1 - INTRODUCTION

Pour garantir la cohérence de la base de données, un gestionnaire de transactions doit résoudre les problèmes de conflits d'accès aux données. Un conflit apparaît lorsque deux transactions non encore validées désirent accéder à une même donnée avec des opérations incompatibles (écriture-lecture par exemple). Dans un Système de Gestion de Base de Données (SGBD), les problèmes de conflits sont résolus grâce à des protocoles de contrôle de concurrence (Bernstein, 1987). Ces protocoles peuvent être divisés en deux principaux groupes utilisant des méthodes duales. La première méthode, dite pessimiste, utilise des verrous pour empêcher les conflits potentiels : l'opération de lecture ou d'écriture est validée avant l'accès à la donnée. Au contraire, la seconde, dite optimiste, laisse les transactions s'exécuter en concurrence et ne vérifie la présence de conflits qu'une fois la phase de validation atteinte : les transactions en conflits sont alors abandonnées puis redémarrées selon certaines hypothèses. La section 2.2 présente plus en détail les méthodes pessimistes et optimistes pour le contrôle de concurrence.

Pour les SGBD Temps Réel (SGBDTR), le problème est plus compliqué : le SGBDTR doit respecter non seulement les contraintes d'intégrité de la base mais aussi les contraintes temporelles individuelles des transactions qui s'expriment par l'attribution d'une échéance à chaque transaction. Les transactions ne sont correctes que si elles sont validées avant leur échéance (Ramamritham, 1993). On distingue trois types de transactions temps réel selon l'importance attribuée à leur échéance (Duvallet, 1999) :

1. les transactions à échéances strictes critiques (*hard*) : une transaction qui rate son échéance peut avoir des conséquences graves sur le système ou sur l'environnement contrôlé ;
2. les transactions à échéances strictes non critiques (*firm*) : si une transaction rate son échéance, elle devient inutile pour le système. Elle est donc simplement ignorée et abandonnée ;
3. les transactions à échéances non strictes (*soft*) : si une transaction rate son échéance, le système ne l'abandonne pas immédiatement car elle peut avoir une certaine utilité pendant un certain temps encore après l'expiration de son échéance, mais la qualité de service qu'elle offre est moindre.

Dans ce contexte, des adaptations des protocoles de contrôle de concurrence existant dans les SGBD non temps réel sont nécessaires pour tenir compte des échéances des transactions. De nombreux protocoles ont ainsi vu le jour (Abbott, 1988, Haritsa, 1992). Il s'agit généralement d'ajouter aux protocoles classiques un ordonnancement temps réel basé sur les échéances des transactions e.g. le protocole 2PL-HP[2] (Abbott, 1988).

Des études de performances ont montré que suivant le type de transactions temps réel utilisé, les protocoles de contrôle de concurrence pessimistes et optimistes ne fournissent pas les mêmes résultats. Pour les transactions de type *firm*, les protocoles optimistes offrent de meilleures performances grâce à leurs propriétés : non-bloquants et libres de tout interblocage (Haritsa, 1990). Mais ces résultats deviennent discutables lorsque les transactions sont de type *soft* (Huang, 1990) : le redémarrage des transactions et la possibilité d'exécution des transactions après échéances entraînent rapidement une surcharge du système.

Dans ce papier, nous nous intéressons à un protocole de contrôle de concurrence temps réel conçu spécialement pour les SGBDTR : le protocole SCC (*Speculative Concurrency Control*) proposé par Bestavros (1992). Ce protocole combine les avantages des protocoles pessimistes et optimistes puisqu'il détecte les conflits dès leur apparition (méthode pessimiste) mais il laisse les transactions s'exécuter en concurrence (méthode optimiste). Le protocole SCC est particulièrement adapté aux SGBDTR dans le sens où il réduit l'impact négatif du blocage et du redémarrage qui sont les inconvénients principaux des

[2] *Two Phase Locking High Priority*.

méthodes pessimistes et optimistes respectivement (Bestavros, 1992).
Des études ont montré que les performances de ce protocole sont meilleures que celles des méthodes pessimistes et optimistes classiques (Bestavros, 1993), même si les premières versions du protocole SCC ne prenaient pas en compte les échéances des transactions. Ensuite, une extension du protocole prenant en compte les échéances et la criticité des transactions a été proposée (Bestavros, 1995).
Dans ce papier, nous proposons une nouvelle extension du protocole SCC permettant de résoudre les deux principaux problèmes qui subsistent dans son utilisation. (1) Nous proposons une autre résolution des conflits de type w-w (*write-write*) et (2) nous proposons un ordonnancement temps réel des transactions pour augmenter les chances des transactions de respecter leur échéance.
Ce document est structuré de la façon suivante. La section 2 expose en détail le protocole SCC (et une de ses extensions : *value-cognizant SCC*) et montre les principaux problèmes de ces protocoles. La section 3 présente notre nouvelle méthode de résolution de conflits. Cette nouvelle méthode est discutée en section 5. Enfin, la section 6 conclut ce travail en donnant quelques pistes de recherche.

## 2 - LE PROTOCOLE SCC

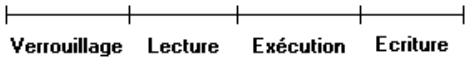
L'intérêt principal du protocole SCC proposé par Bestavros (1992) est qu'il s'agit d'un protocole conçu spécialement pour les SGBDTR. Ses performances résident dans le fait qu'il combine les avantages des méthodes pessimistes et optimistes tout en se débarrassant de leurs principaux inconvénients.

### 2.1 - Notations

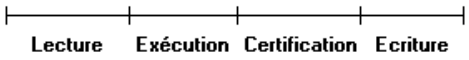
Dans la suite de ce papier, nous noterons $T_i$ les transactions qui s'exécutent en concurrence. Chaque transaction peut être dupliquée et la transaction fantôme de $T_i$ est notée $T_i'$. Les opérations effectuées par les transactions sont *S* (*Start*) pour démarrer, *Rx* (*Read x*) et *Wx* (*Write x*) pour lire et mettre à jour la donnée *x*. *C* et *A* sont utilisés lorsque la transaction est validée (commit) ou abandonnée respectivement.

### 2.2 – Méthodes pessimistes et méthodes optimistes

Les méthodes pessimistes empêchent les conflits avant même qu'ils n'apparaissent en posant des verrous sur les données avant tout accès (figure 1-a). Quand une transaction désire accéder à une donnée, elle doit d'abord obtenir un verrou sur cette donnée. Un verrou peut être posé sur une donnée si et seulement si la donnée n'est pas déjà verrouillée par une autre transaction avec un mode incompatible (lecture et écriture par exemple). Si le mode est incompatible, la transaction requérant le verrou doit attendre que la donnée soit libérée. L'avantage de cette méthode est que les conflits sont détectés dès leur apparition, ce qui permet de garantir la validité de tous les accès aux données. Dans un contexte temps réel, ce type de protocoles possède l'inconvénient majeur d'être un protocole bloquant : les transactions peuvent attendre un verrou pendant un temps indéterminé.
Les méthodes optimistes, de leur coté, laissent les transactions s'exécuter en concurrence et ne vérifient l'apparition de conflits qu'une fois la phase de validation atteinte (figure 1-b). En fait, quand une transaction désire effectuer son opération de *commit*, elle doit vérifier qu'aucun conflit n'est apparu. On utilise pour cela une méthode de certification (Kung, 1981). Les conflits sont détectés soit avec les transactions en cours d'exécution, c'est la certification en avant, soit avec les transactions déjà validées, c'est la certification en arrière. Il est clair que la certification en arrière est difficilement applicable dans un contexte temps réel puisque seule la transaction validante peut être abandonnée en cas de conflit. En revanche pour la certification en avant, on peut utiliser les priorités des transactions pour déterminer le blocage ou l'abandon éventuel de certaines transactions. On peut citer comme exemple le protocole OCC-BC[3] proposé par Haritsa et al. (1992). L'avantage de ce type de méthodes est l'absence de blocage dû aux verrous et l'exécution des transactions en concurrence. Par contre, le redémarrage des transactions peut entraîner une surcharge du système qui peut, à son tour, empêcher des transactions de se terminer avant échéance.




a) *Exécution pessimiste*.




b) *Exécution optimiste*.

**Figure 1 :** Différentes phases des exécutions des transactions selon le type de protocole.

En ce qui concerne le protocole SCC, les conflits sont détectés dès leur apparition comme pour les méthodes pessimistes mais les transactions continuent à s'exécuter en concurrence comme dans les méthodes optimistes.

[3] *Optimistic Concurrency Control with Broadcast Commit*

### 2.3 - Principe du protocole SCC

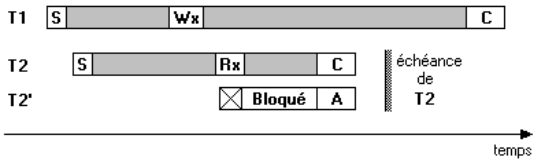
Le protocole SCC a été proposé pour résoudre les problèmes de conflits de type r-w et w-r. Les conflits de type w-w sont résolus par la méthode TWR[4] (Bernstein, 1987, Bestavros, 1994). Cette méthode est basée sur les estampilles des transactions. L'hypothèse est la suivante : seule l'opération d'écriture de la transaction de plus grande estampille (la transaction la plus jeune) sera visible. Avec cette hypothèse, lorsqu'une transaction $T_1$ désire écrire sur une donnée alors qu'une transaction plus jeune $T_2$ a déjà écrit sur cette donnée, l'écriture de $T_1$ est simplement ignorée.

En ce qui concerne les autres types de conflits (r-w et w-r), le protocole SCC propose de dupliquer la transaction en lecture dès qu'un conflit est détecté. La copie de la transaction principale est appelée transaction "fantôme" (ou *shadow transaction*). On a alors deux transactions identiques à un détail près, la transaction principale continue de s'exécuter avec l'*image avant* de la donnée alors que la transaction fantôme reste bloquée avec l'*image après* de la donnée. Deux scénarios peuvent se produire :

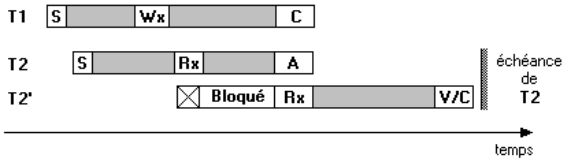
1. La transaction qui accède à la donnée en lecture ($T_2$) se termine et valide avant que la transaction en écriture ($T_1$) ne soit terminée : la transaction fantôme ($T_2$') est alors simplement abandonnée (figure 2-a).
2. $T_1$ se termine et valide avant $T_2$ : $T_2$ doit être abandonnée. Sa transaction fantôme est alors libérée et s'exécute avec l'*image après* de la donnée (figure 2-b). La transaction est alors redémarrée depuis le point de conflit et non depuis le début de la transaction ce qui augmente ses chances de se terminer avant échéance.

Ces exemples simples montrent l'exécution de deux transactions. Mais, avec le protocole SCC de base, une transaction fantôme est créée à chaque apparition de conflit. En limitant le nombre de transactions fantômes par transaction principale, on obtient une classe de protocole SCC-*k*S (*k-shadow* SCC) où *k* désigne le nombre de transactions dupliquées autorisées par transaction, i.e. *k*-1 transactions fantômes (Bestavros, 1992). Parmi cette classe de protocoles, le protocole le plus étudié est le SCC-2S dans lequel on autorise une transaction fantôme par transaction principale. Dans ce cas, les conflits de type r-w et w-r doivent être considérés séparément :

1. Si une transaction $T_2$ désire accéder à une donnée en lecture alors que cette donnée est déjà accédée en écriture par une autre transaction $T_1$ (figures 2-a et 2-b), alors $T_2$ est dupliquée et la transaction fantôme est bloquée. La transaction fantôme sera débloquée si et seulement si la transaction principale doit être abandonnée pour résoudre un conflit.
2. Si $T_1$ désire accéder à une donnée en écriture alors que $T_2$ accède déjà à la donnée en lecture, la transaction fantôme $T_2$' de $T_2$ doit être reconsidérée. Si le point de conflit courant est placé dans la transaction fantôme (figure 3), celle-ci doit être redémarrée depuis le début et est bloquée au nouveau point de conflit. Sinon, la transaction fantôme reste inchangée : le nouveau point de conflit sera pris en compte lorsque $T_2$' sera libérée. Si $T_2$ n'a pas de transaction fantôme, $T_2$' est créée et elle ré-exécute la transaction depuis le début jusqu'au point de conflit.




a) *La transaction en lecture valide la première*.




b) *La transaction en écriture valide la première (promotion de la transaction fantôme)*.

**Figure 2 :** Le protocole SCC.

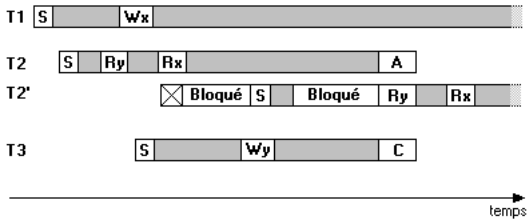



**Figure 3 :** Redémarrage et promotion de la transaction fantôme

Il est à noter que lorsque la transaction fantôme est libérée, elle devient transaction principale et peut être elle-même dupliquée.

### 2.4 - Une extension de SCC : *Value-cognizant SCC*

Bestavros et al. (1995) ont ensuite proposé une extension du protocole SCC de base qui prend en compte les échéances et la criticité des transactions. Ces paramètres permettent de calculer un coefficient de pénalité (*penalty gradient*). Les coefficients des transactions sont utilisés lors de la phase de validation

[4] *Thomas Write Rule*

pour déterminer dans quelle mesure l'opération de *commit* peut être différée. Par exemple, considérons la figure 4 où la situation est initialement la même que sur la figure 2-b. Si on diffère l'opération de *commit* de $T_1$, on peut laisser la transaction $T_2$ s'exécuter jusqu'à validation et ensuite valider $T_1$. Le conflit se résout alors de lui-même. Les coefficients des transactions permettent donc de définir des heuristiques pour différer l'opération de *commit* des transactions si nécessaire.

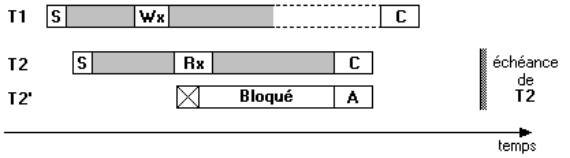



**Figure 4 :** Exemple d'exécution du protocole *Value-cognizant SCC*.

Les détails et les performances de ce type de protocoles peuvent être trouvés dans (Bestavros, 1995), mais nous pouvons simplement dire qu'il s'applique aux transactions à échéances non strictes (*soft*) et que les coefficients ne sont utilisés qu'au moment de la validation. Nous pensons que les échéances des transactions peuvent être utilisées dès la détection des conflits pour augmenter les chances des transactions de respecter leur échéance.

### 2.5 - Discussion de la méthode spéculative

Dans cette section, nous discutons l'utilisation du protocole SCC pour le contrôle de concurrence des transactions temps réel. Pour cela, nous allons différencier les types de conflits : les conflits de type r-w et w-r sont résolus de manière similaire alors que les conflits de type w-w sont résolus séparément.

Dans un premier temps, considérons les conflits de type w-w. Ils sont résolus par la méthode TWR qui utilise les estampilles des transactions pour éventuellement ignorer des opérations d'écriture (cf. section 2.1). Dans un contexte temps réel, l'hypothèse sous-jacente à la méthode TWR est difficilement applicable. En effet, si les estampilles des transactions représentent leur date d'arrivée dans le système, une transaction plus ancienne ($T_1$) avec un temps d'exécution très long et loin de son échéance peut être en conflit avec une transaction plus jeune ($T_2$) mais dont l'échéance est plus proche. Dans ce cas, TWR peut ignorer l'opération d'écriture de $T_1$. Le problème n'est pas résolu de manière correcte puisque $T_2$ a plus de chance de se terminer (valider ou abandonner) avant $T_1$ et cette dernière doit alors réécrire sur la donnée.

Autrement dit, la méthode TWR perd la mise à jour de $T_1$. Ceci est difficilement concevable dans les SGBDTR. En effet, les résultats des transactions temps réel sont importants dès qu'elles ont effectué leur opération de commit (Ramamritham, 1993). Ainsi, la résolution des conflits w-w en utilisant les estampilles des transactions ne semble pas adaptée aux SGBDTR : TWR souffre du problème de perte de mise à jour.

Considérons maintenant les autres types de conflits (r-w et w-r). Avec le protocole SCC, si l'échéance de $T_1$ est plus proche que celle de $T_2$, on peut aboutir à la situation présentée en figure 5 : la transaction $T_2$ ne peut pas respecter son échéance parce que l'opération de commit de $T_1$ ne peut pas être différée et le temps restant à $T_2$' pour s'exécuter n'est pas suffisant. Si les échéances des transactions sont de type *soft,* seule la qualité des résultats de $T_2$ va diminuer (Bestavros, 1995, Duvallet, 1999). Mais si les transactions sont de type *firm* (voire *hard*), $T_2$ ne fournit aucun résultat et peut engendrer des conséquences plus ou moins graves.

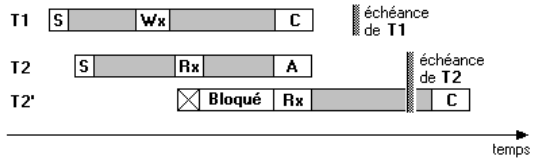



**Figure 5 :** Limites du protocole SCC.

En fait, la résolution des conflits du protocole SCC de base ne tient pas compte des priorités des transactions (échéance et/ou criticité). Une extension de ce protocole a été proposée mais les priorités des transactions ne sont prises en considération qu'au moment de la validation (*value-cognizant SCC*). Nous pensons que les priorités des transactions doivent être considérées dès la détection des conflits pour donner plus de chances aux transactions de respecter leur échéance.

Dans la section suivante, nous proposons une nouvelle extension du protocole SCC, nommée ESCC (*Extended SCC*) qui résout les deux problèmes que nous venons d'exposer (1) en utilisant une nouvelle technique de résolution des conflits w-w et (2) en ordonnançant les transactions temps réel dès la détection d'un conflit.

## 3. EXTENSION DU PROTOCOLE SCC

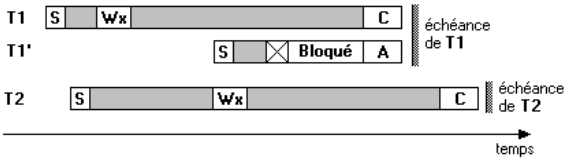
### 3.1 - Conflits w-w

Nous avons montré dans la section précédente que la résolution des conflits basée sur les estampilles des transactions fournit des résultats discutables dans les SGBDTR. Dans ce cas, la méthode TWR, souffrant du problème de perte de mise à jour, n'est pas adaptée. Pour les conflits w-w, nous proposons d'utiliser la même notion que pour les autres types de

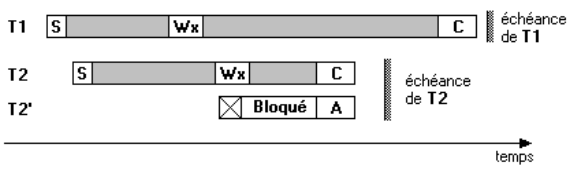
conflits : la duplication des transactions. De cette façon, une transaction fantôme est créée dès l'apparition d'un conflit. Le nombre de transactions fantômes par transaction peut être limité de la même manière que pour le protocole SCC-2S (Bestavros, 1992).

Examinons la possibilité d'appliquer la duplication des transactions aux conflits w-w. Considérons la figure 6. Lorsqu'un conflit de type w-w apparaît, on duplique la transaction dont l'échéance est la plus proche. En effet, si cette dernière doit être redémarrée, alors elle aura moins de chance de respecter son échéance. Deux cas se présentent alors :

- Soit le point de conflit se situe avant le début de la transaction fantôme (figure 6-a) : la transaction fantôme doit être redémarrée et se bloque au nouveau point de conflit.
- Soit le point de conflit est l'instant courant (figure 6-b) : il suffit de bloquer la transaction fantôme à l'instant courant.




a) *$T_1$ est la plus prioritaire*




b) *$T_2$ est la plus prioritaire*

**Figure 6 :** ESCC appliqué aux conflits w-w.

Ainsi, la duplication des transactions peut s'appliquer à tous les types de conflits.

### 3.2 – Autres conflits

La figure 5 montre les limites du protocole SCC pour résoudre les problèmes de conflits r-w et w-r entre des transactions concurrentes. En effet, une seule des transactions se termine avant échéance. Existe-t-il une méthode permettant aux deux transactions de se terminer avant leur échéance ?

Pour cela, nous avons besoin de faire une nouvelle hypothèse : une transaction peut lire les résultats d'une autre transaction non encore validée. Nous relaxons ainsi l'isolation, l'une des propriétés ACID[5] des transactions. Nous discuterons cette hypothèse dans la section suivante. Ainsi, si $T_1$ et $T_2$ sont deux transactions accédant à une même donnée respectivement en écriture et en lecture, notons $d_1$ et $d_2$ leur échéance respective. Un conflit entre ces transactions implique la création d'une transaction fantôme pour $T_2$ : $T_2$'. Le protocole ESCC propose d'ordonnancer $T_2$ et $T_2$' en fonction des échéances de $T_1$ et $T_2$ de la façon suivante :

1. Si $d_1 > d_2$, alors $T_1$ a de grandes chances de se terminer avant $T_2$ (figure 2-a). L'écriture de $T_1$ n'a alors pas d'impact sur le déroulement de $T_2$. De manière optimiste, on peut donc laisser $T_2$ se dérouler comme dans le protocole SCC de base. Dans le cas où $T_1$ se termine avant $T_2$, on peut utiliser le protocole *value-cognizant SCC* pour différer l'opération de commit de $T_1$ et laisser $T_2$ se terminer en premier.
2. Si $d_1 < d_2$, alors on peut penser raisonnablement que $T_1$ va se terminer avant $T_2$ et les modifications qu'elle apporte devront être considérées par $T_2$. L'exécution de SCC peut mener à la situation de la figure 5. C'est pourquoi on choisit ici de bloquer la transaction principale et d'exécuter la transaction fantôme (figure 7). Autrement dit, la transaction en lecture s'exécute avec l'*image après* de la donnée. Les deux transactions ont ainsi plus de chances de se terminer et de valider avant leur échéance.

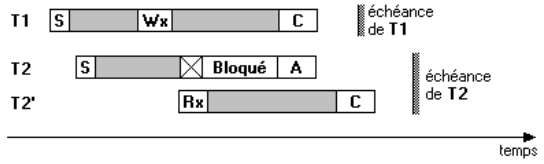



**Figure 7 :** ESCC appliqué aux conflits r-w et w-r.

En résumé, le protocole ESCC propose un ordonnancement des transactions (principales et fantômes) en fonction des échéances des transactions en conflit. Le protocole ESCC, en combinant la duplication des transactions avec un ordonnancement temps réel des transactions, permet donc à davantage de transactions de respecter leur échéance.

## 4 - DISCUSSION SUR LA METHODE

Comme nous l'avons signalé dans la section précédente, le protocole ESCC proposé ne respecte pas la propriété d'isolation des transactions. Cette propriété précise que les résultats d'une transaction ne sont visibles des autres transactions qu'une fois la transaction terminée. Dans un contexte temps réel, cette propriété peut être relaxée (Ramamritham, 1993) dans le sens où l'isolation peut différer l'exécution de certaines transactions et les empêcher de respecter leur échéance. Si on relaxe la propriété d'isolation, alors les transactions peuvent accéder à des données

[5]Atomicité, Cohérence, Isolation, Durabilité

non encore validées et peuvent ainsi fournir des résultats même partiels avant échéance. Autrement dit, dans un contexte temps réel, il peut être plus important d'obtenir des résultats partiels avant échéance plutôt que des résultats complets après échéance (Ramamritham, 1993, Gupta, 1996).
Le fait de relaxer l'isolation des transactions oblige à tenir compte de la validité des données. Lorsqu'une transaction $T_1$ accède à des données déjà utilisées par une autre transaction $T_2$, la validation de $T_1$ est alors dépendante de celle de $T_2$. Si $T_2$ se termine correctement et valide, alors $T_1$ peut continuer normalement son exécution. Mais, dans le cas où $T_2$ doit être abandonnée, $T_1$ doit également être abandonnée puisqu'elle a manipulé des données devenues invalides. Des heuristiques sont utilisées pour limiter les conditions d'accès aux données non encore validées (Gupta, 1996). Par exemple, on peut éviter l'abandon en cascade en ne permettant pas à une transaction accédant déjà à des données non validées d'autoriser à nouveau l'accès aux données qu'elle manipule. On peut également empêcher une transaction proche de son échéance de prêter des données pour limiter le nombre de redémarrage des transactions.
Une autre discussion que l'on peut faire sur le protocole ESCC est le nombre de transactions qu'implique la duplication. En effet, avec *N* transactions et en se limitant à une transaction fantôme par transaction principale, 2*N* transactions peuvent coexister. Mais, comme le montre la discussion précédente, la plupart du temps, une transaction et sa transaction fantôme ne s'exécutent pas en même temps. La transaction fantôme est souvent bloquée pendant que la transaction principale s'exécute sauf dans le cas où la transaction fantôme est redémarrée. On peut donc penser que la surcharge entraînée par cette méthode est négligeable par rapport aux performances en termes de respect des échéances des transactions.
Enfin, il est à noter que la méthode de résolution des conflits w-w du protocole ESCC est plus restrictive que la méthode TWR. En effet, avec le protocole ESCC, toutes les opérations d'écritures doivent être effectivement réalisées (aucune opération ne peut être ignorée). Cette restriction va probablement limiter les performances du protocole ESCC, mais rappelons que cette nouvelle méthode ne souffre pas du problème de perte de mises à jour.

## 5 - CONCLUSION

Dans cet article, nous nous sommes intéressés à un protocole de contrôle de concurrence (SCC) proposé par Bestavros et al. (1992, 1993, 1995) et qui est spécifique aux transactions temps réel. Notre étude de ce protocole montre que malgré ses bonnes performances dans les SGBDTR, il possède quelques inconvénients, notamment pour les conflits de type w-w ou encore dans la prise en compte des échéances des transactions. En effet, il semble que le protocole SCC soit plutôt adapté aux transactions à échéances non strictes (*soft*), mais ses performances décroissent pour les transactions à échéances strictes (critiques ou non). Nous avons donc proposé quelques améliorations pour permettre au protocole SCC de pallier ces problèmes et permettre à davantage de transactions de se terminer avant échéance.
Les améliorations que nous proposons sont basées sur une nouvelle méthode de résolution des conflits de type w-w et la prise en compte des échéances des transactions dès la détection d'un conflit. Le protocole SCC amélioré est appelé ESCC. En fait, le protocole ESCC propose d'utiliser à la fois la duplication des transactions et un ordonnancement temps réel des transactions basé sur leur échéance pour résoudre tous les types de conflits. Des simulations sont actuellement en cours pour évaluer les performances du protocole ESCC par rapport au protocole SCC mais aussi pour évaluer la limitation apportée par la résolution de conflits w-w du protocole ESCC par rapport à la méthode TWR.
Nos travaux portent sur les SGBD distribués temps réel. La répartition des données implique de gérer à la fois le contrôle de concurrence localement sur chaque site mais aussi la validation globale des transactions distribuées. Le protocole ESCC comme le protocole SCC peuvent s'utiliser localement sur chaque site pour le contrôle de concurrence mais il faut tenir compte de la validation globale des transactions. Parmi les protocoles de validation, certains ne sont pas adaptés au contexte temps réel soit parce qu'ils sont bloquants soit à cause du grand nombre de messages échangés entre le coordinateur et les participants (Haritsa, 2000). La validation des transactions distribuées temps réel reste un domaine de recherche ouvert dans les SGBDTR distribués.

## BILBIOGRAPHIE


Abbott R.K., Garcia-Molina, H. (1988), "Scheduling Real-Time Transactions: A Performance Evaluation". In *14th Very Large Data Bases Conference*, pp 1-12, Loas Angeles, California.

Bernstein, P., Hadzilacos, V., Goodman, N. (1987), "Concurrency Control and Recovery in Database Systems". Addison-Wesley.

Bestavros, A. (1992), "Speculative Concurrency Control". Technical Report TR-16-92, Boston University, Boston, MA.

Bestavros, A., Braoudakis, S., Panagos, E. (1993), "Performance Evaluation of Two-Shadow Speculative Concurrency Control". Technical Report 1993-001, Boston University, Boston, MA.

Bestavros, A., Braoudakis S. (1994), "Timeliness via Speculation for Real-Time Databases". In *14th IEEE Real-Time System Symposium*, Puerto Rico.

Bestavros, A., Braoudakis, S. (1995), "Value-cognizant Speculative Concurrency Control". In *21st VLDB Conference*, Zurich, Switzerland.

Duvallet, C., Mammeri, Z., Sadeg, B. (1999), "Les SGBD Temps Réel". *Technique et Science Informatiques*, vol 18, n° 5, pp 479–517.

Gupta, R., Haritsa, J. (1996) "Commit Processing in Distributed Real-Time Database Systems". In *Natl. Conference on Software for Real-Time Systems*, Cochin, India.

Haritsa, J., Carey, M., Livny, M. (1990), "On Being Optimistic about Real-Time Constraints". In 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems.

Haritsa, J. Carey, M., Livny, M. (1992), "Data Acces Scheduling in Firm Real-Time Database Systems". *Real-Time Systems*, vol 4, n°3, pp 203-241.

Haritsa, J., Ramamritham, K., Gupta R. (2000), "Real-Time Commit Processing". In *Real-Time Database Systems*, pp 227–243. Kluwer Academic Publishers.

Huang, J., Stankovic, J. (1990), "Concurrency Control in Real-Time Database System: Optimistic Scheme vs. Two-Phase Locking". Technical Report UM-CS-1990-066, University of Massachusetts.

Kung, H., Robinson, J. (1981), "On Optimistic Methods for Concurrency Control". In *ACM Trans. on Database System*, vol 6, n° 2, pp 213-226.

Ramamritham, K. (1993), "Real-Time Databases". *Journal of Distributed and Parallel Databases*, vol 1, n° 2, pp 199–226.