APPLICABILITE DU CRITERE D'EPSILON-SERIALISABILITE DANS LES SGBD TEMPS REEL

Emna Bouazizi,
Doctorante en Informatique
Emna.Bouazizi@univ-lehavre.fr, +33 (0) 232 744 384

Bruno Sadeg, Claude Duvallet,
Maîtres de conférences en Informatique
Bruno.Sadeg@univ-lehavre.fr, +33 (0) 232 744 405
Claude.Duvallet@univ-lehavre.fr +33 (0) 232 744 405

Adresse professionnelle
LIH, UFR des Sciences et Techniques ★ Université du Havre
25 rue Philippe Lebon ★ BP 540 ★ F-76058 Le Havre Cedex

Résumé : Le critère de sérialisabilité, reconnu comme le critère de correction des transactions dans les *SGBD (Systèmes de Gestion de Bases de Données)* traditionnels, est difficilement applicable dans un contexte temps réel. Ce critère s'avère trop strict pour l'exécution des transactions et pour l'accès aux données temps réel. L'epsilon-sérialisabilité est un critère de correction qui semble davantage adapté aux *SGBDTR (SGBD Temps Réel)* car il permet de relaxer la sérialisabilité lors du traitement des transactions temps réel au détriment de la cohérence stricte des résultats et des données. Dans beaucoup d'applications actuelles, ceci n'est pas très pénalisant à condition que cette incohérence soit bornée et contrôlée. On utilise pour cela des algorithmes de contrôle de divergence qui s'appuient sur le critère d'epsilon-sérialisabilité. L'objectif de cet article est d'étudier une adaptation du concept de l'epsilon sérialisabilité pour les *SGBDTR*.

Abstract: The serialisability, defined as the standard criterion of the transaction correctness in traditional Database Management Systems (DBMS), is poorly supported in a real-time context. This criterion is proved to be too strict for transaction execution and real-time data management. Epsilon-serialisability is a correctness criterion that seems to be more suited to Real-Time DBMS. It allows to relax the serialisability severity for processing of real time transactions to the detriment of the strict consistency of the results and data. Indeed, the Epsilon-serialisability criterion allows to control the inconsistencies by bounding the imported and exported database inconsistencies. For this purpose, we use divergence control algorithms which ensure epsilon-serialisability. This paper aims to study the adaptation of the epsilon-serialisability concept in the real-time context.

Mots clés : *SGBD* temps réel, sérialisabilité, epsilon-sérialisabilité, contrôle de concurrence, contrôle de divergence.

Keywords: real-time databases, serialisability, epsilon-serialisability, concurrency control, divergency control.

Applicabilité du critère d'epsilon-sérialisabilité dans les SGBD temps réel

1 – INTRODUCTION

Dans les *SGBD*, le concept de transaction est fondamental pour maintenir la cohérence des données. Les transactions effectuant des accès concurrents aux données, doivent être contrôlées afin d'éviter les conflits d'accès. Les protocoles de contrôle de concurrence permettent le respect du critère de correction, appelé *sérialisabilité* (*SR*). Ce critère conduit à n'accepter que les exécutions simultanées de transactions produisant les mêmes résultats qu'une exécution séquentielle de ces mêmes transactions. Vu l'importance de ce concept dans la gestion des transactions, la sérialisabilité a fait l'objet d'un grand nombre de travaux (Bernstein, 1987 ; Ullman, 1980 ; Date, 1985). Le concept de sérialisabilité s'est alors enrichi d'un développement récent pour faire apparaître d'autres types de critères de correction, car il s'avérait trop restrictif pour certaines applications (Ramamritham, 1993 ; Duvallet, 1999).

Avec l'avènement des nouvelles applications, une nouvelle notion est apparue dans les *SGBD*. Il s'agit du temps réel ainsi que la notion de contraintes temporelles qui lui est sous-jacente. Avec l'apparition des contraintes temporelles, les *SGBDTR* doivent non seulement gérer une masse importante de données, mais doivent également offrir des mécanismes pour respecter les contraintes temporelles des transactions et des données. Ces contraintes sont souvent explicitées sous forme d'échéances. Dans un contexte temps réel, la sérialisabilité est généralement trop restrictive (Ramamritham, 1993). D'autres critères de correction ont alors été proposés qui semblent davantage adaptés aux *SGBDTR* (Ramamritham 1993 ; Ramamritham, 1992). Dans cet article, nous présentons le critère d'epsilon-sérialisabilité (*ESR*) qui était proposé pour les *SGBD* classiques et nous étudions son applicabilité dans les *SGBDTR*. Nous prouvons qu'il adoucit la sévérité de la *SR* dans le traitement des transactions temps réel en permettant aux transactions, appelées alors Epsilon-Transactions (*ETs*), d'être entachées d'une incohérence contrôlée et limitée.

Par analogie avec les algorithmes de contrôle de concurrence (*CC*) qui supportent efficacement la *SR*, des méthodes de contrôle de divergence (*CD*) sont proposées pour garantir l'*ESR* des *ETs*. Les méthodes de *CD* sont une extension des méthodes de *CC*. Les algorithmes de *CD* sont similaires aux algorithmes de *CC* correspondants sauf que les méthodes de *CD* autorisent des conflits *non-sérialisables*, sous certaines conditions. Dans cet article, nous appliquons un algorithme de contrôle de divergence qui contrôle les incohérences importées et exportées par les *ETs* sujettes à des contraintes temporelles sous forme d'échéances. Des résultats obtenus lors de simulations ont montré que davantage de transactions temps réel respectent leurs échéances par rapport à l'application du critère de sérialisabilité classique.

La suite de cet article est organisée de la façon suivante. La section 2 définit le concept de *SR* et celui du contrôle de concurrence dans les *SGBD* classiques. La section 3 est consacrée à un rappel sur l'*ESR*, et la section 4 aux *ETs*. La section 5 présente les méthodes de contrôle de divergence. L'applicabilité de l'*ESR* est discutée dans la section 6. Dans la section 7, nous présentons les résultats de simulation de nos travaux et discutons des résultats obtenus. Nous concluons cet article en soulignant l'apport de nos travaux et présentons quelques perspectives de recherche.

2 - SERIALISABILITE ET CONTROLE DE CONCURRENCE

L'application des protocoles de *CC* permet de garantir la *SR* et de résoudre les conflits d'accès aux données. Il est alors nécessaire de commencer par définir le concept du *CC* avant de présenter la *SR*.

2.1 - Contrôle de concurrence

Le contrôle de concurrence a pour fonction d'assurer, à chaque transaction, la propriété d'Isolation, une des propriétés *ACID*[1] des transactions. Cette propriété vise à garantir à chacune des transactions un accès aux objets d'une base de données comme si elle était seule à s'exécuter dans le système. Elle est obtenue par différentes méthodes qui ont pour but de synchroniser les accès aux objets, effectués par des transactions concurrentes. Toutes ces méthodes reposent sur le principe de la sérialisabilité des transactions (Bernstein, 1987).

2.2 - La sérialisabilité

La sérialisabilité consiste à n'accepter que les exécutions simultanées de transactions produisant les mêmes résultats qu'une exécution séquentielle de ces transactions. La sérialisabilité représente aussi le critère de correction généralement accepté pour le contrôle de concurrence des transactions (Kamath, 1993 ; Ramamritham, 1992 ; Ramamritham, 1993).

Une exécution sérialisable est une exécution entrelacée des actions d'un ensemble de transactions {T1, T2,...,Tn}, qui donne globalement

[1]Atomicité, Cohérence, Isolation, Durabilité.

et pour chaque transaction participante le même résultat qu'une exécution en série de T1, T2,...,Tn. La sérialisabilité maintient la cohérence de la base de données. En effet, si la base de données exécute seulement des transactions sérialisables et si l'état initial de cette base de données est cohérent, alors la sérialisabilité garantit un passage vers un autre état cohérent de la base.

La sérialisabilité s'appuie sur la relation de précédence entre transactions qui peut être représentée par un graphe de précédence dont les nœuds représentent les transactions et un arc de T_i vers T_j modélise la relation : « T_i précède T_j » dans l'exécution analysée. Une condition suffisante de sérialisabilité est que le graphe de précédence soit sans circuit (acyclique).

2.3 - Limitation de la sérialisabilité classique

Malgré l'importance des services offerts, la *SR* possède aussi ses limitations. Elle exige que les transactions concurrentes, y compris celles effectuant seulement des opérations de lecture, soient ordonnancées dans un ordre sérialisable. Lorsque le nombre de transactions concurrentes augmente, la quantité de conflits augmente, ce qui provoque l'annulation de certaines transactions et la diminution du niveau d'efficacité. Pour les transactions en *lecture seule* qui peuvent tolérer une certaine quantité d'incohérence, la sévérité de la *SR* peut imposer des limitations sur le débit du système et la qualité de service (*QdS*).

3 - EPSILON-SERIALISABILITE

3.1 - Motivations

Pour atténuer les effets négatifs de la *SR*, la notion d'epsilon-sérialisabilité, notée par la suite *ESR*, a été proposée (Pu, 1991). En effet, l'*ESR* permet d'étendre la notion de cohérence et les utilisateurs peuvent alors bénéficier d'un haut niveau de tolérance aux incohérences. Certaines applications peuvent avoir recours à l'*ESR*. De telles applications doivent pouvoir tolérer une certaine quantité d'incohérence, due à l'imprécision naturelle des données ou à des considérations de performances du système et de disponibilité des données. Elle autorise les traitements asynchrones et par conséquent une grande disponibilité et une meilleure autonomie.

3.2 - Définition

La notion d'*ESR* traite le problème du côté des transactions. À chaque requête, est associée une incohérence exportée vers la base de données et une incohérence importée. Plus précisément les écritures exportent une incohérence vers la base et les lectures en importent. Ces incohérences sont rassemblées dans deux accumulateurs et la transaction peut poursuivre son exécution tant que ces accumulateurs ne dépassent pas une certaine valeur (ε). L'idée générale est que l'*ESR* maintient un nombre d'incohérences à l'intérieur de limites spécifiées par ε. ε est alors la quantité d'incohérence autorisée par l'*ESR*. Lorsque ε tend vers 0, l'*ESR* tend vers la *SR* classique (Ramamritham, 1995 ; Sadeg, 2003).

Soit T un ensemble de transactions qui accèdent à la base de données. Nous désignons dans la suite par $\sum_{t\in T} Inc^{import}(t)$ l'incohérence importée par T, et par $\sum_{t\in T} Inc^{export}(t)$ l'incohérence exportée par T. ε^{import} et ε^{export} sont les limites d'incohérence autorisées par l'*ESR*.

La définition de l'*ESR* est établie en supposant l'existence d'une condition sûre pour l'ensemble de transactions T, notée par :

- $\sum_{t\in T} Inc^{import}(t) \le \varepsilon^{import}$
- $\sum_{t\in T} Inc^{export}(t) \le \varepsilon^{export}$

Cette condition exige que l'incohérence importée (resp. exportée) à la base de données ne dépasse pas la limite d'incohérence autorisée.

L'objectif de l'*ESR* est de contrôler la quantité d'incohérence dans les applications qui manipulent de grandes quantités de données. Cette quantité ε dépend de l'application et elle est spécifiée par l'administrateur de la base de données. Elle est mesurée en termes de fonction '*distance*' définie sur les états de la base de données. Désignons par ***P(conflit(L,E))*** le poids d'un conflit entre la lecture (L) et l'écriture (E), qui est représenté par la fonction '*distance*' tel que :

$$\boldsymbol{P(conflit(L,E)) = distance\ (E_{avant}, E_{après})}$$

C'est-à-dire que le poids du *conflit(L,E)* est la distance entre l'état avant l'écriture (E_{avant}) et l'état après l'écriture ($E_{après}$). ***P(conflit(L,E))*** est alors la quantité d'incohérence potentielle introduite par les transactions à cause des conflits.

Supposons que *x1* et *x2* soient deux états de la donnée *x*, la distance entre ces deux états est la différence entre ces deux valeurs :

$$\boldsymbol{Distance((x1),(x2)) = |x1-x2|}$$

Supposons qu'une opération de lecture sur *x* renvoie la valeur 5, et plus tard, qu'une opération d'écriture modifie *x* par 13. Le poids du conflit de lecture/écriture est 8.

$$\boldsymbol{P(conflit(L,E)) = Distance((5),(13)) = |5-13| = 8}$$

Plus généralement, on peut aussi appliquer la fonction de distance sur l'ensemble des états des données de la base. Supposons que la base de données contienne trois données de type numériques *x, y* et *z*. Un état est représenté par le triplet (*x*, *y*, *z*). On peut alors définir la distance comme suit :

$$\boldsymbol{Distance\ ((x1,y1,z1),(x2,y2,z2)) = |x1-x2| + |y1-y2| + |z1-z2|}$$

Par exemple, la distance entre l'état (100,200,400) et l'état (300,500,400) est 500 (i.e, 200+300+0) qui représente la quantité d'incohérence ε introduite dans la base de données. Deux valeurs doivent être modifiées lors de la transition entre les deux états donc le passage de l'état (100,200,400) à l'état (300,500,400) exige au moins deux opérations d'écriture qui peuvent causer des conflits de type lecture/écriture.

4 - EPSILON-TRANSACTION ET INCOHERENCE DES DONNEES

L'Epsilon-Transaction (*ET*) est une extension des transactions classiques (pour la *SR*) par l'ajout d'une spécification de la quantité d'incohérence. Cette quantité est donnée par quelques mesures sur les opérations de la base de données ou de la fonction distance dans l'espace d'états de cette base de données. Pour chaque Epsilon-Transaction, cette spécification est divisée en deux parties (Pu, 1991 ; Pu, 1992) :

1. une incohérence importée, notée « $\boldsymbol{\varepsilon}_{ET}^{import}$ »
2. une incohérence exportée, notée « $\boldsymbol{\varepsilon}_{ET}^{export}$ »

	$\boldsymbol{\varepsilon}_{ET}^{import}$	$\boldsymbol{\varepsilon}_{ET}^{export}$
T	**= 0**	**= 0**
$\boldsymbol{ET_L}$	**> 0**	**= 0**
$\boldsymbol{ET_E}$	**= 0**	**> 0**

Tableau 1: les différents types d'ETs

Le tableau 1 illustre les différents types d' *ETs* :

- des *ETs* qui sont sérialisables (au sens classique), notées "Transaction" (et par la suite T),
- des *ETs* de lecture, notées $\boldsymbol{ET_L}$,
- des *ETs* de mises à jour, notées $\boldsymbol{ET_E}$.

Si $\boldsymbol{\varepsilon}_{ET}^{import}$ = 0 et $\boldsymbol{\varepsilon}_{ET}^{export}$ = 0, alors l'*ET* est sérialisable. Par conséquent les *ETs* considèrent les transactions classiques comme un cas limite. Lorsque $\boldsymbol{\varepsilon}_{ET}^{import}$ > 0 et $\boldsymbol{\varepsilon}_{ET}^{export}$ = 0, les $\boldsymbol{ET_L}$ peuvent importer une incohérence limitée par $\boldsymbol{\varepsilon}_{ET}^{import}$. De même lorsque $\boldsymbol{\varepsilon}_{ET}^{import}$ = 0 et $\boldsymbol{\varepsilon}_{ET}^{export}$ > 0, les $\boldsymbol{ET_E}$ peuvent exporter une incohérence limitée par $\boldsymbol{\varepsilon}_{ET}^{export}$. Cependant, si les *ETs* importent et exportent des incohérences, elles peuvent alors introduire une nouvelle incohérence illimitée dans la base de données. Ces cas sont résumés dans le tableau 1.

À part l'incohérence des transactions, il faut aussi penser à limiter l'incohérence des données. En effet, chaque donnée a une quantité d'incohérence autorisée qu'elle peut importer de (ou exporter dans) la base de données. L'incohérence importée par une epsilon transaction sera la somme des incohérences des données lues. Et l'incohérence exportée sera la somme des incohérences exportées par les données écrites. Il existe alors une forte dépendance entre l'incohérence des transactions et celle des données, représentée par les deux inéquations suivantes :

- Incohérence des données lues ≤ $\boldsymbol{\varepsilon}_{ET}^{import}$
- Incohérence des données modifiées ≤ $\boldsymbol{\varepsilon}_{ET}^{export}$.

On peut illustrer notre définition par l'exemple suivant qui met en évidence la dépendance des deux incohérences.

Soient deux transactions T1 et T2, et trois données *a*, *b* et *c* qui ont initialement pour valeurs : 3, 4 et 2 respectivement (cf. figure 1).

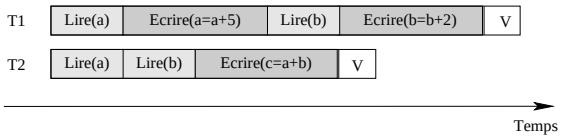

Figure 1 : Importation/Exportation des incohérences

Leur exécution est alors la suivante :

- T1 commence par lire la valeur de *a* (*a=3*) puis la modifier (*a=a+5*), ensuite elle lit la valeur de *b* (*b=4*) puis la modifie (*b=b+2*).
- T2 commence par la lecture de *a* et de b. Ces valeurs n'ont pas encore été modifiées en base car T1 n'est pas encore validée. Une incohérence importée a été alors associée à *a* et à *b*. L'incohérence importée par T2 est la somme des incohérences associées à *a* et à *b*. De même, pour écrire *c*, T2 doit toujours se baser sur les anciennes valeurs de *a* et de *b*, et l'incohérence exportée par T2 est l'incohérence associée à *c*.

5. - CONTROLE DE DIVERGENCE

5.1 - Définition

Comme nous l'avons déjà cité, l'*ESR* adoucit la sévérité de la *SR* dans le traitement des transactions en autorisant une incohérence limitée de la base de données. La limite d'incohérence est automatiquement maintenue par les méthodes de contrôle de divergence (*CD*), de la même façon que la *SR* est maintenue par le mécanisme de contrôle de concurrence (*CC*). Les méthodes de *CD* sont une extension des méthodes de *CC*. Cependant, le contrôle de divergence pour l'*ESR* améliore la concurrence puisque des transactions qui seraient bloquées par le critère de sérialisabilité peuvent ne plus l'être.

Par analogie avec les algorithmes de *CC* qui supportent efficacement la *SR*, les méthodes de *CD*

présentent une conception efficace pour la garantie de l'*ESR*.
Les méthodes de contrôle de divergence (*CD*) effectuent une analyse des méthodes de contrôle de concurrence (*CC*) pour identifier les endroits où les algorithmes de *CC* détectent les conflits des opérations sur les bases de données, i.e. qui provoquent la *non-sérialisabilité* (*non-SR*). Ensuite, elles modifient les algorithmes, qui tiennent compte de la *non-SR* et des opérations conflictuelles (Lecture/Ecriture), de telle façon que la tolérance à l'incohérence soit limitée. Les méthodes de *CD* sont similaires aux algorithmes de *CC* correspondants à la différence près que les méthodes de *CD* autorisent des conflits , sous certaines conditions (Pu, 1997 ; Son, 1993 ; Ulusoy, 1995).

5.2 - Mise en œuvre

Malgré les conceptions différentes pour les méthodes de *CD*, l'idée commune de base consiste à détecter chaque conflit *non-SR* vu par $\boldsymbol{ET_L}$ et à limiter la quantité d'incohérence importée. Par exemple, si les conflits *non-SR* sont comptés uniquement une seule fois pour chaque donnée, alors le nombre de ces conflits est équivalent au nombre total de données lues de façon *non-SR*. Si une $\boldsymbol{ET_L}$ lit des données et une $\boldsymbol{ET_E}$ concurrente est en train de modifier la donnée, alors les transactions ne sont pas sérialisables. Ces données sont donc lues de façon "non-sérialisable".
Les données lues de façon *non-SR* peuvent être utilisées pour modéliser diverses applications pratiques. Si le nombre total des données, lues par une ET_L *non-SR*, est limité par ces spécifications, alors les ET_L peuvent être exécutées simultanément et non-sérialisablement avec d'autres ET_E.
Deux accumulateurs d'incohérence sont associés à chaque *ET*, Accumu-Import et Accumu-Export.
Accumu-Import enregistre la quantité totale d'incohérence que l'*ET* a importé et Accumu-Export maintient et préserve la quantité totale d'incohérence que l'*ET* a exporté. Pour limiter l'incohérence, les méthodes de *CD* s'assurent que pour chaque *ET*, Accumu-Import $\leq \varepsilon_{ET}^{import}$ et Accumu-Export $\leq \varepsilon_{ET}^{export}$.
La conception des méthodes de *CD* repose sur deux phases :

- une « phase d'extension » : les méthodes de *CC* sont étendues par l'identification des endroits où la *non-SR* et les conflits sont détectés.
- une « phase de relaxation » : la *SR* est détendue et relâchée par l'utilisation de l'Accumu-Import et de l'Accumu-Export pour tenir compte de la concurrence des *ETs*.

Dans le *CD,* il s'agit juste d'identifier les conflits *non-SR* et d'empêcher la formation de cycles dans les graphes de sérialisation.
L'étape d'extension isole la partie d'identification des conflits, et l'étape de relaxation modifie la partie de prévention du cycle afin de permettre des incohérences spécifiques. Les modifications varient d'une méthode à l'autre. Il existe plusieurs modifications possibles et plusieurs façons de limiter l'incohérence. Des algorithmes ont été proposés pour garantir l'*ESR* dans les traitements des *ETs* (Wu, 1992 ; Tsang, 1997). Nous étudions le verrouillage à deux phases du *CD* (noté par la suite 2*PLDC*) car le protocole 2*PL*[2] est de loin la technique la plus utilisée (Bernstein, 1987).

5.3 - 2PLDC pour le temps réel

Comme tout protocole de contrôle de divergence (CD), l'application du 2*PLDC* comporte deux phases.

Phase d'extension
C'est la phase d'identification et de détection des conflits en se basant sur une matrice de compatibilité des verrous.
La matrice de compatibilité des verrous pour le 2*PL* standard (*2PLCC*) est présentée par le tableau 2, et celle pour le 2*PLDC* par le tableau 3.

	VL^T	VE^T
VL^T	**ACC**	**X**
VE^T	**X**	**X**

Tableau 2 : ***Compatibilité pour 2PLCC***

Dans le tableau 2 :
- VL^T désigne le verrou de lecture détenu par une transaction,
- VE^T désigne le verrou d'écriture détenu par une transaction.

	VL^{ET_L}	VL^{ET_E}	VE^{ET_E}
VL^{ET_L}	**ACC**	**ACC**	**ACCL-1**
VL^{ET_E}	**ACC**	**ACC**	**X**
VE^{ET_E}	**ACCL-2**	**X**	**X**

Tableau 3 : ***Compatibilité pour 2PLDC***

Dans le tableau 3 :
VL^{ET_L} désigne le verrou de lecture détenu par une ET_L, VL^{ET_E} désigne le verrou de lecture détenu par une ET_E, VE^{ET_E} désigne le verrou d'écriture détenu par une ET_E.
Dans les deux tableaux, les colonnes représentent les verrous détenus et les lignes représentent les verrous demandés. Les cases marquées par ACC

[2]Verrouillage à 2 phases

correspondent aux verrous compatibles (Accord) et les cases marquées d'une croix aux verrous incompatibles.
Le tableau 3 diffère du tableau 2 par l'ajout des deux cases : ACCL-1 et ACCL-2 (Accord Limité). ACCL-1 permet aux ET_L de lire les données non validées (non committées) alors que ACCL-2 permet aux ET_E d'écraser la donnée qui a déjà été lue par une ET_L. On suppose qu'une fois le verrou de VE^{ET_E} acquis, l'ET_E écrit sa nouvelle donnée sur place dans le tampon (buffer) pour que toute autre ET_L puisse immédiatement lire la nouvelle donnée mise à jour mais non encore validée.
L'histoire (1) ci-dessous montre un exemple de verrou ACCL-1sur la donnée 'b' :

$$L_1^{ET_E}(a)\ L_1^{ET_E}(b)\ L_2^{ET_L}(a)\ \underline{E_1^{ET_E}(b)\ L_2^{ET_L}(b)}\ E_1^{ET_E}(a) \qquad (1)$$

Dans cette histoire, la partie soulignée met en évidence un conflit de type E/L sur la donnée 'b'. Le verrou de lecture VL^{ET_L}, demandé par ET_L sur la donnée 'b', peut être acquis si la transaction est proche de son échéance, même si le verrou d'écriture VE^{ET_E} est encore détenu par ET_E.

L'histoire (2) est un exemple de concurrence permise par ACCL-2 sur la donnée '*a'* :

$$L_1^{ET_E}(a)\ L_1^{ET_E}(b)\ E_1^{ET_E}(b)\ \underline{L_2^{ET_L}(a)\ E_1^{ET_E}(a)}\ E_2^{ET_L}(b) \qquad (2)$$

Dans cette histoire, la partie soulignée met en évidence un conflit de type L/E sur la donnée 'a'. Le verrou d'écriture VL^{ET_E}, demandé par ET_E sur la donnée 'a', peut être acquis si la transaction est proche de son échéance, même si le verrou de lecture VL^{ET_L} est encore détenu par ET_L.

Phase de relaxation
Pour contrôler la quantité d'incohérence permise dans les ET_L, nous raffinons la gestion des verrous sur **ACCL-1** et **ACCL-2**. A chaque fois que ET_L demande un verrou VL^{ET_L} dans le cadre de **ACCL-1**, on examine d'abord les valeurs des variables Accumu-Import et Accumu-Export, respectivement de ET_L et ET_E, pour voir si, en les incrémentant d'une unité, elles dépasseront leurs limites (Son, 1993). De la même façon, à chaque fois qu'une ET_E demande un verrou VE^{ET_E} dans le cadre de **ACCL-2**, toutes les ET_L détentrices du verrou VL^{ET_L} en conflit ont leurs Accumu-Import examinés pour voir s'ils ont dépassé leurs $\varepsilon_{ET}^{import}$ après incrémentation d'une unité. Cependant, l'ET_E doit vérifier ses propres Accumu-Export contre (face à) l'incrémentation par le nombre total des ET_L détentrices des verrous VL^{ET_L}. Si l'Accumu-Export de ET_E dépasse son seuil $\varepsilon_{ET}^{export}$ ou si n'importe quel Accumu-Import d'une ET_L dépasse son $\varepsilon_{ET}^{import}$, alors nous devons arrêter le flux d'informations et interdire cet accès particulier. Dans ce cas, on peut choisir de faire attendre le demandeur ou d'annuler la transaction, puis recommencer en retournant les verrous échoués aux *ETs* demandeuses ou en brisant quelques verrous déjà détenus par d'autres *ETs* (en fonction des priorités).

6 - APPLICATION DE L'ESR

6.1 - Applicabilité de l'ESR aux transactions temps réel

L'*ESR* possède un grand nombre d'algorithmes exécutables et efficaces. Sa compatibilité est ascendante puisqu'elle atteint la *SR* classique lorsque ε tend vers 0.
Une classe d'applications où l'*ESR* semble particulièrement souhaitable concerne celles qui manipulent des données dont les valeurs sont mises à jour périodiquement. Cette classe d'applications est souvent caractérisée par :
- une grande quantité de données,
- un rafraîchissement fréquent de la base de données,
- des contraintes temps réel strictes non critiques où les mises à jour doivent être acceptées à l'intérieur d'un intervalle de temps, sinon elles sont rejetées,
- une conception des applications qui accepte que les décisions puissent être basées sur des résultats incomplets/imprécis.

Des algorithmes de *CD* ont été proposés pour limiter l'incohérence vue par les ET_L à un degré qui peut être toléré selon l'application.

6.2 - L'apport de l'application de l'ESR

Dans cet article, nous avons proposé d'utiliser le critère d'epsilon sérialisabilité pour les transactions temps réel. En effet, l'*ESR* est à la fois faisable et largement applicable pour le traitement des transactions temps réel. Des simulations que nous avons effectuées montrent que ce critère est tout à fait adéquat dans les SGBDTR. Par rapport à la sérialisabilité classique, l'*ESR* :
- autorise des traitements asynchrones,
- offre une plus grande disponibilité des données puisqu'elle autorise l'acquisition des verrous sur une donnée qui est déjà verrouillée par une autre transaction,
- permet d'augmenter le nombre des transactions qui réussissent à s'exécuter avant leur échéance car elles ne sont pas obligées d'attendre la libération d'un verrou pour accéder à une donnée verrouillée.

7 - SIMULATION

Dans cette simulation, nous souhaitons mettre en évidence le gain de performance engendré par l'utilisation du critère d'*ESR* dans les *SGBDTR* par rapport au critère de *SR*. On applique pour cela une technique de contrôle de divergence, 2*PLDC*, qui permet de garantir l'*ESR* dans le cadre d'un *SGBD* classique.

Le modèle simplifié que nous avons utilisé consiste en une base de données centralisée dans laquelle s'effectuent des transactions (qui utilisent comme critère de correction la *SR*), puis des Epsilon-Transactions (qui utilisent comme critère de correction l'*ESR*). Pour le *CC*, le protocole *2PLCC* est utilisé alors que pour le *CD*, le *2PLDC* est utilisé. *2PLCC* est mis en œuvre en deux phases : (1) une phase d'acquisition de verrous et (2) une phase de libération de verrous.

Les verrous sont demandés au moyen de l'opération `LOCK(G,M)` de l'objet `LOCKController` et libérés par l'opération `UNLOCK(G)` du même objet. *G* désigne le granule à verrouiller ou à déverrouiller et *M* le mode de verrouillage.

Pour introduire la notion de temps réel, les transactions (et Epsilon-Transactions) possèdent des échéances. Chaque transaction possède son propre contrôleur de temps qui compare l'échéance avec le temps courant. Tant que la transaction n'est pas encore validée, on vérifie si l'instant courant est égal ou non à l'échéance. Si le temps est égal à l'échéance alors que la transaction n'est pas terminée alors la transaction est abandonnée.

Un gestionnaire de file d'attente permet d'insérer, dans une file, les transactions qui sont en attente de libération de verrous. Elles sont ordonnées en fonction de leurs priorités. La priorité la plus élevée correspond à l'échéance la plus proche suivant le principe du protocole Earliest Deadline First (*EDF*) (Liu, 1973). Un gestionnaire de transactions permet de consulter régulièrement la file, de la mettre à jour, de vérifier la validité des transactions (c'est-à-dire si elles n'ont pas encore raté leur échéance) et il envoie celles qui sont encore valides[3] à la base pour qu'elles effectuent une nouvelle tentative de verrouillage.

Une Epsilon-Transaction (*ET*) est caractérisée par sa propre échéance, sa quantité d'incohérence importée ($\varepsilon_{ET}^{import}$), sa quantité d'incohérence exportée ($\varepsilon_{ET}^{export}$) et une liste des opérations de lectures et d'écritures.

Deux accumulateurs (Accumu-Import et Accumu-Export) permettent de contrôler la quantité d'incohérence autorisée par l'*ESR*. À chaque importation d'incohérence (resp. exportation) l'Accumu-Import (resp. l'Accumu-Export) sera incrémenté de 1 jusqu'à la limite autorisée $\varepsilon_{ET}^{import}$ (resp. $\varepsilon_{ET}^{export}$).

Une exécution de notre programme a permis de confirmer les apports de l'*ESR* par rapport à la *SR*. Les résultats ainsi obtenus sont présentés dans le tableau 4.

Le tableau 4 montre que le nombre de transactions qui réussissent par l'*ESR* est beaucoup plus important que celui offert par la *SR* classique. Les transactions qui réussissent sont celles qui respectent leurs échéances, c'est à dire qui valident avant échéance. Le fait d'adoucir les limitations de la *SR* améliore donc le résultat final.

On constate également que le taux de réussite des transactions augmente avec l'augmentation du nombre des transactions à exécuter. Ceci s'explique car : lorsque le nombre de transactions augmente dans le système, le nombre de conflits augmente également (en probabilité). En appliquant le critère de *SR* classique, il y a de fortes chances que beaucoup de verrouillages surviennent. Ces verrouillages conduiront fatalement à davantage de transactions qui manquent leurs échéances. Avec l'*ESR*, beaucoup de conflits sont évités puisqu'on permet que des incohérences contrôlées surviennent. Les transactions, appelées Epsilon-Transactions, ont donc plus de chances de se terminer avant leurs échéances.

Les résultats donnés dans le tableau 4 sont représentés graphiquement sur la figure 2. On constate que l'*ESR* permet à plus de transactions de terminer avant leur échéance. L'amélioration est de l'ordre de 35% par rapport à la *SR* classique.

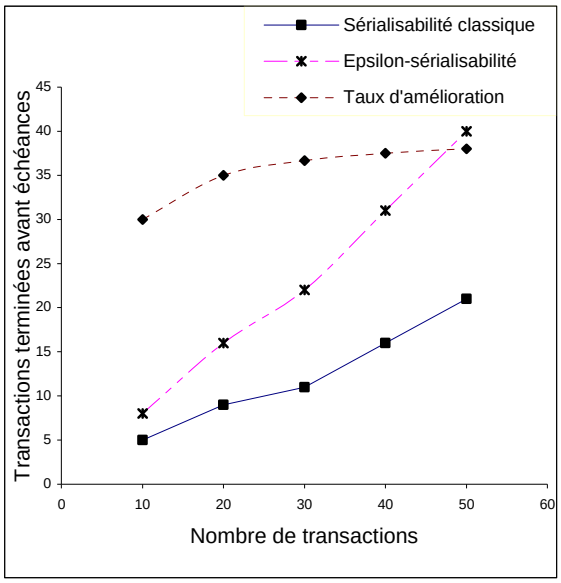

Figure 2: Nombre de transactions qui réussissent par la SR et par l'ESR et taux d'amélioration

[3] qui n'ont pas encore manqué leur échéance

Nombre initial de transactions	Nombre de transactions qui réussissent par la *SR*	Nombre de transactions qui réussissent par l'*ESR*	Taux d'amélioration
10	5	8	0.3
20	9	16	0.35
30	11	22	0.366667
40	16	31	0.375
50	21	40	0.38

Tableau 4: ***Comparaison des résultats entre la SR et l'ESR***

8. - CONCLUSIONS ET PERSPECTIVES

L'objectif de ce travail était d'étudier l'*ESR* et la gestion des conflits entre *ETs* afin de démontrer l'apport de l'*ESR* par rapport à la sérialisabilité classique dans le cadre des *SGBDTR*. Pour cela, un ensemble de concepts ont été définis afin de pouvoir mettre en œuvre l'*ESR*. Les applications pouvant bénéficier de ce type de *SGBDTR* (ayant comme critère l'*ESR*) doivent évidemment tolérer des imprécisions sur les données et/ou les résultats. Il s'agit notamment des applications où un résultat obtenu dans le temps (même partiel ou imprécis), peut être préférable à un résultat complet et précis obtenu en retard. Par exemple, dans les systèmes de prise de décision (marchés boursiers, décision de suivre telle ou telle trajectoire pour un robot, ...).
Afin d'être complète, notre étude a été validée par une implémentation d'un algorithme de contrôle de divergence. Au moyen de cet algorithme, nous avons pu mettre en évidence les améliorations qu'apporte l'*ESR* par rapport à la *SR*. En effet, l'incohérence autorisée par l'*ESR* augmente le nombre d'*ETs* qui se terminent avant échéance et améliore les critères de performance des applications reposant sur un *SGBDTR*.
L'*ESR* traite le problème du coté des transactions. Notre objectif est d'étendre nos travaux pour prendre en compte la spécification de la qualité de service au travers de la qualité des transactions et de la qualité des données. Dans nos futurs travaux, nous pensons utiliser le paramètre ε de l'*ESR* pour faire varier les critères de qualité de service en fonction des besoins des applications.

REMERCIEMENTS

Ces travaux ont été effectués dans le cadre de l'ACI – Jeune Chercheur 1055.

BIBLIOGRAPHIE

Bernstein, P., Hadzilacos, V., Goodman, N. (1987). *Concurrency control and recovery in database systems.* Addison-Wesley.

Date, C.S. (1985). *An Introduction to Database Systems*. Addison-Wesley.

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

Kamath, M., Ramamritham, K. (1993). *Performance Characteristics of Epsilon Serialisability with Hierarchical Inconsistency Bounds*. In Proceedings of the 9th International Conference on Data Engineering. pages 587–594. IEEE Computer Society Press.

Liu, C., Leyland, J. (1973). *Scheduling algorithms for multiprogramming in hard real-time environment.* Journal of the ACM, 20(1): 46–61.

Lam, K., Yau, W. (1998). *On Using Similarity for Concurrency Control in Real-Time Database Systems*. Journal of Systems and Software, 43(3): 223–232,

Pu, C. (1991). *Generalized Transactions Processing with Epsilon Serializability*. In Proceedings of 4th International Workshop on High Performance Transactions Systems, Asilomar, California.

Pu, C., Leff, A. (1992). *Autonomous Transaction Execution with Epsilon Serializability*. In Proceeding of 1992 RIDE Workshop on Transaction and Query Processing. IEEE Computer Society.

Ramamritham, K., Chrysanthis, P.K. (1992). *A Taxonomy of Correctness Criteria in Database Applications*. Journal of Very Large Databases, 4(1): 85–97.

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

Ramamritham, K., Chrysanthis, P. K. (1993). *In Search of Acceptability Criteria: Database Consistency Requirements and Transaction Correctness Properties,* Chapter in "Distributed Object Management", pages 211–229. Morgan Kaufmann Publisher.

Ramamritham, K., Pu, C.A. (1995). *Formal Characterization of Epsilon Serialisability*. IEEE Transaction Knowledge and Data Engineering, 7(6): 997–1007.

Sadeg, B., Amanton, L., Haubert, J. (2003). *Tradding Precision for Timeliness in Distributed Real-Time Databases*. In Proceedings of 5th International Conference on Enterprise Information System (ICEIS'2003), Anger, France.

Son, S.H., Kouloumbis, S. (1993). *A Token-Based Synchronization Scheme Using Epsilon-Serialisability and its Perfo-rmance for Real-Time Distributed Database Systems.* In Proceedings of 3th International Symposium on Database Systems for Advanced Applications, Korea.

Tsang, M.K., Wu, K.L., Yu, P.S. (1997). *Multiversion Divergence Control Time Fuzziness*. Technical Report No. OR 97291-1000, Department of Computer Science and Engineering, Yorktown Heights, NY 10598.

Ullman J.D. (1980). *Principles of Database Systems*. Computer Science Press, Inc., Maryland.

Ulusoy, Ö. (1995). *Research Issues in Real-Time Database Systems*. Information Sciences, 87: 123–151.

Wu, K.L., Yu, P.S., Pu. C. (1992). *Divergence Control Algorithms for Epsilon Serialisability*. In Proceedings of 8th International Conference on Data Engineering, pages 506–515. IEEE Computer Society.