ACCO_CF/RTT : UN ALGORITHME DE CONTROLE DE CONCURRENCE OPTIMISTE POUR LES RELATIONS TEMPORELLES DE TRANSACTION

Rafik BOUAZIZ,
Maître assistant en informatique
Raf.Bouaziz@fsegs.rnu.tn

Achraf MAKNI,
Doctorant en informatique
Achraf.Makni@fsegs.rnu.tn

Adresse professionnelle
Faculté des Sciences Economiques et de Gestion de Sfax ★B.P. 1088★ 3018 Sfax, Tunisie

Résumé : ACCO_CF/RTT est un nouvel algorithme optimiste de contrôle de concurrence pour les relations temporelles de transaction. Cet algorithme procède à l'estampillage des transactions par leur instant d'arrivée et à la notification de toute opération réalisée sur les données. Il valide toute transaction terminée et devenant prioritaire, en utilisant la technique de photos de transactions et celle de marquage de fin de transaction. Les risques d'avortement non justifiés sont ainsi évités.

Summary : ACCO_CF/RTT is a new optimistic concurrency control algorithm for transaction-time relations. This algorithm proceeds to timestamping transactions by their arrival moments and to notify any operation realized on data. It validates any terminated transaction that has the most priority, by using the techniques of transaction snapshots and end of transaction marker. The probabilities of no proof abortion are then eliminated.

Mots clés : Contrôle de concurrence, Méthode optimiste de contrôle de concurrence, Cohérence forte, Bases de données temporelles, Relation temporelle de transaction.

Key words : Concurrency control, optimistic concurrency control method, strong consistency, temporal databases, transaction-time relation.

ACCO_CF/RTT : Un algorithme de contrôle de concurrence optimiste pour les relations temporelles de transaction

1 - INTRODUCTION

La cohérence d'une base de données (BD) exige que le contrôle de concurrence (CC) garantisse une exécution simultanée des transactions qui produit les mêmes résultats que leur exécution séquentielle [Gardarin (1988)]. La cohérence forte d'une BD exige que le CC garantisse une exécution simultanée des transactions qui produit les mêmes résultats que l'exécution séquentielle des transactions dans l'ordre strict de leur arrivée [Rahgozar (1987)].

Le CC prend de nouvelles dimensions lorsqu'on veut l'appliquer aux BD temporelles (BDT), ayant pour objectif de gérer l'historique des données. Peu de travaux ont abordé les problèmes liés à cet aspect [Finger et McBrien (1997)] [Elloumi, Bouaziz et Moalla (1998)] [Castro (1998)]. Nous proposons dans cet article un nouvel algorithme optimiste permettant d'assurer la cohérence forte pour les relations temporelles de transaction (RTT).

La section 2 présente l'état de l'art du CC et de son application aux BDT. La section 3 décrit l'environnement et les objectifs de l'algorithme proposé, ACCO_CF/RTT, dont la structuration est discutée au niveau de la section 4 : définition et ordonnancement des tâches du contrôleur de concurrence pour une RTT, procédure de certification et procédure de validation. La section 5 est consacrée à l'analyse d'un cas d'application de cet algorithme. Finalement, un prototype est présenté dans la section 6.

2 - ETAT DE L'ART DU CC ET DE SON APPLICATION AUX BDT

A notre connaissance, les travaux récents concernant le CC sont peu nombreux. Ils se sont intéressés :

- aux BDT [Finger et McBrien (1997)] [Elloumi, Bouaziz et Moalla (1998)] [Castro (1998)] [Makni et Bouaziz (2003)] qui constituent le domaine de notre contribution ;
- aux BD réparties et aux applications temps réel, avec l'introduction du concept de "epsilon-sérialisation". Ce nouveau critère de correction autorise l'introduction de certaines incohérences, mais bornées et contrôlées. Une transaction poursuit son exécution malgré l'apparition d'incohérences tant que la quantité d'incohérence permise n'est pas dépassée [Lee et Lam (2000)] [Amanton, Sadeg et Haubert (2003)].

Les méthodes de contrôle des accès concurrents sont classifiées dans la littérature en deux grandes catégories :

- Les méthodes pessimistes où le contrôle de cohérence est effectué lors de chaque opération d'une transaction [BERNSTEIN et GOODMAN (1980)] [Miranda et Busta (1986)] [Kumar (1989)] [Yu, Dias et Lavenberg (1990)].
- Les méthodes optimistes où le contrôle de cohérence n'est effectué qu'à la fin de la transaction. Ceci veut dire que ces méthodes autorisent l'exécution simultanée des transactions sans aucun contrôle préalable. C'est au moment de la validation que la cohérence de la base est testée et, le cas échéant, corrigée [Menasce et Nakanishi (1982)] [Chan (1986)] [Yu, Dias et Lavenberg (1990)].

Avec un mécanisme de contrôle de concurrence optimiste, le traitement d'une transaction comporte trois phases : phase de lecture, phase de validation et phase d'écriture.

La validation peut avoir lieu soit après la phase de lecture, soit au cours de cette phase. Selon le moment de la validation, deux types de méthodes optimistes peuvent être envisagés [Yu, Dias et Lavenberg (1990)] : pure optimistic method (POM) et broadcast optimistic method (BOM).

POM, la méthode originale [Kung et Robinson (1979)], adopte une stratégie de validation

basée sur l'histoire des transactions validées. POM présente l'inconvénient d'une définition sévère du conflit, d'où le risque d'avortement non nécessaire de transactions. Cette approche est améliorée par l'introduction de la méthode de marquage de fin de transaction "EOT Marker" qui a permis de surmonter le problème de la définition sévère du conflit.

Quant à la méthode BOM, elle effectue la validation par les photos de toutes les transactions concurrentes qui sont dans leur phase de lecture. Elle permet de détecter les conflits plus tôt que dans les autres méthodes ; une transaction ne doit pas terminer son exécution si on détecte un conflit. Deux variantes de cette méthode ont été proposées :

- BOM avec section critique (SC) : les phases de validation et d'écriture forment ensembles une SC, durant laquelle la BD est verrouillée ; aucune transaction n'a le droit d'y accéder.
- BOM sans SC.

BOM avec SC permet de choisir la transaction à avorter en cas de conflit, ce qui permet d'éviter le problème de famine (starvation problem) qui peut se poser dans les autres méthodes.

Pour accroître le degré de parallélisme, un algorithme de CC peut procéder à la tenue de plusieurs versions pour chaque granule. C'est le cas de l'algorithme que nous proposons ici. Cet algorithme concerne les relations temporelles de transaction (RTT) et exploite les versions tenues par ces relations elles-mêmes.

L'objectif de ces relations consiste à fournir aux applications, devenues nombreuses, non seulement les données courantes, mais aussi tous les états qui se succèdent dans le temps. Pour pouvoir maintenir ces états, il faut que les opérations de mise à jour soient non destructives, tel que c'est proposé, entre autres, par les RTT d'une BDT. Les RTT procèdent au stockage des versions dépassées en les estampillant par les deux temps physiques suivants :

- "temps de début de transaction (TDT)" : temps d'exécution de la transaction qui insère le tuple correspondant. Ce temps est connu a priori.
- "temps de fin de transaction (TFT)" : temps d'exécution de la transaction qui met à jour ou supprime le tuple considéré. Il n'est pas donc connu a priori.

L'intervalle formé par ces deux attributs représente l'intervalle de temps pendant lequel le tuple était le tuple courant [Bouaziz (1991)].

Du fait que ces attributs ne sont jamais modifiés par l'utilisateur, il serait alors possible de rendre compte de l'état de la base à n'importe quel instant du passé, et en particulier pour les états incohérents qui, eux non plus, ne sont pas détruits. Mais, sachant que le temps de début de transaction d'un tuple correspond à l'instant où il est inséré dans la base, il n'est pas possible d'enregistrer des mises à jour postactives, pour insérer des tuples qui se rapportent au futur, ou rétroactives, pour corriger le passé.

Dans le cadre d'une RTT, l'application d'un algorithme de CC qui exploite les anciennes versions des granules maintenues dans la BD serait l'un des buts à atteindre.

Elloumi, Bouaziz et Moalla (1998) ont étudié la possibilité d'appliquer les algorithmes pessimistes dans le cadre des RTT et ont proposé, par la suite, un nouvel algorithme de CC pessimiste qui permet d'assurer la cohérence forte de la BD. Cet algorithme exige une connaissance a priori des granules à manipuler par toute transaction de mise à jour, ce qui constitue une difficulté de mise en œuvre.

Un autre algorithme pessimiste a été proposé dans [Finger et McBrien (1997)]. Il assure une sérialisation temporelle qui permet de garantir une exécution sérialisable des transactions dans l'ordre strict de leurs estampilles (qui correspondent à leurs instants d'arrivée et qui marquent la maturité de ces transactions). Ainsi, 2PL est étendu avec un mécanisme d'ordonnancement de maturité. Dans cet algorithme :

- pour deux transactions Ti et Tj avec $i < j$, si Tj a obtenu un verrou sur un granule x non encore libéré et si Ti demande un verrou non compatible sur x, alors Tj est avortée ;
- une transaction ne peut être validée qu'après validation de toute transaction plus vieille (plus prioritaire) qu'elle.

Cet algorithme a été amélioré par l'utilisation de connaissances a priori, ce qui a permis à une transaction T de valider ses traitements même

si d'autres plus vieilles sont encore en exécution, à condition que ces dernières n'ont pas déclaré a priori des verrous en conflit avec T. Cependant, l'algorithme ne garantit plus la cohérence forte.

Par ailleurs, Castro (1998) a étudié l'amélioration des algorithmes de CC classiques dans les BDT, en essayant de cerner les conflits effectifs sur les données impliquées en fonction des portions temporelles, afin d'optimiser le temps d'exécution global des opérations. Chaque transaction est découpée en deux sous-transactions pour isoler les traitements qui s'adressent à la portion temporelle des données, objet de conflit. La sous-transaction conflictuelle doit être sérialisée avec les sous-transactions concurrentes. Cependant, le concept de l'atomicité d'une transaction est révisé.

Par contre, les algorithmes optimistes n'ont pas été, à notre connaissance, étudiés pour un environnement d'une RTT. Notre travail se propose de le faire afin de construire un algorithme optimiste permettant d'assurer la cohérence forte et d'accroître le parallélisme.

3 - ENVIRONNEMENT ET OBJECTIFS DE L'ALGORITHME PROPOSE

Pour étudier le contrôle de concurrence, nous admettons l'hypothèse adoptée dans [Bernstein, Hadzilacos et Goodman (1987)] consistant à considérer tout système de bases de données (SBD) comme étant formé, tel que le montre la figure 1, des trois modules suivants :

- **Un gestionnaire des transactions** (ou contrôleur d'exécution) qui traite les opérations des transactions reçues ; c'est à travers lui que s'effectue l'interaction entre les transactions et le SBD. Le gestionnaire des transactions lance l'exécution en mode multitâche (que l'on appelle par abus de langage "parallèle") de plusieurs transactions dans différents processus. Lorsqu'une transaction arrive au système, elle est d'abord estampillée par l'instant de son arrivée et stockée dans une file d'attente, puis affectée à un processus dès que le gestionnaire des transactions en trouve un de libre.

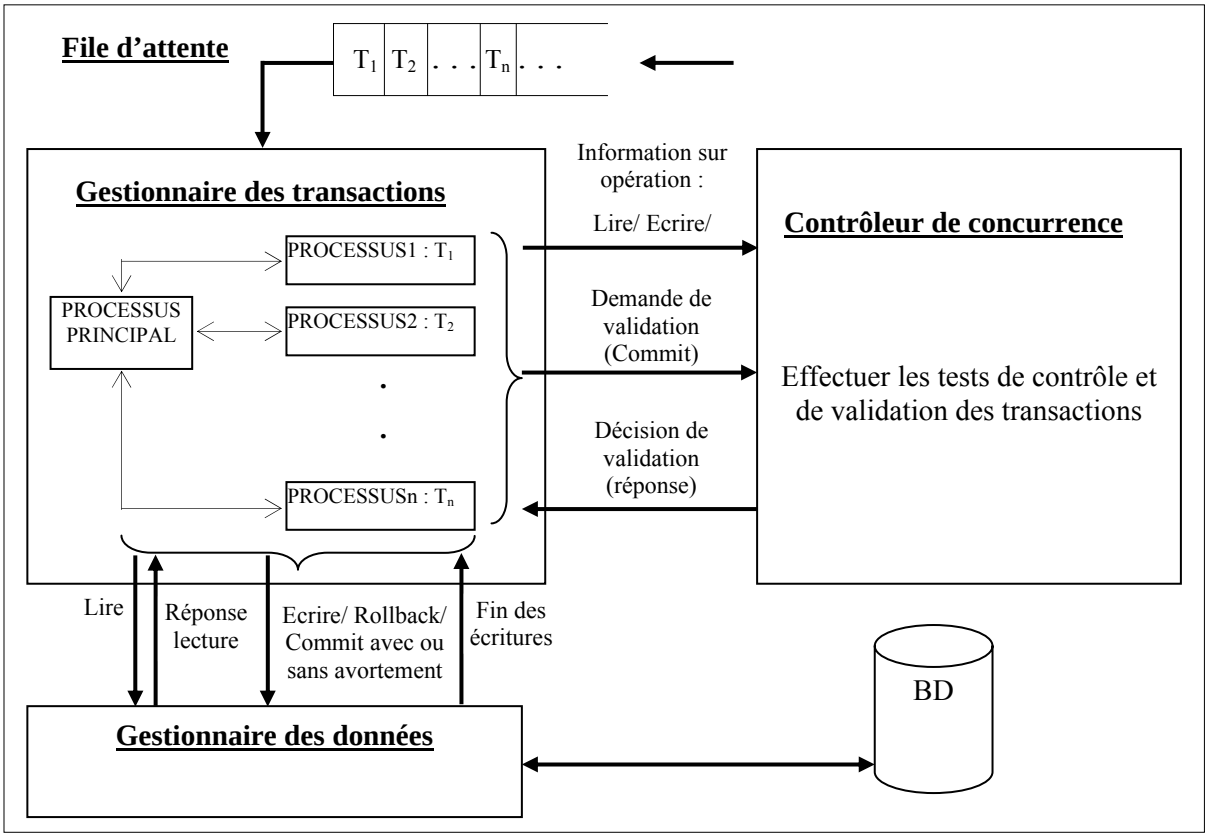


Figure 1. Architecture d'un système de bases de données

- **Un contrôleur de concurrence** qui surveille l'exécution concurrente des transactions afin que cette exécution soit sérialisable. Il a pour rôle d'effectuer les

tests de contrôle et de validation des transactions lancées par le gestionnaire des transactions. Le contrôleur de concurrence communique avec les processus en exécution. En effet, à chaque opération Lire, Ecrire ou Rollback et à chaque demande de validation (Commit), ce contrôleur effectue un traitement approprié.

- **Un gestionnaire des données** qui assure la répercussion sur la BD de tous les effets des transactions validées. C'est lui qui réalise les opérations Lire, Ecrire, Rollback et Commit.

Pour exécuter une opération de lecture, d'écriture ou d'annulation (Rollback), le gestionnaire des transactions fait exécuter cette opération par le gestionnaire des données et en informe le contrôleur de concurrence.

Pour exécuter une opération de validation sur la BD, le gestionnaire des transactions soumet cette opération au contrôleur de concurrence qui décide de l'une des actions suivantes :

- Faire exécuter l'opération par le gestionnaire des données.
- Retarder l'opération ; la transaction est alors mise dans une file d'attente interne pour être validée ultérieurement.
- Annuler la transaction concernée ou les transactions impliquées pour éviter des incohérences.

L'algorithme optimiste que nous nous proposons de concevoir doit garantir les objectifs suivants :

- Maintenir la cohérence forte du système d'information dans un environnement de RTT.
- Exploiter les versions dépassées d'une RTT au lieu de gérer des versions propres au contrôle de la concurrence.
- Minimiser le degré d'avortement des transactions.
- Eviter le problème de famine.
- Détecter les conflits le plus tôt possible.

4 - STRUCTURATION DE L'ALGORITHME ACCO_CF/RTT

4.1 - Définition et ordonnancement des tâches du contrôleur de concurrence pour une RTT

Pour chaque transaction T_i, le contrôleur de concurrence maintient deux ensembles : l'ensemble RS_i (read set), relatif aux objets lus par T_i, et l'ensemble WS_i (write set), relatif aux objets écrits par T_i.

Durant l'exécution d'une transaction, lorsque le contrôleur de concurrence reçoit :

- une opération Lire (T_i, g), il ajoute le granule g à l'ensemble des objets lus ;
- une opération Lire (T_i, g, pt), il n'ajoute le granule g à l'ensemble des objets lus que si pt indique la version courante de g. Si pt désigne une version dépassée, cette opération de lecture ne doit pas être prise en considération dans le test de validation, car elle ne peut en aucun cas produire des conflits ;
- une opération Ecrire (T_i, g), il ajoute le granule g à l'ensemble des objets écrits ;
- une opération Rollback, il élimine les objets lus et écrits de RS_i et WS_i ;
- une opération Commit, il vérifie s'il existe ou non un conflit entre la transaction à valider et les transactions qui ne sont pas encore validées.

Nous proposons de partir de la stratégie de validation des algorithmes BOM. Cette stratégie stipule qu'à chaque exécution, à un instant t, de la commande COMMIT d'une transaction T_i, le contrôleur de concurrence doit effectuer un test de validation de T_i vis à vis des transactions concurrentes qui sont encore dans leur phase de lecture. Pour chacune de ces dernières, le contrôleur de concurrence doit donc examiner l'intersection de l'ensemble des objets qu'elle a lus, jusqu'à l'instant t, avec celui des objets écrits par T_i. Si l'intersection n'est pas vide, c'est qu'il existe un conflit entre les deux transactions. La transaction à avorter est la moins prioritaire.

Afin de pouvoir assurer la cohérence forte, nous proposons de procéder à **l'estampillage des transactions par les instants de leur arrivée**, c'est la transaction la plus jeune qui doit être considérée comme la moins prioritaire. Cependant, d'après les études que nous avons effectuées, estampiller les transactions par les instants de leur arrivée ne suffit pas pour garantir la cohérence de la BD dans le cadre d'une RTT. De ce fait, cette stratégie ne suffit pas pour garantir la cohérence du système d'information dans le

cadre d'une RTT. Ceci est dû à la non synchronisation des transactions. Nous proposons donc de définir un **ordre de validation** entre les transactions lors de l'exécution de la commande COMMIT de toute transaction T_i. Par conséquent, le contrôleur de concurrence doit vérifier, avant de commencer le test de validation, que T_i est la transaction la plus prioritaire de l'ensemble des transactions qui ne sont pas encore validées. Nous proposons alors d'ajouter une **phase de certification** qui précède la phase de validation de chaque transaction.

4.2 - Structuration de la procédure de certification de ACCO_CF/RTT

A l'exécution de la commande COMMIT, T_i passe d'abord par un test de certification. Ce test vérifie que T_i est la plus prioritaire :

- Si ce n'est pas le cas, T_i est mise en attente pour être certifiée ultérieurement.
- Si T_i est la plus prioritaire, le contrôleur de concurrence effectue le test de validation de T_i avec toute transaction T_j en phase de lecture ou en attente de certification. Ces dernières sont forcément plus jeunes que T_i et donc moins prioritaires. Par conséquent :
  - S'il existe un conflit, c'est T_j qui doit être avortée pour être reprise avec la même estampille ;
  - Dans le cas contraire :
    - Si T_j était en attente de certification, le contrôleur de concurrence réexécute de nouveau la procédure de certification pour T_j ;
    - Sinon, le contrôleur d'exécution poursuit l'exécution des instructions de T_j.

Une fois arrivée à sa phase de validation, T_i est validée d'office. Les nouvelles versions des granules manipulés par cette transaction seront enregistrées dans la BD et prendront comme estampille le temps de transaction de T_i, égal à t_i, instant d'arrivée de T_i.

Nous proposons que le test de certification, qui accompagne l'exécution de l'opération COMMIT de toute transaction T_i, soit structuré comme suit :

```
CERTIFICATION (Ti) : entier
   Début
      /* Certification de Ti avec les transactions en attente de certification */
      v := CERTIF_TA (Ti)
      Si v = 1 Alors
      /* Certification de Ti avec les transactions qui sont encore en phase de lecture */
         v := CERTIF_TL (Ti)
      Fin Si
      Retourner (v)
   Fin.
```

Les fonctions "CERTIF_TA" et "CERTIF_TL" permettent de certifier une transaction respectivement par rapport aux transactions en attente de certification et par rapport aux transactions en phase de lecture. Sachant que ces transactions sont ordonnées dans leur liste selon l'ordre croissant de leurs estampilles, ces fonctions sont structurées comme suit :

```
CERTIF_TA (Ti) : entier
   Début
      v := 1
      Si il existe des transactions en attente de certification Alors
         "j prend l'estampille de la première transaction en attente de certification : c'est la transaction la plus prioritaire parmi celles qui sont en attente"
         Si i > j Alors
            Ti est mise en attente de certification
            v := 0
         Fin Si
      Fin Si
      Retourner (v)
   Fin.

CERTIF_TL (Ti) : entier
   Début
      v := 1
      Si il existe des transactions en phase de lecture Alors
         "j prend l'estampille de la première transaction en phase de lecture : c'est la transaction la plus prioritaire parmi celles qui sont en exécution "
         Si i > j Alors
            Ti est mise en attente de certification
            v := 0
         Fin Si
      Fin Si
      Retourner (v)
   Fin.
```

4.3 - Structuration de la procédure de validation de ACCO_CF/RTT

4.3.1 - Validation sous BOM sans SC pour une RTT

Le test de validation de BOM sans SC commence par attribuer un numéro de transaction à T_i, à travers un compteur global. Ce test, effectué avec chaque transaction concurrente T_j, dépend du fait que T_j est en phase de validation ou en phase de lecture.

Mais dans un environnement de RTT, il est nécessaire que l'on procède à l'estampillage des transactions par les instants de leur arrivée, afin de pouvoir assurer la cohérence forte. La procédure de validation doit utiliser ces estampilles et n'a pas donc à attribuer des numéros de transactions. Par ailleurs, le test de validation ne peut concerner que les transactions concurrentes qui n'ont pas encore commencé leur phase de validation. En effet, on ne peut jamais avoir, à un instant donné, deux transactions concurrentes en phase de validation, puisque les transactions passent d'abord par un test de certification. Par conséquent, la procédure de validation vérifie l'absence de conflit uniquement avec les transactions en phase de lecture, y compris celles en attente de certification.

Le test de validation devient alors comme suit :

VALIDATION_SSC (T_i)
Début
"Donner l'accord pour répercuter les écritures sur la BD"
/* Sachant que toutes les transactions qui précèdent Ti devaient être déjà validées */
"j prend l'estampille de la première transaction qui suit T_i"
Tantque j <> "!" **Faire**
Si (CONFLIT (WS_i, RS_j) = 0) **Alors**
"Donner un ordre d'avortement de T_j : ROOLBACK T_j"
Fin Si
"j prend l'estampille de la transaction suivante"
Fin Tantque
Fin.

La fonction "CONFLIT", appelée par la procédure de validation, teste l'absence de conflit entre deux transactions. Elle retourne la valeur 1 si le résultat du test est positif, 0 sinon. Elle est définie comme suit :

CONFLIT (WS_i, RS_j) : entier
Début
r := 1
Si $WS_i \cap RS_j \neq \varnothing$ **Alors**
r := 0
Fin Si
Retourner (r)
Fin.

4.3.2 - Validation sous BOM avec SC pour une RTT

Le test de validation de BOM avec SC comporte une SC durant laquelle il fait appel à la fonction "CONFLIT" et donne l'accord pour répercuter les écritures sur la BD. Toutes les transactions sont donc bloquées durant cette SC. Ce test est alors défini comme suit :

VALIDATION_ASC (T_i)
Début
/* Début de la section critique, annoncée par le symbole < */
< "j prend l'estampille de la première transaction qui suit T_i"
Tantque j <> "!" **Faire**
Si (CONFLIT (WS_i, RS_j) = 0) **Alors**
"Donner un ordre d'avortement de T_j : ROOLBACK T_j"
Fin Si
"j prend l'estampille de la transaction suivante"
Fin Tantque
"Donner l'accord pour répercuter les écritures sur la BD" >
/* Fin de la section critique, annoncée par le symbole > */
Fin.

4.3.3 - Comparaison des validations sous BOM avec et sans SC pour une RTT

Nous constatons que les deux tests de ces deux approches sont composés de deux lots de codes identiques : la vérification de l'absence de conflit et l'accord pour répercuter les écritures sur la BD. Cependant, ils diffèrent sur les deux points suivants :

- L'ordre d'exécution des deux lots : la première approche donne l'accord pour répercuter les écritures sur la BD, puis vérifie l'absence de conflit. La deuxième approche inverse ces lots. Mais cette différence n'a pas d'importance puisque les deux lots se sont avérés indépendants dans le contexte du nouvel algorithme,

contrairement aux contextes des deux variantes originales de BOM. En effet, puisque la transaction à valider ne peut être que la plus prioritaire, alors la répercussion des écritures sur la BD est à exécuter dans tous les cas et indépendamment du résultat de la vérification de l'absence de conflit. Par conséquent, l'ordre d'exécution de ces deux lots n'a pas ici d'influence sur le test de validation.

- La présence de la SC dans la deuxième approche. Vérifions si la présence de la SC est obligatoire ou non. Considérons deux transactions T_i et T_j où T_i est plus prioritaire que T_j. T_i arrive à sa phase de validation alors que T_j est encore dans sa phase de lecture et ces deux transactions ne sont pas, jusque là, en conflit. Supposons qu'après la validation de T_i, T_j effectue une opération de lecture d'un granule manipulé en écriture par T_i.

Si nous appliquons le test de validation avec BOM avec SC, T_i et T_j ne seront pas en conflit. L'opération de lecture de cette dernière sera effectuée sur la nouvelle version créée par T_i.

Si nous appliquons le test de validation avec BOM sans SC, T_i et T_j peuvent être signalées, à tort ou à raison, en conflit. En effet, en l'absence d'une SC, pendant que T_i n'est pas encore arrivée à vérifier l'absence de conflit avec T_j, cette dernière continue son exécution. Elle peut donc atteindre la dite opération de lecture. Cette lecture peut concerner d'une manière aléatoire :

- Soit la nouvelle version du granule créée par T_i ; dans ce cas, il n'existe pas d'incohérence.
- Soit la version précédente ; dans ce cas, il existe une incohérence.

Mais dans les deux cas, le test de validation signale l'existence d'un conflit car il utilise l'ensemble des objets lus de T_j (RS_j) après cette lecture. D'où une définition sévère du conflit.

Par conséquent, ceci implique que l'adaptation de l'approche de BOM sans SC peut conduire à des avortements non justifiés de transactions.

4.3.4 - Procédure de validation

Compte tenu de l'étude du paragraphe précédent, nous proposons de partir de l'approche BOM avec SC et d'imposer un ordre pour la validation des transactions afin de pouvoir garantir la cohérence forte dans le cadre de RTT. Aucune transaction ne peut être validée s'il existe des transactions qui la précèdent et qui n'ont pas été encore validées.

Après avoir validé une transaction T_i, on doit toujours vérifier s'il existe une transaction T_j en attente de certification qui est devenue la plus prioritaire. En effet, la mise en attente de certification de T_j est causée par l'existence d'autres plus prioritaires non encore validées. Deux possibilités peuvent avoir lieu :

- Si T_i était la seule transaction plus prioritaire que T_j, cette dernière, en passant de nouveau le test de certification, s'avère la plus prioritaire ; elle passe donc à la phase de validation.
- Autrement, le test de certification de T_j échoue et elle reste en attente de certification jusqu'à validation de toutes les transactions prioritaires.

Nous proposons alors d'ajouter une fonction de réveil qui sélectionne, le cas échéant, la transaction la plus prioritaire de l'ensemble des transactions mises en attente de certification pour repasser de nouveau le test de certification. Cette fonction doit être exécutée après chaque validation de toute transaction. Nous proposons donc de l'appeler juste après la procédure de validation. Elle est donc exécutée en dehors de la SC du test de validation, ce qui permet d'améliorer le parallélisme.

Il nous reste maintenant de traiter le risque de prolongement de la SC. Cette dernière s'étend durant les deux phases de validation et d'écriture de la transaction T_i. Durant cette période, tous les granules manipulés en écriture par T_i sont verrouillés. Comment alors peut-on réduire cette période ?

La réduction de cette période par l'exclusion de l'une des deux phases de la SC pose des problèmes. En effet :

- Durant la phase d'écriture, tout granule g manipulé en écriture par T_i (transaction à valider) doit être inaccessible par toute transaction concurrente T_j ; cette dernière n'a le droit ni de lire ni d'écrire g pour éviter toute incohérence. Par conséquent, la SC doit couvrir toute la durée de la phase d'écriture.

- Durant la phase de validation de T_i, les ensembles des objets lus des transactions concurrentes doivent être figés. Dans le cas contraire, une définition sévère du conflit peut apparaître.

Cependant, nous pouvons réduire la période de la phase de validation en faisant appel à la technique utilisée par les EOT marker pour une définition correcte des conflits. EOT marker se base sur le principe qu'une transaction concurrente n'a pas besoin d'être bloquée sur l'utilisation d'un granule pour pouvoir définir correctement un conflit. Nous pouvons alors tirer profit de cette technique afin d'assurer une définition correcte de conflit tout en garantissant la cohérence de la BD. Si nous utilisons EOT marker, nous n'aurons pas besoin de verrouiller la BD durant la phase de validation de T_i. Ceci est dû au fait que chaque transaction T_j en phase de lecture prend note de la fin de la transaction T_i à valider. Ainsi, quand cette dernière passe le test de certification avec succès, le contrôleur marque la fin de T_i dans l'ensemble des objets lus par T_j qui continue son exécution. Au moment de la vérification de l'absence de conflit T_i/T_j, l'ensemble des objets lus de T_j concerné par le test de validation se limite aux objets lus du début jusqu'à la marque de fin de T_i (EOT_i).

Appliquons cette technique sur l'exemple du paragraphe précédent, où nous avons constaté une définition sévère du conflit dans le cas de BOM sans SC. Il s'agit de deux transactions T_i et T_j où T_i est plus prioritaire que T_j. T_i arrive à sa phase de validation alors que T_j est encore dans sa phase de lecture et ces deux transactions ne sont pas, jusque là, en conflit. Supposons qu'après la validation de T_i, T_j effectue une opération de lecture d'un granule g manipulé en écriture par T_i.

Si nous intégrons EOT marker, le scénario d'exécution sera comme suit : au moment où T_i arrive à sa phase de validation, T_j en prend note dans son ensemble RS_j. Supposons que RS_j, avant cet instant est égal à {x, y, …, z}. Lorsque T_j prend note de la fin de T_i, RS_j devient {x, y, …, z, EOTi}. Une fois que T_j effectue l'opération Lire (T_j, g), le granule g s'ajoute à RS_j, devenant comme suit : {x, y, …, z, EOT_i, g}. Au moment où le contrôleur de concurrence effectue le test d'absence de conflit T_i/T_j, l'ensemble des objets lus de T_j, concerné par ce test, commence du début de RS_j jusqu'à EOT_i : c'est {x, y, …, z}. Ceci évite l'avortement, à tort, de T_j.

Puisque l'intégration de EOT marker permet de définir correctement les conflits sans avoir besoin de verrouiller la BD dans la phase du contrôle, nous proposons alors que la SC se restreint à la phase d'écriture et à la période de marquage de fin de transaction, sachant que cette dernière est beaucoup plus courte qu'une phase de validation. La procédure de validation que nous proposons est définie alors comme suit :

VALIDATION (T_i)
Début
/* Début de la SC */
< "Donner l'accord pour répercuter les écritures sur la BD"
"j prend l'estampille de la première transaction qui suit T_i"
Tantque j <> "!" **Faire**
"Ajouter l'élément EOT_i à RS_j"
Fin Tantque
> /* Fin de la SC */
"j prend l'estampille de la première transaction qui suit T_i"
Tantque j <> "!" **Faire**
/* $RS_j(T_i)$ est l'ensemble des objets lus de T_j jusqu'à EOT_i */
Si (CONFLIT (WS_i, $RS_j(T_i)$) = 0)
Alors
"Donner un ordre d'avortement de T_j : ROOLBACK T_j"
Fin Si
"j prend l'estampille de la transaction suivante"
Fin Tantque
Fin.

5 - ANALYSE D'UN CAS D'APPLICATION DE L'ALGORITHME PROPOSE

Considérons les granules x, y et z d'une RTT et les transactions TA, TB et TC arrivant au système respectivement aux instants t_1, t_2 et t_3. Ces transactions sont ainsi définies :

TA	TB	TC
Lire (x)	Lire (x)	Lire (x, pt1)
.	Lire (y)	Lire (y, pt2)
Ecrire (x)	.	.
Lire (z)	Ecrire (y)	Ecrire (y)
Ecrire (z)	.	Commit
Commit	Commit	

Supposons qu'il existe 5 versions du granule x, 7 versions du granule y et 3 versions du granule z. Supposons aussi que pt1 appartient à l'intervalle de transaction de la première version de x et pt2 appartient à celui de la troisième version de y.

Notons par ti le temps d'exécution d'une instruction et par g_m^n la version N° n du granule g qui a comme estampille m (le TT). Une exécution sérialisable de ces transactions, basée sur l'application du nouvel algorithme de contrôle de concurrence, peut être comme celle présentée au tableau 1.

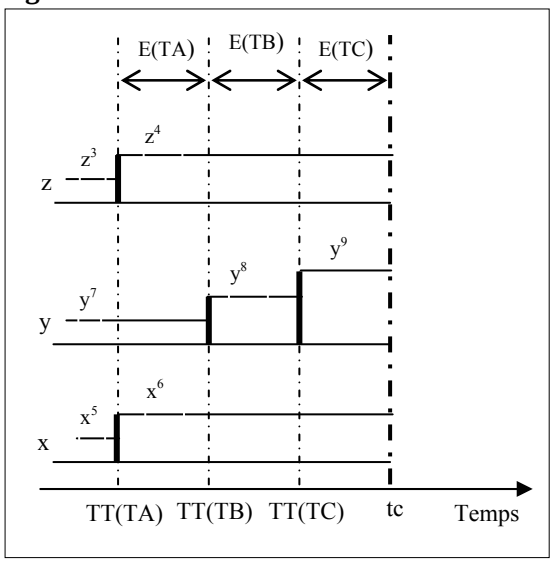
t	TA	TB	TC	Commentaires et suivi des RS et des WS
t_1	Arrivée de TA			
t_2		Arrivée de TB		
t_3			Arrivée de TC	
t_4	Lire (x)			$RS_A = \{x^5\}$
t_5		Lire (x)		$RS_B = \{x^5\}$
t_6			Lire (x, pt1)	pt1 désigne une ancienne version de x et non pas sa version courante : Exécuter la lecture sans prendre note dans RS_C
t_7	Ecrire (x)			$WS_A = \{x_{t1}\}$
t_8		Lire (y)		$RS_B = \{x^5, y^7\}$
t_9			Lire (y, pt2)	pt2 désigne une ancienne version de y et non pas sa version courante : Exécuter la lecture sans prendre note dans RS_C
t_{10}	Lire (z)			$RS_A = \{x^5, z^3\}$
t_{11}		Ecrire (y)		$WS_B = \{y_{t2}\}$
t_{12}			Ecrire (y)	$WS_C = \{y_{t3}\}$
t_{13}	Ecrire (z)			$WS_A = \{x_{t1}, z_{t1}\}$
t_{14}		Commit		Le test de certification de TB échoue car TA est plus prioritaire : mise en attente de TB.
t_{15}			Commit	Le test de certification de TC échoue car TA et TB sont plus prioritaires : mise en attente de TC.
t_{16}	Commit Validation de TA et création des nouvelles versions x_{t1}^6 et z_{t1}^4			- TA est la plus prioritaire, elle passe le test de certification avec succès. Elle est validée. $RS_B = \{x^5, y^7, EOT_A\}$ $RS_C = \{ EOT_A\}$ - $WS_A \cap RS_B \neq \varnothing$ - $WS_A \cap RS_C = \varnothing$ - Avortement de TB qui doit être reprise. - Exécution de la procédure Réveil : TC non certifiée car TB est plus prioritaire.
t_{17}		Lire (x)		$RS_B = \{x^6\}$
t_{18}		Lire (y)		$RS_B = \{x^6, y^7\}$
t_{19}		Ecrire (y)		$WS_B = \{y_{t2}\}$
t_{20}		Commit Validation de TB et création de la nouvelle version y_{t2}^8		- TB est la plus prioritaire, elle passe le test de certification avec succès. Elle est validée. $RS_C = \{EOT_A, EOT_B\}$ - $WS_B \cap RS_C = \varnothing$ - Exécution de la procédure Réveil : réveil de TC.
t_{21}			Commit Validation de TC et création de la nouvelle version y_{t3}^9	- TC est la seule transaction non encore terminée, elle passe le test de certification avec succès. Elle est validée. - Aucune transaction concurrente pour le test de validation. - Exécution de la procédure Réveil : aucune transaction en attente.

Tableau 1 : Exemple d'exécution de transactions concurrentes

La figure 2 donne l'ensemble des états pris par la base suite à cette exécution sérialisable. Sur cette figure :

- L'état E(TA) est formé par les nouvelles versions de x et de z (x^6et z^4) et l'ancienne version de y (y^7) qui continue à être courante. Cet état s'étend de TT(TA) jusqu'à TT(TB). Les nouvelles versions x^6et z^4 des granules x et z ont pris effet à partir de l'instant TT(TA).
- L'état E(TB) est formé par la nouvelle version de y (y^8) et des anciennes versions de x et de z (x^6 et z^4) qui continuent à être courantes. Cet état s'étend de TT(TB) jusqu'à TT(TC). La nouvelle version y^8 du granule y a pris effet à partir de l'instant TT(TB).
- L'état E(TC) est formé par la nouvelle version de y (y^9) et des anciennes versions de x et de z (x^6 et z^4) qui continuent à être courantes. Cet état s'étend de TT(TC) jusqu'au temps courant (tc). La nouvelle version y^9 du granule y a pris effet à partir de l'instant TT(TC).

Figure 2. Suivi des états cohérents de la RTT suite à l'exécution du cas d'application



6 - PROTOTYPE REALISE

Le prototype, réalisé sous VC++6.0, utilise un ensemble de structures de données pouvant être appelées par les différents modules du SBD. Ces structures sont des listes chaînées permettant de mémoriser toutes les informations relatives aux transactions arrivées au système. Le contrôleur de concurrence ACCO_CF/RTT se charge de la gestion des listes des objets lus et écrits par les transactions pour la vérification de la cohérence.

L'interface montre en détail l'avancement de l'exécution des opérations des différentes transactions. L'instruction sélectionnée est celle en cours d'exécution. Les états que peut avoir une transaction durant son exécution sont également affichées ("en exécution", "non certifiée", "Mise en attente", …). L'affichage des ensembles des objets lus et écrits des transactions montre leur évolution suite à l'application des règles de gestion et les éventuels conflits afin de pouvoir suivre la vérification de l'absence de conflits. Enfin, pour mieux suivre le fonctionnement attendu du prototype, la succession de l'exécution des différents modules du contrôleur de concurrence est affichée. Nous nous limitons ici à la présentation de l'image-écran qui correspond à l'instant d'exécution t_{21} où on arrive à Commit (TC).

Puisque TB est la plus prioritaire des transactions en phase de lecture, elle passe le test de certification avec succès. Puis, la marque de la fin de TB est notée dans l'ensemble des objets lus de TC (unique transaction encore en attente de certification). La vérification de l'absence de conflit TB/TC montre qu'il n'existe pas de conflit. Lorsque la fonction REVEIL ( ) est exécutée, TC est la seule transaction dans le SBD et donc la plus prioritaire. Elle est alors réveillée et elle passe le test CERTIFICATION (T_j) vis-à-vis des transactions non encore validées. Le résultat du test est positif, TC est donc certifiée et validée (figure 3).

7 - CONCLUSION

L'algorithme optimiste ACCO_CF/RTT traite du contrôle de concurrence (CC) dans le cadre des relations temporelles de transaction (RTT) et assure leur cohérence forte. La phase de certification n'autorise une transaction T à passer à la phase de validation que si elle est la plus prioritaire parmi toutes les transactions qui sont encore en phase de lecture ou en attente de certification. Cet ordre de priorité correspond à l'ordre défini par l'estampillage des transactions par les instants de leur arrivée. Une fois arrivée à la phase de validation, la transaction T est validée d'office. En cas de conflit avec une autre transaction T', forcément plus jeune que T, c'est T' qui est à

avorter. La répercussion des écritures sur la BD se déroule durant une section critique (SC) pour pouvoir maintenir la cohérence des RTT. Durant cette SC, l'accès à l'ensemble des granules manipulés par la transaction à valider est interdit pour toutes les transactions concurrentes. Cependant, la SC est réduite au maximum, elle ne comprend pas la période de recensement de conflits.

L'algorithme est caractérisé par un degré de parallélisme élevé dû à sa nature optimiste, à l'exploitation de versions multiples maintenues dans les RTT sans se charger de leur gestion et à l'intégration de la technique EOT Marker qui a permis de diminuer le temps de blocage des transactions par la réduction de la SC. Par ailleurs, la stratégie de validation basée sur les photos des transactions concurrentes, améliorée par la technique EOT Marker, permet, d'une part, d'accélérer la détection des conflits et, d'autre part, d'éviter le risque d'avortements non justifiés. Ceci améliore nettement les temps d'exécution des transactions.

Cet algorithme se limite aux RTT et ne peut pas être appliqué en tant que tel aux relations temporelles de validité et aux relations bitemporelles. Cependant, il nous ouvre la voie à des études plus approfondies du CC pour les BD temporelles. Par ailleurs, nous prévoyons d'étudier la robustesse de cet algorithme, à travers la vérification formelle sous l'environnement SPIN et le langage PROMERA.

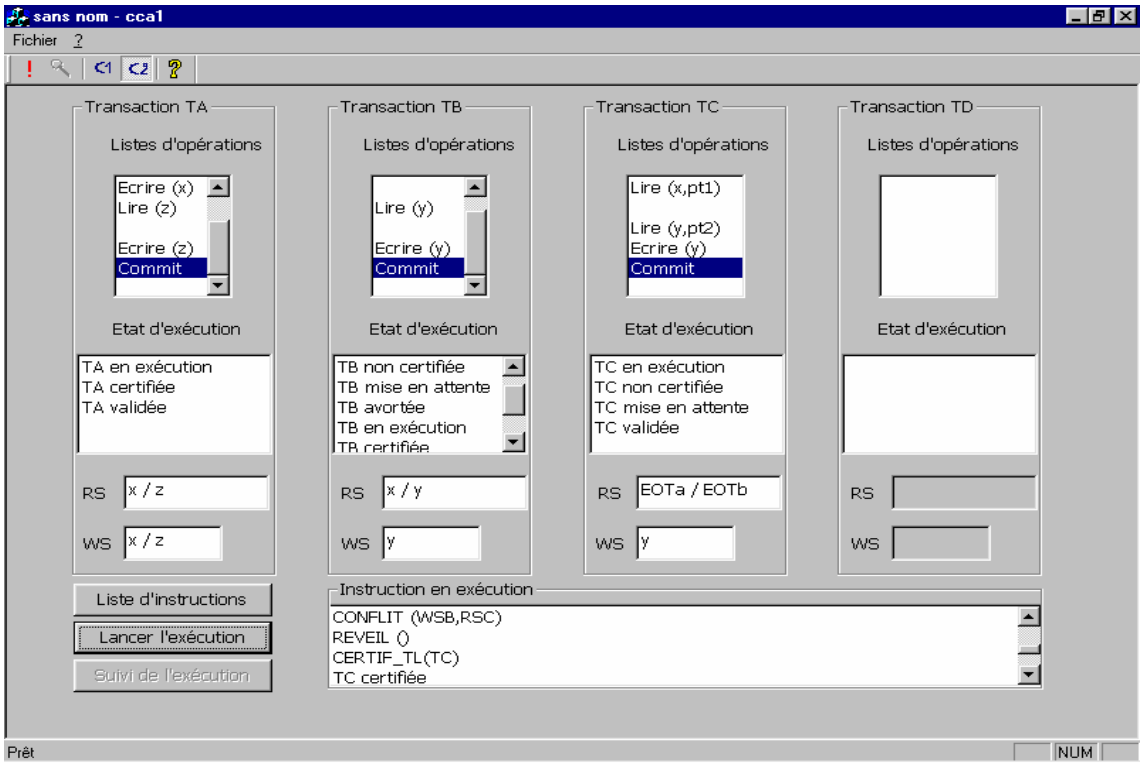


Figure 3. Réveil et validation de TC

Bibliographie

Amanton, L., Sadeg, B., Haubert, J. (2003), "Tradding Precision for Timeliness in Distributed Real-Time Databases", *International Conference on Enterprise Information Systems*, Vol 1, pp558-561.

Bernstein, P. A., Goodman, N. (1980), "Timestamp-based algorithms for concurrency control in distributed database systems", *Very Large DataBases* VLDB'80.

Bernstein, P. A., Hadzilacos, V., Goodman, N. (1987), "*Concurrency control and recovery in DBS* ", Edition ADDISON-WESLEY.

Bouaziz, R. (1991), "*Gestion temporelle et historisation des données dans les systèmes d'information*" , Thèse de doctorat de spécialité en informatique présentée à la Faculté des Sciences de Tunis, Université

des Sciences, des Techniques et de Médecine de Tunis.

Castro, C. (1998), "On concurrency management in temporal relational databases", *SEBD*, pp189-202.

Chan, M. Y. (1986), "Database concurrency control using READ/WRITE set information", in *Information Systems*, Vol 11, n° 4.

Elloumi, Dhouib, S., Bouaziz, R., Moalla, M. (1998), "Contrôle de concurrence multiversion dans les bases de données temporelles", *Bases de Données Avancées BDA'98*, pp135-155.

Finger, M., McBrien, P. (1997), "Concurrency Control for Perceivedly Instantaneous Transactions in Valid-Time Databases", in *TIME*, pp 112-118.

Gardarin, G. (1988), "*Bases de données : Les systèmes et leurs langages*", Edition EYROLLES.

Kumar, V. (1989), "A study of the behaviour of the read/write ratio under two-phase locking schemes", in *Information Systems*, Vol 14, n° 1.

Kung, H. T., Robinson, J. T. (1979) "On optimistic methods for concurrency control", *Very Large DataBases* VLDB'79.

Lee, V. C., Lam, K. W. (2000), "Conflict free transaction scheduling using serialisation graph for real-time databases", in *journal of Systems and Software*, Vol 55, n° 1, pp57-65.

Makni, A., Bouaziz, R. (2003), "Principe de base d'un algorithme optimiste de contrôle de concurrence d'accès", *Génie Electrique et Informatique*, Vol 1, pp207-211.

Menasce, D., Nakanishi, T. (1982), "Optimistic versus pessimistic concurrency control mechanisms in database management systems", in *Information Systems*, Vol 7, n° 1.

Miranda, S., Busta, J. M. (1986), "*L'art des bases de données : Les base de données relationnelles*", Edition EYROLLES.

Rahgozar, M. (1987), "*Contrôle de concurrence par gestion des événements*", Thèse de doctorat de l'Université de Paris 6.

Yu, S. P., Dias, D. M., Lavenberg, S. S. (1990), "*On modeling database concurrency control*", IBM Research Center, 195 RC 15386 (#68455).