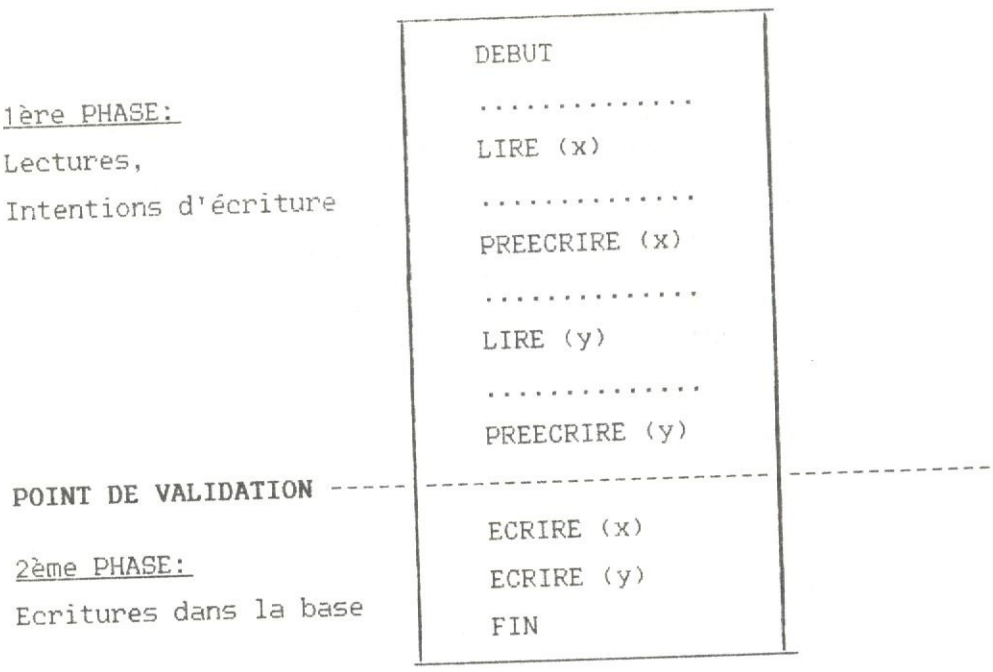
travail alloué à la transaction à cet effet.

Exemple:la transaction T ci-dessous doit lire et écrire les objets x et y.



I.4- TRANSACTION VIVANTE - TRANSACTION VALIDEE

Une transaction validée est une transaction qui s'est exécutée correctement et donc dont les mises à jour sur les objets de la base ont été rendues effectives. Dans la suite, nous noterons T* toute transaction T validée.

Par opposition à une transaction validée, une transaction sera dite vivante si elle est encore en cours d'exécution.

I.5- CONFLIT - DEPENDANCE

Un conflit naît quand, sur deux transactions voulant effectuer une lecture ou une préécriture sur un même objet, l'un des accès est une préécriture.

Il existe trois types de conflit:
- -le conflit LP (Lecture/Préécriture) survient quand la lecture d'un objet par une transaction précède sa préécriture par une deuxième.
- -le conflit PL (Préécriture/Lecture) est le cas inverse du précédent.
- -le conflit PP (Préecriture/Préécriture) se produit entre deux transactions qui veulent effectuer une préécriture sur le même objet.

Un conflit entre deux transactions engendre une dépendance entre elles. Cette dernière est soit effective, si un ordre entre les transactions est déjà imposé, soit potentielle, si cet ordre n'est fixé qu'à la validation de l'une d'entre elles: la dépendance potentielle deviendra effective à cette validation.

L'établissement des dépendances entre deux transactions vivantes T_1 et T_2 s'effectue en fonction de la chronologie de leurs opérations de lecture, de préécriture et de validation. Suivant les conflits, plusieurs cas peuvent

se présenter.

-si T_1 lit un objet x avant sa préécriture par T_2 (conflit LP), alors, T_1 précède T_2 dans l'ordre de sérialisation . Cette dépendance est effective et sera notée $T_1 \longrightarrow T_2$ ou encore $T_1 <_x T_2$ (T_1 précède T_2 au niveau de l'objet x).

-dans le cas où T_1 effectue une préécriture sur un objet avant sa lecture par T_2, le conflit PL consistera à mettre en attente T_2 jusqu'à la validation de T_1: ainsi, T_2 lira la valeur écrite par T_1 et on aura $T_1^* \longrightarrow T_2$. Si T_2 n'est pas mise en attente et effectue une lecture immédiate, on retrouve le schéma du conflit LP avec $T_2 \longrightarrow T_1$.

-le seul cas de dépendance potentielle est exprimé par le conflit PP: quand T_1 et T_2 effectuent une préécriture sur le même objet, l'ordre des écritures effectives dans la base ne sera déterminé qu'à la validation de l'une d'elles. Si la validation de T_1 est effectuée avant celle de T_2, on aura $T_1^* \longrightarrow T_2$ et dans le cas contraire, $T_2^* \longrightarrow T_1$.

I.6- GRAPHE DE DEPENDANCES

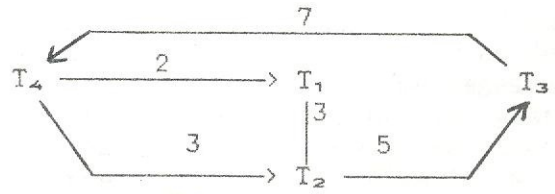
L'exécution d'un ensemble de transactions peut être visualisée par un graphe dont les noeuds représentent les transactions et les arcs les dépendances, effectives et potentielles, induites par leurs conflits: L'apparition d'un conflit relatif à une lecture ou à une préécriture entraîne le rajout d'un arc et le rejet d'une transaction signifie la suppression du noeud qui la figure ainsi que de ses arcs incidents.

L'existence d'un circuit dans ce graphe traduit une incohérence dans l'exécution concurrente de plusieurs transactions. Quelle que soit la méthode utilisée, le contrôle de la sérialisabilité revient à garantir que le graphe G* des transactions validées (terminées et non rejetées) est sans circuit.

Exemple: soient les transactions T_1, T_2, T_3 et T_4.

T_1	T_2	T_3	T_4
2 PREECRIRE x	3 PREECRIRE x		1 LIRE x
	4 LIRE y	5 PREECRIRE y	
		6 LIRE z	7 PREECRIRE z
8 valider T_1			

A l'instant 7 et après l'exécution de l'instruction PREECRIRE (z), le graphe comportera quatre dépendances effectives se rapportant toutes à des conflits LP et une dépendance potentielle qui correspond au conflit PP existant entre T_1 et T_2.



N d' Ordre :
N SIRC :

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE (U. S. T. H. B.) ALGER

INSTITUT D' INFORMATIQUE

THESE

PRESENTEE A L'U.S.T.H.B. POUR L'OBTENTION DU GRADE DE
MAGISTER EN INFORMATIQUE
PAR

Mme NORA-LISA BELKHODJA NEE KHIMECHE

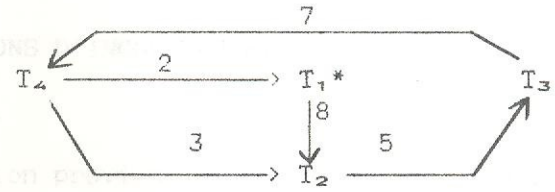
CONTROLE DE CONCURRENCE DANS LES SYSTEMES REPARTIS

METHODE MULTIVERSION PAR INTERVALLES D'ESTAMPILLES

SOUTENUE PUBLIQUEMENT LE : 03 OCTOBRE 1989
DEVANT LE JURY COMPOSE DE MESSIEURS :

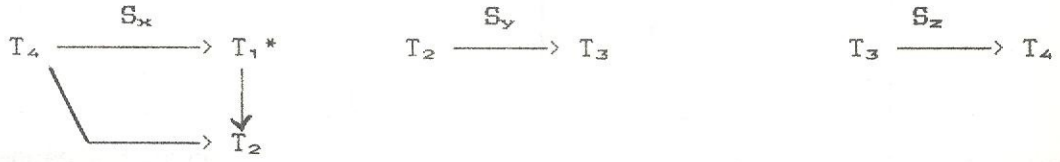
KHELLADI	A.	PROFESSEUR	U.S.	PRESIDENT
BEN-YELLES	C.B.	PROFESSEUR	U.S.T.H.B.	RAPPORTEUR
ATROUN	M.A.	CHARGE DE COURS	U.S.T.H.B.	EXAMINATEUR
BADACHE	N.	CHARGE DE COURS	U.S.T.H.B.	EXAMINATEUR
PONS	J.F.	MAITRE DE CONFERENCES	U.S.T.L.	EXAMINATEUR

A l'instant 8, lors de la validation de T_1, cette dépendance potentielle devient effective et sera schématisée par



La validation des trois autres transactions donnera alors lieu à une exécution concurrente non équivalente à une exécution série et cela provoquera l'incohérence de la base. Pour rendre cette exécution sérialisable, il suffira de casser le circuit en rejetant une des trois transactions non encore validées.

Plaçons cet exemple dans le contexte d'un environnement réparti où chacun des objets x, y et z se trouve sur son site propre. Chacun des trois sites n'appréhendera qu'une partie du graphe G et ne peut donc, à partir de son graphe local, prendre une décision correcte quant à la sérialisation. A l'instant 8, on aura la situation ci-après:



A la validation de T_2, T_3 et T_4 et bien qu'il y ait une incohérence de la base suite à une exécution non sérialisable, les graphes locaux ne contiendront pas de circuit.

REMARQUES:

Dans le contexte d'un environnement réparti, un site donné n'aura qu'une connaissance partielle du graphe de dépendances. Par le biais d'un échange de messages, il ne pourra qu'approcher la connaissance de son ensemble: entre l'émission et la réception d'un message, le graphe aura évolué. C'est la contradiction propre à tout système distribué ("Existence d'opérations globales / Absence d'état global perceptible" [RAYN 85]).

Le graphe de sérialisation d'une exécution est un outil très utilisé dans les preuves et démonstrations d'algorithmes. Cependant, d'un point de vue pratique, sa mise en oeuvre reste, sinon impossible, du moins très contraignante.

Sa gestion par un seul site donnera lieu à un échange de messages, avec les autres sites, assez important pour générer un goulot d'étranglement au niveau de ce site privilégié. De plus, et sans compter les éventuelles défaillances de ce site -pannes-, cette gestion centralisée pénalise le parallélisme.

Le trafic de messages entre sites ne serait pas moins intense dans le cas où une copie du graphe serait maintenue sur chacun d'eux. Un état identique de ces copies sur tous les sites ne s'obtient qu'au détriment du niveau de parallélisme, les délais de transmission des messages étant variables. Parmi les solutions qui ont été proposées pour la gestion de ces copies locales, on peut citer l'exclusion mutuelle entre sites et les algorithmes de maintien de la cohérence de copies multiples [CORN 81].

d'une transaction dépendra de celle de la transaction non validée dont elle aura lu la sortie.

-l'approche de [AGRA 86] n'effectue pas de contrôle sur les transactions lectrices et les prend en compte lors des validations des transactions écrivains.

Les techniques multiversions présentées diminuent les possibilités de rejet mais héritent de tous les inconvénients liés aux techniques classiques qu'elles utilisent: ainsi, l'estampillage statique peut être à l'origine de rejets de transactions écrivains.

Nous avons ensuite élaboré la méthode MPIE en étendant la technique des intervalles d'estampilles aux méthodes multiversions. Nos principales motivations ont été la diminution des rejets et l'amélioration du parallélisme. La technique des intervalles introduit un caractère dynamique dans le calcul de l'ordre de sérialisation qui est construit au fur et à mesure des accès aux versions des objets et permet un ordre de sérialisation global proche de l'ordre réel induit par les dépendances entre les transactions. Par rapport aux méthodes multiversions classiques, où il est fixé à priori, cela représente un avantage certain.

D'un autre côté, notre méthode présente un niveau de parallélisme plus élevé que celui d'une méthode monoversion: elle offre aux transactions de lecture un choix de plusieurs versions pour un même objet et permet un ordre de sérialisation des transactions différent de l'ordre chronologique de leurs certifications.

Nous avons résolu les cas de rejets de lectrices pures en permettant à leurs opérations de lecture d'accéder à un nombre fixé de versions, à travers l'utilisation d'un ensemble d'intervalles courants, et en reportant le contrôle sur les transactions écrivains.

Ceux-ci effectuent un contrôle en avant en phase de certification locale: ils tronquent leurs propres intervalles pour laisser aux lectrices pures de plus grandes possibilités de sérialisation.

En dernier lieu et toujours pour améliorer ses performances, nous avons proposé une adaptation de notre méthode à un modèle de transaction parallèle. La suppression du point de synchronisation de la fin des phases de lectures locales augmentera le parallélisme, mais rendra la méthode moins adaptée aux transactions de type interactif (CAO).

Perspectives:

Un prolongement de ce travail pourrait être l'implantation de la méthode que nous avons proposée dans un environnement physiquement distribué et la réalisation d'une étude comparative avec les principales autres méthodes de contrôle de concurrence.

Cette implantation devrait s'effectuer sur les différents modèles de transaction présentés, de manière à mesurer l'impact de la présence ou de l'absence des différents points de synchronisation sur le comportement de la méthode.

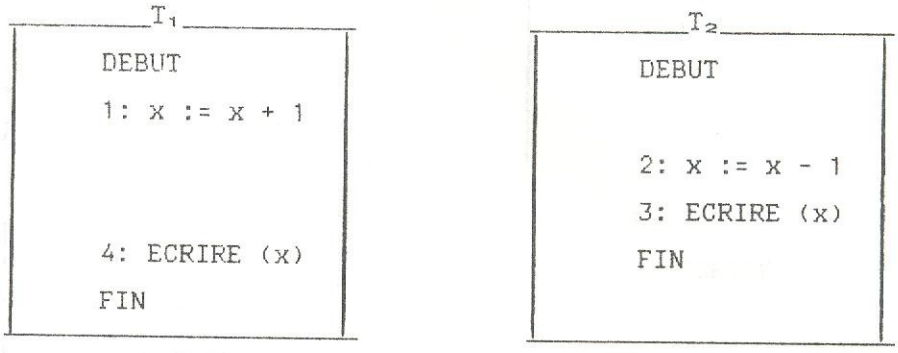
Plus généralement, une des principales perspectives dans le domaine du contrôle de la concurrence reste l'approche sémantique des opérations ([WEIH 83], [HERL 86]) qui débouche sur l'utilisation des types abstraits. Dans le modèle que nous avons utilisé, la transaction est séquentielle et se présente comme une suite d'opérations de lectures et d'écritures. Nous n'avons émis d'hypothèses, ni sur la nature des opérations, ni sur celle des objets accédés.

Toutes les raisons évoquées ci-dessus font qu'aucun algorithme développé jusqu'à présent ne s'est appuyé directement sur le graphe de sérialisation.

I.7- EXEMPLES DE SITUATIONS D'INCOHERENCE

I.7.1- PERTE D'OPERATION.

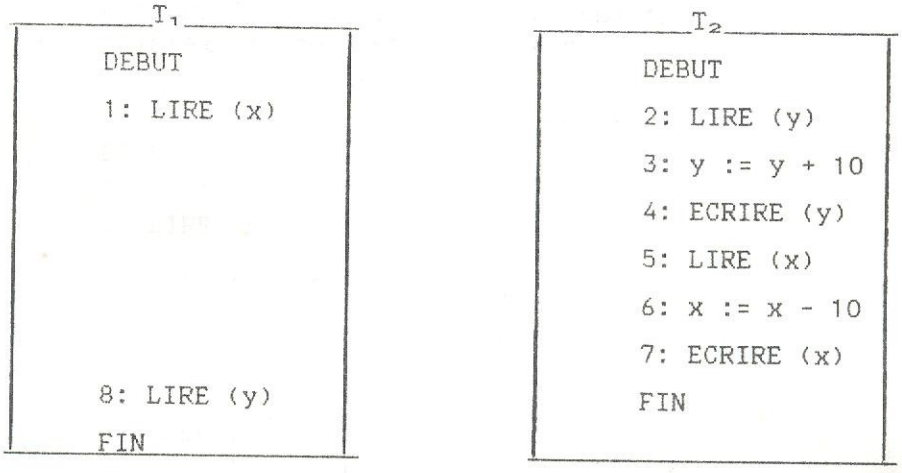
La perte d'opération provient du caractère de simultanéité qui définit l'exécution concurrente de plusieurs transactions: la modification d'un objet de la base effectuée par une transaction est détruite par celle qu'exécute simultanément une deuxième transaction. Les accès concurrents non contrôlés peuvent donc mener à des pertes de mises à jour sur les données. Prenons un exemple de deux transactions T_1 et T_2 accédant simultanément à la même donnée x.



Cet accès concurrent à la donnée x conduit à perdre la mise à jour effectuée par la transaction T_1. La cohérence de la base de données a été cependant maintenue.

I.7.2- SITUATION D'INCOHERENCE.

La cohérence des données repose sur le respect des contraintes d'intégrité qui relient les objets de la base. La non vérification de ces contraintes lors d'une exécution de transactions peut mener à des résultats incohérents. Soient T_1 et T_2 deux transactions accédant aux objets x et y reliés par la contrainte d'intégrité x + y = 100.



x + y = 110 (pour T_1)

x + y = 100 (sur la base)

La cohérence de la base a été maintenue mais T_1 a perdu sa vue cohérente: T_1 a lu x avant que T_2 ne le mette à jour. On a donc, au terme de l'exécution de T_1, x + y = 110. Cette altération de la contrainte d'intégrité provient du fait que T_1 et T_2 ne sont pas sérialisables (T_1 précède T_2 pour x et T_2 précède T_1 pour y).

Remarque: Si aucune contrainte d'intégrité ne lie les objets accédés par les transactions, la sérialisabilité peut s'avérer trop forte comme condition de maintien de la cohérence.

I.7.3- PERTE DE LA COHERENCE

Les transactions T_1 et T_2 ci-après accèdent aux objets x et y. Ces deux objets sont reliés par la contrainte d'intégrité x = y.

T_1	T_2
DEBUT	DEBUT
1: x := x + 1	
2: ECRIRE (x)	3: x := x * 2
	4: ECRIRE (x)
	5: y := y * 2
	6: ECRIRE (y)
7: y := y + 1	FIN
8: ECRIRE (y)	
FIN	

Au terme de ces exécutions, la contrainte x = y se vérifie pour chacune des deux transactions puisqu'elles ont fait subir à x et à y, chacune de son côté, les mêmes modifications. Néanmoins, il y a eu perte de cohérence au niveau de la base de données tout entière car l'exécution simultanée des deux transactions a conduit à l'état incohérent $x \neq y$.

I.7.4- SERIALISABILITE ET COHERENCE (ASPECT SEMANTIQUE)

Si on tient compte de la sémantique des opérations, la sérialisabilité des transactions devient une condition de cohérence trop forte. Prenons deux transactions T_1 et T_2 accédant aux objets x et y et soit (x R y) la contrainte d'intégrité reliant ces objets.

T_1	T_2
DEBUT	DEBUT
.....	
2: LIRE (x)	
.....	
4: ECRIRE (x)	
	5: ECRIRE (x)
	6: ECRIRE (y)
7: LIRE (y)	FIN
.....	
9: ECRIRE (y)	
FIN	

Chacune de ces transactions doit assurer, à son niveau, le respect de la contrainte d'intégrité. Aussi, à l'instant 9, la transaction T_1 va modifier l'objet y de façon à conserver (x R y) relativement à la valeur de x qu'elle aura écrite à l'instant 4. Or cette valeur n'a plus cours puisque l'objet x a été remodifié à l'instant 5 par T_2. L'exécution concurrente de T_1 et T_2 n'est pas sérialisable puisque T_1 précède T_2 pour x et T_2 précède T_1 pour y. Son déroulement altèrerait donc la cohérence de la base.

Considérons maintenant la nature des opérations et supposons que T_1, à l'instant 9, donne à y la même valeur que T_2 à l'instant 6: on remarque immédiatement que l'état final de la base demeure cohérent, bien que l'exécution parallèle de T_1 et T_2 ne soit pas sérialisable.

CHAPITRE II

CONTROLE DE CONCURRENCE CONTINU

II.1 Verrouillage à deux phases
II.2 Estampillage statique

II.3 Estampillage dynamique
II.3.1 Intervalles d'estampilles

II.4 Conclusion

Les méthodes basées sur le contrôle de concurrence continu supposent une fréquence élevée des conflits et sont qualifiées de méthodes pessimistes. Elles vérifient la sérialisabilité tout au long de la vie de la transaction. L'approche pessimiste consiste à éviter les exécutions non sérialisables par l'attente ou le rejet lors de chaque opération effectuée par une transaction.

Elles n'admettent pas la notion de dépendance potentielle: en cas de conflit PP, la dépendance qui se produit entre deux transactions est immédiatement rendue effective par le choix d'un ordre entre elles. On impose donc, dès la préécriture, un ordre de validation et celle-ci ne pourra plus modifier les dépendances. Cela représente un avantage mais le respect de l'ordre préétabli augmente de façon notable le nombre de rejets ou de mises en attente des transactions.

Les techniques qu'utilisent généralement ces méthodes sont le verrouillage à deux phases (verrouillage 2 PL), l'estampillage statique et l'estampillage dynamique.

II.1- VERROUILLAGE A DEUX PHASES [ESWA 76], [TRAI 82]

Cette technique s'appuie sur le principe de la mise en attente et du rejet des transactions en conflit: une transaction n'accède à un objet, qu'après avoir obtenu un verrou.

L'ensemble des transactions vivantes sur les sites doivent se conformer aux deux règles:

-Deux transactions ne peuvent verrouiller le même objet.

-Tous les verrous détenus par une transaction doivent être libérés à la fin de sa phase de validation.

Ce protocole qui s'exécute en deux temps est qualifié de protocole 2 phases.

-La première phase consiste en l'acquisition par la transaction de tous les verrous des objets qu'elle doit accéder. Le terme de cette phase constitue "le point de verrouillage maximum" et correspond au point de validation.

-La deuxième phase comprend les écritures dans la base et la libération des verrous.

Le verrouillage des objets s'effectue en fonction de la chronologie des demandes d'accès, quelle que soit la nature du conflit. La transaction détentrice du verrou mettra alors en attente, jusqu'à sa validation, toutes celles qui doivent accéder au même objet.

Aucune transaction vivante ne pourra ainsi précéder une transaction validée et l'ordre de sérialisation des transactions correspondra à l'ordre chronologique de leurs validations.

Des attentes entre transactions en conflit peuvent se créer inutilement: si on prend deux transactions T_1 et T_2 telles que T_1 effectue une préécriture sur l'objet x avant que T_2 ne la lise -T_1 a donc déjà verrouillé x-,

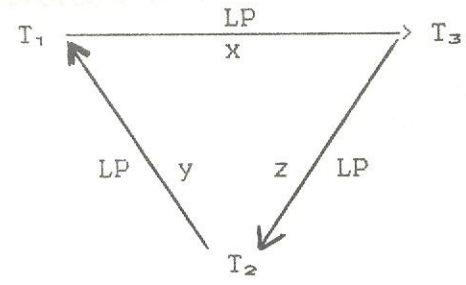
on voit que le conflit PL engendré ne peut être résolu que par la mise en attente de T_2 jusqu'à la validation de T_1. Sous un autre protocole, il aurait pu être solutionné sous la forme du conflit LP, en laissant T_2 effectuer une lecture immédiate de l'objet x, sans pour cela altérer la cohérence de la base.

D'un autre côté, des mises en attente "en cascade" peuvent être générées: une transaction verrouille une partie des objets qu'elle doit accéder et retarde une deuxième transaction alors qu'elle est elle-même en attente d'autres verrous. On aboutit ainsi très facilement à des situations d'interblocage.

L'utilisation de cette technique ne peut pas induire des incohérences sur la base puisque l'existence d'ordres sérialisables incompatibles donne lieu invariablement à une situation d'interblocage. Prenons par exemple les transactions T_1, T_2 et T_3 telles que:

T_1	T_2	T_3
1 LIRE x		
	2 LIRE y	
		3 LIRE z
4 PREECRIRE y		
	5 PREECRIRE z	
		6 PREECRIRE x

L'exécution concurrente de ces trois transactions n'est pas sérialisable puisque T_1 précède T_3 pour l'objet x, T_2 précède T_1 pour y et T_3 précède T_2 pour z. Cette non sérialisabilité provoque, à l'instant 6, une situation d'interblocage sur le graphe des dépendances des transactions vivantes.



. T_1 attend la validation de T_2

. T_2 attend la validation de T_3

. T_3 attend la validation de T_1

II.2- ESTAMPILLAGE STATIQUE.

Un ordre de sérialisation exprimé au moyen d'estampilles est fixé à priori [BERN 81]. On affecte à chaque transaction une estampille et on se base sur la valeur de celle-ci pour exprimer, de manière systématique, une dépendance entre chaque couple de transactions.

En prenant t_i pour estampille de la transaction T_i et t_j pour celle de T_j, la dépendance entre T_i et T_j sera établie comme suit:

$$t_i < t_j \quad ==> \quad T_i \longrightarrow T_j$$

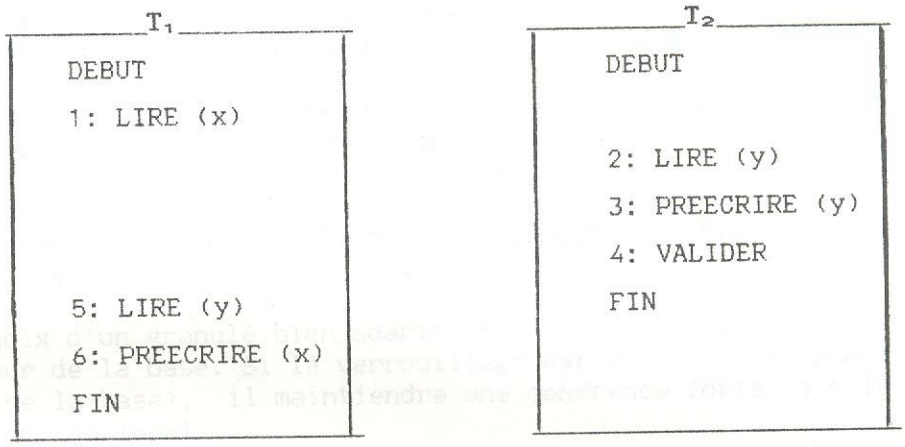
Une estampille d'écriture E(x) et une estampille de lecture L(x) sont gé-

rées au niveau de chaque objet x. E(x) correspondra à l'estampille de la dernière transaction validée ayant écrit l'objet x et L(x), à celle de la transaction la plus récente qui l'aurait lu.

Les accès aux objets seront ensuite traités en fonction de ces deux estampilles, c'est-à-dire, en fonction de l'ordre de sérialisation imposé à priori. Ainsi, une transaction qui doit effectuer une lecture sur l'objet x sera rejetée si son estampille est inférieure à E(x) et une opération de préécriture ne sera acceptée que si l'estampille associée majore max(L(x),E(x)).

Il y a donc risque de privation puisqu'une transaction peut être rejetée indéfiniment. De plus, et du fait de l'imposition systématique d'une dépendance à chaque couple de transactions, on peut avoir des rejets sans qu'il y ait conflit.

Exemple:
Soient deux transactions T_1 et T_2 et soient t_1 et t_2 les estampilles qui leurs sont associées telles que $t_1 < t_2$.



A l'instant 5, l'opération LIRE (y) de T_1 provoquera le rejet de cette transaction puisque $E(y) = t_2$ et $t_1 < t_2$.

Ce rejet ne se justifie pas puisque le conflit PL $T_2^* \longrightarrow T_1$ induit par l'opération LIRE (y) de T_1 n'engendre pas de circuit dans le graphe des dépendances.

II.3- ESTAMPILLAGE DYNAMIQUE

L'estampillage dynamique se propose de pallier aux inconvénients de l'estampillage statique et attend l'apparition du premier conflit d'une transaction pour lui affecter une estampille qui tiendra compte de la dépendance induite par ce conflit.

Cependant, les problèmes que pose l'estampillage statique ne sont que reportés à ce premier conflit puisqu'après, une dépendance est établie entre la transaction conflictuelle et toutes les autres.

II.3.1- INTERVALLES D' ESTAMPILLES.

Cette technique consiste à associer à chaque transaction vivante un intervalle d'estampilles.

Soient deux transactions T_i et T_j, $[a_i, b_i]$ l'intervalle associé à T_i et $[a_j, b_j]$ celui de T_j. La dépendance imposée entre T_i et T_j sera exprimée comme suit:

- $b_i < a_j$ ==> $T_i \longrightarrow T_j$
- $b_j < a_i$ ==> $T_j \longrightarrow T_i$

Si les intervalles de deux transactions en conflit sont disjoints, ils sont déjà ordonnés et comme pour l'estampillage statique, cet ordre sera respecté. S'ils ne le sont pas, une valeur sera fixée arbitrairement dans leur intersection et servira à les tronquer selon la dépendance voulue.

II.4- CONCLUSION

Deux problèmes se posent au niveau de la technique du verrouillage 2 PL, celui de la granularité des verrous et celui du type de verrouillage.

La granularité des verrous désigne le mode d'affectation de ces derniers dans la base. Le verrou peut porter sur un seul objet, un t-uple, une partie ou la totalité de la base.

Le type de verrouillage indique le degré d'affectation d'un verrou à un objet de la base: on peut verrouiller toutes les copies de l'objet, certaines seulement ou, à la rigueur, aucune.

Le niveau de parallèlisme est largement tributaire des solutions apportées.

Le choix d'un granulé bien adapté (t-uple, variable) revient à l'administrateur de la base. Si le verrouillage est global (concernant tous les objets de la base), il maintiendra une cohérence forte mais le parallèlisme sera nul ou local.

Le parallélisme sera nettement plus élevé si le maintien de la cohérence utilise un verrou logique local qui consisterait en la gestion d'une file d'attente de transactions au niveau de chaque site [HERM 79], [BERN 81].

La gestion des verrous peut devenir rapidement assez lourde et le temps de réponse prohibitif [ATRO 84].

D'autre part, le contrôle continu s'effectue au niveau de chaque lecture/écriture d'une transaction, ce qui dégrade les performances et ne s'avère utile que si la probabilité de conflit est forte.

Suite à ces contraintes, d'autres méthodes de contrôle qui, elles, "laissent aller la transaction" et ne vérifient la cohérence qu'au niveau de la validation, ont vu le jour: c'est le contrôle de concurrence par certification.

S O M M A I R E

INTRODUCTION..... 8

CHAPITRE I- CONCEPTS ET EXEMPLES..... 11
I.1- Sérialisabilité..... 13
I.2- Recouvrabilité - Atomicité..... 13
I.3- Modèle de transaction..... 13
I.4- Transaction vivante - Transaction validée..... 14
I.5- Conflit - Dépendance..... 14
I.6- Graphe de dépendances..... 15
I.7- Exemples de situations d'incohérence..... 17
I.7.1- Perte d'opération..... 17
I.7.2- Situation d'incohérence..... 17
I.7.3- Perte de la cohérence..... 18
I.7.4- Sérialisabilité et cohérence (aspect sémantique)..... 18

CHAPITRE II- CONTROLE DE CONCURRENCE CONTINU..... 20
II.1- Verrouillage à deux phases..... 22
II.2- Estampillage statique..... 23
II.3- Estampillage dynamique..... 24
II.3.1- Intervalles d'estampilles..... 24
II.4- Conclusion..... 25

CHAPITRE III- CONTROLE DE CONCURRENCE PAR CERTIFICATION..... 26
III.1- Certification par ensembles d'objets..... 28
III.1.1- Etude en environnement centralisé:
méthode de KUNG & ROBINSON..... 28
III.1.2- Etude en environnement réparti..... 30
III.1.2.1-Méthode de Ceri & Owiki..... 30
III.1.2.2-Modèle de transaction répartie..... 31
III.1.2.3-Modèle de certification-validation répartie... 31
III.2- Concurrence et certification..... 33
III.3- Construction d'un ordre de sérialisation global..... 36
III.4- Conclusion..... 37

CHAPITRE IV- CERTIFICATION PAR INTERVALLES D'ESTAMPILLES 39
IV.1- Structure des données..... 42
IV.2- Accès aux objets..... 43
IV.3- Certification - validation répartie..... 43
IV.4- Conclusion..... 45

CHAPITRE III

CONTROLE DE CONCURRENCE PAR CERTIFICATION

III.1 Certification par ensembles d'objets
III.1.1 Etude en environnement centralisé: méthode de KUNG & ROBINSON

III.1.2 Etude en environnement réparti
III.1.2.1 Méthode de Ceri & Owiki
III.1.2.2 Modèle de transaction répartie
III.1.2.3 Modèle de certification-validation répartie

III.2 Concurrence et certification
III.3 Construction d'un ordre de sérialisation global
III.4 Conclusion

Les méthodes optimistes ou par certification permettent à la transaction d'exécuter librement ses accès en lecture ou en préécriture. Elles ne procèdent au contrôle que lors d'une étape finale de certification. La transaction est alors rejetée s'il s'avère qu'elle peut provoquer une incohérence dans la base ou qu'elle a lu des données non cohérentes.

Ces méthodes utilisent donc un schéma de transaction à deux ou trois phases, selon le cas: une phase de lecture/préécriture, une phase de certification et éventuellement une phase de validation.

La transaction effectue ses accès aux objets durant la première phase. Celle-ci est toujours déroulée entièrement: on laisse "aller" la transaction et on lui permet de manipuler des données incohérentes.

La transaction est contrôlée lors de sa phase de certification. On vérifie alors si ses lectures se sont effectuées sur des données cohérentes et si ses préécritures ne risquent pas d'entraîner une incohérence dans la base. Si la certification échoue, la transaction est rejetée. Sinon, elle rend les résultats de ses lectures à l'utilisateur et, dans le cas où elle a effectué des préécritures, la phase de validation locale est initiée.

Durant cette dernière phase, les préécritures de la transaction sont transformées en écritures dans la base.

III.1- CERTIFICATION PAR ENSEMBLE D'OBJETS

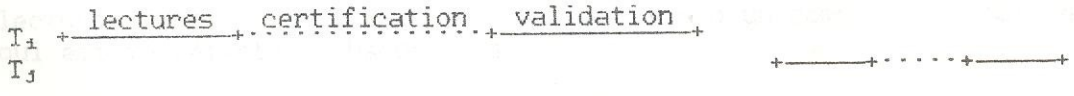
III.1.1- ETUDE EN ENVIRONNEMENT CENTRALISE: METHODE DE KUNG & ROBINSON

Cette méthode a été présentée dans [KUNG 81]. Elle utilise le schéma de transaction optimiste à trois phases: accès aux objets, certification et, éventuellement, validation. Son principe est qu'à chaque transaction, on fait correspondre une estampille unique et deux ensembles d'objets. L'un, noté O_L, comprendra les objets accédés en lecture et l'autre, noté O_E, ceux accédés en écriture. Les dépendances existant entre deux transactions sont alors données par l'interaction des ensembles d'objets qui leurs sont associés.

Pour certifier une transaction T, cette méthode ne tient compte que de ses dépendances avec les transactions déjà validées. Elle se base sur le graphe G* des transactions validées et vérifie que G_T*, composé de G* et de la transaction vivante T, est sans circuit.

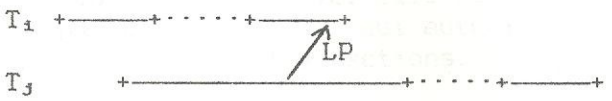
Pour que deux transactions T_i et T_j, d'estampilles t_i et t_j avec $t_i<t_j$, soient sérialisables, c'est-à-dire pour que leur exécution concurrente soit équivalente à leur exécution série, l'une des quatre alternatives suivantes doit être vérifiée.

CAS 1: T_i termine sa phase d'écriture avant que T_j ne débute sa phase de lecture. Cela correspond à une véritable exécution série des deux transactions qui sont totalement indépendantes "dans le temps". On ne fera peser aucune contrainte sur les ensembles d'objets.

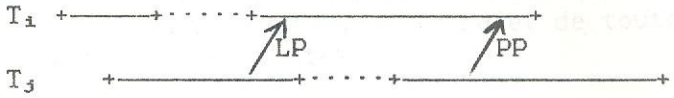


CAS 2: T_i termine sa phase d'écriture avant que T_j ne débute la sienne. Dans ce cas, il existe un parallèlisme entre les deux transactions: T_j effectue ses lectures avant que T_i n'ait terminé sa phase de validation donc avant que les préécritures de T_i ne soient répercutées dans la base.

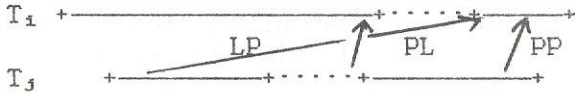
La transaction T_j ne doit pas accéder en lecture à un objet préécrit par T_i (conflit LP) car cet accès s'exécutera sur une donnée que la mise à jour de T_i rendra "dépassée". Pour cela, on devra restreindre les relations sur les ensembles d'objets des deux transactions et imposer $O_E(T_i) \cap O_L(T_j) = \emptyset$.



CAS 3: T_i termine sa phase de lecture avant que T_j ne termine la sienne. Dans ce cas et en plus du conflit LP du deuxième cas (T_j ne doit pas lire un objet préécrit par T_i), on a une occurrence de conflit PP: T_j ne doit écrire aucun objet avant que T_i ne l'écrive. Le parallèlisme est plus fort et les relations entre les ensembles d'objets de T_i et T_j plus restreintes. La seule intersection non vide permise est $O_L(T_i) \cap O_E(T_j)$.



CAS 4: Il n'y a plus de restrictions dans l'ordre de déroulement des phases des deux transactions, le parallélisme entre T_i et T_j est maximum et on ne permet plus aucune intersection d'ensembles d'objets non vide. On se trouve devant un troisième type de conflit: le conflit PL. La transaction T_j ne doit pas effectuer une préécriture sur un objet lu précédemment par T_i.



MISE EN OEUVRE DE LA METHODE:

Dans cette méthode, l'ordre de sérialisation des transactions correspond à l'ordre chronologique de leur entrée en phase de certification. Deux modes de validation peuvent être envisagés: la validation série et la validation parallèle.

-**Validation série:** Elle correspond à la mise en oeuvre des alternatives 1 et 2 et s'adapte bien à un environnement monoprocesseur avec une phase d'écriture en mémoire principale. La certification d'une transaction correspond au contrôle des dépendances qui la lient aux transactions qui sont en phase de validation durant sa phase de lecture.

Chaque transaction reçoit son estampille à la fin de sa phase de lecture. Cet estampillage se fait au moyen d'un compteur global tnc qui est incrémenté à chaque fois.

Chaque transaction gère à son niveau deux autres compteurs, t_début et t_fin. Ces deux compteurs recoivent la valeur de tnc à deux instants différents: t_début au démarrage de T et t_fin à la fin de sa phase de lecture. Les transactions qui sont en phase d'écriture durant la phase de lecture de T seront donc celles d'estampilles comprises entre (t_début+1) et t_fin.

La transaction T sera rejetée s'il existe une transaction T', d'estampille comprise entre (t_début+1) et t_fin, telle que $O_E(T') \cap O_L(T) \neq \emptyset$

-Validation parallèle: Elle est relative à la mise en oeuvre des alternatives 3 et 4 qui autorisent le parallélisme entre les phases d'écritures des transactions.

Chaque transaction T gère l'ensemble des transactions qui lui sont concurrentes et qui la précèdent dans l'ordre de sérialisation. Son rejet découlera ensuite de l'existence d'une transaction T' appartenant à cet ensemble et telle que $O_E(T') \cap [O_L(T) \cup O_E(T)] \neq \emptyset$.

L'inconvénient est que la transaction T' est une transaction vivante. Après avoir provoqué le rejet de T, elle peut être elle-même rejetée par la suite. Les auteurs proposent la mise en attente de la transaction T et le report de la décision quant à sa validation ou son rejet jusqu'à la validation ou le rejet de toutes les transactions T' qui lui sont concurrentes.

III.1.2- ETUDE EN ENVIRONNEMENT REPARTI

III.1.2.1- METHODE DE CERI & OWIKI [CERI 82]

Ceri et Owiki ont étendu la méthode de Kung & Robinson à un environnement réparti. Chaque site assure à son niveau la sérialisabilité des transactions selon l'ordre d'arrivée des demandes de certification. La certification locale d'une transaction T correspond aux contrôles suivants.

CAS 1: S'il existe une transaction T_i telle que $O_E(T_i) \cap O_L(T) \neq \emptyset$, et que T_i ait effectué une préécriture durant la phase de lecture de T, alors, T sera rejetée (cas 2 de Kung & Robinson).

CAS 2: T sera aussi rejetée s'il existe une transaction T_i précédant T dans l'ordre d'arrivée sur le site -donc dans l'ordre de sérialisation- n'ayant pas démarré sa phase de validation et telle que $O_E(T_i) \cap [O_L(T) \cup O_E(T)] \neq \emptyset$.

. $O_L(T) \cap O_E(T_i) \neq \emptyset$ nous ramène au cas 2 de Kung & Robinson: le conflit LP est de type $T \longrightarrow T_i$ et est incompatible avec l'ordre de sérialisation. T ne doit pas lire une donnée qu'une mise à jour de T_i rendra "dépassée".

. $O_E(T) \cap O_E(T_i) \neq \emptyset$ induit un conflit PP. Une dépendance potentielle sera établie selon l'ordre des mises à jour. Comme rien ne garantit que cet ordre sera identique à l'ordre de sérialisation, on procède au rejet à priori de T.

CAS 3: Soit D_x le graphe de dépendances "local" relatif à l'objet x réalisé à partir de toutes les transactions en cours de validation, donc non achevées. On pose:

$T_i \longrightarrow T_j$ dans D_x <===>

- $[O_L(T_i) \cap O_E(T_j) \neq \emptyset$
- ou $O_E(T_i) \cap O_L(T_j) \neq \emptyset$
- ou $O_E(T_i) \cap O_E(T_j) \neq \emptyset]$

et T_i précède T_j dans l'ordre chronologique des certifications.

et soit B l'ensemble des transactions T_i telles que $T_i \longrightarrow T$ dans D_x.

Le contrôle de concurrence doit vérifier que toutes les transactions appartenant à l'ensemble B sont achevées, c'est-à-dire ont effectué leurs mises à jour ou ont avorté. Le contrôleur procède en leur accordant un délai -Time Out- à l'expiration duquel T est rejetée si elle n'est pas achevée.

Le Time Out est un moyen de détection de cycles éventuels dans le graphe D_x, cycles causés par des ordres chronologiques incompatibles d'un objet à un autre.

III.1.2.2- MODELE DE TRANSACTION REPARTIE

La phase de lecture/préécriture d'une transaction comporte les accès qu'elle exécute du site où elle a été lancée (appelé site origine) à ceux où se trouvent les objets qu'elle doit lire ou mettre à jour.

Chaque transaction présente un premier point de synchronisation à la fin de toutes les phases de lectures "locales". Après passage de ce point, le site origine emettra les demandes de certifications de la transaction vers les sites auxquels elle a accédé (voir fig. 1 ci-dessous: phase de vie et émission de C(T)).

III.1.2.3- MODELE DE CERTIFICATION - VALIDATION REPARTIE [BOKS 85]

La certification d'une transaction T démarre après la réception du message C(T) ("certifier T") par chacun des sites concernés. Elle se réalise en deux phases, une phase de certification locale et une phase de contrôle global (voir fig. 1 ci-dessous: phases de certifications locales, de certification globale et de validations locales).

-Phase de certification locale LC:
Les phases de certifications locales LC(T,x) se déroulent en parallèle sur chacun des sites auxquels la transaction T a accédé. Chaque site S_x accédé contrôle la sérialisabilité locale de T. En vue d'une certification globale, les résultats des phases de certifications locales doivent être regroupées, d'où la nécessité d'un deuxième point de synchronisation.

-Phase de contrôle global GC:
La phase de certification globale GC(T) de la transaction T peut être exécutée de manière centralisée sur le site origine de T ou de manière répartie sur chaque site objet (auquel la transaction a accédé). Chaque site doit alors avoir pris, au préalable, connaissance de tous les résultats locaux.

La réussite du contrôle global déclenche les phases de validations locales: Chaque site S_x effectue à son niveau la validation locale LV(T,x) de la transaction T: les préécritures effectuées par T sont transformées en écritures dans la base.

L'échec du contrôle global entraîne le rejet de la transaction.

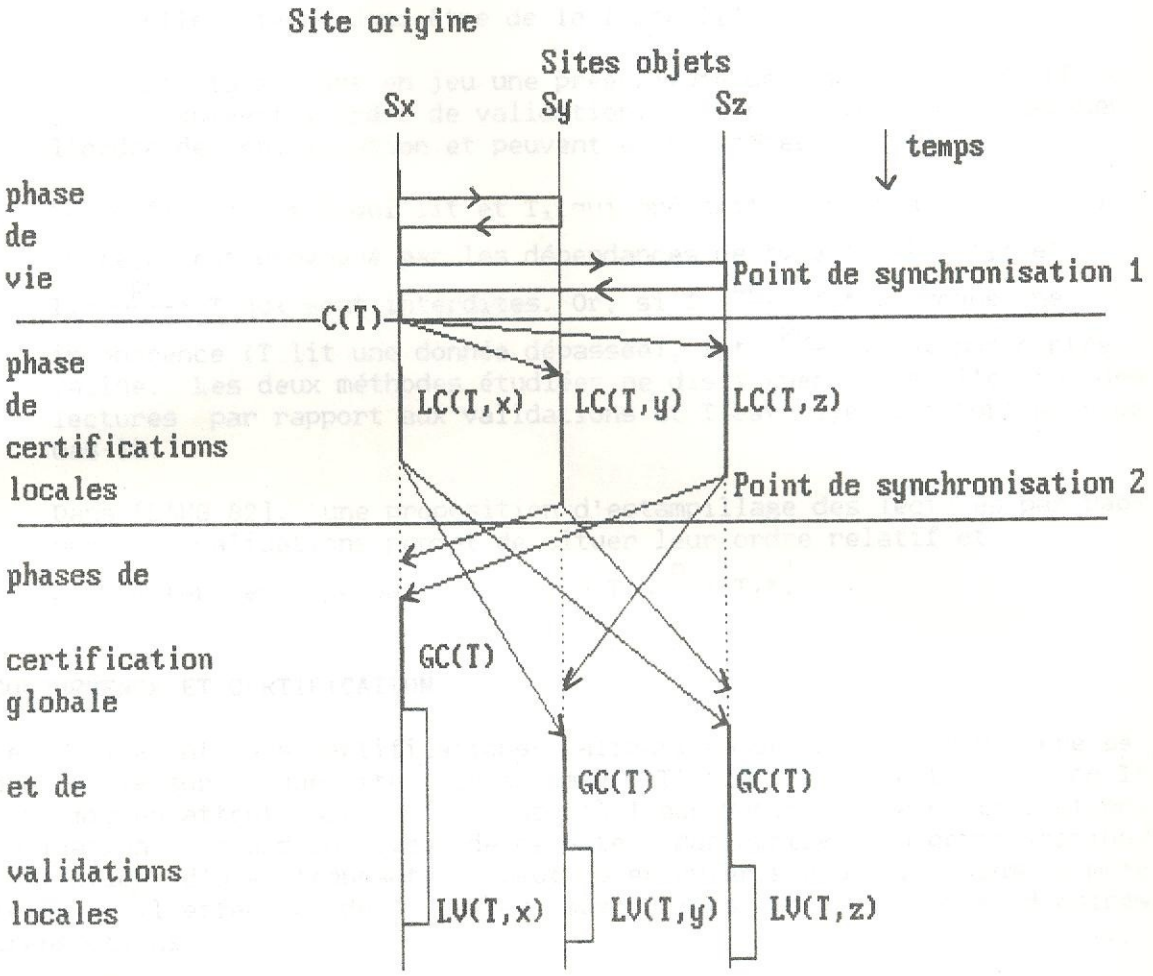


FIGURE 1: MODELE DE TRANSACTION REPARTIE [BOKS 85]

Remarques:
Pour certifier une transaction, la méthode de Kung & Robinson, ainsi que celle de Ceri & Owiki ne se basent que sur le graphe G* des transactions validées. Elles ne tiennent compte que des dépendances qui existent entre la transaction en cours de certification et celles qui sont déjà validées et ne prennent pas en charge celles qui lient cette même transaction aux autres transactions vivantes. Pour cela, on dit de ces méthodes qu'elles utilisent un contrôle "en arrière". Un contrôle plus complet devrait considérer également les contraintes existant entre la transaction à certifier et les autres transactions vivantes.

D'un autre côté, ces méthodes utilisent un ordre de sérialisation conforme à celui de leur entrée en phase de certification. Les dépendances qui lient une transaction T aux transactions T_i* validées avant elle doivent donc être de la forme T_i* ⟶ T.

Les conflits mettant en jeu une préécriture de T sont de type PP ou LP et suivent l'ordre de validation. Ils respectent par conséquent l'ordre de sérialisation et peuvent être ignorés.

Le conflit entre T qui lit et T_i qui préécrit aboutit au rejet de T. Ce rejet est provoqué par les dépendances de type T $\xrightarrow{LP}$ T_i* et T_i* $\xrightarrow{PL}$ T qui sont interdites. Or, si T $\xrightarrow{LP}$ T_i* provoque une incohérence (T lit une donnée dépassée), T_i* $\xrightarrow{PL}$ T est par contre valide. Les deux méthodes étudiées ne distinguent pas l'ordre des lectures par rapport aux validations et T est rejetée à tort dans ce cas-là.

Dans [LAUS 82], une proposition d'estampillage des lectures par rapport aux validations permet de situer leur ordre relatif et d'accepter les dépendances de type T $\xrightarrow{LP}$ T_i*.

III.2- CONCURRENCE ET CERTIFICATION

Le traitement des certifications- validations se fait d'une manière séquentielle sur chaque site. Le message C(T) "certifier la transaction T" est mis en attente sur un site jusqu'à l'achèvement de la certification-validation en cours au niveau de ce site. Par contre, la certification-validation d'une transaction peut s'exécuter sur un site dans le même temps qu'il effectue des accès aux objets en provenance d'autres transactions.

Dans la certification de transactions différentes, le recouvrement entre les phases de certification locale et de contrôle global - validation locale est admis. Une transaction peut exécuter sa certification locale sur un site pendant qu'une deuxième exécute sa certification globale ou sa validation locale sur un autre (voir fig. 2 ci-après).

Lorsque le contrôle est réparti, la demande de certification d'une transaction T émise par le site origine S_x peut être reçue, par un des sites destinataires S_y, après le résultat de la certification locale LC(T,x) en provenance d'un autre site S_x.

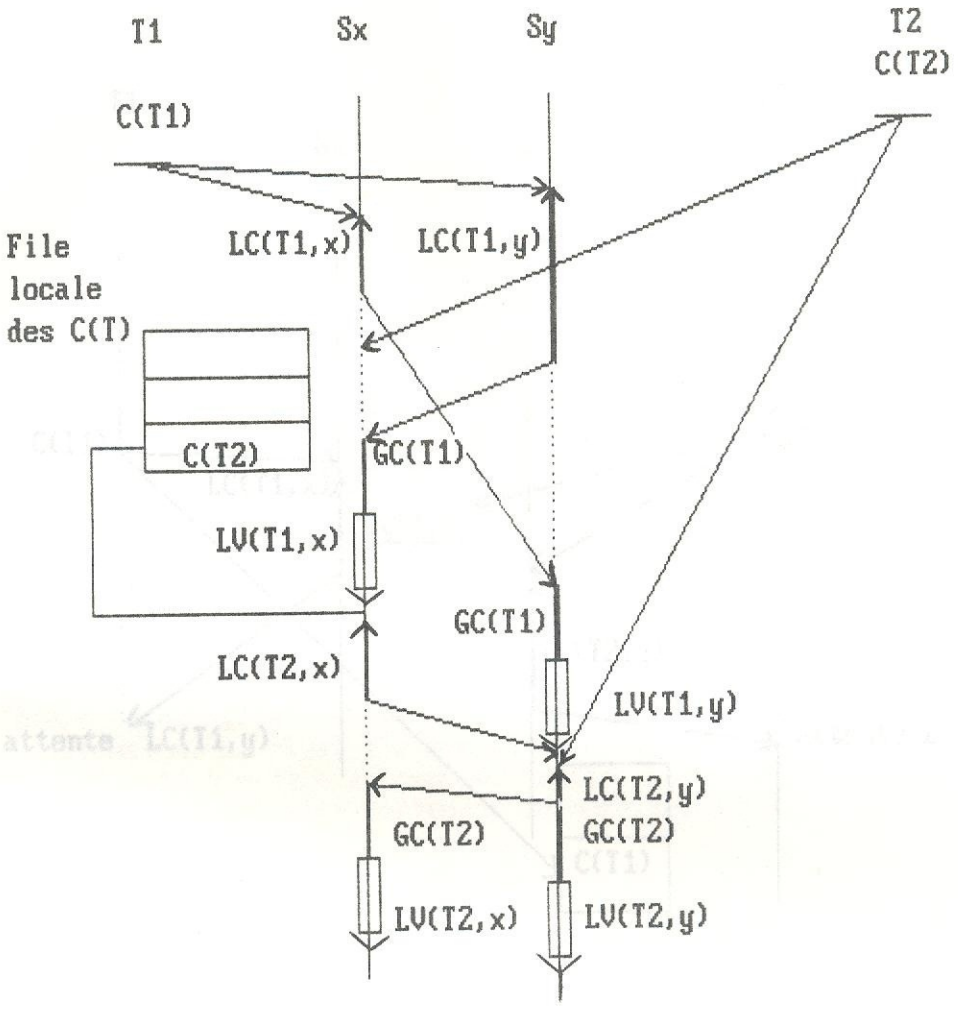


FIGURE 2: PARALLELISME EN CERTIFICATION SERIE REPARTIE (CONTROLE GLOBAL REPARTI)

Interblocage:
Le regroupement des résultats des certifications locales, donc l'attente qu'il nécessite en section critique sur chaque site peut provoquer une situation d'interblocage.

L'exemple de la figure 3 ci-après, choisi en contrôle global centralisé, montre la transaction T_1 en attente du résultat de sa certification locale sur le site S_y et la transaction T_2 en attente du résultat de la sienne sur le site S_x. Or S_y ne peut faire démarrer $LC(T_1,y)$ avant d'avoir le résultat du contrôle global de T_2 et S_x n'initiera $LC(T_2,x)$ qu'à l'achèvement de la certification-validation de T_1.

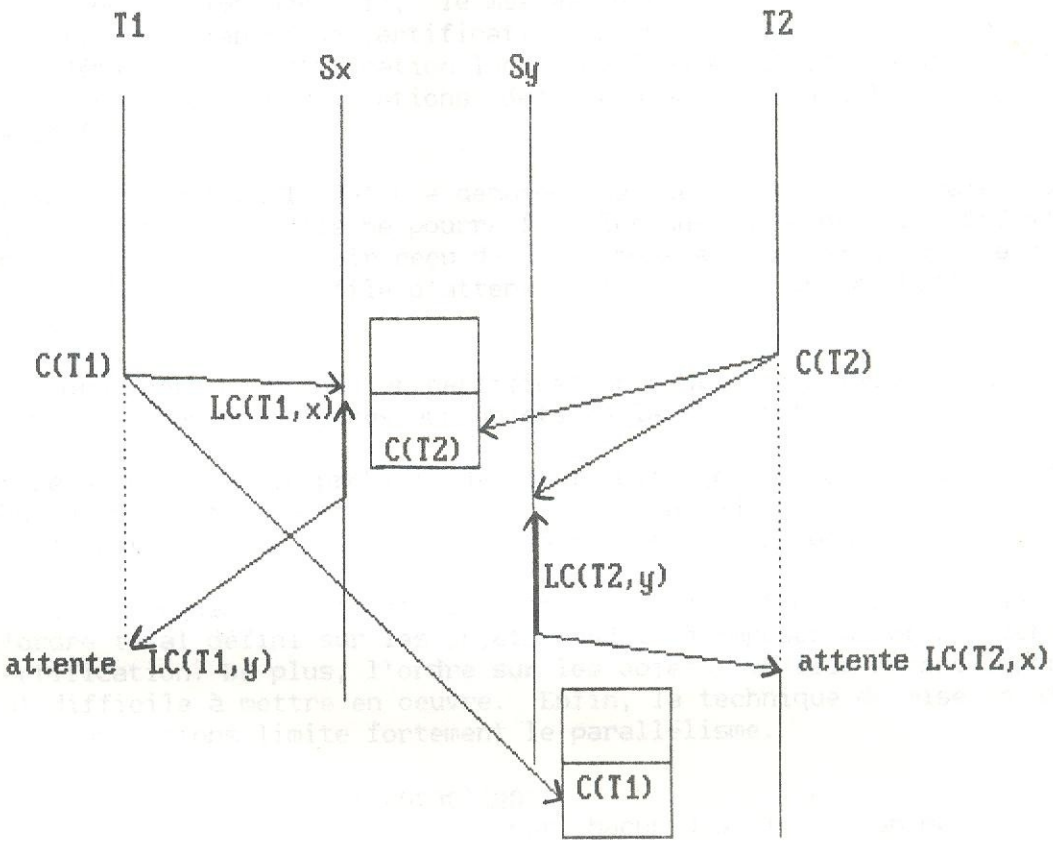


FIGURE 3: INTERBLOCAGE EN CERTIFICATION SERIE REPARTIE [PONS 86] (CONTROLE GLOBAL CENTRALISE)

Contrairement au cas du verrouillage 2 PL, cet interblocage ne traduit pas une exécution non sérialisable. Pour l'éviter, deux approches peuvent être envisagées [PONS 86].

- Ordonner les demandes de certification des transactions selon un ordre total établi sur les objets.

- Définir un ordre global entre les transactions.

CHAPITRE V- CONTROLE DE CONCURRENCE ET METHODES MULTIVERSIONS 47
V.1- Concepts 49
V.1.1- Exécutions cohérentes équivalentes 49
V.1.2- Exécution multiversion 51
V.1.3- Exécutions multiversions équivalentes 52
V.1.4- Sérialisabilité 53
V.2- Estampillage et multiversion 55
V.2.1- Algorithme de Reed 55
V.2.2- Algorithme de Carey 56
V.3- Méthodes multiversions "optimistes" 58
V.3.1- Approche de Ming Yee Lai 58
V.3.2- Approche d'Agrawal & Bernstein 60
V.3.2.1- Environnement centralisé 60
V.3.2.1.1- Techniques d'intégration et de compression 62
V.3.2.2- Environnement distribué 63
V.4- Conclusion 65

CHAPITRE VI- METHODE MULTIVERSION PAR INTERVALLES D' ESTAMPILLES 66
VI.1- Structure et gestion des données utilisées 70
VI.1.1- Exemple 72
VI.2- Ordre de sérialisation 72
VI.3- Accès aux objets 74
VI.3.1- Accès en lecture 75
VI.3.2- Accès en écriture 75
VI.4- Certification et validation répartie 76
VI.4.1- Certification locale 76
VI.4.2- Certification globale 77
VI.4.3- Validation locale 78
VI.5- Réajustement des intervalles de sérialisation 79
VI.6- Etude des rejets 81
VI.6.1- Rejet local d'une transaction 81
VI.6.2- Rejet global d'une transaction 82
VI.7- Choix de l'estampille de certification 82
VI.7.1- Choix de l'intervalle 83
VI.7.2- Choix de l'estampille 83
VI.8- Contrôle mixte (continu et par certification) 84
VI.8.1- Accès aux objets 87
VI.8.2- Certification et validation 88
VI.8.3- Deuxième approche 90
VI.8.3.1- Procédures 91
VI.9- Prolongement: Modèle de transaction parallèle 94
VI.10- Etude de la sérialisabilité 96
VI.11- Exemples d'application 99
VI.12- Privation 104
VI.13- Conclusion 105

CONCLUSIONS & PERSPECTIVES 107

REFERENCES BIBLIOGRAPHIQUES 111

Ordre global sur les objets:
La mise en oeuvre de cette solution passe par la suppression du message C(T) "certifier la transaction T", qui constituait un point de synchronisation à la suite de tous les accès aux objets effectués par T. Ce message est remplacé par une demande d'entrée en phase de certification, DC(T,x), que le site origine de T adresse au site S_x au moment où T accède en lecture/écriture à ce dernier.

Comme pour le message C(T), le message DC(T,x) est mis en attente dans une file de demandes de certifications au niveau du site S_x. Celui-ci ne fera démarrer la certification locale de T qu'à l'achèvement de toutes les certifications-validations des transactions qui la précèdent dans cette file.

Si une transaction T émet une demande de certification locale DC(T,x) vers un site S_x, elle ne pourra formuler une autre demande DC(T,y) vers un site S_y qu'après avoir reçu de S_x un message signifiant que sa demande DC(T,x) n'est plus en file d'attente et que la phase LC(T,x) a été lancée.

Les demandes d'entrée en certification sont donc notifiées au fur et à mesure des besoins d'accès aux objets de la transaction.

On résoudra ainsi le problème de l'interblocage: si la transaction T_1 accède avant la transaction T_2 sur S_x, elle accèdera avant T_2 sur S_y et l'alternative de la figure 3 ci-dessus ne se posera plus.

Comme conséquence, l'ordonnancement des certifications locales selon l'ordre total défini sur les objets conduit à imposer un ordre global de certification. De plus, l'ordre sur les objets, vu leur nombre important, est difficile à mettre en oeuvre. Enfin, la technique de mise en attente des transactions limite fortement le parallélisme.

Ordre global entre les transactions:
Cette deuxième approche impose, sur chacun des sites, un même ordre de traitement des certifications-validations des transactions [BOKS 85].

L'estampillage des transactions permet de construire un ordre de sérialisation global qui fixera l'ordre d'entrée en phase de certification. Les ordres de sérialisation locaux sont ensuite définis comme des sous-ordres de cet ordre global.

Une généralisation de la méthode de [KUNG 81] à un environnement réparti qui respecte ces contraintes a été présentée dans [PONS 86].

III.4- CONSTRUCTION D'UN ORDRE DE SERIALISATION GLOBAL

Plutôt que d'imposer aux ordres de sérialisation locaux de suivre un ordre de sérialisation global fixé avant les phases de certifications locales, il serait moins contraignant de pouvoir suivre la procédure inverse, c'est-à-dire, construire un ordre de sérialisation global en fonction des ordres de sérialisation locaux [PONS 86].

Les ordres de sérialisation locaux et l'ordre de sérialisation global correspondent, respectivement, à des graphes de dépendances locaux et à un graphe de dépendances global (union des graphes locaux) sans circuit.

Une gestion centralisée du graphe de dépendances global sur un site privilégie ce site et amoindrit le parallélisme. D'un autre côté, la maintenance sur chaque site d'une copie de ce graphe impose une lourde gestion des copies multiples et un nombre important de messages à échanger [CORN 81].

Actuellement, on s'efforce de trouver des mécanismes qui auraient des propriétés assurant la compatibilité entre les ordres de sérialisation locaux et l'ordre de sérialisation global sans passer par la gestion des graphes. Certains de ces mécanismes pourraient être les suivants:

1)- $T_i <_x T_j \Longrightarrow T_i <_G T_j$

2)- Si la transaction T n'est pas localement sérialisable, alors elle ne sera pas globalement sérialisable. Le rejet local d'une transaction entraîne son rejet global ou, s'il existe un circuit global, il y a au moins un site où l'ordre de sérialisation local n'est pas compatible avec l'ordre de sérialisation global et ceci quel que soit l'ordre de certification.

3)- Si, pour tout x, T est localement sérialisable sur S_x, alors T sera globalement sérialisable pour x appartenant à $O_L(T)$.

Une méthode reposant sur ces propriétés comporterait des avantages certains:

-La sérialisation de T ne dépend que de l'ensemble des sites accédés par T.
-La phase de contrôle global se réduit à l'intersection des résultats booléens des phases de certifications locales.

Les éventuels rejets inutiles (provoqués par les dépendances de type $T \xrightarrow{LP} T_i^*$) limitent le parallélisme. Par contre, plus une méthode minimisera localement le risque de conflit, par le rejet ou l'attente, plus le contrôle global sera allégé.

La méthode de Kung & Robinson sacrifie les "vieilles lectures" puisque, pour certifier une transaction, elle ne tient compte que de ses dépendances avec les transactions déjà validées et ignore celles qui la lient aux autres transactions vivantes. Si T_2 a effectué une lecture avant la préécriture de T_1 ($T_2 \xrightarrow{LP} T_1$) et si T_1 précède T_2 dans l'ordre de validation, alors T_2 sera rejetée.

Une variante basée sur le contrôle avant impose à toute transaction ayant effectué une écriture d'être certifiée après la validation des transactions lectrices pures [SHLA 81]. Là encore l'ordre de sérialisation global sera égal à l'ordre global des certifications et on résout ce problème par l'attente.

III.4- CONCLUSION

Les méthodes optimistes éliminent les attentes liées à la technique du verrouillage 2 PL et les attentes et rejets induits par la technique d'estampillage [PONS 86]. Ce fait de "laisser aller" la transaction et de ne pas la mettre en attente ou la rejeter au premier conflit permet aussi de rassembler un maximum d'informations sur l'établissement des dépendan-

ces et donc de mieux appréhender la sérialisabilité.

Du fait que le contrôle n'a lieu que lors de l'étape de certification, le synchronisme lecture/préécriture est supprimé et le parallélisme entre les transactions s'en trouve amélioré.

L'efficacité de ces méthodes, qualifiées d'optimistes, se vérifie quand les conflits sont rares. Dans ce cas-là, une méthode de contrôle continu dégrade inutilement le niveau de parallélisme en contrôlant systématiquement tous les accès aux objets des transactions.

Elles s'adaptent bien aux systèmes qui n'effectuent que des lectures. Un accès en lecture peut s'effectuer sur une donnée incohérente, ce qui provoque l'échec de sa certification et son rejet. Par contre et de par sa nature, il ne peut, de lui-même, engendrer une incohérence dans la base. De ce fait, les systèmes qui n'effectuent que des lectures vérifient l'hypothèse de la rareté des conflits.

En revanche, ces méthodes imposent aux transactions un ordre d'entrée en phase de certification identique sur tous les sites. Pour que ceux-ci prennent une décision unique, il leur faut la même connaissance des conflits et dépendances entre transactions vivantes et transactions validées; ils doivent donc considérer les mêmes ensembles de transactions vivantes et de transactions validées.

Il en découle un ordre de sérialisation des transactions identique à l'ordre chronologique de leur entrée en phase de certification.

Ces méthodes posent aussi le problème de la privation: elles sont basées sur le rejet de la transaction qui provoque l'incohérence. Celle-ci peut être rejetée indéfiniment, après chacun de ses redémarrages.

ces et donc de mieux appréhender la sérialisabilité.

Du fait que le contrôle n'a lieu que lors de l'étape de certification, le synchronisme lecture/préécriture est supprimé et le parallélisme entre les transactions s'en trouve amélioré.

L'efficacité de ces méthodes, qualifiées d'optimistes, se vérifie quand les conflits sont rares. Dans ce cas-là, une méthode de contrôle continu dégrade inutilement le niveau de parallélisme en contrôlant systématiquement tous les accès aux objets des transactions.

Elles s'adaptent bien aux systèmes qui n'effectuent que des lectures. Un accès en lecture peut s'effectuer sur une donnée incohérente, ce qui provoque l'échec de sa certification et son rejet. Par contre et de par sa nature, il ne peut, de lui-même, engendrer une incohérence dans la base. De ce fait, les systèmes qui n'effectuent que des lectures vérifient l'hypothèse de la rareté des conflits.

En revanche, ces méthodes imposent aux transactions un ordre d'entrée en phase de certification identique sur tous les sites. Pour que ceux-ci prennent une décision unique, il leur faut la même connaissance des conflits et dépendances entre transactions vivantes et transactions validées; ils doivent donc considérer les mêmes ensembles de transactions vivantes et de transactions validées.

Il en découle un ordre de sérialisation des transactions identique à l'ordre chronologique de leur entrée en phase de certification.

Ces méthodes posent aussi le problème de la privation: elles sont basées sur le rejet de la transaction qui provoque l'incohérence. Celle-ci peut être rejetée indéfiniment, après chacun de ses redémarrages.

CHAPITRE IV

CERTIFICATION PAR INTERVALLES D' ESTAMPILLES

Ce chapitre particularise l'étude d'une méthode de contrôle de concurrence par certification qui a été développée dans [SOKS 84] et qui généralise l'estampillage dynamique en s'appuyant sur la technique des intervalles d'estampilles.

Elle a été élaborée sur le principe de l'affectation d'un intervalle d'estampilles à chaque transaction vivante et d'une estampille, prélevée de l'intervalle associé, à chaque transaction validée.

L'intervalle d'estampilles d'une transaction est construit de manière dynamique et matérialise les dépendances qui lient cette transaction à celles qui sont déjà validées.

L'intervalle associé à une transaction T' accédant à l'objet x est noté I(T',x) et représente la place de T' vivante dans l'ordre local relatif à x. Il sera borné à gauche et à droite par les estampilles des transactions validées ayant accédé à ce même objet x: à gauche par les estampilles de celles qui précèdent T' dans l'ordre de sérialisation et à droite, par les estampilles de celles qui suivent T' dans ce même ordre de sérialisation.

On aura donc, pour toute transaction T* validée, d'estampille t et ayant accédé à x:

. Si T' ——> T*, alors I(T',x) < t (quel que soit t' ∈ I(T',x), on a t' < t)
. Si T* ——> T', alors t < I(T',x) (quel que soit t' ∈ I(T',x), on a t < t')

Cette méthode introduit un ordre de sérialisation des transactions différent de l'ordre chronologique des certifications. Elle admet les dépendances de type T'——> T*: une transaction précédant, dans l'ordre de sérialisation, une autre validée, peut être certifiée et validée après elle.

Les contraintes existant entre la transaction T vivante et les transactions validées sur l'ensemble des objets accédés sont groupées dans l'intervalle global $I_G(T)$,

$$I_G(T) = \bigcap_{x \in X} I(T,x) \qquad \text{avec } X = \{\text{objets accédés par T}\}$$

et la transaction T est sérialisable si et seulement si l'intervalle global $I_G(T)$ qui lui est associé est non vide.

La sérialisabilité locale de la transaction T accédant au site S_x sera donc exprimée par le critère $I(T, x) \neq \emptyset$.

L'étude de la méthode peut être effectuée sans dénaturer les caractéristiques des intervalles sous l'hypothèse simplifiée d'un objet par site. Dans le cas de plusieurs objets par site et pour une transaction T accédant au sous-ensemble d'objets X_i du site S_i, l'intervalle de sérialisation global se calcule ainsi:

$$\text{On a } I_G(T) = \bigcap_{S_i \in S} I(T, S_i) \qquad \text{avec } S = \{\text{sites accédés par T}\}$$

$$\text{et } I(T, S_i) = \bigcap_{x \in X_i} I(T, x) \qquad \text{avec } X_i = \{\text{objets accédés par T sur } S_i\}$$

$$\text{donc } I_G(T) = \bigcap_{S_i \in S} I(T,S_i) = \bigcap_{S_i \in S} \left(\bigcap_{x \in X_i} I(T,x) \right) = \bigcap_{x \in X} I(T,x)$$

$$\text{avec } X = \{\text{objets accédés par T}\}$$

Le critère de sérialisation globale $I_G(T) \neq \emptyset$ est donc indépendant de la répartition sur les sites des objets accédés par T.

IV.1- STRUCTURES DES DONNEES

a) Estampilles associées aux transactions:
A toute transaction validée on associe une estampille numérique positive unique qui lui est attribuée lors de la phase de certification globale. Elle est choisie dans l'intervalle d'estampilles adjoint à la transaction.

b) Estampilles associées aux objets:
Une estampille d'écriture E(x) et une estampille de lecture L(x) sont gérées au niveau de chaque objet x. Elles sont mises à jour à la validation de toute transaction T, d'estampille t et ayant accédé à x. Si T est une transaction de lecture pure, L(x) sera positionnée à max[L(x), t], sinon on affectera t à E(x). Ainsi E(x) désignera l'estampille de la dernière transaction validée ayant écrit l'objet x et L(x), celle de la transaction la plus récente - dans l'ordre de sérialisation- qui aurait lu ce même objet x.

c) Intervalles locaux associés aux transactions:
A toute transaction vivante T et à chaque objet x qu'elle accède, on fait correspondre l'intervalle d'estampilles I(T,x), d'abord initialisé à $[0,+\infty[$, ensuite tronqué à gauche ou à droite de manière à exprimer les dépendances existant entre T et les transactions validées ayant accédé à x. Désignons par T_E la dernière transaction validée ayant écrit x et par T_L, la plus récente qui l'aurait lu. Ces deux transactions ont donc pour estampilles E(x) et L(x) et la construction de I(T,x) s'effectuera selon les règles ci-après:

. Lors d'un accès en lecture effectué par T et si T_E précède T dans l'ordre de sérialisation ($T_E <_x T$), T lira la mise à jour exécutée par T_E: son estampille devra donc être supérieure à celle de T_E et I(T,x) sera tronqué à gauche.

$$T_E <_x T \Rightarrow I(T,x) = I(T,x) \cap [E(x) + 1, +\infty [$$

. Dans le cas ou T précède T_E dans l'ordre de sérialisation ($T <_x T_E$), son estampille devra être inférieure à celle de T_E. Lors de la validation de T_E, I(T,x) sera par conséquent tronqué à droite.

$$T <_x T_E \Rightarrow I(T,x) = I(T,x) \cap [0, E(x)-1]$$

. Si T accède en préécriture à x et si T_E et T_L précèdent T dans l'ordre de sérialisation, celle-ci ne pourra réécrire x qu'après son écriture par T_E et sa lecture par T_L. Son estampille devra donc majorer celles de T_E et de T_L : la troncature de I(T,x) s'effectuera à gauche, lors de la phase de certification locale.

$$T_E <_x T \text{ et } T_L <_x T \Rightarrow I(T,x) = I(T,x) \cap [\max(E(x), L(x)) + 1, +\infty [$$

d) Ensemble de transactions:
Chaque site S_x gère l'ensemble des transactions vivantes ayant accédé à l'objet x et fait correspondre un intervalle local à chacune d'elles.

Lors des accès à x, elles seront réparties en deux ensembles, lectrices(x) et écrivains(x). Ainsi, on pourra différencier les transactions ayant effectué des lectures et qui par conséquent, nécessitent une gestion des intervalles, de celles ayant accédé uniquement en préécriture. On notera qu'une transaction peut effectuer une lecture et une préécriture sur l'objet x et donc appartenir à la fois à lectrices(x) et à écrivains(x).

IV.2- ACCES AUX OBJETS

Ces accès se feront conformément au modèle de transaction défini en I.3 (cf. page 13): ils s'effectuent sur l'espace de travail affecté à la transaction et ne sont exécutés sur la base de données qu'une fois la transaction validée.

Procédure lire (T, x) du site S_x

```
DEBUT
SI    T ∉ (lectrices(x) U écrivains(x))
ALORS I(T,x) := [0, +∞ [
FSI
lectrices(x) := lectrices(x) U {T}
I(T,x)       := I(T,x) ∩ [E(x) + 1, +∞ [
v            := valeur lue (x)
RENVOYER v au site S_T    /* renvoi au site qui a initié T */
FIN
```

Procédure préécrire (T, x) du site S_x

```
DEBUT
écrivains(x) := écrivains(x) U {T}
ECRIRE x sur l'espace de travail de T
FIN
```

Lorsque la transaction T effectue sa lecture sur l'objet x, I(T,x) sera tronqué et borné à gauche par l'estampille d'écriture E(x) de cet objet, de manière à exprimer la dépendance de T avec la transaction validée qui a écrit x.

Les préécritures ne seront contrôlées qu'au niveau de la phase de certification locale.

IV.3- CERTIFICATION - VALIDATION REPARTIE

Le protocole utilisé ne privilégie pas de site particulier. La certification-validation d'une transaction se déroule en trois phases sur l'ensemble des sites auxquels elle a accédé.

a) Phase de certification locale:
Les phases de certification locales LC(T,x) se déroulent en parallèle sur tous les sites concernés et débutent, au niveau de chacun d'eux, dès réception du message C(T) -"certifier T"- émis par le site S_T qui a initié T et qui gère ses accès. Chaque site contrôle alors la sérialisabilité de T à son niveau.

Procédure de certification locale LC(T,x) du site S_x

```
DEBUT
SI    T ∈ écrivains(x)
ALORS I(T,x) := I(T,x) ∩ [max (E(x),L(x)) + 1, +∞ [
FSI
DIFFUSER I(T,x)
FIN
```

C'est au niveau de la phase de certification locale qu'on exécute la troncature à gauche de l'intervalle de la transaction qui a effectué une préécriture. L'estampille qu'on affectera par la suite à cette transaction, si elle n'est pas rejetée, doit être supérieure à celles des transactions validées qui ont effectué la dernière écriture et la dernière lecture sur l'objet.

A la fin de cette phase, l'intervalle local est diffusé à tous les sites accédés par la transaction pour qu'ils entament leur phase de certification globale.

b) Phase de certification globale:
Elle s'exécute sur l'ensemble des sites accédés par la transaction et chacun de ceux-ci ne la mettra en oeuvre qu'après avoir reçu tous les intervalles locaux des autres sites. Chaque site concerné doit donc connaître tous les autres sites accédés: on peut transmettre cette information aux sites concernés via le message C(T) émis par S_T.

Procédure de certification globale GC(T,x) du site S_x

```
DEBUT
IG(T) :=  ∩ I(T,x)
          x ∈ {objets accédés par T}
SI IG(T) ≠ ∅
ALORS FAIRE
      CHOISIR l'estampille t
      RGC(T) := (vrai,t)
      FAIT
SINON RGC(T) := (faux,?)   /* cas de rejet global */
FSI
FIN
```

La variable RGC(T), composée d'un booléen et d'une estampille, donne le résultat de la certification globale. Dans le cas où T est sérialisable, c'est-à-dire si l'intervalle global $I_G(T)$ associé est non vide, le choix de l'estampille doit s'effectuer de la même façon sur tous les sites concernés.

c) Phase de validation locale:
Le déroulement de cette phase dépend du résultat de la certification globale.

Si le booléen de RGC(T) affiche la valeur "vrai" et si T a lu l'objet x, on procède à la mise à jour de l'estampille L(x). Dans le cas où T a effectué une préécriture, on affecte l'estampille t choisie pour T à E(x) et on procède à la troncature des intervalles des transactions vivantes de lectrices(x) et à l'écriture effective de x dans la base.

Procédure de validation locale LV(T,x) du site S_x

```
DEBUT
SI RGC(T) ≠ faux
ALORS FAIRE
      SI T ∈ écrivains(x)
      ALORS FAIRE
            /* validation écriture : */
            Section Critique: «
            E(x) : = t
                  POUR T'∈ [lectrices(x) - {T}]
                  FAIRE I(T',x) := I(T',x) ∩ [0, E(x) - 1]
                  FAIT
            ECRIRE (x) »
            FAIT
      FSI
      SI     T ∈ lectrices(x)
      ALORS  L(x) := max[(L(x),t]
      FSI
      FAIT
FSI
DETRUIRE (T,x)
FIN
```

Le traitement de validation associé à une préécriture doit s'effectuer en exclusion mutuelle avec toute opération de lecture.

La vie de la transaction T s'achève par sa destruction locale. Elle est supprimée des deux ensembles lectrices(x) et -ou- écrivains(x). Les intervalles local I(T,x) et global $I_G(T)$ sont également détruits.

IV.4- CONCLUSION

La méthode de certification par intervalles d'estampilles utilise le modèle de certification-validation répartie introduit dans [KUNG 81] et dé-

L'agencement d'ordinateurs en réseaux a amené la mise en place et le développement d'applications et de bases de données réparties.

Une base de données répartie se définit comme un ensemble d'objets distribués sur différents sites d'un réseau et constituant un seul système logique [CERI 84]. Ces objets sont liés par des relations appelées "contraintes d'intégrité" et la cohérence du système dépendra du respect de ces contraintes.

Pour assurer le contrôle de concurrence dans les systèmes informatiques répartis, plusieurs méthodes et techniques ont été élaborées. Elles synchronisent les accès concurrents de plusieurs utilisateurs à la base de données répartie. Ces accès se font par le biais de transactions qui sont des unités d'exécution cohérente: une transaction exécutée seule sur un état cohérent conduit à un autre état cohérent de la base.

Les méthodes de contrôle de concurrence ont à respecter les deux grandes règles suivantes:

-Toute transaction soumise au système pourra s'exécuter.
-Chaque transaction s'exécutera comme si elle était unique dans le système.

Elles sont donc conduites à assurer l'exécution en parallèle des transactions et à prendre en charge les effets des pannes qui pourraient survenir au sein du système.

Ces méthodes ont été développées sur un modèle de transaction général -la transaction est vue comme une suite de lectures/écritures- et reposent sur le concept de l'atomicité des transactions: celles-ci doivent se dérouler totalement ou pas du tout et doivent être sérialisables [PAPA 79], c'est-à-dire que l'exécution concurrente et l'exécution en série d'un même ensemble de transactions doivent être équivalentes. Elles se répartissent à l'heure actuelle en deux grandes classes.

Le contrôle de concurrence continu adopte comme hypothèse la fréquence élevée des conflits (hypothèse pessimiste): La sérialisabilité est vérifiée tout au long de la vie de la transaction. Cette vérification utilise des techniques d'estampillages -statique ou dynamique- [BERN 81] ou de verrouillage à deux phases (verrouillage 2 PL) [ESWA 76], [TRAI 82]. Les conflits y sont traités par l'attente et le rejet de la transaction. Les méthodes de ce type sont celles qui sont le plus couramment utilisées.

Le contrôle de concurrence par certification se base sur la rareté des conflits (hypothèse optimiste): La cohérence de la base de données n'est contrôlée qu'au niveau de la phase terminale de certification de la transaction [KUNG 81], [CERI 82], [CARE 83]. Lorsque la rareté des conflits est effective, ce type de contrôle augmente notablement le parallélisme des transactions.

Le développement des méthodes multiversions a été effectué dans le but de diminuer les rejets de transactions. Ces techniques consistent à conserver un ensemble de versions d'un même objet: elles fournissent aux transactions une succession de vues sur chaque objet de la base de données.

Dans un premier temps, nous avons effectué une synthèse des principales méthodes existantes, appliquées sur des bases de données monoversions ou multiversions.

veloppé dans [BOKS 85] (cf. fig. 1 page 32) et bénéficie donc des avantages qui en découlent: le contrôle de la sérialisabilité d'une transaction ne porte que sur les sites auxquels elle a accédé et sa certification globale se réduit à une intersection des résultats de la phase de certification locale précédente.

Cette méthode permet aux transactions vivantes de précéder dans l'ordre de sérialisation celles qui sont déjà validées. L'ordre exprimé est proche de l'ordre partiel engendré par les dépendances entre les transactions. Il permet notamment à une transaction qui a lu un objet de ne pas être rejetée si, entre cette lecture et sa validation, une deuxième effectue une écriture validée sur l'objet.

Néanmoins, ses possibilités demeurent en deçà de celles des méthodes théoriques basées sur la gestion du graphe de sérialisation: elle peut provoquer des rejets inutiles.

Soient par exemple deux transactions T_1 et T_2 telles que l'intersection de leurs ensembles d'objets accédés soit vide (tous leurs accès se sont effectués sur des sites différents). Leurs intervalles associés sont donc quelconques: prenons les tels que $I_G(T_2) < I_G(T_1)$. Désignons enfin par t_1 et t_2 les estampilles choisies dans $I_G(T_1)$ et $I_G(T_2)$ et associées à T_1 et T_2 lors de leurs validations. Toute transaction vivante T qui précède T_2 pour l'objet y et suit T_1 pour l'objet x ($T_1 <_x T$ et $T <_y T_2$) vérifiera alors $I_G(T) = \emptyset$ et sera rejetée.

Les hypothèses exprimées ci-dessus se traduisent par:
- $T_1^* \longrightarrow T$ sur S_x ===> $t_1 < I(T, x)$ (quel que soit $t \in I(T, x)$, $t_1 < t$)
- $T \longrightarrow T_2^*$ sur S_y ===> $I(T, y) < t_2$ (quel que soit $t \in I(T, y)$, $t < t_2$)
- $I_G(T_2) < I_G(T_1)$ (quels que soient $t_1 \in I_G(T_1)$ et $t_2 \in I_G(T_2)$, $t_2 < t_1$)

Les trois assertions énoncées ci-dessus étant contradictoires, on aura forcément $I_G(T) = \emptyset$.

Le rejet de la transaction T est cependant inutile: elle ne peut pas introduire un circuit dans le graphe de sérialisation puisqu'aucun arc ne relie T_1 à T_2.

Le choix des estampilles associées aux transactions qui sont en phase de validation influe sur le taux de rejets puisque les intervalles des transactions encore vivantes sont bornés par ces estampilles.

L'estampille étant prélevée dans l'intervalle global associé à la transaction, on peut même diriger ce choix de manière à avantager localement une classe particulière de transactions. Prenons une transaction vivante T dont l'intervalle sera tronqué à droite par l'estampille de la transaction T_E qui valide. Il est clair que plus cette estampille sera proche de la borne gauche de $I_G(T_E)$, plus l'intervalle local associé à T sera restreint et T défavorisée.

Cette technique de certification par intervalles d'estampilles est considérée comme une méthode de contrôle arrière puisque la sérialisation d'une transaction vivante n'est exprimée qu'en fonction des transactions déjà validées. On peut réaliser une forme de contrôle avant en prenant en charge les transactions vivantes et en réajustant en conséquence l'intervalle de celle qui certifie localement [HAER 82]: On optimiserait ainsi le taux de rejets.

V.1 Concepts
- V.1.1 Exécutions cohérentes équivalentes
- V.1.2 Exécution multiversion
- V.1.3 Exécutions multiversions équivalentes
- V.1.4 Sérialisabilité

V.2 Estampillage et multiversion
- V.2.1 Algorithme de Reed
- V.2.2 Algorithme de Carey

V.3 Méthodes multiversions "optimistes"
- V.3.1 Approche de Ming Yee Lai
- V.3.2 Approche d'Agrawal & Bernstein
  - V.3.2.1 Environnement centralisé
    - V.3.2.1.1 Techniques d'intégration et de compression
  - V.3.2.2 Environnement distribué

V.4 Conclusion

Les méthodes multiversions sont basées sur le principe de la conservation d'un ensemble de versions d'un même objet. Ces méthodes fournissent aux transactions une succession de vues sur les objets de la base de données. Elles éliminent ainsi les problèmes liés aux contrôles sur les lectures, particulièrement le rejet de la transaction qui ne peut plus lire une donnée parce que cette dernière aura été modifiée (non rejet des "vieilles lectrices").

D'un point de vue fonctionnel, la base de données garde son unicité vis-à-vis de l'utilisateur: ce dernier ne peut accéder qu'à une version de l'objet. C'est le contrôleur -scheduler- qui s'appuie sur cet ensemble de versions pour minimiser le nombre de rejets de transaction et améliorer ses performances.

Nous avons vu que dans la plupart des méthodes monoversions, la cohérence de la base était assurée par la sérialisabilité, autrement dit par l'équivalence entre l'exécution en parallèle et l'exécution en série d'un même ensemble de transactions. Le contrôle portait donc surtout sur l'ordre de traitement des opérations de lectures/écritures.

Dans une approche multiversion, l'analyse d'un algorithme de contrôle de concurrence s'effectuera en fonction d'une notion de sérialisabilité plus étendue.

Cette généralisation de la sérialisabilité s'appuiera sur deux structures de données.

-L'historique multiversion qui illustre les exécutions des différentes opérations sur la base de données: chaque objet y figure sous un certain nombre de versions et chacune d'elles correspond à une écriture de celui-ci par une transaction quelconque.

-L'historique monoversion où chaque objet n'est représenté que par une seule version et qui représente la projection de l'historique multiversion à un instant t. Il correspond aussi à la vue de la base de données qu'à l'utilisateur à ce même instant.

Cette vue doit être cohérente et on montrera dans la suite, notamment à travers la notion de 1-sérialisabilité, que la cohérence d'un historique multiversion passe par celle de tous les historiques monoversions qu'il engendre.

Dans cette partie, nous introduirons d'abord les concepts propres aux méthodes multiversions et la notion de cohérence qu'elles sous-tendent. Nous présenterons ensuite quelques algorithmes qui les illustrent: ceux de [REED 78] et de [CARE 83], précurseurs dans l'approche multiversion, celui de [MING 84] dont l'originalité réside dans la prise en charge des transactions en phase de certification en permettant aux autres transactions d'accéder à leurs résultats et enfin celui de [AGRA 86] qui offre un bon niveau de parallélisme en permettant une validation parallèle des transactions.

V.1- CONCEPTS

V.1.1- EXECUTIONS COHERENTES EQUIVALENTES.

D'une manière intuitive, deux exécutions cohérentes effectuées par un même ensemble de transactions sont équivalentes si elles réalisent les mêmes calculs et si elles renvoient les mêmes valeurs, c'est à dire si

elles lisent la même occurrence d'un objet et si elles effectuent sur un objet la même mise à jour finale.

A partir de cette assertion, nous allons définir la notion d'équivalence entre les exécutions cohérentes de deux suites d'opérations dans une approche multiversion.

Le schéma de transaction sur lequel nous nous baserons sera l'ensemble partiellement ordonné $T_i = (\Sigma_i, <_i)$ où

. Σ_i désigne l'ensemble des opérations de lecture/écriture issues de l'exécution de la transaction T_i et
. $<_i$ l'ordre qui régit ces opérations.

Prenons par exemple la transaction T_1 telle que

Cette transaction effectue les lectures sur les objets x et z en parallèle et exécute ensuite une opération d'écriture sur x.

Dans la suite, nous noterons $<$ la relation de précédence liant deux opérations de deux transactions distinctes.

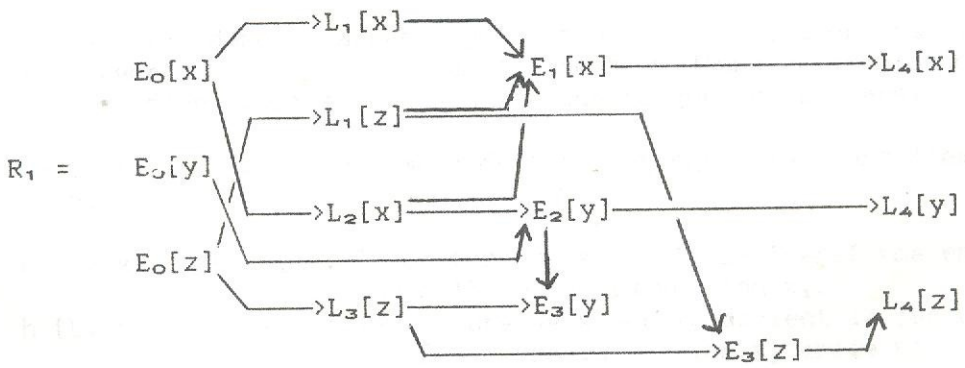
Soit R une exécution sur un ensemble de transactions $\{T_0,.....,T_n\}$. Nous dirons que T_j "lit x" de T_i si:

1- $E_i(x)$ et $L_j(x)$ sont des opérations dans R.
2- $E_i(x) < L_j(x)$
3- $\nexists E_k(x)$ tel que $E_i(x) < E_k(x) < L_j(x)$.

Autrement dit, si l'opération "Ecrire x" initiée par T_i précède l'opération "Lire x" lancée par T_j et si aucune transaction T_k ne vient modifier l'objet x avant cette lecture.

DEFINITION:
Deux exécutions cohérentes effectuées par un même ensemble de transactions seront dites équivalentes si leurs relations "lire de" renvoient les mêmes résultats.

EXEMPLE:
Soient R_1 et R_2 deux exécutions possibles de l'ensemble des transactions $\{T_0,T_1,T_2,T_3,T_4\}$.



R_2 = $E_0[x]E_0[y]E_0[z]L_2[x]E_2[y]L_1[x]L_1[z]E_1[x]L_3[z]E_3[y]E_3[z]L_4[x]...$
$L_4[y]L_4[z]$

Dans R_1 comme dans R_2, l'objet x est créé par T_0 avant sa lecture par T_1 et aucune autre transaction ne le réécrit avant cette lecture: T_1 lit x de T_0. De même,

T_1 lit z de T_0,
T_2 lit x de T_0,
T_3 lit z de T_0,
T_4 lit x de T_1,
T_4 lit y de T_3 et
T_4 lit z de T_3.

Les deux opérations "Lire Objet" lancées par une même transaction dans R_1 et R_2 ramènent la même version de cet objet et cela se vérifie pour chaque couple d'opérations ainsi définies. R_1 et R_2 sont donc équivalentes.

REMARQUE:
La définition ci-dessus n'impose aucune contrainte sur l'état final de la base de données. De ce fait, deux exécutions cohérentes équivalentes peuvent ne pas fournir les mêmes résultats. Par exemple, $R = E_0[x]E_1[x]$ et $R' = E_1[x]E_0[x]$ sont deux exécutions équivalentes dans lesquelles les deux versions finales de l'objet x, générées par deux transactions distinctes, sont différentes.

Or, l'état final de la base de données doit être unique: dans l'ensemble des exécutions possibles, la même transaction devrait écrire la seule valeur finale de l'objet x.

On peut cependant utiliser la définition proposée sans préjudice pour cette unicité: il suffit de rajouter, dans chaque exécution, une même transaction finale, transaction qui ne devrait effectuer que des lectures (rôle de T_4 dans l'exemple précédent). L'équivalence entre deux exécutions signifie alors les mêmes résultats pour ces lectures, donc des valeurs finales uniques pour les objets de la base de données.

V.1.2- EXECUTION MULTIVERSION

Dans une approche multiversion, une opération sur un objet de la base de données doit se traduire par une opération sur une version possible de cet objet.

On définira donc ci-après une fonction h qui transformera l'écriture ou la réécriture d'un objet en la création d'une version de celui-ci et la lecture d'un objet en celle d'une quelconque de ses versions.

Soit $\{T_0,\ldots,T_n\}$ un ensemble de transactions effectuant l'exécution R. On aura,

$h\ [E_i(x)] = E_i(x_i)$: l'écriture de x par T_i se transforme en la création par celle-ci de la version x_i.

$h\ [L_i(x)] = L_i(x_j)$: la lecture de x par T_i devient la lecture, par celle-ci, de la version x_j créée par T_j.

Une exécution multiversion effectuée par un ensemble de transactions $\{T_0,\ldots,T_n\}$ induit sur celui-ci un ordre partiel: toujours sur la base du schéma de transaction $T_i = (\Sigma_i, <_i)$ présenté en V.1.1 (cf. page 49), on pose $R = (\Sigma, <)$ où

1- Toutes les opérations soumises par une transaction sont traduites en des opérations multiversions: $\Sigma = h\left(\bigcup_{i=1}^{n} \Sigma_i\right)$.

2-Une exécution multiversion tient compte de l'ordre imposé par les transactions: quelles que soient les opérations O_i et O_i', initiées par T_i, si $O_i < O_i'$ alors $h(O_i) < h(O_i')$.

3- Une transaction ne peut lire une version que si celle-ci a été précédemment créée: Si $h[L_j(x)] = L_j(x_i)$ alors $E_i(x_i) < L_j(x_i)$.

V.1.3- EXECUTIONS MULTIVERSIONS EQUIVALENTES

Les règles et définitions de cette partie se déduisent, par analogie, de celles qui ont été énoncées pour les exécutions cohérentes équivalentes d'une approche monoversion. La notion d'objet y sera remplacée par celle de version d'un objet.

Soit R une exécution sur l'ensemble des transactions $\{T_0\ldots,T_n\}$: T_j "lit x" de T_i si T_j lit la version x créée par T_i, c'est à dire x_i. On dira donc que T_j "lit x" de T_i si et seulement si T_j lit x_i.

Or, l'application de la fonction h vue ci-dessus à la relation T_j "lit x" de T_i, soit h $[L_j(x)]$, donne $L_j(x_i)$ -T_j lit x_i-.

Les relations "lire de" sont donc entièrement déterminées par la fonction h et on pourra étendre aux exécutions multiversions la définition sur l'équivalence d'exécutions monoversions.

DEFINITION:
Deux exécutions multiversions sur un même ensemble de transactions sont équivalentes si leurs relations "lire de" renvoient les mêmes résultats, c'est-à-dire si elles effectuent les mêmes opérations.

REMARQUE:
Sous une approche multiversion, le seul type de conflit possible sera le conflit PL (Préécriture/Lecture) : la lecture d'une version est conditionnée par la création de celle-ci [$E_i(x_i) < L_j(x_i)$].

Le conflit PP (Préécriture/Préécriture) se trouve éliminé du fait que les préécritures correspondent, une fois validées, à des créations de versions. On peut toujours implanter une nouvelle version, à condition qu'elle n'invalide aucune lecture: aucun conflit ne peut exister entre deux transactions qui n'effectuent que des écritures.

Le conflit LP (Lecture/Préécriture), qui se traduit par $[L_j(x_i) < E_i(x_i)]$ ne pourra pas non plus se produire, T_j ne pouvant lire x_i si celle-ci n'a pas encore été créée. T_j lira la dernière version validée disponible.

V.1.4- SERIALISABILITE.

Une exécution multiversion peut revêtir deux aspects.

-Chacune des transactions qui l'effectue lit toujours la dernière version disponible d'un objet: on peut alors assimiler cette exécution à une exécution monoversion.

-Une transaction quelconque lit une version antérieure à la dernière disponible.

EXEMPLE:
Les exécutions R_1 et R_2 illustrent ces deux cas.

$$R_1 = E_0(x_0)\ L_2(x_0)\ L_1(x_0)\ E_1(x_1)\ E_1(y_1)\ L_3(x_1)$$
$$R_2 = E_0(x_0)\ E_0(y_0)\ L_1(x_0)\ E_1(y_1)\ L_2(y_0)$$

Dans R_2, la transaction T_2 a, pour résultat de sa lecture sur l'objet y, la version y_0 créée par T_0 et non la version y_1 qui est la dernière disponible. Dans R_1, par contre, chaque transaction a pour résultat de sa lecture la dernière version de l'objet créé.

DEFINITION:
Une exécution est "1-sérialisable" si pour tout couple de transactions (T_i,T_j) et tout objet x, si T_j "lit x" de T_i alors T_i est la plus récente transaction qui ait créé une version de l'objet x.

THEOREME:
Soit R une exécution multiversion possible sur $\{T_0,\ldots,T_n\}$.
R sera équivalente à une exécution sérielle non multiversion si elle est 1-sérialisable.

Au vu de ce qui précède, les critères déjà utilisés pour la mise en conformité des algorithmes des méthodes monoversions peuvent servir de base à la correction des algorithmes figurant les approches multiversions.

La démonstration de la "1-sérialisabilité" d'une exécution s'appuiera sur le graphe de sérialisation multiversion.

Ce graphe illustre l'ordre des opérations de lecture/écriture d'une exécution R sur l'ensemble des transactions $\{T_0,\ldots,T_n\}$. L'existence d'un circuit en son sein signifie, comme dans le cas d'une approche monoversion, une exécution non sérialisable.

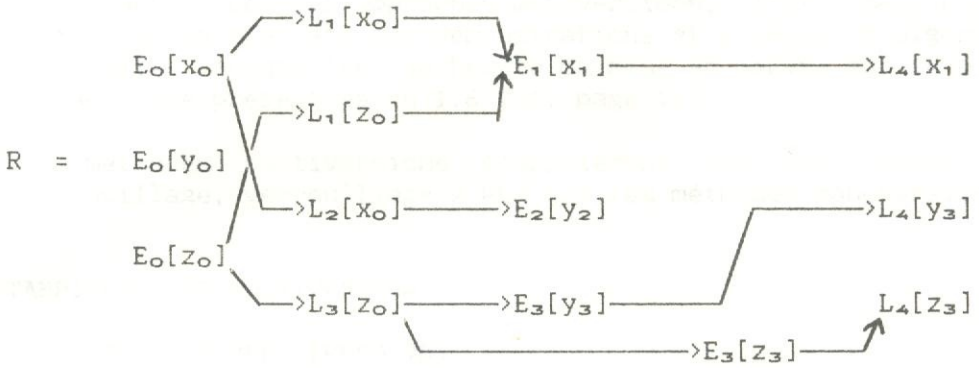
Les noeuds représentent les transactions et un couple (T_i,T_j) de celles-ci ne sera assimilé à un arc que s'il existe un objet x tel que T_j lit x de T_i. L'aspect monoversion du graphe de dépendance se définit ainsi.

Pour présenter son aspect multiversion, on prendra un ordre total quelconque sur les versions d'un objet et on définira l'ordre de versions "«" sur R en étendant cet ordre total à tous les objets.

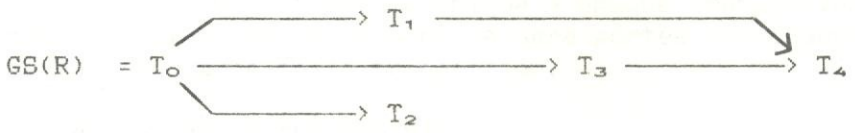
DEFINITION :
Soit R une exécution multiversion sur l'ensemble $\{T_0,\ldots,T_n\}$ et "$\ll$" un ordre de production de versions sur R. Le graphe de dépendances multiversion sera le graphe monoversion muni des arcs supplémentaires définis comme suit:

Pour tout couple d'opérations $L_k(x_j)$ et $E_i(x_i)$ dans R avec $k \neq i$, on rajoutera dans le graphe l'arc (T_i,T_k) si $x_i \ll x_j$ ou l'arc (T_k,T_i) dans le cas contraire.

EXEMPLE:
Soit R une exécution possible de l'ensemble de transactions $\{T_0,T_1,T_2,T_3,T_4\}$.



La transaction T_0 précède les transactions T_1, T_2 et T_3 puisque dans l'exécution R, T_1 "lit x" de T_0, T_1 "lit z" de T_0, T_2 "lit x" de T_0 et T_3 "lit z" de T_0. D'autre part, T_1 et T_3 précèdent T_4 puisque T_4 "lit x" de T_1, T_4 "lit y" de T_3 et T_4 "lit z" de T_3. L'aspect monoversion du graphe de R s'illustre donc de la manière suivante:



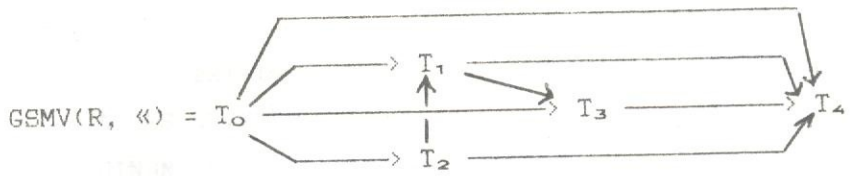
Soit alors un ordre de versions sur R tel que:

$$\begin{cases} x_0 \ll x_1 \\ y_0 \ll y_2 \ll y_3 \\ z_0 \ll z_3 \end{cases}$$

On pourra donc rajouter les arcs

(T_0,T_4) : T_4 lit x_1, T_0 écrit x_0 et $x_0 \ll x_1$.
(T_2,T_4) : T_4 lit y_3, T_2 écrit y_2 et $y_2 \ll y_3$.
(T_2,T_1) : T_2 lit x_0, T_1 écrit x_1 et $x_0 \ll x_1$.
(T_1,T_3) : T_1 lit z_0, T_3 écrit z_3 et $z_0 \ll z_3$.

D'où le graphe de dépendances multiversion:



THEOREME DE SERIALISABILITE EN MULTIVERSION:
Une exécution multiversion R est "1-sérialisable" si et seulement si il existe un ordre "«" tel que le graphe de sérialisation multiversion associé à R soit acyclique [BERN 83].

REMARQUES:
Comme dans le cas des approches monoversions, l'utilisation du graphe de sérialisation s'arrête aux démonstrations et preuves d'algorithmes: les difficultés auxquelles se heurte sa mise en oeuvre sur le plan pratique ont déjà été présentées en I.6 (cf. page 16).

Les méthodes multiversions s'appuieront sur les mêmes techniques -estampillage, verrouillage 2 PL- que les méthodes monoversions.

V.2- ESTAMPILLAGE ET MULTIVERSION

V.2.1- ALGORITHME DE REED [REED 78]

Premier à être élaboré dans le cadre d'une approche multiversion, cet algorithme illustre quelques préceptes qui seront repris par d'autres auteurs, particulièrement le fait qu'une transaction ne tient pas compte des mises à jour effectuées par une autre transaction vivante -la version d'un objet qu'elle lira étant celle qu'aura créée la dernière transaction validée avant qu'elle ne débute- et celui que les transactions lectrices peuvent invalider des transactions écrivains.

On affecte une estampille unique à chaque transaction, cela au début de son exécution. Cette estampille sera portée sur chacune des opérations de Lecture/Ecriture de la transaction.

Procédures de Lectures/Ecritures:
Nous noterons ts(i) l'estampille associée à la transaction T_i.

Procédure LIRE Req(T_i,x)

DEBUT

RENVOYER la version x_k créée par T_k telle que

- T_k validée.
- ts(k) > ts(j) quelle que soit T_j transaction validée ayant créé x_j.
- ts(k) < ts(i).

FIN

Procédure ECRIRE Req(T_i,x)

```
DEBUT
S' il existe Tj ayant lu xk avec ts(k) < ts(i) < ts(j)
ALORS REJETER Ti.
SINON créer "nouvelle version" xi.
FSI
FIN
```

Toutes les exécutions produites par cet algorithme sont 1-sérialisables [BERN 83].

L'algorithme de REED a été développé et enrichi par CAREY.

V.2.2- ALGORITHME DE CAREY [CARE 83]

Cet algorithme favorise lui aussi les transactions de lecture pure. Chaque transaction T_i y est estampillée quand elle entame son exécution et les différentes versions d'un objet x sont visualisées à travers les couples (x_i, ts(i)) où x_i représente la valeur de la version créée par la transaction T_i et ts(i) l'estampille de cette dernière.

Le contrôle de la concurrence s'appuie sur un historique géré au niveau de chaque objet x et formé d'une suite d'intervalles, chacun d'entre eux correspondant à une version de x: il est borné à gauche par l'estampille de la transaction qui a créé cette version et à droite par celle de la transaction la plus récente qui l'aurait lue.

De la même manière que précédemment, une demande de lecture sur un objet est toujours satisfaite et elle l'est avec la version validée la plus récente de cet objet.

Dans le cas où la transaction qui effectue la lecture est la plus récente, la mise à jour de l'historique se traduit par l'agrandissement de l'intervalle associé à la version renvoyée. L'estampille de la transaction lectrice, supérieure à la borne droite de l'intervalle, remplacera celle-ci.

Les transactions lectrices étant favorisées, une demande d'écriture sera rejetée si l'estampille de la transaction qui l'a émise appartient à un intervalle de l'historique. Dans le cas contraire, il y aurait création d'une nouvelle version et rajout d'un intervalle d'origine cette estampille et de longueur nulle au niveau de l'historique.

EXEMPLE:
Prenons comme historique au niveau de l'objet x la suite d'intervalles [3, 6], [10, 13] où,

- 3 représente l'estampille d'écriture de la version x_1,
- 6 l'estampille de la lecture la plus récente de cette version,
- 10 l'estampille d'écriture de la version x_2 et
- 13 l'estampille de la lecture la plus récente de x_2.

Cette synthèse montre la rigidité des méthodes classiques de contrôle continu (ordre de sérialisation imposé dans l'estampillage statique, création de problèmes d'interblocage dans le verrouillage 2PL) et les limites des méthodes optimistes (ordre de sérialisation des transactions identique à l'ordre chronologique de leur entrée en phase de certification). Les méthodes multiversions, quant à elles, héritent de tous les inconvénients liées aux techniques classiques qu'elles utilisent.

Toujours dans le souci de diminuer les rejets de transactions et, plus particulièrement, d'éliminer les inconvénients de l'estampillage statique dans les méthodes multiversions, nous avons conçu une méthode de contrôle optimiste utilisant la technique des intervalles d'estampilles sur une base de données multiversion.

La méthode Multiversion Par Intervalles d'Estampilles (MPIE) associe ainsi les avantages des méthodes multiversions au dynamisme des intervalles d'estampilles. En calculant l'ordre de sérialisation, elle diminue les risques de conflits, donc les risques de rejets de transactions et augmente ainsi le niveau de parallélisme.

Nous l'avons d'abord construite sur un schéma de transaction optimiste. Ensuite, et pour éviter le rejet des lectrices pures, nous l'avons reprise avec l'introduction d'une forme de contrôle continu sur un modèle de transaction séquentielle. Enfin, et toujours dans le but d'améliorer ses performances, nous avons procédé à son adaptation sur un modèle de transaction parallèle.

Ce document présente la méthode Multiversion Par Intervalles d'Estampilles. Il sera structuré comme suit:

Le chapitre I introduira les notions de base du contrôle de concurrence et illustrera à travers quelques exemples des occurrences de situation d'incohérence.

Nous présenterons ensuite les caractéristiques des techniques de contrôle continu, dites pessimistes, dans le chapitre II et celles des méthodes par certification ou optimistes dans le chapitre III.

Le chapitre IV particularisera l'étude de la technique des intervalles d'estampilles et soulignera son côté dynamique. Cette technique, présentée dans [BOKS 84] et particulièrement bien adaptée à la répartition, sera utilisée dans l'élaboration de notre méthode.

Nous montrerons l'intérêt des méthodes multiversions dans le chapitre V. Nous les introduirons par leurs concepts théoriques propres et les illustrerons à travers quelques approches d'auteurs, celles de [REED 78] et de [CARE 83], précurseurs de l'approche mutiversion, celle de [MING 84] qui tient compte des transactions en phase de certification en laissant leurs résultats accessibles aux autres transactions et celle de [AGRA 86] qui a pour particularité le non-contrôle des transactions lectrices.

La méthode que nous avons élaborée fera l'objet du chapitre VI. Nous y donnerons les différentes étapes de sa conception et mettrons en évidence les améliorations apportées au fur et à mesure par chacune d'elles.

En conclusion, nous présenterons le bilan de notre étude et évoquerons les perspectives d'évolution dans le domaine.

Cela peut être visualisé par le schéma suivant, aucune création de version ne pouvant s'effectuer dans les zones hachurées.

Les règles énoncées ci-dessus seront mises en évidence par les trois cas suivants:

- Une demande de lecture d'estampille 7 accède à la version x_1. L'intervalle correspondant est agrandi et devient [3, 7].

- Une demande d'écriture d'estampille 11 sera rejetée: la version x_2 ne peut être modifiée puisqu'elle a déjà été lue par la transaction d'estampille 13.

- Une demande d'écriture d'estampille 8 entraînera la création de la version x_3 et le rajout de l'intervalle [8, 8] au niveau de l'historique qui deviendra alors [3, 7], [8, 8], [10, 13].

Procédures de Lectures/Ecritures:

Procédure LIRE Req(T_i,x)

DEBUT
RENVOYER la version x_k créée par T_k telle que
- T_k validée.
- ts(k) > ts(j) quelle que soit T_j, transaction validée ayant créé x_j.
- ts(k) < ts(i).

/* Mise à jour de l'intervalle [ts(k),ts(j)] correspondant */
SI ts(i) > ts(j)
ALORS AGRANDIR l'intervalle en [ts(k), ts(i)].
SINON ne rien faire.
FSI
FIN

Procédure ECRIRE Req(T_i,x)

DEBUT
SI ts(i) appartient à un intervalle de l'historique de x
ALORS REJETER T_i
SINON FAIRE
CREER la version x_i
RAJOUTER à l'historique l'intervalle [ts(i), ts(i)]
FAIT
FSI
FIN

V.3- METHODES MULTIVERSIONS "OPTIMISTES"

V.3.1- APPROCHE DE MING YEE LAI [MING 84].

D'une manière traditionnelle, l'exécution d'une transaction se déroule en trois phases: la phase de lecture où la transaction lit les objets de la base de données et où une écriture se traduit par la création d'une nouvelle version, celle de certification dans laquelle on vérifie la 1-sérialisabilité de la transaction et enfin la phase de validation où les résultats des mises à jour effectuées sont communiqués afin qu'ils soient pris en compte par les autres transactions.

Contrairement à ce qui a été vu dans [KUNG 81] où les transactions de lecture supportent tout le poids du contrôle, l'approche multiversion est utilisée pour garantir à toute transaction une vue cohérente de la base de données. Pour cela, elle ne prend pas en compte l'éventuelle modification, effectuée par une autre transaction vivante, d'un objet qu'elle doit manipuler. La version qu'elle aura de cet objet sera celle qu'aura figée la dernière transaction validée avant qu'elle n'effectue sa lecture.

Pour mettre en oeuvre cette méthode, Ming Yee Lai a utilisé une technique d'estampillage: on associe à toute transaction T, au commencement de son exécution, une estampille de lecture -read time stamp- rts(T). A chacune de ses demandes de lecture on enverra la version de l'objet existant dans la base de données à l'instant rts(T). Ainsi, les transactions de lecture pure ont toujours une vue cohérente de la base et sont donc toujours validées.

Toute transaction d'écriture T qui entame sa phase de certification se voit attribuer l'estampille -write time stamp- wts(T): le contrôle vérifie ensuite que les objets accédés par la transaction n'ont subi aucune modification tout au long de la vie de celle-ci.

Pour une transaction T, d'estampilles rts(T) et wts(T), ce contrôle se traduira par la vérification du critère:

$$(1) \qquad \left[\bigcup_i WS(T_i) \right] \cap \left[RS(T) \cup WS(T) \right] = \emptyset$$

où RS(T) désigne l'ensemble des objets accédés en lecture par T, WS(T) celui des objets accédés en écriture par cette même transaction et T_i une transaction ayant été validée dans l'intervalle [rts(T), wts(T)].

Dans le cas où le critère (1) est vérifié, la transaction T sera validée. En effet, aucune transaction validée durant l'intervalle [rts(T), wts(T)] n'a modifié un objet accédé par T.

Dans le cas contraire, il existe au moins une transaction qui aurait été validée durant la vie de T et qui aurait modifié un objet accédé par celle-ci. T sera donc rejetée.

Le critère de la 1-sérialisabilité se base d'abord sur l'ordonnancement des estampilles d'écriture wts. Les estampilles rts ne sont prises en compte que dans le cas où elles auraient été attribuées à des transactions de lecture pure (approche de Carey).

Le choix de l'estampille de lecture rts(T) d'une transaction T est effectué par le site qui initie cette dernière. Cette estampille est unique et

est utilisée sur chacun des sites accédés par T: il suffit que l'un d'entre eux ne puisse fournir une vue cohérente de la base en fonction de cette estampille pour que T soit rejetée.

L'étape de certification se réalise à travers un protocole 2 phases. Un des sites accédés par T est choisi comme coordonnateur: il fixe alors une estampille d'écriture wts(T), unique elle ausi, et délivre un message aux sites concernés afin qu'ils fassent démarrer leurs phases de certifications locales. Chacun de ceux-ci procède alors, à son niveau, à la certification de la transaction et envoie son issue au coordonnateur lequel, après réception de tous les résultats leur diffuse celui de la certification globale.

La mise en oeuvre de ce protocole est assez contraignante: la certification y est centralisée et ne débute qu'après choix du coordonnateur.

De plus le parallélisme n'est possible que si aucun accès en écriture ne figure dans un ensemble d'accès parallèles sur un même objet: on a vu que la validation de la transaction qui aurait réalisé cette écriture invaliderait automatiquement toutes celles qui auraient effectué les accès parallèles.

L'utilisation du protocole 2 phases supprime l'atomicité de la partie certification-validation: une transaction quelconque peut donc accéder aux résultats d'une autre certifiée localement mais non encore validée. On dira que celle-ci est semi-certifiée ou bien encore "pendante".

Dans un tel cas, on pourrait soit différer cette lecture conflictuelle jusqu'à la validation de la transaction pendante, soit alors, T_i étant une transaction voulant lire la sortie de T_j semi-certifiée, réajuster l'estampille de lecture $rts(T_i)$ de manière à ce qu'elle précède $wts(T_j)$ et faire lire ainsi la version précédente à T_i.

Sous l'hypothèse d'une rareté de conflits, on dégraderait ainsi inutilement les performances -perte de parallélisme ou lecture d'une version dépassée-.

La solution "optimiste" consisterait à laisser les sorties des transactions pendantes disponibles à celles qui voudraient les lire. Ainsi, ces dernières n'enregistreraient pas de retards induits par leurs besoins en lectures et bénéficieraient des versions les plus récentes.

Par contre, leurs validations dépendraient de celles des transactions semi-certifiées qu'elles auraient lues. Il faudrait donc intégrer celles-ci dans la relation (1) -cf. page 58- en les faisant figurer dans $\bigcup_i WS(T_i)$.

Cette union comprendrait alors les transactions validées dans l'intervalle [rts(T), wts(T)] et celles semi-certifiées avant rts(T).

De plus, sachant qu'un site ne connaît l'éventuelle certification d'une transaction qu'au niveau de sa validation, cette solution optimiste suppose l'attente et le report de la validation d'une transaction après celle de la transaction pendante dont elle aurait lu la sortie.

Le problème des avortements en cascade se pose: si une transaction semi-certifiée est rejetée, toutes les transactions qui auraient lu ses sor-

ties seront rejetées elles aussi; s'il y a, parmi ces dernières, des transactions semi-certifiées qui auraient fourni des résultats à d'autres transactions, celles-ci seraient rejetées à leur tour et ainsi de suite.

Cette solution n'est donc valable que si on suppose que les conflits sont rares et que les transactions semi certifiées seront, dans la plupart des cas, validées.

V.3.2- APPROCHE D'AGRAWAL & BERNSTEIN [AGRA 86].

Cette approche offre un bon niveau de parallélisme: les transactions de lecture s'y déroulent de manière asynchrone et ne comportent pas de phase de validation. De plus, elle n'engendre pas de problème d'interblocage et ne présente pas de risques d'avortements en cascade.

Les validations des transactions d'écriture tiennent compte du non contrôle des lectrices et, de ce fait, peuvent être mises en attente -mais non rejetées-. Cette méthode sera donc particulièrement destinée aux systèmes à taux de lectrices élevé.

V.3.2.1- ENVIRONNEMENT CENTRALISE.

L'ordre de sérialisation des transactions est celui de leur entrée en phase de validation. Or, les estampilles numérotant les transactions sont affectées à la fin de la phase suivante, celle d'écriture. Comme l'ordre d'entrée des transactions en phase de validation n'est pas forcément identique à celui dans lequel elles achèvent leurs phases écriture, l'ordre induit par les estampilles ne sera pas nécessairement l'ordre de sérialisation.

L'algorithme élaboré par Agrawal et Bernstein, et étendu par eux aux bases de données relationnelles, remédie à cela en utilisant, au niveau de chacune des transactions, deux compteurs qui garantissent un ordre sérialisable aux exécutions de celles-ci.

De plus, il présente l'intéressante particularité de permettre à plusieurs transactions de se trouver en même temps en cours de validation ou en phase écriture: il admet la validation parallèle.

Les transactions en phase de validation ou d'écriture sont placées dans une file d'attente. Chaque entrée de celle-ci contient le numéro de la transaction -la file est ordonnée suivant ces derniers- et précise son type -"En Cours de Validation" ou "Ecriture Terminée"-.

Le compteur **VTNC** est affecté à chaque transaction à l'instant où elle entame son exécution. Sa valeur est supérieure ou égale au numéro de la dernière transaction qui aurait eu une vue cohérente de la base de données.Toutes les transactions de numéro inférieur ou égal à **VTNC** auront donc été traitées et auraient effectué leurs mises à jour dans le cas où elles auraient été validées.,

Le compteur **CTNC**, lui, est incrémenté à chaque fois qu'une transaction T entre en phase de validation. Il lui est alors affecté comme numéro de transaction et est utilisé pour l'insérer dans la file d'attente, dans une entrée de type "En Cours de Validation". Ce type se transformera en "Ecriture Terminée" dès que T achèvera sa phase d'écriture. On attendra alors que toutes les transactions de numéro inférieur à celui de T -qui sont donc entrées en phase de validation avant T- aient ter-

miné leur phase écriture pour communiquer les résultats des mises à jour effectuées par T, détruire son entrée dans la file d'attente et prendre son numéro comme nouvelle valeur de **VTNC**.

Cette façon de procéder supprime le risque de lecture des résultats de T avant celle des mises à jour réalisées par les transactions qui la précèdent dans l'ordre de sérialisation: ainsi, les résultats des transactions sont communiqués dans l'ordre de leur entrée en phase de validation et donc dans l'ordre de sérialisation.

Procédure de Validation Parallèle de T

```
DEBUT
«fn   <-- CTNC
 CTNC <-- CTNC + 1
 ALLOUER ENTREE dans FILE
 ENTREE.TYPE   <-- "En Cours de Validation"
 ENTREE.NUMERO <-- CTNC
 METTRE ENTREE dans FILE  »
VALID <-- vrai
    POUR i = (sn(T) + 1) A fn
    FAIRE
       SI WS (Ti) ∩ [RS (T) U WS (T)] ≠ ø
       ALORS VALID = faux
       FSI
    FAIT
SI VALID
ALORS FAIRE
      [Phase Ecriture]
      ENTREE.TYPE <-- "Ecriture Terminée"
      FAIT
SINON FAIRE
      « DETRUIRE ENTREE dans FILE »
      REJETER T
      FAIT
FSI
  «TANT QUE TETE (FILE).TYPE = "Ecriture Terminée"
   FAIRE
   VTNC <-- TETE (FILE).NUMERO;
   DETRUIRE TETE (FILE)
   FAIT »
FIN
```

Un algorithme usuel de contrôle de concurrence gère toujours un compteur de transactions qui est affecté à chacune d'elles, à l'instant où elle entame son exécution, comme numéro début de transaction.

Dans l'algorithme ci-dessus et pour une transaction T, on notera **sn(T)** ce numéro -on a vu qu'il était initialisé avec la valeur du compteur **VTNC-**, **fn** désignant celui de la dernière transaction entrée en validation avant T. Ils serviront à délimiter le début et la fin de T: ainsi les transactions dont l'exécution peut avoir interféré avec celle de T auront forcément des numéros compris entre **sn(T)** et **fn**.

T sera alors validée si les intersections entre chacun des ensembles des relations accédées en écriture par ces transactions et celui des relations accédées en lecture ou en écriture par T sont toutes égales à l'ensemble vide.

V.3.2.1.1- TECHNIQUES D'INTEGRATION ET DE COMPRESSION.

L'extension de cet algorithme aux bases de données relationnelles et le fait de s'assurer du choix de la bonne version d'une relation nuit au niveau de parallélisme. Aussi, des techniques d'optimisation, réalisables parallèlement à l'exécution d'autres transactions, ont été mises au point par ses auteurs.

INTEGRATION:
C'est le procédé de création -ou de destruction- de versions de t-uples d'une relation de la base.

Toutes les versions de t-uples d'une relation sont rangées dans un fichier et deux champs -"create et "delete"- sont gérés au niveau de chacune d'elles. Le champ "create" de la version t d'un t-uple est noté c(t) et contient le numéro de la transaction qui l'a créée. Le champ "delete" de cette même version est noté d(t) et contient, soit le numéro de la transaction qui l'aurait détruite, soit l'infini dans le cas où elle ne l'aurait pas encore été.

Une transaction T ne pourra donc lire une version t d'un t-uple que si le début de son exécution se situe entre la création et la destruction de celle-ci, autrement dit, que si l'inégalité
$c(t) \leq sn(T) < d(t)$ est vérifiée.

Les intégrations sont réalisées dans la phase écriture des transactions. La destruction d'une version t d'un t-uple ne se traduit que par l'affectation d'un numéro de transaction à d(t): cette version demeure dans le fichier, disponible à la lecture de toute transaction T_i telle que $sn(T_i) < d(t)$.

La création d'une version t d'un t-uple consiste en l'adjonction de cette version à la relation, avec un numéro de transaction assigné à c(t) et d(t) mis à l'infini. De même que pour une destruction, cette opération peut être effectuée indépendamment de l'exécution d'autres transactions puisque aucune transaction T_i telle que $sn(T_i) < c(t)$ ne pourra lire la nouvelle version. L'intégration parallèle est donc possible.

COMPRESSION:
C'est la technique de destruction -effective- de versions annulées par des transactions ayant été validées: on affecte à toute relation R une étiquette bt(R) égale au numéro de la transaction qui a créé la plus vieille version encore en cours et une transaction T ne s'exécutera que si sn(T) est supérieur à bt(R).

La compression proprement dite est alors effectuée par la recopie du fichier contenant les versions des t-uples conjuguée à l'élimination de celles dont le champs "delete" est inférieur à bt(R). Cette étiquette sera ensuite réajustée avec le numéro de transaction approprié.

Comme l'intégration, la compression est également indépendante de toute exécution de transaction.

V.3.2.2- ENVIRONNEMENT DISTRIBUE.

Dans un environnement réparti, chaque site gère ses propres compteurs de transactions. La validation suivra un protocole 2 phases: les validations locales d'une transaction précèderont sa phase écriture et seront lancées par un site coordonnateur choisi au préalable.

Durant celles-ci, les créations ou destructions de versions ne s'effectueront, au niveau d'un site donné, que sur les t-uples locaux à ce site.

La généralisation de l'algorithme de validation parallèle à un niveau réparti ne se limite pas à l'exécution de celui-ci sur chaque site, en assimilant une transaction globale à un ensemble de transactions locales indépendantes.

Prenons l'exemple où deux transactions T_1 et T_2 accèdent aux relations R_1 et R_2 implantées sur un même site avec, en notant RS l'ensemble des t-uples lus et WS celui des t-uples écrits,

$$\begin{cases} RS(T_1) = RS(T_2) = [R_1, R_2]. \\ WS(T_1) = [R_2]. \\ WS(T_2) = [R_1]. \end{cases}$$

On voit que $WS(T_1) \cap RS(T_2) \neq \emptyset$ (1)
et que $WS(T_2) \cap RS(T_1) \neq \emptyset$ (2)

Donc, d'après l'algorithme de validation parallèle, la validation de T_1 entraîne le rejet de T_2 et réciproquement.

Supposons maintenant que R_1 figure au niveau d'un site S_1, R_2 à celui d'un site S_2 et que le temps d'acheminement des messages fait que T_1 entre en validation sur le site S_1 avant T_2 et que la situation inverse se produise au niveau de S_2.

Dans ce cas, les intersections (1) et (2) ci-dessus seraient toutes les deux égales à l'ensemble vide et les deux transactions seraient validées, ce qui n'est pas le résultat recherché.

Le passage d'un environnement centralisé à un environnement distribué s'effectue à travers la gestion de deux compteurs, **gsn** -numéro début global de transaction- et **gtn** -numéro global de transaction-.

Le numéro début global d'une transaction T, **gsn(T)**, lui sera affecté à son début d'exécution par le site coordonnateur et sa valeur sera celle du compteur **VTNC** de ce dernier. T ne pourra alors lire les résultats d'une autre transaction que si celle-ci a un numéro inférieur ou égal à **gsn(T)**.

Lorsque T accèdera à un site S_i, **gsn(T)** ne réalisera pas forcément l'égalité avec le compteur $\mathbf{VTNC_i}$ puisque ce dernier n'est géré qu'au niveau de S_i.

Si **gsn(T)** est inférieur à $\mathbf{VTNC_i}$, T ne lira pas la version la plus récente sur S_i mais celle créée par la dernière transaction validée de numéro inférieur à **gsn(T)**. La vue globale que T a de la base restera cohérente.

Dans le cas contraire, T sera retardée sur S_i jusqu'à ce que toutes les transactions de numéro inférieur ou égal à **gsn(T)** et en phase de validation ou d'écriture sur ce site soient achevées. Pour empêcher alors qu'une transaction encore en phase lecture ne mette en attente T, on réajuste automatiquement le compteur $\mathbf{CTNC_i}$, au niveau de S_i, de façon à ce qu'il soit toujours au moins égal à **gsn(T)**. Ainsi, toute transaction entrant en validation sur un site auquel T aurait déjà accédé aura un numéro supérieur à **gsn(T)**.

Il n'y a donc pas de problème d'interblocage puisqu'une transaction ne peut être retardée que lors de sa phase lecture et qu'elle ne peut l'être que par une autre en phase de validation ou d'écriture.

La cohérence est aussi fonction d'un numéro de transaction unique sur tous les sites accédés. Ce numéro global de transaction sera noté **gtn(T)** pour une transaction T et sera assigné de la manière suivante.

Au début de la phase validation, chaque site envoie la valeur de son compteur **CTNC** au coordonnateur. Celui-ci fixe alors **gtn(T)** comme étant la valeur maximum des **CTNC** augmentée d'un facteur de sécurité. Ce facteur est rajouté pour essayer de compenser le temps d'acheminement des messages: c'est une estimation du nombre de transactions entrées en validation sur un site après l'envoi de son **CTNC**.

Comme pour **gsn(T)**, un problème de synchronisation va se poser: si, au niveau du site Si, **gtn(T)** est supérieur à $\mathbf{CTNC_i}$, il suffirait de réajuster ce dernier à la valeur de **gtn(T)**.

Dans le cas contraire, il existerait une transaction T_1, de numéro supérieur ou égal à **gtn(T)**, déjà en phase de validation sur S_i. L'ordre 1-sérialisable des transactions étant celui de leur entrée en phase de validation, T_1 précède alors T dans cet ordre et **gtn(T)** doit être rectifié à la hausse. C'est le site coordonnateur qui s'occupera de cette correction et de la diffusion de la nouvelle valeur de **gtn(T)**.

La validation de T ne sera pas pour autant retardée sur les autres sites. Avant la réception du nouveau numéro, ils continueront à l'effectuer avec l'ancien. En effet, l'ensemble des transactions qui pour-

raient avoir interféré avec T, donc de numéro compris entre **gsn(T)** et "ancien" **gtn(T)**, est inclus dans celui des transactions de numéro compris entre **gsn(T)** et "nouveau" **gtn(T)** puisque **gsn(T)** n'a pas varié et que "ancien" **gtn(T)** est inférieur à nouveau **gtn(T)**. Les sites compléteront la validation de T avec ce dernier dès qu'il leur sera connu.

V.4- CONCLUSION

Dans une base de données multiversion, les opérations sur les objets initiées par l'utilisateur sont traduites en opérations sur des versions d'objets.

Le contrôle de concurrence ne porte plus sur les transactions de lecture pure qui disposent d'un choix de plusieurs versions pour un même objet.

En particulier, et du fait qu'une mise à jour sur un objet est matérialisée par la création d'une version de cet objet, les vieilles lectrices ne sont plus rejetées comme dans le cas d'une base de données monoversion.

Les méthodes multiversions diminuent donc le nombre de rejets et offrent un meilleur niveau de parallélisme. Par contre, la cohabitation de plusieurs versions d'un objet en mémoire rend assez délicate la gestion de celle-ci. Par exemple, pour éviter les "surcharges", on doit procéder régulièrement à une destruction de versions et celle-ci devra tenir compte des éventuelles demandes de lecture des transactions actives.

Les algorithmes présentés mettent en évidence les avantages de ces méthodes sur les méthodes monoversions.

-Une amélioration de la technique de l'estampillage statique est illustrée par les algorithmes de [REED 78] et de [CARE 83]. L'estampillage multiversion permet de ne pas rejeter les transactions de lecture pure dont l'ordre chronologique diffère de l'ordre de sérialisation.

-L'approche de [MING 84] généralise la certification basée sur la gestion d'ensembles d'objets aux bases de données multiversions. De plus et si les conflits sont effectivement rares, elle améliore le niveau de parallélisme en tenant compte des transactions semi-certifiées.

-L'algorithme de [AGRA 86] se base sur un double estampillage des transactions et une gestion de files d'attente de celles-ci pour contrôler les écrivains et ne renvoyer que des valeurs cohérentes aux lectrices pures. Sur le plan pratique, il pose le problème de la dérive des compteurs et, comme celui de [MING 84], présente l'inconvénient d'un contrôle global centralisé (site coordonnateur privilégié).

VI.1 Structure et gestion des données utilisées
VI.1.1 Exemple

VI.2 Ordre de sérialisation
VI.3 Accès aux objets
VI.3.1 Accès en lecture
VI.3.2 Accès en écriture

VI.4 Certification et validation répartie
VI.4.1 Certification locale
VI.4.2 Certification globale
VI.4.3 Validation locale

VI.5 Réajustement des intervalles de sérialisation
VI.6 Etude des rejets
VI.6.1 Rejet local d'une transaction
VI.6.2 Rejet global d'une transaction

VI.7 Choix de l'estampille de certification
VI.7.1 Choix de l'intervalle
VI.7.2 Choix de l'estampille

VI.8 Contrôle mixte (continu et par certification)
VI.8.1 Accès aux objets
VI.8.2 Certification et validation
VI.8.3 Deuxième approche
VI.8.2.1 Procédures

VI.9 Prolongement: Modèle de transaction répartie
VI.10 Etude de la sérialisabilité
VI.11 Exemples d'application
VI.12 Privation
VI.13 Conclusion

I.1 Sérialisabilité
I.2 Recouvrabilité - Atomicité
I.3 Modèle de transaction
I.4 Transaction vivante - Transaction validée
I.5 Conflit - Dépendance
I.6 Graphe de dépendances

I.7 Exemples de situations d'incohérence
- I.7.1 Perte d'opération
- I.7.2 Situation d'incohérence
- I.7.3 Perte de la cohérence
- I.7.4 Sérialisabilité et cohérence (aspect sémantique)

Les méthodes multiversions exposées précédemment restent soumises à tous les inconvénients liés aux techniques classiques qu'elles utilisent. Par exemple, l'estampillage statique multiversion, bien qu'il constitue une amélioration de la technique de base, instaure une dépendance systématique entre chaque couple de transactions et impose un ordre qui ne tient pas compte des contraintes induites par les conflits. On a vu qu'une transaction à l'origine d'un conflit n'engendrant pas de circuit dans le graphe de sérialisation est quand même rejetée si la dépendance induite par ce conflit ne correspond pas à l'ordre fixé par les estampilles.

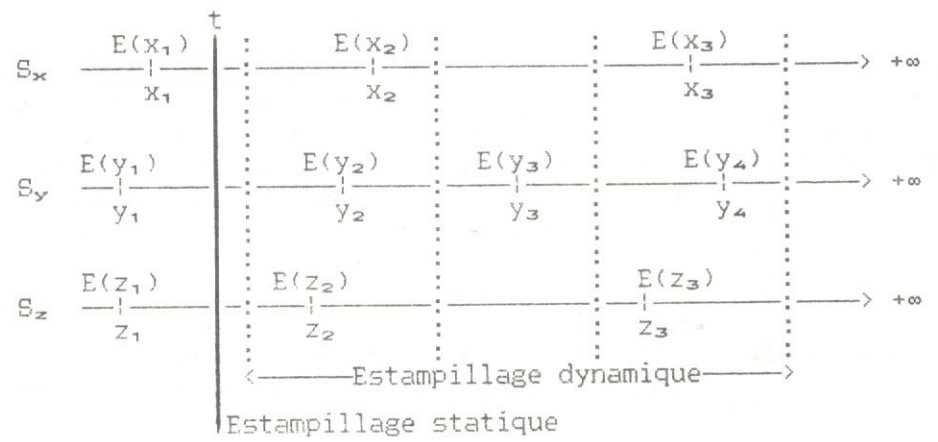
Pour pallier aux inconvénients de cet aspect statique, nous proposons une mise en pratique de la technique des intervalles d'estampilles sur les méthodes multiversions. La méthode "Multiversion Par Intervalles d'Estampilles" construite de cette manière et notée MPIE par la suite associera ainsi les avantages liés aux méthodes multiversions à ceux qui correspondent à l'utilisation de la technique des intervalles d'estampilles.

Son aspect multiversion minimisera les possibilités de rejet -grâce à la gestion de l'historique des données- et offrira un niveau de parallélisme plus élevé que celui des méthodes monoversions, qu'elles soient continues ou par certification. Il favorise les transactions de lecture pure qui disposent d'un choix de versions pour un même objet et qui ne sont donc pas contrôlées et offre à toutes les transactions de plus larges éventualités de sérialisation.

La technique des intervalles d'estampilles admet qu'une transaction vivante précède dans l'ordre de sérialisation une autre déjà validée: l'utilisation de cette technique permettra un ordre de sérialisation différent de l'ordre chronologique des certifications. La disponibilité d'un ensemble de versions pour chaque objet de la base contribue aussi à cet avantage mais la technique des intervalles introduira un caractère dynamique dans le calcul de l'ordre de sérialisation. Cela constitue une amélioration par rapport aux méthodes multiversions classiques.

Exemple 1: l'estampillage dynamique offre aux transactions de lecture un ensemble de possibilités de sérialisation.

Prenons la base multiversion représentée par les objets x, y et z.



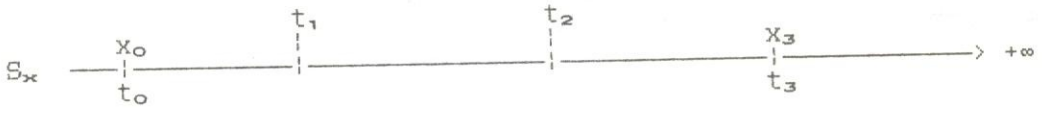
Dans le cas d'un estampillage statique, les versions renvoyées par une lecture seront pleinement déterminées par la relation $E(o_i) < t$ où $E(o_i)$ représente l'estampille de création de la version o_i et t celle associée à la lecture. La valeur de t étant fixée au préalable, une seule possibilité est donc offerte.

Quand le calcul de l'ordre de sérialisation se fait d'une manière dynamique, la valeur de t est choisie à postériori, dans la phase terminale de certification: la relation $E(o_i) < t$ ne fige plus une situation et les lectures pourront s'effectuer suivant un ensemble de possibilités.

Exemple 2: L'estampillage statique peut être à l'origine de rejets de transactions écrivains.

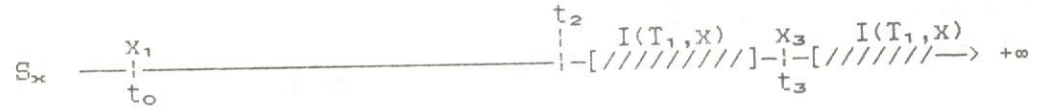
Soient les transactions T_1 et T_2 d'estampilles respectives t_1 et t_2 avec $t_1 < t_2$ et soient x_0 et x_3 les deux versions disponibles de l'objet x.

T_1	T_2
1 LIRE x	2 LIRE x
3 PREECRIRE x	4 CERTIFIER (T_2)
5 CERTIFIER (T_1)	
6 ECRIRE x	



Dans le cas d'un estampillage statique, la transaction écrivain T_1 sera rejetée à l'instant 5 puisqu'elle invalide la lecture effectuée par T_2.

Au contraire, une méthode multiversion utilisant la technique des intervalles d'estampilles empêcherait ce rejet: T_2 certifie avec l'estampille t_2 avant T_1 mais T_1 peut sérialiser avec une estampille choisie supérieure à t_2. En fait, T_1 a deux possibilités de sérialisation: soit entre t_2 et t_3, en ayant lu la version x_0, soit après t_3, en ayant lu la version x_3.



La méthode MPIE a été construite, d'après le principe des méthodes optimistes, sur un schéma de transaction qui comporte trois phases: accès aux objets, certification et validation. Les accès en lecture ou en préécriture ne sont pas contrôlés et on permet à la transaction de manipuler des données incohérentes. Un premier point de synchronisation est observé à la fin des phases de lectures locales. Le contrôle de concurrence s'effectue lors de l'étape de certification de la transaction: elle sera rejetée si elle provoque une incohérence ou si elle a lu des données non cohérentes. Dans le cas contraire, les résultats des lectures sont soumis à l'utilisateur. Si, de plus, la transaction a mis à jour des objets, elle déroule sa phase de validation: c'est durant celle-ci que les modifications effectuées sont prises en compte dans la base.

En tant que méthode multiversion , elle gère un ensemble de versions pour chaque

objet de la base: la lecture d'un objet sera celle d'une version de cet objet et une écriture dans la base se traduira par la création d'une nouvelle version.

Du fait de l'utilisation de la technique des intervalles d'estampilles, la sérialisation d'une transaction vivante ne tient compte que de ses dépendances avec les transactions déjà validées. Cela implique un ensemble d'intervalles de sérialisation possible au niveau de chaque transaction vivante. Cet ensemble, construit dynamiquement, exprimera ces dépendances et fournira l'estampille que l'on associera à la transaction lors de sa validation.

Dans ce chapitre, nous présenterons d'abord la méthode MPIE sur un schéma de transaction optimiste. Nous en donnerons les principaux mécanismes, notamment en ce qui concerne les accès aux objets, avant de développer son étude dans le contexte d'un environnement réparti.

Nous reprendrons ensuite l'étude en introduisant une forme de contrôle continu, par le biais d'un intervalle courant global, sur un modèle de transaction séquentielle. Pour améliorer les performances de la méthode, nous l'adapterons aussi à un modèle de transaction parallèle.

Enfin, quelques exemples illustreront la méthode et clôtureront le chapitre.

VI.1- STRUCTURE ET GESTION DES DONNEES UTILISEES

a) Estampilles associées aux transactions:
On associe une estampille numérique positive unique à chaque transaction validée. Elle est prélevée dans l'ensemble d'intervalles d'estampilles adjoint à la transaction et désigne l'emplacement de T dans l'ordre de sérialisation des transactions validées.

b) Estampilles associées aux versions:
A chaque version x_i d'un objet x, on fait correspondre une estampille d'écriture $E(x_i)$ et une estampille de lecture $L(x_i)$. Ces estampilles seront affectées -cas des estampilles d'écriture- ou mises à jour -pour les estampilles de lecture- lors de la validation des transactions.

Soit t l'estampille associée à une transaction validée T:
si T a lu la version x_i, alors, $L(x_i) = \max[L(x_i), t]$,
si T a créé la version x_i, alors, $E(x_i) = t$.

De cette manière, $E(x_i)$ désignera l'estampille de la transaction validée qui a créé x_i et $L(x_i)$ celle de la transaction validée la plus récente, dans l'ordre de sérialisation, qui l'a lue.

c) Ensembles d'intervalles associés aux objets:
Au niveau de chaque objet de la base, on assurera la maintenance de deux ensembles "locaux" d'intervalles. Ces ensembles devront représenter les possibilités de sérialisation des transactions vivantes accédant à l'objet. Ils devront matérialiser toutes les dépendances liant ces transactions à celles, déjà validées, qui ont accédé à ce même objet. L'un sera noté "LSP" et sera destiné aux accès en lecture, l'autre, "ESP", concernera les accès en écriture.

Au niveau d'un objet x, ils seront définis en fonction des versions x_i de cet objet. Chaque intervalle qui les compose correspondra à une de ces versions.

Si la transaction vivante T accède en lecture à la version x_i, l'intervalle associé -où elle pourra sérialiser- est borné, à gauche par $E(x_i)$ et à droite, par $E(x_{i+1})$, x_{i+1} étant la version suivante si elle existe. Les dépendances existant entre T et les transactions validées ayant accédé à x seront alors exprimées ainsi:

. T_{Ei}, d'estampille $E(x_i)$, transaction validée qui a créé x_i, précède T dans l'ordre de sérialisation: l'estampille de T devra être supérieure à celle de T_{Ei}.
. T_{Ei+1}, d'estampille $E(x_{i+1})$, transaction validée qui a créé x_{i+1}, suit T dans l'ordre de sérialisation: l'estampille de T devra être inférieure à celle de T_{Ei+1}.

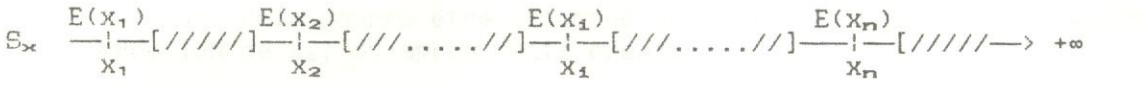
Si T crée une version postérieurement à une version x_i, l'intervalle associé est borné, à gauche par $L(x_i)$ -ou par $E(x_i)$ si x_i n'a pas encore été lue- et à droite par $E(x_{i+1})$, x_{i+1} étant la version suivante si elle existe.

En notant T_{Li}, d'estampille $L(x_i)$, la plus récente transaction validée qui a lu x_i, les dépendances qui lient T aux transactions validées seront illustrées, au niveau de l'objet x, par:

. T_{Ei} et T_{Li} précèdent T dans l'ordre de sérialisation: l'estampille de T devra majorer celles de T_{Ei} et de T_{Li}.
. T précède T_{Ei+1} dans l'ordre de sérialisation: son estampille devra être inférieure à celle de T_{Ei+1}.

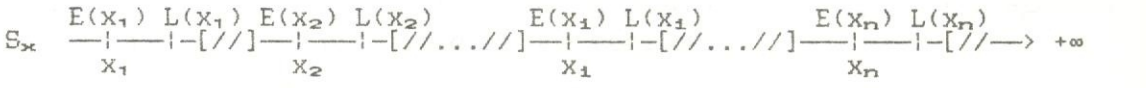
Au niveau d'un site S_x, les intervalles de sérialisation locaux sont donnés par les zones hachurées des diagrammes ci-dessous.

. Accès en lecture: $LSP(x) = \bigcup_{i=1}^{n-1} [E(x_i)+1,\ E(x_{i+1})-1] \cup [E(x_n)+1,\ +\infty[$



. Accès en écriture: $ESP(x) = \bigcup_{i=1}^{n-1} [L'(x_i)+1,\ E(x_{i+1})-1] \cup [L'(x_n)+1,\ +\infty[$

où $L'(x_i)$ est égale soit à $L(x_i)$, estampille de la transaction la plus récente ayant lu la version x_i, soit à $E(x_i)$, si la version x_i n'a pas encore été lue.



d) Ensembles d'intervalles locaux associés aux transactions:
Suivant les accès effectués par une transaction vivante (lecture, écriture ou lecture/écriture), une copie de l'un de ces ensembles (LSP ou ESP) ou une synthèse des deux sera maintenue et gérée au niveau de celle-ci. Cette copie représentera l'ensemble des intervalles de sérialisation possible de la transaction et désignera son emplacement dans l'ordre local relatif au site S_x. Elle sera désignée par EI(T,x) pour une transaction vivante T accédant à l'objet x et obéira aux principes suivants.

. Pour un accès en lecture, on effectue: $EI(T,x) := EI(T,x) \cap LSP(x)$

. Lors d'une préécriture $EI(T,x)$ ne subit pas de modification. Mais en phase de certification locale et pour un accès en écriture, on effectue: $EI(T,x) := EI(T,x) \cap ESP(x)$

e) Ensembles de transactions:

Chaque site S_x gère l'ensemble des transactions vivantes ayant accédé aux versions de l'objet x et fait correspondre un ensemble local d'intervalles de sérialisation possible à chacune d'elles. Ces transactions sont définies par leur état (vivante, validée) et leur mode d'accès (lecture ou écriture).

Elles seront distinguées, suivant que leurs accès à l'objet x soient des lectures ou des écritures, en deux ensembles, désignés par lectrices(x) et écrivains(x). Une transaction peut appartenir à la fois à lectrices(x) et à écrivains(x).

VI.1.1- EXEMPLE

Prenons un objet x représenté par les versions x_1 et x_2. Soient $E(x_1)$ et $E(x_2)$ les estampilles des transactions validées qui ont créé ces versions et $L(x_1)$ celle de la transaction validée la plus récente, dans l'ordre de sérialisation, qui a lu la version x_1.

- Un accès en lecture sur l'objet x renverra, soit la version x_1, soit la version x_2. La sérialisation de cet accès pourra se faire après la création de x_1 et avant celle de x_2 ou après la création de x_2.

- Un accès en écriture se traduira par la création d'une troisième version. Cela pourra être effectué après la lecture de x_1 et avant la création de x_2, ou après cette création.

Les deux ensembles d'intervalles de sérialisation locaux seront:

$$LSP(x) = [E(x_1) + 1,\ E(x_2) - 1] \cup [E(x_2) + 1,\ +\infty[,$$
$$ESP(x) = [L(x_1) + 1,\ E(x_2) - 1] \cup [E(x_2) + 1,\ +\infty[$$

L'estampille de lecture $L(x_1)$ est utilisée pour le réajustement de l'intervalle -troncature à gauche- dans le cas où T_{L1}, dernière transaction ayant lu x_1 et donc d'estampille $L(x_1)$, existe et que $T_{L1} <_x T$. Cette estampille sert au calcul de l'endroit où une transaction de lecture/écriture peut insérer une version.

VI.2- ORDRE DE SERIALISATION

Comme dans le cas monoversion, les dépendances entre la transaction T vivante et celles qui sont validées seront matérialisées dans un ensemble global $EI_G(T)$, au niveau de tous les objets accédés, de la forme:

$$EI_G(T) = \bigcap_{x \in X} EI(T,x) \qquad \text{avec } X = \{\text{objets accédés par T}\}$$

- Sur le site S_y, l'accès en lecture de l'objet y ne pourra s'effectuer qu'à travers celle de la version y_1, seule créée.

$$EI(T,y) = [E(y_1) + 1, +\infty [$$

- Sur le site S_z, l'accès en lecture de l'objet z peut porter sur la version z_1, donc après sa création et avant celle de z_2, ou sur la version z_2, après sa création.

$$EI(T,z) = [E(z_1) + 1, E(z_2) - 1] \cup [E(z_2) + 1, +\infty [$$

L'ensemble global des intervalles de T, intersection des ensembles locaux sera $EI_G(T) = EI(T,x) \cap EI(T,y) \cap EI(T,z)$

En tenant compte des inégalités exprimées par le schéma, à savoir $L(x_1) = \max [L(x_1), E(x(_1), E(y_1), E(z_1)]$ et $E(z_2) < E(x_2)$, on obtient

$$EI_G(T) = [L(x_1)+1, E(z_2)-1] \cup [E(z_2)+1, E(x_2)-1] \cup [E(x_2)+1, +\infty[$$

et toute estampille choisie dans cet intervalle respectera les dépendances de T avec les transactions déjà validées et ayant accédé aux objets x, y et z.

REMARQUE:
Le critère de sérialisation globale $EI_G(T) \neq \emptyset$ ne dépend pas de la répartition des objets accédés par T sur les sites.

Généralisons à plusieurs objets par site et désignons par S l'ensemble des sites accédés par une transaction T, par X_i l'ensemble des objets du site S_i qu'elle a lus ou préécrits et enfin par X l'ensemble des objets auxquels elle a accédé sur tous les sites.

L'ensemble global d'intervalles de sérialisation $EI_G(T)$ sera de la forme:

$$EI_G(T) = \bigcap_{S_i \in S} EI(T,S_i) = \bigcap_{S_i \in S} \left(\bigcap_{x \in X_i} EI(T,x) \right) = \bigcap_{x \in X} EI(T,x)$$

Notre étude pourra donc se baser sur l'hypothèse d'un objet par site.

Cette hypothèse, qui devra être reconsidérée dans le cas d'une implémentation effective, n'altèrera ni les caractéristiques de la méthode, ni ses propriétés.

VI.3- ACCES AUX OBJETS.

Le modèle de transaction défini dans toutes les méthodes étudiées -et que nous avons adopté pour la méthode MPIE- assure l'atomicité de la transaction en lui allouant un espace de travail au début de son exécution: mis à part la première lecture qui a lieu sur la base, tous les accès aux objets, en lecture comme en écriture, s'effectuent sur cet espace de travail et les écritures ne sont répercutées sur la base de données qu'au niveau de la phase de validation.

De plus, dans cette méthode, les accès se font uniquement en fonction des transactions déjà validées. Les autres transactions vivantes et accédant aux mêmes objets ne sont pas prises en compte. L'ensemble des intervalles de sérialisation possible d'une transaction, pour une lecture comme pour une préécriture, n'est fonction que des estampilles des transactions validées.

- Sur le site S_y, l'accès en lecture de l'objet y ne pourra s'effectuer qu'à travers celle de la version y_1, seule créée.

$$EI(T,y) = [E(y_1) + 1, +\infty [$$

- Sur le site S_z, l'accès en lecture de l'objet z peut porter sur la version z_1, donc après sa création et avant celle de z_2, ou sur la version z_2, après sa création.

$$EI(T,z) = [E(z_1) + 1, E(z_2) - 1] \cup [E(z_2) + 1, +\infty [$$

L'ensemble global des intervalles de T, intersection des ensembles locaux sera $EI_G(T) = EI(T,x) \cap EI(T,y) \cap EI(T,z)$

En tenant compte des inégalités exprimées par le schéma, à savoir $L(x_1) = \max [L(x_1), E(x(_1), E(y_1), E(z_1)]$ et $E(z_2) < E(x_2)$, on obtient

$$EI_G(T) = [L(x_1)+1, E(z_2)-1] \cup [E(z_2)+1, E(x_2)-1] \cup [E(x_2)+1, +\infty[$$

et toute estampille choisie dans cet intervalle respectera les dépendances de T avec les transactions déjà validées et ayant accédé aux objets x, y et z.

REMARQUE:
Le critère de sérialisation globale $EI_G(T) \neq \emptyset$ ne dépend pas de la répartition des objets accédés par T sur les sites.

Généralisons à plusieurs objets par site et désignons par S l'ensemble des sites accédés par une transaction T, par X_i l'ensemble des objets du site S_i qu'elle a lus ou préécrits et enfin par X l'ensemble des objets auxquels elle a accédé sur tous les sites.

L'ensemble global d'intervalles de sérialisation $EI_G(T)$ sera de la forme:

$$EI_G(T) = \bigcap_{S_i \in S} EI(T,S_i) = \bigcap_{S_i \in S} \left(\bigcap_{x \in X_i} EI(T,x) \right) = \bigcap_{x \in X} EI(T,x)$$

Notre étude pourra donc se baser sur l'hypothèse d'un objet par site.

Cette hypothèse, qui devra être reconsidérée dans le cas d'une implémentation effective, n'altèrera ni les caractéristiques de la méthode, ni ses propriétés.

VI.3- ACCES AUX OBJETS.

Le modèle de transaction défini dans toutes les méthodes étudiées -et que nous avons adopté pour la méthode MPIE- assure l'atomicité de la transaction en lui allouant un espace de travail au début de son exécution: mis à part la première lecture qui a lieu sur la base, tous les accès aux objets, en lecture comme en écriture, s'effectuent sur cet espace de travail et les écritures ne sont répercutées sur la base de données qu'au niveau de la phase de validation.

De plus, dans cette méthode, les accès se font uniquement en fonction des transactions déjà validées. Les autres transactions vivantes et accédant aux mêmes objets ne sont pas prises en compte. L'ensemble des intervalles de sérialisation possible d'une transaction, pour une lecture comme pour une préécriture, n'est fonction que des estampilles des transactions validées.

VI.3.1- ACCES EN LECTURE

Procédure LIRE (T,x)

```
DEBUT
SI      T ∉ (lectrices(x) U écrivains(x))
ALORS   EI(T,x) := [0, +∞ [
FSI

lectrices(x) := lectrices(x) U {T}
EI(T,x) := EI(T,x) ∩ LSP(x)
CHOISIR intervalle I dans EI(T,x) tel que
        borne inf(I) > borne inf(I') quelquesoit I' ∈ EI(T,x)
v := valeur lue (x_i) /* xi est telle que E(x_i)=borne inf(I) - 1 */
ENVOYER v au site S_T /* S_T: site qui a initié T */
EI(T,x) := I
FIN
```

Dans l'algorithme proposé, le choix de l'intervalle de lecture I dans EI(T,x) sera effectué de manière à accéder à la dernière version de x.

Après toute lecture d'une version x_i, EI(T,x) sera tronqué et borné à gauche par $E(x_i)$, de manière a exprimer la dépendance qui lie cette lecture à la transaction validée qui a créé x_i. En particulier, une éventuelle écriture de l'objet x par la même transaction T ne pourra donner une version antérieure à x_i.

En désignant la plus ancienne version disponible par x_1, l'intersection entre EI(T,x) et LSP(x) sera vide si la borne droite de EI(T,x) est inférieure à l'estampille d'écriture $E(x_1)$: la transaction T, trop vieille lectrice, sera rejetée dans ce cas-là.

La gestion de la mémoire à travers la destruction des vieilles versions devra tenir compte des demandes de lecture des transactions encore actives. Pour éviter le rejet des lectrices, il faudra déterminer la plus vieille transaction vivante sur les sites avant de procéder à l'élimination des versions inutilisées.

VI.3.2- ACCES EN ECRITURE

Les accès en écriture se traduisent par des préécritures, c'est-à-dire se limitent à des écritures sur l'espace de travail associé à la transaction. Ces préécritures ne sont pas prises en compte lors de l'accès et n'entraînent pas de réajustement d'intervalles. Elles seront contrôlées ultérieurement, lors de la certification de la transaction.

```
Procédure PREECRIRE (T,x)
DEBUT
SI      T ∉ (lectrices(x) U écrivains(x))
ALORS   EI(T,x) := [0, +∞ [
FSI
écrivains(x) := écrivains(x) U {T}
ECRIRE  l'objet x dans l'espace de travail de T
FIN
```

VI.4- CERTIFICATION ET VALIDATION REPARTIE

Le modèle de certification-validation répartie utilisé est une adaptation du modèle présenté dans [KUNG 81] pour un environnement centralisé (cf. Fig. 1 page 32). Il comporte trois phases: une de certification locale, une de certification globale et une de validation locale qui n'est initiée que si la certification globale réussit. Ces trois phases se déroulent toutes sur chacun des sites accédés. Il n'y a pas de site privilégié. Seul le site qui a initié la transaction joue un rôle de coordonnateur: il lance les phases de certifications locales et, dans le cas d'un rejet à ce niveau, fait avorter la transaction sur l'ensemble des sites auxquels elle a accédé sans attendre la phase de certification globale.

Les certifications-validations de transactions distinctes peuvent s'exécuter en parallèle sur les sites: par exemple, une transaction peut dérouler sa certification globale ou sa validation locale sur un site pendant qu'une deuxième exécute sa certification locale sur un site différent.

En général, la certification-validation d'une transaction ne s'exécute pas en phase d'exclusion mutuelle sur un site: d'autres transactions peuvent y dérouler parallèlement leurs accès aux objets.

VI.4.1- CERTIFICATION LOCALE

Elle démarre sur le site S_x dès que celui-ci reçoit le message "certifier T" émis par le site S_T qui a initié la transaction T. Ce message fera connaître à S_x les autres sites accédés. Les phases de certifications locales LC(T,x) se déroulent en parallèle sur tous les sites S_x des objets x accédés par T. Chacun de ces sites contrôle la sérialisabilité locale de T: l'ensemble des intervalles de sérialisation possible au niveau local sera déterminé au cours de cette phase.

Par contre, le choix de l'intervalle de sérialisation ne s'effectue pas à ce niveau: au terme de cette phase et en cas de non rejet local, c'est l'ensemble des intervalles de sérialisation possible au niveau local, EI(T,x), qui sera diffusé aux sites qui ont été accédés par T. L'envoi de ces messages entre les sites constitue le point de synchronisation 2 (cf. Fig. 1 page 32). La réception sur un site des messages de tous les autres, déclenchera, au niveau de ce site, la phase de certification globale. La détermination de l'intervalle de sérialisation se fera lors de celle-ci. Ainsi, le choix aura lieu sur la totalité des ensembles d'intervalles de sérialisation et les rejets dus à un mauvais choix d'intervalle au niveau local seront évités.

Une transaction T est rejetée par un site S_x si l'intersection de

EI(T,x) avec ESP(x) est vide. Le site S_x envoie alors un message de rejet local au site qui a initié T.

Procédure de certification locale LC(T,x) au site S_x

```
DEBUT
SI   T ∉ écrivains(x)      /* cas où T est une lectrice pure */
ALORS DIFFUSER EI(T,x)  à tout site Sy / y ∈ {objets accédés}
SINON FAIRE
      EI(T,x) := EI(T,x) ∩ ESP(x)
       /* test sur le rejet local: */
       SI EI(T,x) =  ∅
       ALORS ENVOYER  message de rejet à ST
       SINON DIFFUSER EI(T,x) à tout site Sy / y ∈ {objets accédés}
       FSI
      FAIT
FSI
FIN
```

Dans le cas où un rejet est détecté au niveau d'un site quelconque, le site qui a initié T, destinataire du message de rejet local, fait avorter la transaction sur tous les sites auxquels elle a accédé, sans attendre les résultats des autres certifications locales. La phase suivante, celle de certification globale, ne démarrera donc que si aucun rejet local n'a été signalé.

VI.4.2- CERTIFICATION GLOBALE

La certification globale d'une transaction T s'exécute sur l'ensemble des sites auxquels elle a accédé et elle ne débute sur chacun d'entre eux qu'après qu'il ait reçu les résultats des certifications locales de tous les autres - rappelons qu'un site accédé par une transaction est informé des autres sites qu'elle accède par le message qui déclenche la phase de certification locale -.

Procédure de certification globale GC(T)

```
DEBUT
EIG(T) :=   ∩   EI(T,x)   /*  X = {objets accédés par T} */
          x ∈ X
SI    EIG(T) = ∅
ALORS RGC(T) := (faux, ?)   /* Cas de rejet global */
SINON  FAIRE
       CHOISIR intervalle I dans EIG(T) puis estampille t dans I
       RGC(T) :=  (vrai,t)
       FAIT
FSI
FIN
```

Nous introduisons dans ce chapitre quelques notions de base, règles et définitions régissant la plupart des méthodes du contrôle de concurrence.

Nous y présentons le modèle de transaction utilisé par toutes les méthodes développées à ce jour ainsi que le graphe visualisant les dépendances entre transactions et illustrons, à travers des exemples, quelques occurrences de situations d'incohérence.

I.1- SERIALISABILITE

La plupart des méthodes du contrôle de concurrence reposent sur le critère de la sérialisabilité. Il garantit l'équivalence entre l'exécution concurrente d'un ensemble de transactions, d'une part, et l'exécution en série de ces mêmes transactions, d'autre part. Ainsi, la cohérence de la base est assurée et l'ensemble des transactions ont une vue cohérente de celle-ci.

Propriété suffisante pour assurer la cohérence, la sérialisabilité est cependant une condition trop forte [PAPA 79]. En tenant compte de la sémantique des opérations, il existe des exécutions non sérialisables qui n'altèrent pas la cohérence de la base.

I.2- RECOUVRABILITE - ATOMICITE

La recouvrabilité d'une transaction est assurée par son exécution totale et entière. L'exécution partielle d'une transaction, en cas de panne hardware par exemple, peut entraîner une incohérence de la base. Les effets des exécutions inachevées doivent donc être éliminés.

Une transaction qui vérifie les contraintes de recouvrabilité et de sérialisabilité sera dite atomique.

I.3- MODELE DE TRANSACTION

Dans la plupart des articles traitant du contrôle de concurrence, tant continu que par certification, les transactions illustrant les méthodes sont d'un modèle assez général: elles y sont définies comme une suite d'opérations de Lectures/Ecritures.

Pour assurer la contrainte d'atomicité d'une transaction, un espace de travail, inaccessible aux autres transactions, lui est alloué au début de son exécution. Tous les accès aux objets qu'elle exécute sont alors effectués sur des copies de ces objets, dans cet espace de travail: une lecture retourne la valeur de la copie ou celle de l'objet dans la base si la copie n'existe pas et une écriture, qui s'exécute toujours sur l'espace de travail, met à jour la copie après l'avoir éventuellement créée. Ce n'est qu'au niveau de l'étape de validation de la transaction que les écritures sont répercutées sur la base de données. L'espace de travail est alors détruit.

L'intention d'écrire un objet dans la base doit être annoncée avant que la transaction n'atteigne son point de validation. Une opération de préécriture a donc été définie: elle signale que la transaction doit effectuer une écriture sur l'objet désigné et elle réalise cette écriture sur l'espace de

Le résultat de la certification globale de la transaction T, RGC(T), est composé d'une variable booléenne et d'une variable estampille. Le booléen est mis à vrai ou à faux suivant que le contrôle global a réussi ou pas. Dans l'affirmative, le choix de l'estampille de T se fera, à ce niveau, dans l'ensemble des intervalles de sérialisation possible $EI_G(T)$, intersection de tous les ensembles d'intervalles locaux. Ce choix doit s'effectuer de la même façon sur chacun des sites concernés. Ils devront mettre en oeuvre le même algorithme de calcul de l'estampille, de manière à ce que sa valeur soit la même sur l'ensemble des sites accédés. Une fois déterminée, cette valeur est affectée à la variable estampille de RGC(T).

Une étude plus détaillée sur le choix de l'estampille et sur les conséquences de celui-ci, tant sur les transactions vivantes que sur celles qui débuteraient leur exécution ultérieurement, sera introduite en VI.7 (cf. page 82).

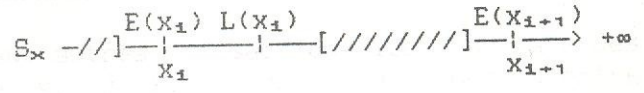
VI.4.3- VALIDATION LOCALE

Quand le contrôle global d'une transaction T réussit, la variable RGC(T) véhicule la valeur "vrai" et l'estampille choisie t et la phase de validation locale est mise en oeuvre sur chacun des sites accédés par T.

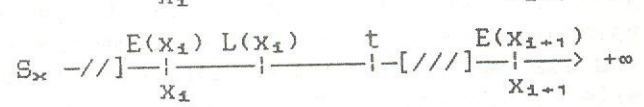
Si celle-ci a effectué une préécriture, il y aura mise à jour de la base de données à ce niveau. La nouvelle version prendra pour estampille d'écriture l'estampille t choisie pour T. Le résultat de cette validation est ensuite communiqué aux transactions postérieures à travers la mise à jour des ensembles LSP(x) et ESP(x) qui prennent en compte l'estampille d'écriture de la nouvelle version.

Pour T ayant effectué une lecture, l'estampille de lecture de la version lue prendra la valeur de t si elle n'existe pas encore ou si elle existe et lui est inférieure. Ce dernier cas peut se produire puisque, pour un accès en lecture, le choix de t s'effectue dans un intervalle de sérialisation borné à gauche par l'estampille d'écriture de la version et non par sa dernière estampille de lecture. L'ensemble des intervalles de sérialisation ESP(x) est ensuite réajusté: la portion d'intervalle comprise entre l'estampille de lecture précédente et la valeur de t en sera retranchée (voir schéma ci-dessous).

Avant la lecture:



Après la lecture d'estampille t:



Les dépendances qui lient la transaction T qui valide aux autres encore vivantes doivent aussi être prises en compte. Les intervalles associés à ces transactions et englobant l'estampille choisie pour T devront être réajustés en fonction de ce choix [BOKS 84]. Cette question sera traitée en VI.5 (cf. page 79).

La vie de la transaction T se termine par sa destruction locale sur chaque site S_x qu'elle a accédé. Cela sous-entend la destruction des ensembles EI(T,x) et $EI_G(T)$ ainsi que la suppression de T des ensembles lectrices(x) et, ou, écrivains(x).

Procédure de validation locale LV (T,x) au site S_x

```
DEBUT
SI  RGC(T) ≠ faux                          /* certification réussie */
ALORS FAIRE
      SI    T ∈ écrivains(x)
      ALORS FAIRE                          /* Validation écriture */
            ECRIRE x                       /* création d'une version x_i */
            E(x_i)  := t
            LSP(x) := (LSP(x) ∩ [0,E(x_i)[) U (LSP(x) ∩ ]E(x_i),+∞[)
            ESP(x) := (ESP(x) ∩ [0,E(x_i)[) U (ESP(x) ∩ ]E(x_i),+∞[)
            FAIT
      FSI
      SI    T ∈ lectrices(x)               /* lecture d'une version x_i */
      ALORS SI t > L(x_i)
            ALORS FAIRE
                  L(x_i) := t
                  ESP(x):=(ESP(x)∩ [0,E(x_i)[) U (ESP(x)∩ ]L(x_i),+∞[)
                  FAIT
            FSI
      FSI
      FAIT
FSI
FIN
```

En cas d'échec de la certification globale (RGC(T) = faux), des phases locales de rejet remplacent les phases de validations.

VI.5- REAJUSTEMENTS DES INTERVALLES DE SERIALISATION

On a vu que, pour communiquer aux autres transactions les résultats des mises à jour effectuées lors de la validation locale de l'une d'entre elles, on intégrait les nouvelles estampilles des versions aux ensembles d'intervalles ESP et LSP. Comme une transaction ne peut effectuer ses accès qu'à travers l'utilisation d'au moins un de ces deux ensembles, elle aura donc connaissance d'une création de version ou d'une lecture antérieures à ses accès.

Cela est valable pour les transactions qui démarrent leur exécution après cette validation, quelle que soit leur nature, et pour les transactions vivantes qui effectuent des écritures. Celles-ci n'utilisent l'ensemble des intervalles ESP qu'au niveau de leur certification locale: elles baseront donc leurs accès sur un ensemble déjà mis à jour et, dans le cas d'un conflit LP, ne risquent pas de prendre une estampille inférieure à celle du dernier lecteur de la version précédente.

Par contre, une lectrice encore vivante lors de la validation d'un écrivain ne peut prendre en charge la mise à jour qu'il a exécutée. Elle a déjà effectué sa lecture sur une version antérieure et elle a accédé à celle-ci sur la base d'un ensemble d'intervalles LSP non encore modifié. Pour respecter les dépendances qui découlent de la validation de l'écrivain, il faut procéder au réajustement de l'intervalle de sérialisation de cette lectrice -rappelons que celui-ci a été choisi lors de l'accès en lecture-.

Notons t l'estampille de la transaction écrivain qui valide. Celle qui sera affectée à la lectrice lors de sa validation, pour être cohérente avec t, doit lui être inférieure et l'intervalle dans lequel elle serait choisie doit donc être borné à droite par (t-1).

Soit T' une lectrice vivante telle que EI(T',x) contienne t et supposons que EI(T',x) n'a pas encore été réajusté. On a donc, avant le premier réajustement,

$EI(T',x) = [E(x_{i-1}) + 1, +\infty[$

```
                     EI(T',x)
Sx   --|-----[/////////////////////////---> +∞
     x(i-1)
```

Et après ce réajustement:

```
               EI(T',x)         t
Sx   --|-----[////////////]-----|----------> +∞
     x(i-1)                     x(i)
```

Soit $EI(T',x) = EI(T',x) \cap [0,t-1] = [E(x_{i-1}) + 1, t - 1]$

La validation d'un écrivain, dans l'algorithme de validation locale LV(T,x) de la page précédente, deviendra alors:

```
...
SI     T ∈ écrivains(x)
ALORS FAIRE                     /* Validation écriture */
       ECRIRE x                 /* création d'une version xi */
       E(xi)  := t
       ..........................................................
       :  /* Réajustement des lecteurs vivants: */              :
       :  POUR T' ∈ (lectrices(x) - {T}) / t ∈ EI(T',x)         :
       :  FAIRE                                                 :
       :  EI(T',x) := EI(T',x) ∩ [0,t - 1]                      :
       :  FAIT                                                  :
       :........................................................:
       LSP(x) := (LSP(x) ∩ [0,E(xi)[) U (LSP(x) ∩ ]E(xi),+∞[)
       ESP(x) := (ESP(x) ∩ [0,E(xi)[) U (ESP(x) ∩ ]E(xi),+∞[)
       FAIT
FSI
...
```

VI.6- ETUDE DES REJETS

VI.6.1- REJET LOCAL D'UNE TRANSACTION

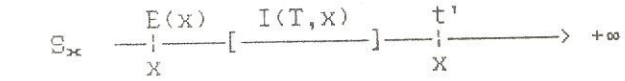
Il y aura rejet de la transaction T au niveau du site S_x si son ensemble local d'intervalles de sérialisation possible, EI(T,x), devient égal à l'ensemble vide à la suite d'un accès ou d'un réajustement, si T est lectrice, ou lors de la phase de certification locale, pour T écrivain.

Cela ne peut se produire pour une lectrice pure ou un écrivain pur: la première a toujours accès à la dernière version -au moment de cet accès- et la deuxième peut toujours créer sa version sous la seule condition qu'elle passe après la dernière lecture de la version précédente.

Examinons l'éventualité où T est une transaction VLE -"vieux lecteur écrivain"-: dans une base de données monoversion, c'est un cas de rejet local.

. T est d'abord un vieux lecteur: l'objet qu'il a lu a été remis à jour postérieurement à sa lecture et l'intervalle de sérialisation qui lui est associé a été tronqué à droite par l'estampille t' de cette remise à jour.

$I(T,x) = [E(x) + 1, t' - 1]$

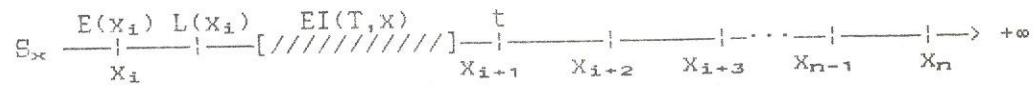


. T est ensuite un écrivain qui veut effectuer une écriture sur l'objet qu'il a lu: l'estampille de cette écriture doit être supérieure à celle de la dernière mise à jour de l'objet. Lors de la certification locale de T, I(T,x) sera donc tronqué à gauche par t'.

$I(T,x) = I(T,x) \cap [t'+1, +\infty[$

$I(T,x) = [E(x)+1,t'-1] \cap [t'+1, +\infty[= \emptyset$

Dans une base de données multiversion, la transaction VLE T n'a certes pas lu la dernière version de l'objet et son ensemble local d'intervalles de sérialisation, EI(T,x), a été tronqué à droite suite à la création d'une version plus récente. Mais, et c'est là un avantage procuré par le côté multiversion de la méthode, T peut toujours insérer sa version après la plus récente lecture de la version qu'elle a elle-même lue. Dans une base de données multiversion, aucune écriture n'est ignorée.



Le seul cas de rejet local concédé par la méthode MPIE sera donc celui des trop vieilles lectrices, c'est-à-dire celui des transactions qui essaient de lire une version qui a été supprimée parce que trop ancienne.

VI.6.2- REJET GLOBAL D'UNE TRANSACTION

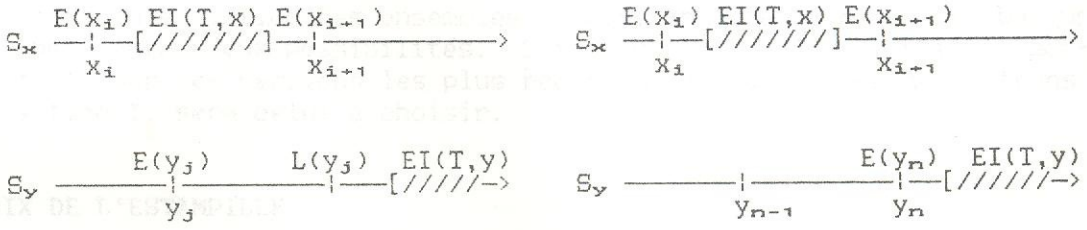
On dit qu'une transaction T est rejetée au niveau global si et seulement si l'ensemble $EI_g(T)$ des intervalles de sérialisation possible qui lui est associé au niveau global est vide alors que les ensembles correspondant aux niveaux locaux ne le sont pas.

Dans ce cas-là aussi, un écrivain pur, qui crée des versions sur plusieurs sites, n'est jamais rejeté: les ensembles d'intervalles de sérialisation locaux qui lui sont associés sont tous de borne supérieure infinie et donc d'intersection non vide.

Par contre, cette borne n'est plus infinie pour une transaction qui a effectué une lecture sur un site et dont l'intervalle local sur ce site a été tronqué à droite suite à la création d'une nouvelle version. L'intervalle global qui sera associé à cette transaction, intersection de tous les intervalles locaux, peut donc se trouver égal à l'ensemble vide, ce qui provoquerait le rejet de la transaction.

EXEMPLES:

T lecteur/écrivain T lit x et écrit y	T lecteur pur T lit x et lit y



Dans les deux cas de figure, on a $EI_g(T) = EI(T,x) \cap EI(T,y) = \emptyset$

On verra ultérieurement, en VI.8 (cf. page 84), comment l'utilisation d'un intervalle courant global nous permettra de détecter beaucoup plus rapidement les rejets et surtout comment il nous servira à éviter les cas de rejets de lectrices pures en les synchronisant.

VI.7- CHOIX DE L'ESTAMPILLE DE CERTIFICATION

La méthode MPIE gère les accès aux objets à l'aide de deux ensembles d'intervalles de sérialisation possible ESP et LSP, maintenus au niveau de chaque objet et réajustés au fur et à mesure des validations de transactions, en fonction des estampilles choisies.

Le choix de l'estampille influera donc sur les accès aux objets, aura une incidence sur le taux de rejet et, comme on le verra plus loin, pourra même être dirigé de manière à avantager une classe de transaction.

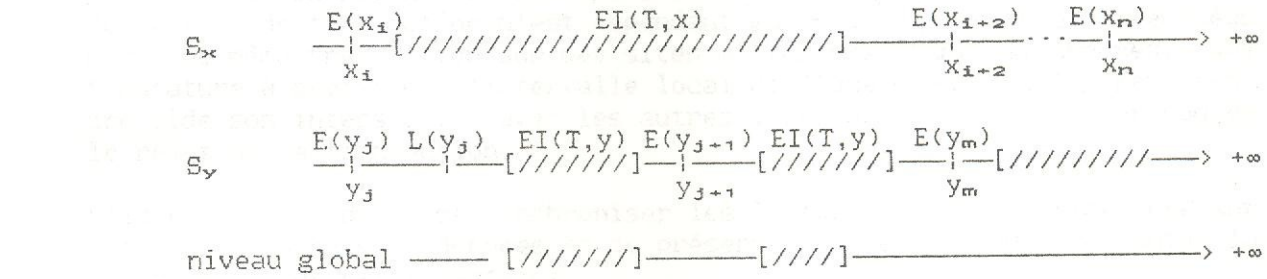
La méthode présente deux niveaux de choix: celui de l'intervalle et celui de l'estampille au sein de ce dernier.

VI.7.1- CHOIX DE L'INTERVALLE

La méthode repousse ce choix au niveau de la certification globale. Il s'effectuera sur la totalité des ensembles des intervalles de sérialisation et les rejets dus à un mauvais choix d'intervalle au niveau local sont ainsi évités.

Dans le souci de toujours fournir à l'utilisateur les données les plus récentes, l'intervalle choisi devra, lors d'un accès en lecture, donner la dernière version d'un objet dans le temps logique et pour un accès en écriture, permettre la création de la version la plus récente, toujours dans le temps logique.

EXEMPLE: Sur les sites S_x et S_y, les possibilités de sérialisation de la transaction T sont données par les zones hachurées.



Après intersection des ensembles locaux, on aura au niveau global un choix entre deux possibilités. L'intervalle le plus à droite, portant sur les versions les plus récentes lors des accès de la transaction T, sera celui à choisir.

VI.7.2- CHOIX DE L'ESTAMPILLE

L'intervalle $I_G(T) = [t1, t2]$ à l'intérieur duquel sérialisera la transaction T étant fixé, on procède alors au choix de l'estampille.

Suivant le type de la transaction qui valide, nous allons étudier l'impact qu'aura ce choix sur les autres transactions.

-T lectrice pure:
Soit T une transaction de lecture pure qui a lu une version x_i de l'objet x et qui valide avec une estampille t. Lors de cette validation, l'ensemble d'intervalles ESP est réajusté de manière à ne pas permettre aux transactions écrivains d'insérer une version après l'estampille d'écriture de x_i et avant cette estampille t.

Le choix qui avantagerait le plus les transactions écrivains sera donc t1, borne gauche de $I_G(T)$: en minimisant t, il élargit à gauche un des intervalles de l'ensemble ESP.

-T écrivain:
Soit t l'estampille associée à une transaction écrivain T en phase de validation.

Cette validation provoque la troncature à droite, par (t - 1), des intervalles de sérialisation des lectrices vivantes en conflit avec T.

Donc, plus t est pris proche de t1, plus l'intervalle de la lectrice se réduit à droite et plus cette lectrice se trouve désavantagée. A l'inverse, t proche de t2 la favoriserait.

Nous pouvons donc conclure que le choix de l'estampille avantage toujours une classe de transactions. Il serait en fait intéressant d'avoir une politique de choix dynamique qui évoluerait suivant la nature des transactions en conflit sur le site. Cette méthode complétée par une stratégie de choix permettrait de réaliser une forme de contrôle avant (prise en compte des conflits avec les autres transactions vivantes).

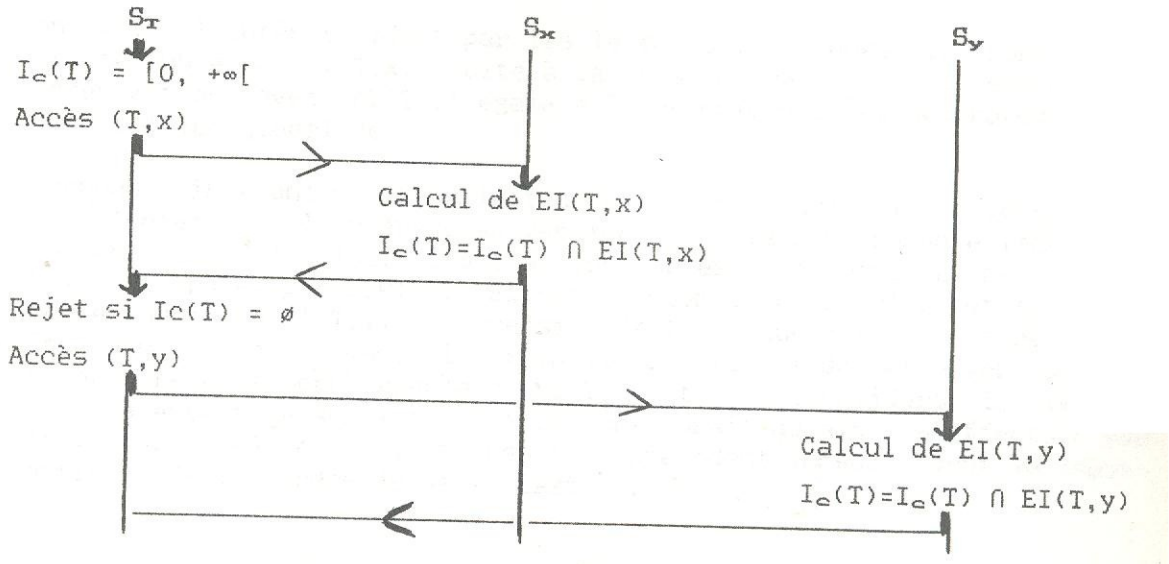
VI.8- CONTROLE MIXTE (CONTINU ET PAR CERTIFICATION)

Comme toute approche multiversion, la méthode MPIE favorise les lectrices pures en ne contrôlant pas les opérations de lectures. Le risque de rejet de ce type de transaction n'est cependant pas nul. Les lectures effectuées par une même transaction sur des sites différents étant asynchrones, une troncature à droite de l'intervalle local de l'une d'entre elles peut rendre vide son intersection avec les autres intervalles locaux et provoquer le rejet de la transaction.

L'idée serait donc de synchroniser les lectures d'une lectrice pure sur une vue de la base de données et de préserver cette vue durant toute la phase de vie de cette lectrice.

Nous avons introduit un intervalle courant global qui synchronise les opérations de lectures en véhiculant, sur tous les sites, la vue de la base de données que possède la transaction à l'instant de son premier accès en lecture.

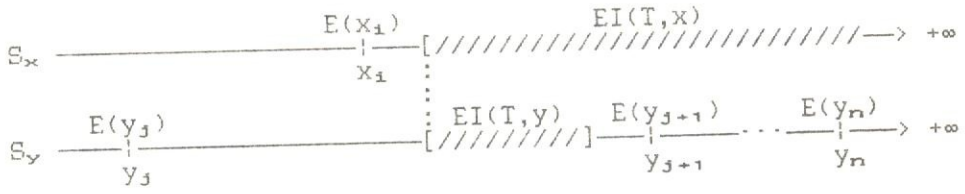
L'utilisation de cet intervalle induit un schéma de transaction synchrone en phase de vie. Dans le modèle de transaction précédent, S_T, site origine de la transaction T, effectue ses accès aux objets en parallèle sur les sites concernés. Dans ce nouveau modèle, S_T n'accède qu'à un seul site à la fois. Il attend le résultat de cet accès -et un intervalle courant modifié- pour accéder au site suivant et ainsi de suite. Les accès se font d'une manière séquentielle, un site après l'autre, comme le montre le schéma ci-après, où $I_c(T)$ désigne l'intervalle courant global maintenu auprès de la transaction T.



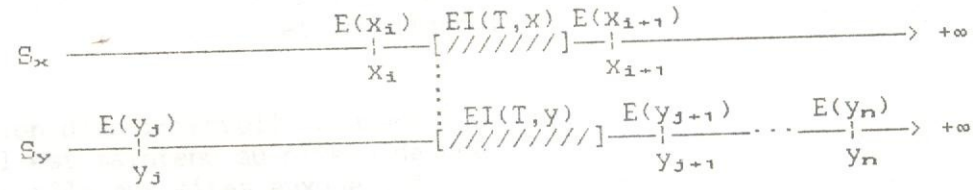
La synchronisation des lectures sur le premier accès est réalisée en fixant la borne inférieure de l'intervalle courant sur la valeur de celle de l'intervalle local associé au premier accès en lecture sur tous les sites. Le mode d'accès qu'on a proposé doit donc être modifié: seule, la première lecture de la transaction s'exécute sur la dernière version disponible lors de l'accès. Les lectures postérieures sur les autres sites doivent s'exécuter sur des versions synchrones à celle du premier accès et ces versions ne seront pas forcément les plus récentes.

Pour préserver ces lectures et empêcher les transactions écrivains de les invalider, nous avons reporté le poids du contrôle sur ces dernières. Les transactions écrivains effectuent un contrôle "en avant" en tenant compte des lectrices pures vivantes: lors de leur phase de certification locale, ils tronquent à gauche leurs propres intervalles de manière à ne pas pénaliser ces lectrices. Les opérations de lecture pure doivent donc être typées (par exemple LIRE$ x) afin qu'on puisse les reconnaître et un troisième ensemble, qui ne comporterait que les lectrices pures, doit être géré avec les ensembles lectrices(x) et écrivains(x).

EXEMPLE: Soit une lectrice pure T qui lit les objets x et y.



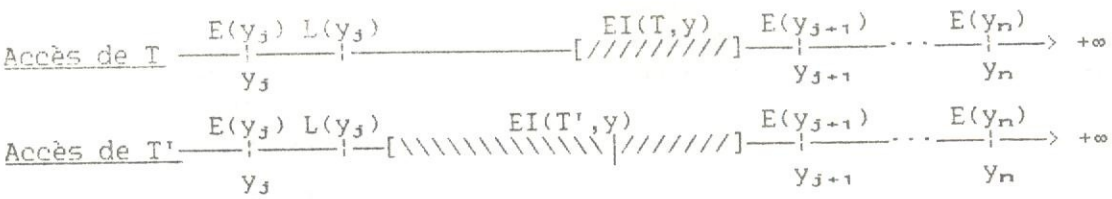
Le premier accès en lecture qui a lieu sur S_x s'effectue sur la version x_i et force le deuxième accès en lecture, qui a lieu sur S_y, à s'exécuter sur la version y_j et non sur y_n, dernière version disponible sur S_y lors de cet accès. En supposant ensuite qu'un écrivain valide la version x_{i+1} sur le site S_x comme ceci,



on voit l'intêret offert par les lectures synchrones: la troncature à droite subie par EI(T,x), suite à la création de x_{i+1}, ne rend pas son intersection avec EI(T,y) égale à l'ensemble vide et ne provoque donc pas le rejet global de T.

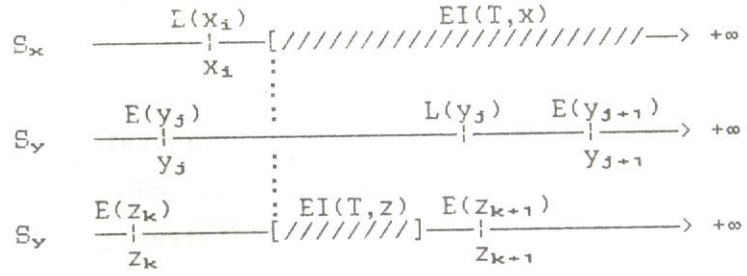
Prenons maintenant un vieux lecteur écrivain T' qui a lu la version y_j de l'objet y: il insérera sa version après la plus récente lecture de la version qu'il a lue, c'est-à-dire après l'estampille de lecture $L(y_j)$ fournie par ESP(y). Si cette estampille est plus petite que la borne gauche de EI(T,y), la version de T' risque d'être insérée entre les deux. Dans ce cas, l'estampille d'écriture de cette version sera inférieure à la borne gauche de EI(T,y) et elle invalidera la lecture que T a effectuée sur y_j: cette lecture aurait dû alors s'effectuer sur la version créée par T', transaction précédant immédiatement la transaction T dans l'ordre de sérialisation.

Plaçons nous sur le site S_y:



Pour préserver la lecture que T a effectuée sur y_j, l'écrivain T' devra faire du contrôle en avant en tronquant son propre intervalle de manière à ce que sa borne gauche soit supérieure à celle de EI(T,y). En complément à cela, on prendra cette dernière comme estampille de la lectrice T, lors de sa validation.

La synchronisation des lectures -à travers l'utilisation de l'intervalle courant- diminue aussi le risque de rejet des lecteurs écrivains, transactions qui effectuent à la fois des lectures et des écritures. Ce risque n'est cependant pas éliminé totalement: on ne peut synchroniser une écriture avec les autres accès si cette synchronisation invalide les lectures effectuées par d'autres transactions.



Dans le schéma ci-dessus, une écriture qu'exécuterait une transaction T sur l'objet y ne peut être synchronisée avec les lectures qu'elle a effectuées sur les objets x et z: pour qu'elle le soit, cette écriture se placerait entre $E(y_j)$ et $L(y_j)$ et invaliderait la lecture d'estampille $L(y_j)$.

REMARQUE:
L'utilisation d'un intervalle courant global induit une forme de contrôle continu. Il est maintenu au niveau de chaque transaction T et est transmis à tour de rôle aux sites auxquels T accède en lecture ou en préécriture. Chacun de ceux-ci procède à son intersection avec l'ensemble local des intervalles de sérialisation possible associé à T avant de le renvoyer à S_T, porteur des dépendances locales avec les transactions déjà validées.

Cette forme de contrôle continu est presque une construction site par site de l'ensemble global des intervalles de sérialisation possible $EI_\sigma(T)$. Mais l'intervalle courant global reste seulement une approximation de l'intersection des ensembles locaux EI(T,x) puisqu'il ne prend pas en compte les réajustements effectués sur les intervalles des lectrices vivantes lors des validations des transactions écrivains.

Par contre, il véhicule plus d'informations que n'importe quel ensemble local EI(T,x) sur les contraintes globales qui pèsent sur la transaction T. Dans le cas d'une hypothèse pessimiste, si on suppose que le nombre de rejets est élevé, il permet, au prix d'un échange de messages entre sites, un rejet de caractère global avant la phase de certification globale.

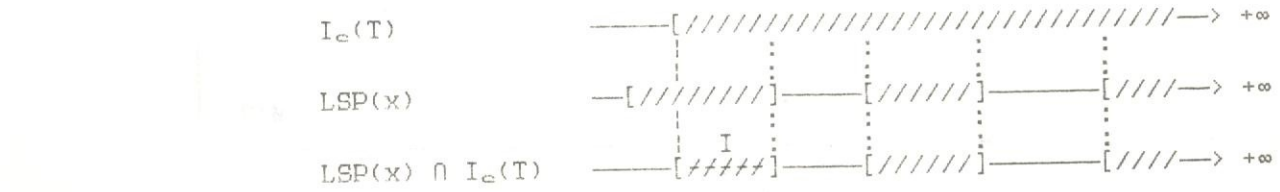
VI.8.1- ACCES AUX OBJETS

Comme précédemment, les accès en écriture se traduisent par des préécritures sur l'espace de travail de la transaction, préécritures qui ne seront prises en compte que lors de la phase de certification locale. L'algorithme déjà présenté ne subira donc pas de modifications, contrairement à celui de l'accès en lecture qui est repris ci-après.

Procédure LIRE (T, x, $I_c(T)$)

```
/* Ic(T), intervalle courant de T, a été initialisé à [0, +∞[ */
/* au niveau de ST, site origine de T */
DEBUT
SI      T ∉ (lectrices(x) U écrivains(x))
ALORS   FAIRE
        EI(T,x) := Ic(T) ∩ LSP(x)
        CHOISIR intervalle I dans EI(T,x) tel que
             borne inf(I) > borne inf(I') quel que soit I' ∈ EI(T,x)
        Ic(T), EI(T,x) := I
        FAIT
FSI
lectrices(x) := lectrices(x) U {T}
SI LIRES x    /*LIRES: opération de lecture d'une lectrice pure */
ALORS lectrices_pures(x) := lectrices_pures(x) U {T}
FSI
Ic(T) := EI(T,x)
CHOISIR intervalle I dans Ic(T) tel que
        borne inf(I) < borne inf(I') quel que soit I' ∈ Ic(T)
Ic(T), EI(T,x) := I
v := valeur lue (xi) /* xi est telle que E(xi)=borne inf(I) - 1 */
ENVOYER (v, Ic(T)) au site ST          /* ST: site qui a initié T */
ATTENDRE l'accusé de reception
FIN
```

Le choix de l'intervalle de lecture I ne sera effectué de manière à accéder à la dernière version de x que pour le premier accès de la transaction. Pour les accès ultérieurs et afin de synchroniser les lectures, le choix de I se fera dans l'intersection de l'intervalle courant avec l'ensemble local de sérialisation et portera sur l'intervalle le plus à gauche comme le montre le schéma ci-dessous.



VI.8.2- CERTIFICATION ET VALIDATION

Les phases de certifications locales d'une transaction T débutent sur les sites accédés dès réception du message "certifier T", sous la réserve que l'intervalle courant ne soit pas d'intersection vide avec l'ensemble des intervalles de sérialisation associé à la transaction au niveau local.

Procédure de certification locale LC(T,x,I_c(T)) au site S_x

```
DEBUT
EI(T,x) := EI(T,x) ∩ Ic(T)
SI EI(T,x) = ∅
ALORS ENVOYER message de rejet au site ST  /*ST: site origine de T*/
SINON FAIRE
      SI   T ∉ écrivains(x)  /* cas où T est une lectrice pure */
      ALORS DIFFUSER EI(T,x)  à tout site Sy / y ∈ {objets accédés}
      SINON FAIRE
            EI(T,x) := EI(T,x) ∩ ESP(x)
            /* test sur le rejet local: */
            SI EI(T,x) =  ∅
            ALORS ENVOYER  message de rejet au site ST
            SINON FAIRE
                  /* contrôle en avant de T: réajustement EI(T,x)*/
                    POUR TOUT T' ∈ lectrices_pures(x) tel que
                                   borne inf Ic(T') ∈ EI(T,x)
                    FAIRE
                    EI(T,x):= EI(T,x) ∩ [borne inf Ic(T') + 1, +∞[
                    FAIT
                  SI EI(T,x) =  ∅
                  ALORS ENVOYER  message de rejet au site ST
                  SINON DIFFUSER EI(T,x) à tout site Sy  tel que
                                                y ∈ {objets accédés}
                  FSI
                  FAIT
            FSI
            FAIT
      FSI
      FAIT
FSI
FIN
```

De la même manière que précédemment (cf. VI.4.1, page 76), le site S_T, dès reception du message de rejet, fait avorter la transaction sur tous les sites auxquels elle a accédé.

La procédure de certification globale d'une transaction est modifiée au niveau du choix de l'estampille: celui-ci s'effectue toujours dans l'intersection de tous les ensembles de sérialisation locaux mais, pour une lectrice pure, il sera guidé et portera obligatoirement sur la borne gauche de l'intervalle courant associé à cette lectrice. Ce choix dirigé est allié au contrôle en avant effectué par les transactions écrivains en phase de certification locale et sert à ne pas invalider les lectures.

Remarquons que, grâce à l'intervalle courant, certaines contraintes globales ont déjà été intégrées dans les différents ensembles de sérialisation locaux.

Procédure de certification globale GC(T) au site S_x

```
DEBUT
EI_G(T) := ∩ EI(T,x)   /*  X = {objets accédés par T} */
           x ∈ X
SI    EI_G(T) = ∅
ALORS RGC(T) := (faux, ?)   /* Cas de rejet global */
SINON  FAIRE
       ┌──────────────────────────────────────┐
       │ SI   T ∉ lectrices_pures(x)          │
       │ ALORS  CHOISIR intervalle I dans EI_G(T)
       │                  puis estampille t dans I
       │ SINON  t := borne inf I_G(T)         │
       │ FSI                                  │
       └──────────────────────────────────────┘
         RGC(T) :=  (vrai,t)
         FAIT
FSI
FIN
```

La phase de validation locale est la même que celle définie en VI.4.3. Elle consistera, selon les cas, en la création effective d'une version, le réajustement des intervalles des lectrices vivantes en conflit avec la transaction écrivain qui valide, la mise à jour des estampilles de lecture et d'écriture, celles des ensembles locaux de sérialisation associés aux objets, etc...(cf. algorithmes pages 79 et 80).

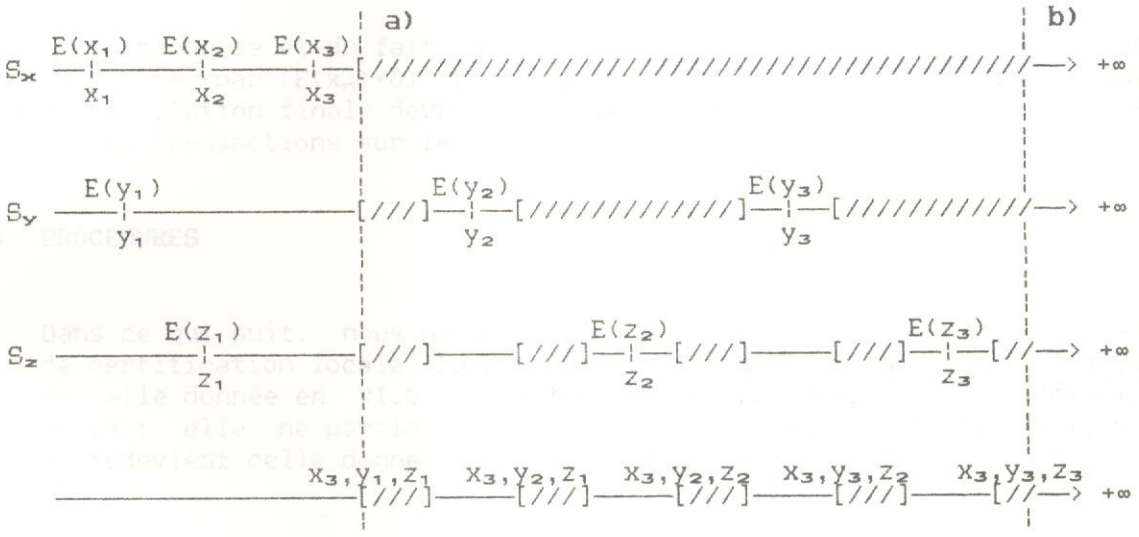
VI.8.3- DEUXIEME APPROCHE

Dans l'approche précédente, l'intervalle courant global, solution au problème du rejet des lectrices pures, les soumet à une forme d'estampillage statique et fait tout reposer sur le premier accès en lecture de la transaction: les sites n'évoluent pas forcément de la même façon et celle-ci risque d'avoir une vue "passée" de la base de données.

Ces désavantages peuvent être évités en donnant pour résultat à une lecture un nombre fixé de versions -et non plus une seule- à travers l'utilisation d'un ensemble d'intervalles courants. Ainsi, cela ne serait plus une seule, mais un ensemble de vues de la base de données qui serait véhiculé sur les sites.

Cela ne sera que lors de la certification globale, après avoir procédé à l'intersection des ensembles d'intervalles courants sur tous les sites accédés, que l'on récupère l'intervalle qui fournit la vue la plus récente.

EXEMPLE: Prenons sur les sites S_x, S_y et S_z la situation initiale ci-dessous et supposons une lectrice pure T qui effectue son premier accès sur S_x avant de lire les objets y et z.

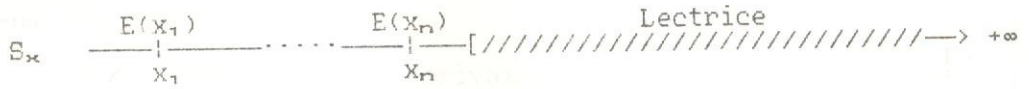


a)- L'utilisation d'un seul intervalle courant global synchronise les trois opérations de lecture sur le premier accès et donne à T la vue "passée" (x_3, y_1, z_1) de la base de données.

b)- L'approche préconisée et l'utilisation d'un ensemble d'intervalles courants fournit à T la vue la plus récente de la base, soit (x_3, y_3, z_3) .

Comme dans l'approche précédente, le contrôle est reporté sur les transactions écrivains: ceux-ci effectuent un contrôle "en avant" en phase de certification locale et réajustent leurs propres intervalles pour laisser aux lectrices pures de plus grandes possibilités de sérialisation.

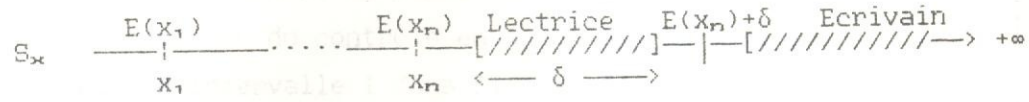
La troncature à gauche de l'intervalle de la transaction écrivain ne s'effectue plus par rapport à une valeur finie (borne inférieure de l'intervalle courant dans l'approche précédente): cela suppose, dans le cas où l'ensemble des intervalles associé à une lectrice pure est de borne supérieure infinie, un rejet systématique de tous les écrivains en conflit avec elle.



Pour résoudre ce problème il suffit de ramener la borne infinie de l'ensemble d'intervalles LSP associé à la lectrice

(soit $\bigcup_i [E(x_i) + 1, E(x_{i-1}) - 1] \cup [E(x_n) + 1, +\infty[$) à la valeur

$(E(x_n) + \delta)$ avec $\delta > 0$: la transaction écrivain peut alors sérialiser à droite de cette borne et ne sera pas rejetée.



En contrepartie et du fait de la troncature à droite de l'intervalle de la lectrice (par $(E(x_n)+\delta)$), le risque de rejet de celle-ci n'est plus nul. La solution finale devra donc être, en définitive, adaptée à la nature des transactions sur les sites.

VI.8.3.1- PROCEDURES

Dans ce qui suit, nous ne fournirons que les procédures de lecture et de certification locale. La procédure de certification globale diffère de celle donnée en VI.8.2 (lectures synchronisées sur le premier accès): elle ne particularise plus le traitement des lectrices pures et redevient celle donnée en VI.4.2 (cf. page 77).

La procédure de validation locale est celle qui a été donnée aux pages 79 et 80, à la différence que le réajustement des lectrices vivantes lors de la validaton d'une écriture (cf. page 80) ne concerne pas les lectrices pures: ou cela a déjà été fait (elles ont exécuté l'opération $(E(xn) + \delta)$ pour éviter les rejets d'écrivains), ou cela n'est pas nécessaire (les écrivains font du contrôle en avant et sérialisent à droite des bornes supérieures des lectrices pures).

Procédure LIRE (T, x, $EI_c(T)$)

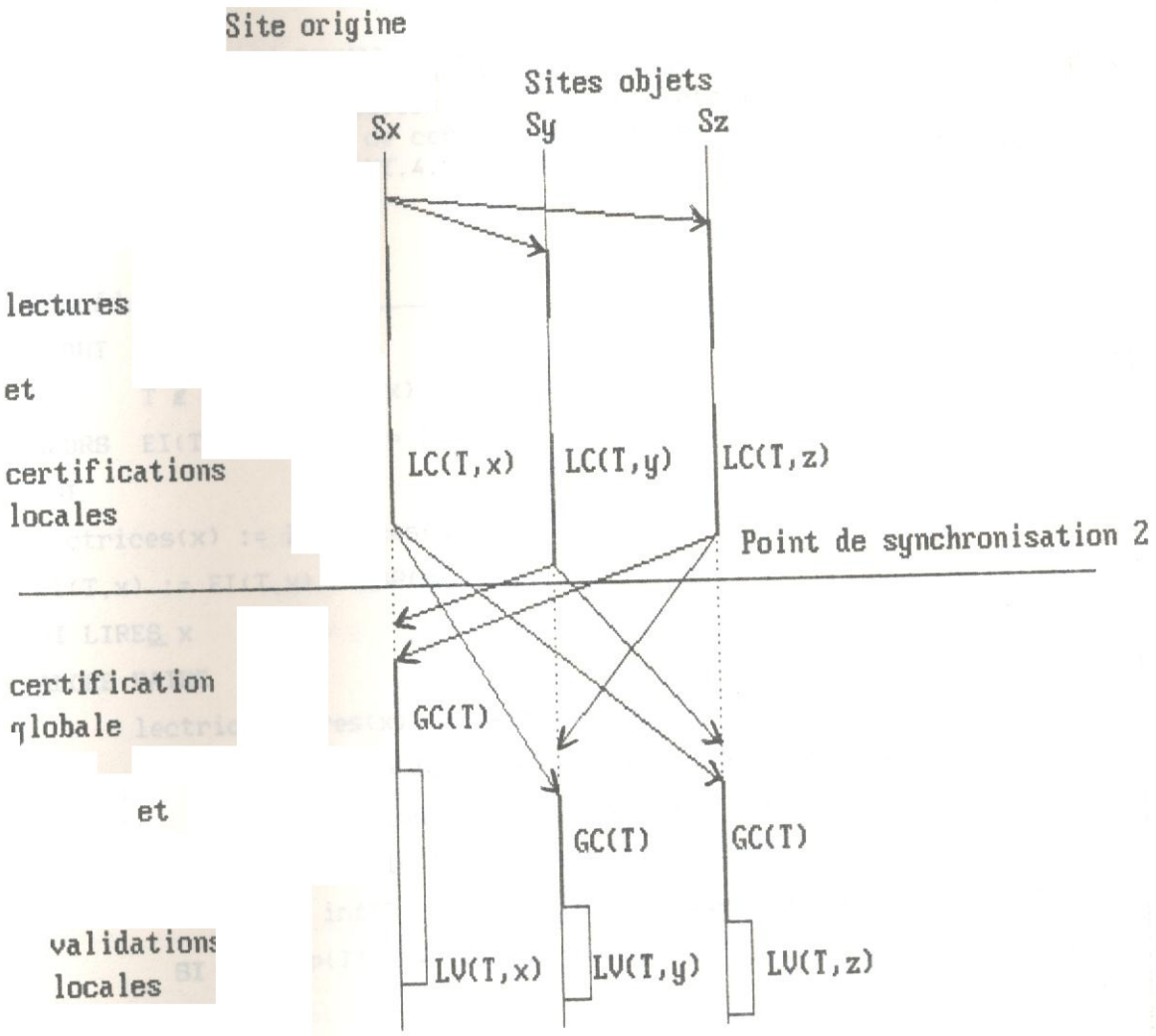
```
/* EIc(T), ensemble d'intervalles courant de T */
/* a été initialisé à [0, +∞[ au niveau de ST, site origine de T*/
DEBUT
SI     T ∉  (lectrices(x) U écrivains(x))
ALORS EI(T,x), EIc(T) := EIc(T) ∩ LSP(x)
FSI
lectrices(x) := lectrices(x) U {T}
SI LIRES x   /*LIRES: opération de lecture d'une lectrice pure */
ALORS FAIRE
      lectrices_pures(x) := lectrices_pures(x) U {T}
      /* suppression de la borne infinie d'une lectrice pure en
         prévision du contrôle en avant des écrivains */
      CHOISIR intervalle I dans EI(T,x) tel que
         borne inf(I) > borne inf(I') quel que soit I' ∈ EI(T,x)
      SI borne sup(I) = ∞  ALORS  borne sup(I):=borne inf(I) + δ
      FSI
      FAIT
FSI
EIc(T) := EI(T,x)
   POUR TOUT Ii ∈ EIc(T)
   FAIRE
   Vi:=valeur lue (xi) /*xi est telle que E(xi)=borne inf(Ii)-1*/
   FAIT
ENVOYER (V1,...,Vn, EIc(T)) au site ST  /*ST:site initiateur T */
ATTENDRE l'accusé de reception
FIN
```

Procédure de certification locale $LC(T,x,EI_c(T))$ au site S_x

```
DEBUT
EI(T,x) := EI(T,x) ∩ EI_c(T)
SI EI(T,x) = ∅
ALORS ENVOYER message de rejet au site S_T
SINON FAIRE
      SI   T ∉ écrivains(x)  /*cas où T est une lectrice pure
      ALORS DIFFUSER EI(T,x) à tout site S_y / y ∈ {objets accéd
      SINON FAIRE
            EI(T,x) := EI(T,x) ∩ ESP(x)
            /* test sur le rejet local: */
            SI EI(T,x) = ∅
            ALORS ENVOYER  message de rejet au site S_T
            SINON FAIRE
                  /*contrôle en avant de T: réajustement EI(T,x)*
                    POUR TOUT T' ∈ lectrices_pures(x)
                    FAIRE
                    CHOISIR l'intervalle I ∈ EI(T',x) tel que
                            borne inf(I) > borne inf(I')
                            quel que soit I' ∈ EI(T',x)
                    EI(T,x):= EI(T,x) ∩ [borne sup(I) + 1, +∞[
                    FAIT
                  SI EI(T,x) = ∅
                  ALORS ENVOYER  message de rejet au site S_T
                  SINON DIFFUSER EI(T,x) à tout site S_y  tel que
                  FSI                          y ∈ {objets accédés}
                  FAIT
            FSI
            FAIT
      FSI
      FAIT
FSI
FIN
```

VI.9- PROLONGEMENT: MODELE DE TRANSACTION PARALLELE

Pour disposer d'un niveau de parallélisme plus élevé, nous avons procédé à l'adaptation de la méthode MPIE sur le modèle de transaction parallèle ci-dessous.



MODELE DE TRANSACTION PARALLELE

Ce modèle ne comprend plus le point de sychronisation 1 qui figure sur le modèle "Contrôle global réparti" et qui se situe à la fin de la phase des lectures (cf. Fig. 1 page 32).

Contrairement au point de synchronisation 2, qui empêche une transaction d'exister en phase de vie sur un site et en phase de validation sur un autre, le point de synchronisation 1 peut être supprimé: il ne sert pas à garantir l'atomicité de la transaction et n'est pas non plus utile au contrôle de concurrence.

Cependant, et du fait que l'existence de ce point suppose un renvoi immédiat des résultats des lectures au site origine de la transaction, le modèle proposé sera moins adapté que le précédent aux transactions de type interactif (CAO).

D'un autre côté, le modèle parallèle n'utilise pas l'ensemble d'intervalles courants du modèle séquentiel et ne bénéficie donc pas des avantages de cette utilisation: un rejet de caractère global ne pourra pas être détecté avant la phase de contrôle global.

Pour tenir compte de ce nouveau schéma de transaction, les procédures de lecture et de certification locale ont été modifiées et figurent ci-dessous. Les procédures de certification globale et de validation locale sont celles définies en VI.4.2 et VI.4.3 (cf. algorithmes pages 77, 79 et 80).

Procédure LIRE (T,x)

```
DEBUT
SI      T ∉ (lectrices(x) U écrivains(x))
ALORS  EI(T,x) := [0, +∞ [
FSI
lectrices(x) := lectrices(x) U {T}
EI(T,x) := EI(T,x) ∩ LSP(x)
SI LIRES x
ALORS FAIRE
      lectrices_pures(x) := lectrices_pures(x) U {T}
      /* suppression de la borne infinie d'une lectrice pure en
         prévision du contrôle en avant des écrivains */
      CHOISIR intervalle I dans EI(T,x) tel que
            borne inf(I) > borne inf(I') quel que soit I' ∈ EI(T,x)
      SI borne sup(I) = ∞  ALORS  borne sup(I) := borne inf(I) + δ
      FSI
      FAIT
FSI
   POUR TOUT I_i ∈ EI(T)
   FAIRE
   v_i := valeur lue (x_i) /*x_i telle que E(x_i)=borne inf(I_i) - 1*/
   FAIT
ENVOYER (v_1,...,v_n, EI(T)) au site S_T  /*S_T: site qui a initié T*/
/* sans attendre l'accusé de réception */
FIN
```

Procédure de certification locale LC(T,x) au site S_x

```
DEBUT
SI   T ∉ écrivains(x)      /* cas où T est une lectrice pure */
ALORS DIFFUSER EI(T,x)  à tout site Sy / y ∈ {objets accédés}
SINON FAIRE
      EI(T,x) := EI(T,x) ∩ ESP(x)
       /* test sur le rejet local: */
       SI EI(T,x) = ∅
       ALORS ENVOYER  message de rejet à ST
       SINON FAIRE
             /* contrôle en avant de T: réajustement EI(T,x)*/
                POUR TOUT T' ∈ lectrices_pures(x)
                FAIRE
                CHOISIR l'intervalle I ∈ EI(T',x) tel que
                        borne inf(I) > borne inf(I')
                        quel que soit I' ∈ EI(T',x)
                EI(T,x):= EI(T,x) ∩ [borne sup(I) + 1, +∞[
                FAIT
              SI EI(T,x) = ∅
              ALORS ENVOYER  message de rejet au site ST
              SINON DIFFUSER EI(T,x) à tout site Sy  tel que
                                        y ∈ {objets accédés}
              FSI
             FAIT
        FSI
       FAIT
 FIN
```

VI.10- ETUDE DE LA SERIALISABILITE

Nous avons montré en VI.2 (cf. page 72) que, pour qu'une transaction soit sérialisable, il suffit que l'ensemble des intervalles de sérialisation possible qui lui est associé au niveau global ne soit pas vide: l'estampille que l'on affecte à la transaction lors de sa validation est alors prélevée de cet ensemble.

Nous allons utiliser maintenant les concepts théoriques sur la 1-sérialisabilité que l'on a introduits en V.1 (cf. page 49) pour démontrer que la méthode MPIE n'induit pas de circuit dans le graphe de sérialisation.

Nous montrerons dans un premier temps que cette méthode vérifie les quatre propriétés que doit posséder tout algorithme utilisant l'estampillage comme technique de sérialisation pour être correct [BERN 87].

Dans la suite, nous désignerons par R une exécution possible d'un ensemble de transactions, par $e_j(x_j)$ l'opération d'écriture de la version x_j par la transaction T_j et par $l_j(x_i)$, celle de lecture effectuée par T_j sur x_i.

PROPRIETE I:

énoncé: A chaque transaction T_i correspond une estampille unique t_i telle que $t_i = t_j$ équivaut à $i = j$.

Selon la nature de la transaction, la méthode MPIE vérifie totalement ou en partie cette première propriété.

Le choix de l'estampille d'une transaction écrivain s'effectue dans l'intersection des ensembles locaux

$$\bigcup_{i=1}^{n-1} [L'(x_i) + 1, E(x_{i+1}) - 1] \cup [L'(x_n) + 1, +\infty [$$

où $L'(x_i)$ s'identifie soit à l'estampille de lecture $L(x_i)$, soit à celle d'écriture $E(x_i)$, si x_i n'a pas encore été lue.

A chaque fois qu'une transaction exécute sa phase de validation, l'estampille qui lui est associée est supprimée de ces ensembles locaux. Une transaction écrivain ne peut donc bénéficier d'une estampille qui a été déjà affectée.

Par contre le choix de l'estampille d'une lectrice pure s'opère dans l'intersection des ensembles

$$\bigcup_{i=1}^{n-1} [E(x_i) + 1, E(x_{i+1}) - 1] \cup [E(x_n) + 1, +\infty [$$

qui ne sont pas mis à jour lors de la validation des transactions de lecture pure: une transaction de ce type peut donc se voir associer une estampille qui aurait été déjà affectée.

On a donc:
$t_i = t_j$ équivaut à $i = j$ ou à T_i et T_j sont deux lectrices pures.

PROPRIETE II:

énoncé: Pour chaque opération $l_k(x_j)$ -T_k lit x_j- dans R, il existe une opération $e_j(x_j)$ -T_j crée x_j- dans R telle que $e_j(x_j)$ précède $l_k(x_j)$ et $t_j \leq t_k$.
La création d'une version x_j par la transaction T_j précède sa lecture par la transaction T_k et T_k ne peut lire que les versions d'estampille inférieure à la sienne.

Cette propriété est vérifiée par la méthode MPIE: l'accès en lecture d'une transaction s'effectue sur la version de l'objet qui précède immédiatement l'intervalle associé à cette transaction. Donc,

-la transaction T_j qui a créé la version x_j précède la transaction T_k qui la lit.
-l'estampille t_j associée à T_j est inférieure à la borne gauche de l'intervalle associé à T_k. Elle sera donc inférieure à l'estampille t_k qui sera prélevée de cet intervalle pour être affectée à T_k lors de sa validation.

PROPRIETE III:

énoncé: Pour chaque couple d'opérations $l_k(x_j)$ -T_k lit x_j- et $e_i(x_i)$ -T_i crée x_i- dans R, une des trois assertions suivantes est vérifiée:

a)- $t_i < t_j$.

b)- $t_k < t_i$.

c)- i=k et l'opération $l_k(x_j)$ précède l'opération $e_i(x_i)$.

La lecture qu'effectue une transaction T_k doit porter sur la version x_j créée par T_j, transaction qui précède immédiatement T_k dans l'ordre de sérialisation. La création d'une version x_i par la transaction T_i doit être soit antérieure à la création de x_j, soit postérieure à sa lecture par T_k. Elle ne peut pas s'insérer entre la création de x_j et sa lecture.
Si T_k est une transaction lecteur écrivain, qui lit x_j et crée la version x_k (cas c), alors l'opération de lecture de x_j doit précéder l'opération d'écriture de x_k.

Dans la méthode MPIE, l'accès en lecture d'une transaction T_k porte sur la version x_j qui précède immédiatement l'intervalle associé à T_k.

.Si x_j est la plus récente version, son estampille correspondra à la borne gauche de l'intervalle de lecture de T_k. Si ce n'est pas le cas (lectures synchronisées par l'utilisation d'un intervalle courant), les transaction écrivains font du contrôle en avant et tronquent à gauche leurs propres intervalles de manière à ne pas insérer leurs versions entre la création de x_j et sa lecture.

.Si un écrivain T_i crée une version x_i et que l'estampille t_i qui lui est associée se trouve dans l'intervalle de lecture de T_k, il y a troncature à droite de ce dernier par $(t_i - 1)$: ainsi, l'estampille t_k qui sera affectée à T_k sera inférieure à t_i.

Si T_k est un lecteur écrivain, qui exécute une lecture et une écriture sur le même objet, son opération de lecture sera effectuée en premier puisqu'elle est immédiate. Sa création de version se traduira par une préécriture sur son espace de travail et ne deviendra effective que lors de sa validation.

PROPRIETE IV:

énoncé: En notant c_j la validation de la transaction T_j, on aura:
Si les opérations $l_j(x_i)$ -T_j lit x_i- et c_j appartiennent à R et $i \neq j$ alors, c_i précède c_j.

Dans la méthode MPIE, les accès aux objets ne s'effectuent qu'en fonction des transactions déjà validées. Cette dernière propriété est donc vérifiée.

PREUVE DE LA 1-SERIALISABILITE:

Pour qu'un algorithme soit correct, il suffit que le graphe de sérialisation multiversion GSMV(R,«) qui lui correspond soit acyclique.

Soient t_i et t_j les estampilles d'écritures des versions x_i et x_j -donc celles associées au transactions validées T_i et T_j qui ont créé ces versions-. On pose un ordre sur les versions des objets que l'on structu-

re comme suit: $x_i \ll x_j \iff t_i < t_j$.

Pour prouver que le graphe GSMV(R,«) ne comporte pas de circuits, nous devons montrer que pour tout arc $T_i \longrightarrow T_j$ (T_j lit x_i de T_i) qui lui appartient, on a $t_i < t_j$.

Prenons, dans le cas monoversion, l'arc $T_i \longrightarrow T_j$ (T_j lit x de T_i) du graphe de sérialisation SG(R). D'après la propriété II, T_j ne peut lire une écriture d'estampille supérieure à la sienne. On a donc nécessairement $t_i \leq t_j$.

D'un autre côté et suite à la propriété I, T_i étant une transaction écrivain, a une estampille unique: on ne peut avoir $t_i = t_j$. Donc tout arc $T_i \longrightarrow T_j$ de SG(R) vérifie $t_i < t_j$.

Généralisons au cas multiversion et considérons les opérations $l_k(x_j)$ -T_k lit x_j- et $e_i(x_i)$ -T_i crée x_i- dans R avec i,j et k distincts. Ces opérations portent sur deux versions x_i et x_j d'un même objet x et ces deux versions se situent l'une par rapport à l'autre, selon l'ordre défini plus haut, suivant deux possibilités:

-<u>cas trivial ($x_i \ll x_j$):</u> d'après l'ordre défini, on a $t_i < t_j$ pour l'arc $T_i \longrightarrow T_j$ de GSMV(R, «).

-<u>cas général ($x_j \ll x_i$):</u> Appliquons la propriété III au couple d'opérations $l_k(x_j)$ et $e_i(x_i)$. On peut avoir l'un des trois cas:

a)- $t_i < t_j$.
b)- $t_k < t_i$.
c)- i=k et l'opération $l_k(x_j)$ précède l'opération $e_i(x_i)$.

. le cas a) est impossible puisque $x_j \ll x_i$ équivaut à $t_j < t_i$.
. le cas b) donne $t_j < t_k < t_i$
($t_j < t_k$ d'après la propriété II -$t_j \leq t_k$- et la propriété I -t_j unique-).
. le cas c) nous ramène au cas trivial $x_j \ll x_k$ équivaut à $t_j < t_k$.

Les arcs sont donc ordonnés suivant leurs estampilles et le graphe GSMV(R,«) est acyclique.

VI.11- EXEMPLES D'APPLICATION

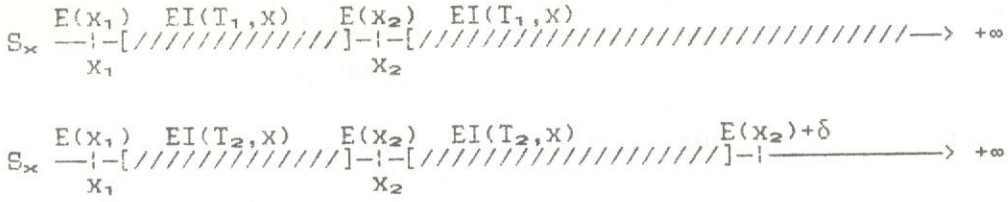
A/-<u>EXEMPLE 1:</u> <u>Intérêt de l'utilisation des intervalles</u>

Soient les transactions T_1 et T_2, d'estampilles t_1 et t_2.

T_1	T_2
LIRE x	LIRE x
PREECRIRE x	CERTIFIER (T_2)
CERTIFIER (T_1)	

L'utilisation d'une technique d'estampillage statique suppose le rejet de la transaction écrivain T_1, si t_1 est plus petit que t_2: T_1 invalide alors la lecture effectuée par T_2.

Prenons cet exemple sous la méthode MPIE, les possibilités de sérialisation des opérations de lecture de T_1 et T_2 sont représentées par les diagrammes suivants:



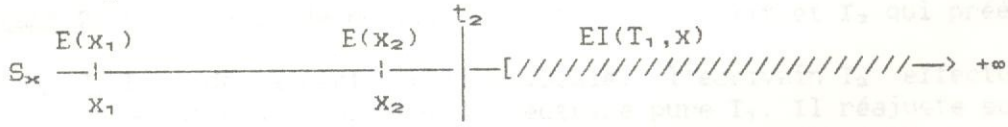
La lectrice pure T_2 tronque son ensemble d'intervalles à droite à la valeur $(E(x_2) + \delta)$, de manière à ne pas invalider un écrivain qui effectue un contrôle en avant.

Suivant que T_1 certifie avant ou après T_2, deux cas sont à étudier:

-Si l'écrivain T_1 certifie avant T_2, il effectue un contrôle en avant lors de sa certification locale, de manière à ce que son estampille soit supérieure à $(E(x_2) + \delta)$ pour préserver la lectrice pure T_2.

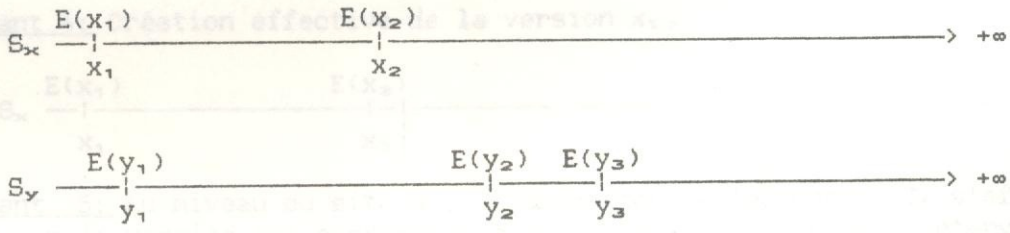
Le choix de l'estampille qui sera associée à T_2 se fera ensuite dans l'intervalle $[E(x_2) + 1, E(x_2) + \delta]$: ainsi, cela sera la version x_2, version la plus récente, qui sera récupérée.

-Si T_2 certifie avant T_1, T_1 voit T_2 validée et crée sa version après la version lue par T_1 (cette version sera la version la plus récente, soit x_2)



B/-**EXEMPLE 2:** Illustration de la méthode

Prenons, sur deux sites S_x et S_y, la situation de départ suivante:



et soit deux transactions T_1 et T_2 qui accèdent à ces deux sites:

T_1	T_2
1 LIRE x	
	2 PREECRIRE x
	3 CERTIFIER (T_2)
	4 ECRIRE x
5 LIRE y	
6 CERTIFIER (T_1)	

Nous allons dérouler la méthode MPIE sur les trois cas de fig possibles:

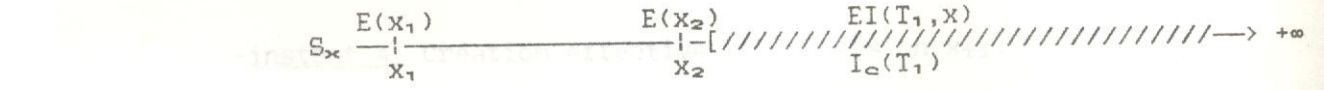
CAS 1: utilisation d'un modèle de transaction séquentiel avec lec res synchronisées sur le premier accès par un seul intervalle cour au niveau global.

CAS 2: utilisation d'un modèle de transaction séquentiel avec ren de plusieurs versions à une lecture à travers un ensemble d'interv les courants au niveau global.

CAS 3: utilisation d'un modèle de transaction parallèle.

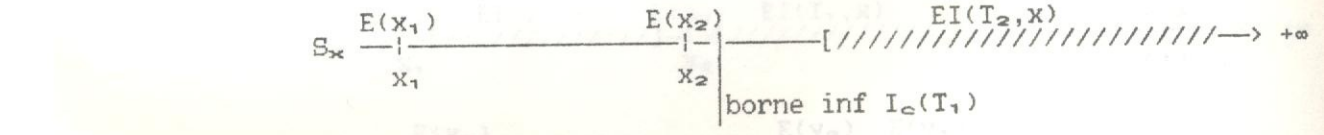
a)- **CAS 1: Transaction séquentielle et un seul intervalle courant.**

-instant 1: L'opération de lecture de T_1 portera sur la version la p récente de l'objet x, soit x_2. Les possibilités de sérialisations de seront alors:

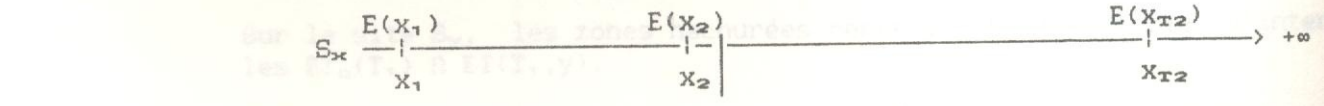


-instant 2: Occurrence de conflit LP entre T_1 qui lit et T_2 qui préécr

-instant 3: Lors de sa certification locale, l'écrivain T_2 effectue contrôle en avant pour préserver la lectrice pure T_1. Il réajuste son semble d'intervalles, de manière à ce que sa borne gauche soit supéri à celle de $I_c(T_1)$.

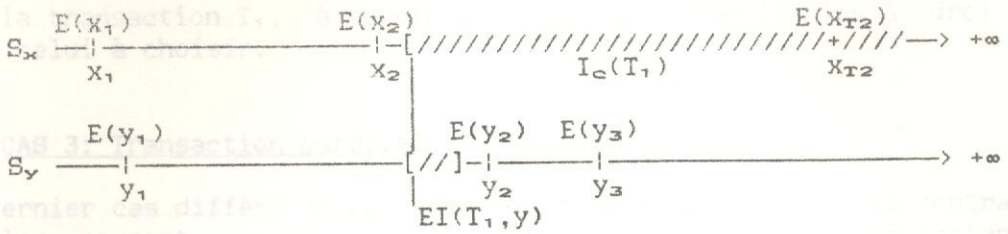


-instant 4: Création effective de la version x_{T2}:



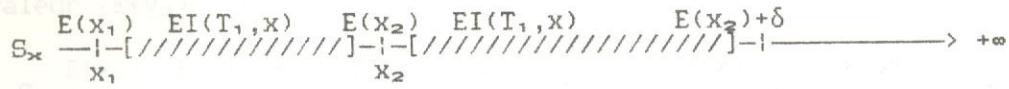
-instant 5: Au niveau du site S_y, l'opération de lecture de T_1 s'ef tuera sur la version qui correspond à la borne inférieure de l'interv courant $I_c(T_1)$, soit y_1, et non sur la version y_3 qui est la récente. L'estampille qui sera affectée à T_1 lors de sa phase de cert

cation globale sera la borne gauche de $I_c(T_1)$.



b)- CAS 2: Transaction séquentielle et ensemble d'intervalles courants.

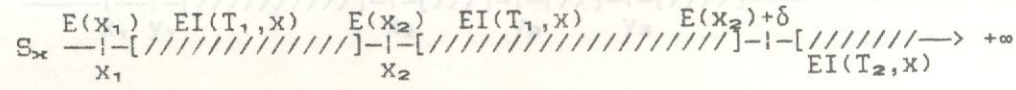
-instant 1: L'ensemble des intervalles de sérialisation possible associé à T_1 au début de son exécution, $EI(T_1,x)$, est visualisé par le schéma:



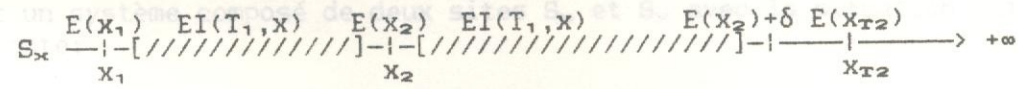
A son premier accès en lecture, T_1, lectrice pure, ramène la borne supérieure infinie de son ensemble d'intervalles à la valeur $(E(x_2) + \delta)$. Ainsi, un éventuel écrivain pourrait effectuer un contrôle en avant -pour préserver T_1- sans être rejeté.

-instant 2: Occurrence de conflit LP entre T_1 qui lit et T_2 qui préécrit.

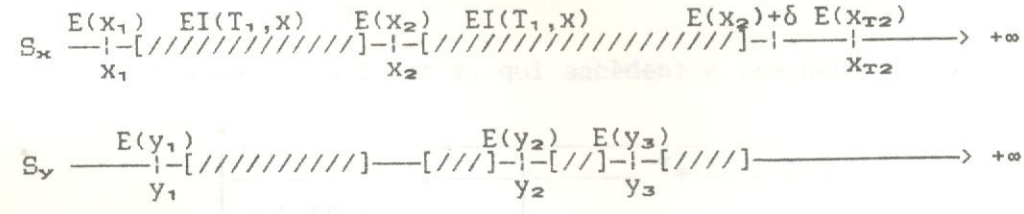
-instant 3: Pour ne pas invalider la lecture effectuée par la lectrice pure T_1, la transaction écrivain T_2 effectue un contrôle en avant lors de sa phase de certification locale et réajuste son ensemble d'intervalles de manière à sérialiser à la droite du dernier intervalle associé à T_1.



-instant 4: Création effective de la version x_{T2}:

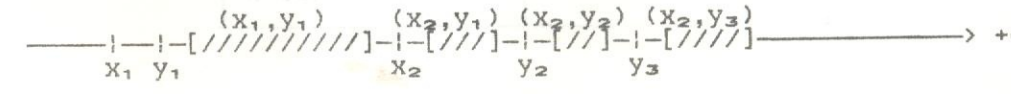


-instant 5: T_1 accède en lecture au site S_y: les contraintes pesant sur T_1 et établies sur S_x sont reportées sur S_y, par le biais de l'ensemble des intervalles courants $EI_c(T_1)$.



Sur le site S_y, les zones hachurées représentent l'ensemble d'intervalles $EI_c(T_1) \cap EI(T_1,y)$.

-instant 6: Au niveau global, la sérialisation de la lectrice pure T_1 peut s'effectuer dans l'un des quatre intervalles suivants:

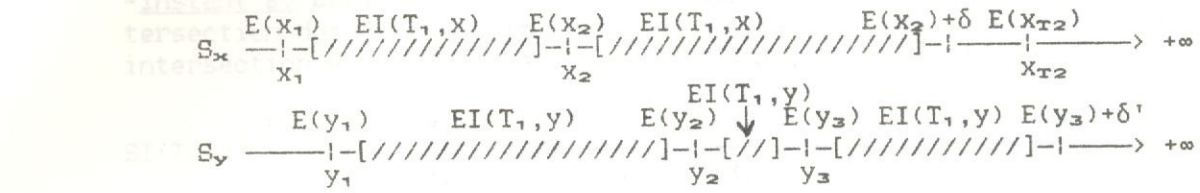


L'intervalle qui fournira les versions les plus récentes qui n'invalident pas la transaction T_1, à savoir x_2 et y_3 est situé le plus à droite et sera celui à choisir.

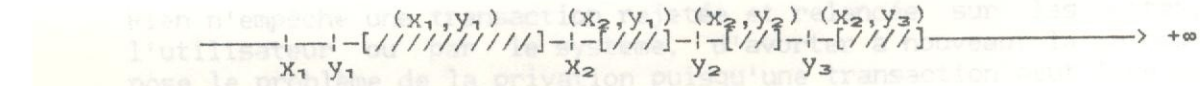
c)- **CAS 3:** Transaction parallèle.

Ce dernier cas diffère du précédent dans le seul fait que les contraintes locales ne sont plus véhiculées sur les sites. L'exposé des instants 1, 2, 3 et 4 correspondant au seul site S_x sera donc identique à celui du cas 2.

-instant 5: Premier accès en lecture de T_1 sur le site S_y: T_1, lectrice pure, ramène la borne supérieure infinie de son ensemble d'intervalles à la valeur $(E(y_3) + \delta')$.

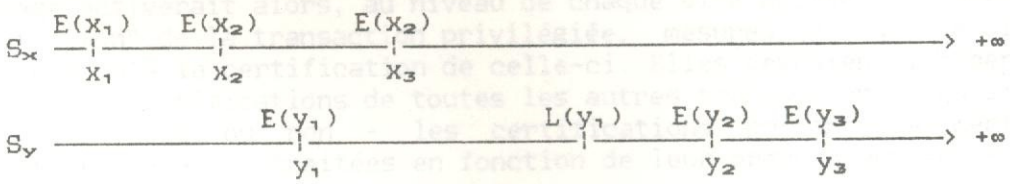


-instant 6: On procède, au niveau global, à l'intersection des ensembles d'intervalles où la sérialisation de T_1 est possible. Le choix portera ensuite sur l'intervalle qui fournira les versions des objets x et y les plus récentes. Cela sera, comme dans le cas précédent, celui qui fournira les versions x_2 et y_3.



C/-**EXEMPLE 3:** Cas de rejet de transactions

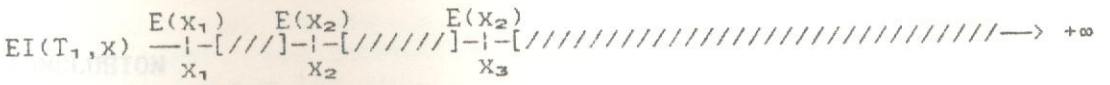
Soit un système composé de deux sites S_x et S_y avec la situation initiale suivante:



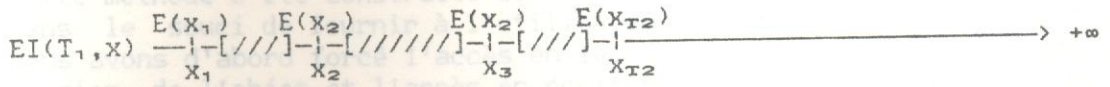
et soit deux transactions T_1 et T_2 qui accèdent à ces deux sites:

T_1	T_2
1 LIRE x	
	2 PREECRIRE x
	3 CERTIFIER (T_2)
	4 ECRIRE x
5 PREECRIRE y	
6 CERTIFIER (T_1)	

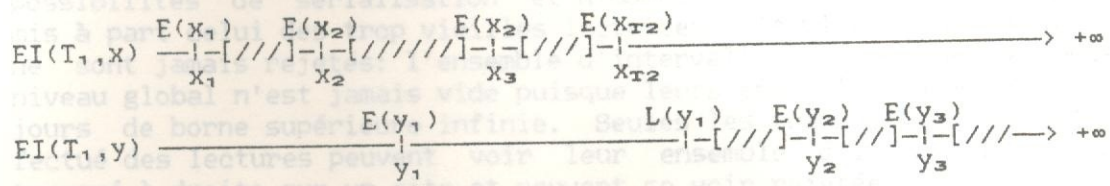
-instant 1: L'ensemble des intervalles de sérialisation possible associé à T_1 lors de son premier accès en lecture sur le site S_x est donné ci-après:



-instant 4: Il y a création effective de la version x_{T2} par la transaction T_2: lors de cette étape (validation locale de T_2), l'intervalle $EI(T_1,x)$ qui comprend l'estampille $E(x_{T2})$ associée à la transaction validée T_2 sera tronqué à droite.



-instant 6: Lors de la certification globale de T_1, on procède à l'intersection des deux ensembles d'intervalles, $EI(T_1,x)$ et $EI(T_1,y)$: cette intersection étant égale à l'ensemble vide, T_1 sera rejetée.



VI.12- PRIVATION

Rien n'empêche une transaction rejetée et relancée sur les sites, par l'utilisateur ou par le système, d'avorter à nouveau: la méthode MPIE pose le problème de la privation puisqu'une transaction peut être rejetée indéfiniment.

Pour pallier à cela, on pourrait privilégier certaines transactions en faisant porter la mention "certification prioritaire" sur le message qui lance leur certification locale.

Ce message activerait alors, au niveau de chaque site accédé, des mesures de "protection" de la transaction privilégiée, mesures qui ne seraient désactivées qu'à la certification de celle-ci. Elles devraient inhiber et retarder les certifications de toutes les autres transactions, qu'elles soient prioritaires ou non - les certifications portant la mention "prioritaire" seraient traitées en fonction de leur ordre d'arrivée -.

Ainsi, la certification de la transaction privilégiée ne prendrait en considération que les dépendances effectives qui la relient aux transactions déjà validées lors de son démarrage. Il ne serait pas tenu compte de ses dépendances potentielles avec les autres transactions vivantes: pas de réajustement de l'intervalle d'une lecture suite à la validation d'une écriture ni celui de l'intervalle d'une écriture afin de privilégier une lecture pure. Durant la certification de la transaction prioritaire les autres transactions vivantes ne pourraient dérouler que leur phase d'accès aux objets. Elles n'entameraient leur certification qu'après exécution complète de la privilégiée.

Cette manière de procéder garantit l'exécution totale de la transaction

et résout le problème de la privation. Elle peut cependant nuire aux lectrices pures. Une transaction prioritaire qui exécute une écriture n'effectue plus de contrôle en avant pour privilégier les transactions de lecture pure et peut les invalider.

VI.13- CONCLUSION

Notre étude suit les différentes phases de l'édification de la méthode MPIE et souligne les améliorations amenées au fur et à mesure par chacune d'elles.

Cette méthode a été construite sur un schéma de transaction optimiste. Dans le souci de fournir à l'utilisateur les données les plus récentes, nous avons d'abord forcé l'accès en lecture à s'exécuter sur la dernière version de l'objet et l'accès en écriture à créer la version la plus récente dans le temps logique.

Conçue de cette manière, la méthode fournit aux transactions de larges possibilités de sérialisation et n'admet pas de rejet au niveau local, mis à part celui des trop vieilles lectrices. De plus les écrivains purs ne sont jamais rejetés: l'ensemble d'intervalles qui leur est associé au niveau global n'est jamais vide puisque leurs ensembles locaux sont toujours de borne supérieure infinie. Seules les transactions qui ont effectué des lectures peuvent voir leur ensemble d'intervalles associé tronqué à droite sur un site et peuvent se voir rejetées.

Nous avons donc introduit, au niveau global, un intervalle courant qui synchronise les opérations de lecture d'une lectrice pure sur son premier accès: il véhicule, sur tous les sites, la vue de la base de données que possède la lectrice à l'instant de cet accès. Nous avons ensuite résolu le rejet des lectrices pures en associant à cette synchronisation un contrôle en avant effectué par les transactions écrivains en phase de certification locale. Pour ne pas invalider la lectrice pure et pour lui laisser de plus larges possibilités de sérialisation, les écrivains réajustent leurs propres intervalles en les tronquant à gauche. Ainsi et à l'instar des méthodes multiversions classiques, la méthode MPIE favorise les lectures en leur offrant un choix de versions pour un même objet et en reportant le contrôle sur les écritures.

Cependant, l'utilisation de l'intervalle courant induit un schéma de transaction séquentiel en phase de vie et pénalise légèrement le parallélisme. De plus, la synchronisation des lectures sur le premier accès peut donner à la lectrice une vue passée de la base de données. Nous avons donc étendu la notion d'intervalle courant et permis à une opération de lecture d'avoir pour résultat un certain nombre de versions -et non plus une seule-, à travers un ensemble d'intervalles courants. L'intervalle qui fournit la vue la plus récente de la base de données est récupéré ensuite en phase de certification globale. Dans cette solution aussi, le contrôle est reporté sur les transactions écrivains.

La méthode que nous avons proposé met en oeuvre la technique des intervalles d'estampilles sur une base de données multiversion. Nous l'avons élaborée dans le but d'améliorer les performances des méthodes multiversions en supprimant les inconvénients dus à l'estampillage statique et à l'ordre de sérialisation qu'il impose. Cet ordre n'est plus fixé à priori, mais est calculé d'une manière dynamique, au fur et à mesure des

accès aux versions des objets. Il est différent de l'ordre chronologique des certifications et admet qu'une transaction vivante précède une autre déjà validée.

Grâce à la technique des intervalles d'estampilles, la certification d'une transaction ne dépend que de ses contraintes avec les transactions déjà validées. Les intervalles expriment un ordre proche de l'ordre partiel engendré par les dépendances entre les transactions.

Le côté multiversion offre plusieurs possibilités de sérialisation aux transactions et minimise les risques de rejets par rapport à l'application de la technique des intervalles d'estampilles sur une base de données monoversion.

Conclusions:

Nous avons présenté brièvement dans cette thèse les deux grands types de contrôle développés à ce jour sur des bases de données monoversions ou multiversions: le contrôle pessimiste qui vérifie la sérialisabilité de la transaction de manière continue et les méthodes optimistes qui ne contrôlent la transaction qu'au niveau d'une étape finale de certification. Nous y avons exposé ensuite la conception d'une méthode de contrôle optimiste, la méthode Multiversion Par Intervalles d'Estampilles qui utilise la technique des intervalles d'estampilles sur une base de données multiversion.

L'étude des méthodes classiques de contrôle continu fait ressortir une certaine rigidité des techniques qu'elles utilisent pour réaliser le contrôle global et assurer la sérialisabilité des transactions. Le contrôle s'effectue au moyen de techniques d'estampillages [BERN 81] et de verrouillage à deux phases [ESWA 76],[TRAI 82]. L'estampillage utilisé est statique (l'ordre des estampilles est fixé à priori) ou dynamique (l'ordre n'est pas fixé au préalable mais lors du premier conflit). Dans le verrouillage à deux phases, l'ordre de sérialisation global est construit à partir des ordres locaux d'accès aux objets et ceux-ci peuvent être incompatibles. Cette technique a pour principal inconvénient la création de problèmes d'interblocage qu'il faudra prévenir ou guérir.

Le contrôle réalisé à postériori dans les méthodes par certification ([KUNG 81], [CERI 82], [CARE 83]) autorise plus de parallélisme quand les conflits sont rares. Par contre, ces méthodes imposent un ordre de sérialisation des transactions identique à l'ordre chronologique de leur entrée en phase de certification. Elles posent aussi le problème de la privation puisqu'elles sont basées sur le rejet de la transaction qui provoque l'incohérence. Dans ce type de contrôle, la technique des intervalles d'estampilles qui a été présentée dans [BOKS 84] et qui généralise l'estampillage dynamique s'avère particulièrement intéressante: le caractère dynamique de cette technique, allié au caractère optimiste du contrôle par certification, augmente notablement le niveau de parallélisme. Elle permet en plus un ordre chronologique des certifications différent de l'ordre de sérialisation lequel est construit à partir des conflits.

Les méthodes mises en oeuvre sur des bases de données multiversions ont pour principal objectif le non rejet de lectrices et l'élimination des problèmes liés au contrôle sur les opérations de lectures. Pour cela, elles s'appuient sur la gestion d'un historique de versions au niveau de chaque objet. Elles permettent ainsi aux vieilles lectrices -transactions voulant lire une donnée modifiée- de ne plus être rejetées. En revanche, elles posent le problème de la gestion de l'espace mémoire relatif aux versions des objets.

Pour illustrer les méthodes multiversions, nous avons présenté les approches suivantes:

-l'algorithme de [REED 78] ainsi que sa version développée et enrichie dans [CARE 83]: il permet aux transactions lectrices d'invalider les transactions écrivains.

-l'approche multiversion optimiste de [MING 84] qui laisse les résultats des transactions en phase de certification accessibles aux autres transactions. Ainsi, les demandes en lecture n'induiraient pas de retards et bénéficieraient toujours des versions les plus récentes. Cette méthode pose cependant le problème des rejets en cascade puisque la validation