

MASTER INFORMATIQUE DE L'UNIVERSITÉ BORDEAUX 1
 BASES DE DONNÉES AVANCÉES INF 559 (CHRISTIAN RETORÉ)
 EXERCICES CORRIGÉS ACCÈS CONCURRENTS (31 DÉCEMBRE 2007)

Le cours a présenté trois techniques de gestion des accès concurrents :

1. énoncé du problème : sérialisabilité, sérialisabilité par permutation, test par construction du graphe de précédence ;
2. verrous simples : transactions cohérentes (actions précédées de verrouillage ; déverrouillage ultérieur), schéma légaux (au plus un verrou sur un élément) ; verrous de plusieurs types (partagés et exclusifs) , modification des notions de cohérence et de légalité en conséquences
3. estampillage (time stamps) simple ou avec copies multiples
4. validation

Un rappel peut-être utile (niveau lycée) : étant données deux suites $a_1 \cdots a_n$ et $b_1 \cdots b_p$ sans élément commun il y a $C_{n+p}^p = C_{n+p}^n = (n+p)! / p!n!$ suites différentes contenant chaque élément de chaque suite exactement une fois, et préservant l'ordre interne à chaque suite : a_i avant a_j si et seulement si $i < j$ et b_i avant b_j si et seulement si $i < j$. Il est amusant de montrer cela par récurrence ; cela conduit à une formule $f(n, p) = f(n-1, p) + f(n, p-1)$ avec $f(0, p) = f(n, 0) = 1$ et on conclut en utilisant la formule du triangle dit de Pascal : $C_{n-1}^{p-1} + C_{n-1}^p = C_n^p$.

Pour les exercices de réflexion, pas d'exercice type, mais vous pouvez réfléchir aux questions suivantes : quel type de correction garantissent les techniques de gestion des accès concurrents ? comment les garantissent-elles ?

Notation $o_i(E)$: la transaction i effectuée sur l'élément E l'opération x qui peut être :

- r lecture (read)
- w écriture (write)

Exercice A

Dans un système par verrouillage l'opération x ci-dessus peut aussi être :

- l verrouillage (lock)
- u déverrouillage (unlock)

On considère les transactions suivantes :

- $S_1 : r_1(A); A := A + 2; w_1(A); r_1(B); B := B * 3; w_1(B);$
- $S_2 : r_2(B); B := B * 2; w_2(B); r_2(A); A := A + 3; w_2(A);$
- $T_1 : l_1(A); r_1(A); A := A + 2; w_1(A); u_1(A); l_1(B); r_1(B); B := B * 3; w_1(B); u_1(B);$
- $T_2 : l_2(B); r_2(B); B := B * 2; w_2(B); u_2(B); l_2(A); r_2(A); A := A + 3; w_2(A); u_2(A);$
- $U_1 : l_1(A); r_1(A); A := A + 2; w_1(A); l_1(B); r_1(B); B := B * 3; w_1(B); u_1(A); u_1(B);$
- $U_2 : l_2(B); r_2(B); B := B * 2; w_2(B); l_2(A); r_2(A); A := A + 3; w_2(A); u_2(B); u_2(A);$

(A.1) Trouver un schéma d'exécution de S_1 et S_2 (c'est-à-dire T_1 et T_2 sans verrouillage ni déverrouillage) qui soit sérialisable mais pas sérialisable par permutation.

On remarque que les opérations $X := X * 3$ et $X := X * 2$ ainsi que $Y := Y + 2$ et $Y := Y + 3$ commutent. Le schéma $S : r_1(A); A := A + 2; w_1(A); r_2(B); B := B * 2; w_2(B); r_2(A); A := A + 3; w_2(A); r_1(B); B := B * 3; w_1(B)$; est donc équivalent, quel que soit l'état initial de la base au schéma sériel : $S_1; S_2 : r_1(A); A := A + 2; w_1(A); r_1(B); B := B * 3; w_1(B); r_2(B); B := B * 2; w_2(B); r_2(A); A := A + 3; w_2(A)$; Mais en permutant les actions qui commutent, on ne pourra jamais transformer S en $S_1; S_2$ (puisque $w_2(B)$ ne peut passer après $r_1(B)$) ni en $S_2; S_1$ (puisque $w_1(A)$ ne peut passer après $r_2(A)$).

(A.ii) Donner un schéma d'exécution de T_1 et T_2 interdit par les verrous.

Voici un schéma que les verrous empêchent : $S' : l_1(A); r_1(A); A := A + 2; w_1(A); l_2(B); r_2(B); B := B * 2; w_2(B); u_2(B); l_2(A); r_2(A); A := A + 3; w_2(A); u_2(A); u_1(A); l_1(B); r_1(B); B := B * 3; w_1(B); u_1(B);$
L'opération encadrée n'est pas possible, A étant toujours verrouillé par la transaction T_1 .

(A.iii) Combien y a-t-il de schémas légaux pour T_1 et T_2 ?

On va compter les opérations de multiplication et d'augmentation de A et de B par 2 et par 3, puisqu'elles sont mentionnées dans le schéma, mais on pourrait aussi ne considérer que les lectures et écritures.

On appelle T_1^A les opérations de T_1 qui portent sur A (les 5 premières) ; T_1^B les opérations de T_1 qui portent sur B (les 5 dernières) ; T_2^B les opérations de T_2 qui portent sur B (les 5 premières) ; T_2^A les opérations de T_2 qui portent sur A (les 5 dernières). Ainsi $T_1 = T_1^A; T_1^B$ et $T_2 = T_2^B; T_2^A$.

Il y a diverses manières de compter les cas, et diverses formulations du résultat ; mais du moment qu'elles n'en oublient pas et qu'elles ne comptent pas deux fois les mêmes, elles sont toutes valables. Parmi $5 + 5 = 10$ premières opérations du schéma d'exécution,

1. soit $l_1(B)$ apparaît,
2. soit $l_2(A)$ apparaît,
3. soit aucun des deux n'apparaît.

Comptons maintenant tous les schémas légaux possibles, dans chacun des cas définis ci-dessus :

1. Si $l_1(B)$ apparaît, alors les 5 opérations de T_1^A on toutes été effectuées, mais les 5 de T_2^B n'ont pas pu être toutes effectuées, et en particulier la dernière opération de T_2^B , c.-à-d. $u_2(B)$ n'a pas été effectuée. Aucune opération de de T_2^B n'a donc été effectuée, car sinon $l_1(B)$ n'aurait pu avoir lieu, B étant toujours verrouillé. Si aucune n'a été effectuée c'est que les 10 premières opérations du schéma sont T_1 : ce schéma est donc $T_1; T_2$.
2. Si $l_2(A)$ apparaît, un raisonnement symétrique au précédent montre que les schéma est $T_2; T_1$.
3. Si ni $l_1(B)$ ni $l_2(A)$ n'apparaît dans les 10 premières opérations, c'est que les 10 premières opérations sont constituées des 5 opérations de T_1^A et des 5 opérations de T_2^B . Il y a C_{10}^5 telles suites possibles, et toutes donnent des schémas légaux (pas de conflits entre les verrouillages sur A et ceux sur B). Pour poursuivre, les C_{10}^5 suites possibles obtenues par entrelacement des 5 opérations de T_1^B avec les 5 opérations de T_2^A , conviennent toutes aussi. Cela fait $(C_{10}^5) * (C_{10}^5)$ schémas légaux possibles.

Au total on trouve $2 + (C_{10}^5)^2 = 63506$ schémas légaux possibles.

(A.iv) Combien y a-t-il de schémas légaux et sérialisables pour T_1 et T_2 ?

Comme dit ci-dessus, tous les schéma sont sérialisables (notions pour laquelle seule importe les r_i et les w_i), car les additions sur A commutent entre elles ainsi que les multiplications sur B .

(A.v) Combien y a-t-il de schémas légaux et sérialisables par permutation pour T_1 et T_2 ?

Comme T_1 et T_2 ne sont pas des transactions à deux phases, on ne peut pas affirmer *a priori* que tous les schémas légaux soient sérialisables par permutation.

Reprenons la partition des schémas que nous avons utilisé en (A.iii) pour compter les schémas légaux. Nous allons montrer qu'aucun des schémas dans lesquels ni $l_1(B)$ ni $l_2(A)$ n'apparaît dans les 10 premières opérations n'est sérialisable par permutation. Un tel schéma, disons S est constitué d'un entrelacement de T_1^A et de T_2^B , disons S' , puis d'un entrelacement de T_1^B et T_2^A , disons S'' . La première partie S' contient $w_1(A)$ et la seconde S'' contient $w_2(A)$, on a donc T_1 nécessairement avant T_2 dans tout schéma sériel équivalent par permutation à S (un arc de T_1 vers T_2 dans le graphe qui teste la sérialisabilité par permutation). La première partie S' contient $w_2(B)$ et la seconde S'' contient $w_1(A)$, on a donc T_2 nécessairement avant T_1 dans tout schéma sériel équivalent par permutation à S (un arc de T_2 vers T_1 dans le graphe qui teste la sérialisabilité par permutation). Il n'y a donc pas de schéma sériel équivalent par permutation à S (on a un cycle $T_1 \rightarrow T_2 \rightarrow T_1$ dans le graphe dont l'acyclicité équivaut à la sérialisabilité par permutation).

Les deux autres schémas, $T_1; T_2$ et $T_2; T_1$ sont sériels et donc sérialisables par permutation.

(A.vi) T_1 et T_2 sont-elles des transactions à deux phases ?

T_1 n'est pas une transaction à deux phases : en effet, le verrouillage de B intervient après le déverrouillage de A . T_2 n'est pas une transaction à deux phases : en effet, le verrouillage de A intervient après le déverrouillage de B .

(A.vii) U_1 et U_2 sont-elles des transactions à deux phases ?

Oui. Dans l'une comme dans l'autre tous les verrouillages précèdent tous les déverrouillages.

(A.viii) Combien y a-t-il de schémas légaux d'exécution de U_1 et U_2 ? (en ne prenant en considération que les opérations effectives, les r et les w).

On peut compter comme on l'a fait précédemment en s'interrogeant sur la présence de $l_1(B)$ ou de $l_2(A)$ dans les 8 premières opérations. Mais on peut aussi raisonner sur l'ordre d'apparition des 4 verrouillages dans l'exécution, $l_1(A), l_1(B), l_2(B), l_2(A)$, et c'est ce qu'on va faire.

Les deux premiers verrouillages ne peuvent pas être $l_1(A), \dots, l_2(B)$. Si tel était le cas, le verrouillage suivant ne pourrait pas être $l_1(B)$ car il faudrait qu'intervienne avant $l_1(B)$ le déverrouillage $u_2(B)$, et on aurait donc forcément déjà rencontré $l_2(A)$. Le verrouillage suivant ne pourrait pas non plus être $l_2(A)$, car il faudrait qu'intervienne avant $l_2(A)$ le déverrouillage $u_1(A)$, et on aurait donc forcément déjà rencontré $l_1(B)$. Et comme il faut bien que les 4 verrouillages figurent dans le schéma, c'est que les deux premiers verrouillages ne peuvent pas être $l_1(A), \dots, l_2(B)$.

Pour des raisons symétriques, les deux premiers verrouillages ne peuvent pas non plus être $l_2(B), \dots, l_1(A)$.

Les deux premiers verrouillages ne peuvent pas non plus être $l_1(A), \dots, l_2(A)$ puisque dans U_2 $l_2(A)$ est précédé de $l_2(B)$.

Pour des raisons symétriques, les deux premiers verrouillages ne peuvent pas non plus être $l_2(B), \dots, l_1(B)$ puisque dans U_1 $l_1(B)$ est précédé de $l_1(A)$.

Les deux premiers verrouillages sont donc

1. soit $l_1(A), \dots, l_1(B)$
2. soit $l_2(B), \dots, l_2(A)$.

Calculons dans chacun de ces deux cas, les nombres de tels schémas légaux :

1. Dans le premier cas, peut-il y avoir des opérations de U_2 entre les deux verrouillages de U_1 ? Non, car s'il y en a, B est verrouillé par U_2 et pour qu'il soit déverrouillé et permette $l_1(B)$, il faudrait atteindre $u_2(B)$, et donc être passé par $l_2(A)$, ce qui est impossible car U_1 déverrouille A après $l_1(B)$. Après $l_1(B)$ aucune opération de U_2 ne peut prendre place tant que $u_1(B)$ n'a pas été atteinte, puisque $l_2(B)$ est la première opération de U_2 . Il faut donc atteindre $u_1(B)$, la dernière opération de U_1 : le schéma est donc $U_1; U_2$.
2. Dans le second cas, un argument symétrique montre que le schéma est nécessairement $U_2; U_1$.

Il n'y a que donc deux schémas légaux — sériels et obtenus à partir de transaction à deux phases : deux raisons d'être sérialisables par permutation !

La question se restreint aux r et aux w car l'exercice a été obtenu par simplification d'un exercice où il était demandé de placer soi-même les verrous pour obtenir les transitions à deux phases, et si on inverse les déverrouillages en fin de transaction ($u_1(B); u_1(A)$ et $u_2(A); u_2(B)$), il y a des schémas en plus (par exemple avec la séquence $\dots, u_1(B); l_2(B); r_2(B); u_1(A), \dots$ au changement de transaction) mais qui ont le même ordre de w et de r .

Exercice B

Dans un système avec des verrous partagés, l'opération de déverrouillage est inchangée, mais l'opération de verrouillage l ci-dessus est remplacée par deux opérations de verrouillage :

- xl verrouillage avec un verrou exclusif
- sl verrouillage avec un verrou possiblement partagé

On considère deux schémas d'exécution construits sur trois transactions 1, 2, 3 :

- $\mathcal{S}_1 : r_1(A); r_2(B); r_3(C); w_1(B); w_2(C); w_3(D);$
- $\mathcal{S}_2 : r_1(A); r_2(B); r_3(C); r_1(B); r_2(C); r_3(D); w_1(C); w_2(D); w_3(E);$

(B.i) Pour chacun des deux schémas :

- Insérer des verrous partagés avant chaque lecture si elle n'est pas suivie d'une écriture du même élément par la même transaction.
- Placer un verrou exclusif en face de toutes les autres écritures ou lectures.
- Placer les déverrouillages à la fin de chaque transaction.

(B.ii) Décrire l'exécution de la suite d'opérations ainsi obtenue.

À titre d'exemple, traitons le schéma \mathcal{S}_2 . Celui-ci devient :

$sl_1(A); r_1(A); sl_2(B); r_2(B); sl_3(C); r_3(C); sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(D); r_3(D);$
 $xl_1(C); w_1(C); u_1(A); u_1(B); xl_2(D); w_2(D); u_2(B); u_2(C); u_2(D); xl_3(E); w_3(E); u_3(C); u_3(D); u_3(E);$

Les séquences d'opérations correspondant à chaque transactions sont donc :

$T_1 : sl_1(A); r_1(A); sl_1(B); r_1(B); xl_1(C); w_1(C); u_1(A); u_1(B);$
 $T_2 : sl_2(B); r_2(B); sl_2(C); r_2(C); xl_2(D); w_2(D); u_2(B); u_2(C); u_2(D);$
 $T_3 : sl_3(C); r_3(C); sl_3(D); r_3(D); xl_3(E); w_3(E); u_3(C); u_3(D); u_3(E);$

Jusqu'à $r_3(D)$ incluse, toutes les opérations passent dans l'ordre indiqué. Mais ensuite $xl_1(C)$ ne peut être accordé, en raison de $sl_3(C)$ et de $sl_2(C)$; le schéma écrit ne se déroulera donc pas comme prévu. Explicitons une suite possible. Le $xl_1(C)$ et toutes les opérations suivantes de T_1 sont mises en attente, C étant toujours verrouillé par T_2 et T_3 . Le $xl_2(D)$ et toutes les opérations suivantes de T_2 sont mises en attente, D étant toujours verrouillé par T_3 . Le $xl_3(E)$ passe, et les deux autres transactions T_1 et T_2 restent bloquées. Après $u_3(C)$, on peut relancer T_1 . Si, par exemple, le contrôleur lance toutes les opérations de T_1 en attente, elles passent toutes, il peut alors exécuter $u_3(D)$ puis les opération de T_2 en attente puis $u_3(E)$, la dernière opération de T_3 . Pour résumer, voici un schéma d'exécution possible lorsque \mathcal{S}_2 est lancé :

$sl_1(A); r_1(A); sl_2(B); r_2(B); sl_3(C); r_3(C); sl_1(B); r_1(B); sl_2(C); r_2(C); sl_3(D); r_3(D); xl_3(E); w_3(E);$
 $u_3(C); xl_1(C); w_1(C); u_1(A); u_1(B); u_3(D); xl_2(D); w_2(D); u_2(B); u_2(C); u_2(D); u_3(E);$

Exercice C

Notation spécifique pour l'estampillage :

– st_i : la transaction i commence.

On considère les séquences d'opérations suivantes, ou st_i signifie que la transaction i commence :

$\mathcal{S}_0 : st_1; st_2; st_3; st_4; r_1(A); w_1(A); w_4(A); r_3(A); w_3(A); w_2(A); r_2(A); st_5; st_6; st_7; w_5(A); r_2(A); r_7(A); r_6(A)$

$\mathcal{S}_1 : st_1; st_2; st_3; st_4; w_1(A); w_3(A); r_4(A); r_2(A);$

$\mathcal{S}_2 : st_1; st_2; st_3; st_4; w_1(A); w_4(A); r_3(A); w_2(A);$

$\mathcal{S}_3 : st_1; st_2; st_3; st_4; w_1(A); w_2(A); w_3(A); r_2(A); r_4(A);$

(C.i) Décrire l'exécution de chacune des suites d'opérations avec un système d'estampillage simple.

Les instructions $commit_i$ ne sont pas indiquées on est libre de les déclencher dès lors qu'il n'y a plus d'opération de la transaction i . C'est après (et pour nous ce sera juste après) que le booléen $c(X)$ est mis à 1 pour tous les X écrits par T_i Il reste alors à suivre l'algorithme donné en cours. Voici une possibilité d'exécution pour le schéma S_0 . Il y a un degré de liberté pour le moment où on relance les transactions comme pour celui où on les commit.

<i>action</i>	$RT(A)$	$WT(A)$	$c(A)$	
st_1	0	0	1	$T_1 : 10$
st_2	0	0	1	$T_2 : 20$
st_3	0	0	1	$T_3 : 30$
st_4	0	0	1	$T_4 : 400$
$r_1(A)$	10	0	1	
$w_1(A)$	10	0	0	
$commit_1$	10	10	1	
$w_4(A)$	10	10	0	
$commit_4$	10	400	1	
$r_3(A)$	10	400	1	T_3 annulée : $400 > 300$ et $c(A) = 1$
$commit_3$	10	400	1	n'a plus de sens
$w_2(A);$	10	400	1	ignorée $WT(A) > \#T_2$
st_3	10	400	1	$T_3 : 3000$ reprise de T_3
$r_3(A)$	3000	400	1	
$w_3(A)$	3000	400	0	
$r_2(A)$	3000	3000	0	lecture mise en attente $c(A) : 0$
st_5	3000	3000	0	$T_5 : 5000$
st_6	3000	3000	0	$T_6 : 6000$
st_7	3000	3000	0	$T_7 : 7000$
$w_5(A)$	3000	3000	0	écriture mise en attente $c(A) : 0$
$commit_3$	3000	3000	1	
$r_2(A)$	3000	3000	1	$r_2(A)$ est relancée T_2 est annulée
$w_5(A)$	3000	3000	0	T_5 reprend
$r_7(A)$	3000	3000	0	lecture en attente
$commit_5$	3000	5000	1	
$r_7(A)$	7000	5000	1	T_7 reprend
$r_6(A)$	7000	3000	1	$RT(A)$ inchangé

(C.ii) Décrire l'exécution de chacune des suites d'opérations avec un système d'estampillage multiple.

L'estampillage multiple permet d'éviter certains échecs de lecture ou d'écriture. Pour éviter ces derniers, on crée une nouvelle copie datée de l'élément, chacune étant munie du numéro de la transaction ayant effectué la dernière lecture – ce nombre est initialisé à la transaction ayant créé cette copie (du reste, il ne peut lui être inférieur).

<i>action</i>	A_0					
st_1	0					$T_1:10$
st_2	0					$T_2:20$
st_3	0					$T_3:30$
st_4	0					$T_4:400$
$r_1(A)$	10					
$w_1(A)$	–	A_{10}	suppression de A_0 (plus de transaction < 10)			
$w_4(A)$	–	10	A_{400}			
$r_3(A)$	–	30	400			
$w_3(A)$	–	30	400	A_{30}		
$w_2(A)$	–	30	400	30	T_2 annulée. A_{400} n'est pas créé	
st_2	–	30	400	30	$T_2:2000$ reprise de T_2	
$w_2(A);$	–	30	400	30	A_{2000}	
$r_2(A)$	–	30	400	30	2000	
st_5	–	30	400	30	2000	$T_5:5000$
st_6	–	30	400	30	2000	$T_6:6000$
st_7	–	30	400	30	2000	$T_7:7000$
$w_5(A)$	–	30	400	30	2000	A_{5000}
$r_7(A)$	–	30	400	30	2000	7000
$r_6(A)$	–	30	400	30	2000	7000

Exercice D

Notation spécifique pour les événements utiles à un système par validation :

- R_iX la transaction i commence, et son ensemble d'éléments lus est X .
- V_i la transaction i essaie de valider.
- W_iX la transaction i finit, et son ensemble d'éléments écrits est X .

On considère les suites suivantes d'événements :

$\mathcal{S}_0 : R_1\{A, B\}; R_2\{B, C\}; V_1; R_3\{C, D\}; V_3; W_1\{A\}; V_2; W_2\{A\}; W_3\{B\};$

$\mathcal{S}_1 : R_1\{A, B\}; R_2\{B, C\}; V_1; R_3\{C, D\}; V_3; W_1\{C\}; V_2; W_2\{A\}; W_3\{D\};$

$\mathcal{S}_2 : R_1\{A, B\}; R_2\{B, C\}; R_3\{C\}; V_1; V_2; V_3; W_1\{A\}; W_2\{B\}; W_3\{C\};$

$\mathcal{S}_3 : R_1\{A, B\}; R_2\{B, C\}; R_3\{C\}; V_1; V_2; V_3; W_1\{C\}; W_2\{B\}; W_3\{A\};$

$\mathcal{S}_4 : R_1\{A, B\}; R_2\{B, C\}; R_3\{C\}; V_1; V_2; V_3; W_1\{A\}; W_2\{C\}; W_3\{B\};$

Décrire dans chacun des cas l'exécution de la séquence d'opérations dans un système par validation.

Traitons le cas \mathcal{S}_0 .

Lorsque T_1 cherche à valider, il n'y a pas de transaction ayant validé, il n'y a rien à vérifier, T_1 valide.

Lorsque T_3 cherche à valider en V_3 , T_1 a déjà validé.

(RS) On a $FIN(T_1) > START(T_3)$ il faut vérifier que $RS(T_3) = \{C, D\}$ et $WS(T_1) = \{A\}$ sont disjoints, c'est le cas.

(WS) On a $FIN(T_1) > VAL(T_3)$ il faut vérifier que $WS(T_3) = \{B\}$ et $WS(T_1) = \{A\}$ sont disjoints, c'est aussi le cas.

T_3 valide.

T_1 finit.

Lorsque T_2 cherche à valider en V_2 , T_1 et T_3 ont validé.

T_1

RS On a $FIN(T_1) > START(T_2)$ il faut donc vérifier que $RS(T_2) = \{B, C\}$ et $WS(T_1) = \{A\}$ sont disjoints. C'est le cas.

WS On n'a pas $FIN(T_1) > VAL(T_2)$, rien à vérifier sur $WS(T_1)$ et $WS(T_2)$.

T_3

(RS) On a $FIN(T_3) > START(T_2)$ il faut donc vérifier que $RS(T_2) = \{B, C\}$ et $WS(T_3) = \{B\}$ sont disjoints, ce n'est pas le cas, T_2 est annulée.

(WS) T_2 étant annulée, cette vérification est inutile.

T_2 est annulée.

Si T_2 redémarre immédiatement, et que les instructions $R_2\{B, C\}$ et V_2 arrivent avant $W_3\{B\}$, alors T_2 sera annulée pour la même raison. Comme l'annulation prend un certain temps, il est vraisemblable que T_3 finisse d'abord.

Si T_2 cherche à valider ensuite en V_2 après $W_3\{B\}$, T_1 et T_3 ont validé. Comme $FIN(T_1) < FIN(T_3) < START(T_2) < VAL(T_2)$ il n'y a donc pas lieu de vérifier que des ensembles d'écriture et de lectures sont disjoints, T_2 valide sans problème.