

## 1 Notions clés

### 1.1 Schémas sériels

Si on traite les actions arrivant de diverses transactions et machines au fur et à mesure de leur arrivée sur le serveur, encore faut-il veiller à ce que tout se passe comme si une transaction avait lieu en premier, puis une autre transaction, ... chacune étant effectuée dans son intégralité.

Une *transaction*  $T_i$  est une séquence d'actions élémentaires d'écriture et de lecture, notées  $w_i(A)$  ou  $r_i(A)$  où  $w$  signifie écriture et  $r$  lecture, où  $A$  dénote l'élément lu ou écrit. Etant donnée une famille  $I$  de transactions, un *schéma*  $S$  est une suite d'actions élémentaires telle que  $S$  restreint aux actions d'une des transactions est exactement la séquence des actions de la transaction  $i$  — dans le même ordre — et telle que toute action élémentaire de  $S$  appartient à l'une des transactions de  $I$ .

Un schéma est dit *sériel* s'il consiste en une transaction complète, puis une autre transaction complète etc. : entre deux actions de la même transaction  $i$  il n'y a que des actions de cette même transaction  $i$ .

### 1.2 Schémas sérialisables

Un schéma constitué des transactions  $T_1, \dots, T_N$  est dit *sérialisable* s'il existe un schéma sériel constitué des transactions  $T_1, \dots, T_N$  qui produise le même effet, quels que soient l'état initial de la base et les valeurs inscrites ou lues, .

Voici un exemple de schéma non sérialisable.  $(T_1)$  multiplie par deux  $A$  puis  $B$ ,  $T_1 : r_1(A), (A := A * 2), w_1(A), r_1(B), (B := B * 2), w_1(B)$  et  $(T_2)$  ajoute 100 à  $A$  puis à  $B$ :  $T_2 : r_2(A), (A := A + 100), w_2(A), r_2(B), (B := B + 100), w_2(B)$  Si les deux valeurs de  $A$  et de  $B$  sont égales à 100 au début, elles doivent l'être ensuite. Néanmoins, si on fait la partie  $A$  de  $T_1$  c'est-à-dire  $(T_1/A)A := A * 2$ , puis  $(T_2)A := A + 100$  puis la partie  $B$  de  $T_1$  c'est-à-dire  $(T_1/B)B := 2 * B$  on obtient  $A = 300$  et  $B = 400$ ! Ce schéma n'est pas sérialisable.

Remarque: si au lieu d'ajouter 100 la deuxième transaction multiplie  $A$  et  $B$  par trois, le schéma ne serait toujours pas sérialisable, car cela dépend des opérations effectuées sur les éléments.

### 1.3 Schémas sérialisables par permutation

Afin de voir si un schéma est sérialisable, on peut déjà voir si un schéma est sérialisable par permutation: peut-on échanger les actions deux à deux de manière à obtenir un schéma sériel tout en étant sûr que l'effet sur la base soit le même? Dans ce cas, le schéma est dit *sérialisable par permutation* (conflict serialisable). On observe que deux actions ne commutent pas lorsque les deux portent sur un même élément, et que l'une des deux actions est une écriture.

Un schéma sérialisable par permutation est toujours sérialisable. Par contre, un schéma peut très bien être sérialisable sans être sérialisable par permutation:

$$w_1(Y)w_2(Y)w_2(X)w_1(X)w_3(X) \equiv w_1(Y)w_1(X)w_2(Y)w_2(X)w_3(X)$$

## 1.4 Test de sérialisabilité par permutation

On construit un graphe dont les sommets sont les transactions et on dessine un arc de  $T_i$  vers  $T_j$  lorsqu'il existe des actions  $a_i$  de  $T_i$  et  $a_j$  de  $T_j$  telles que:  $a_i$  est avant  $a_j$  dans le schéma,  $a_i$  et  $a_j$  concernent le même élément, et l'une au moins des deux actions est une écriture. Ce graphe est acyclique si et seulement si le schéma est sérialisable par permutation. En effet, s'il y a un cycle, on voit qu'on a un cycle de contraintes d'ordre, car chaque arc correspond à  $T_i$  doit être avant  $T_j$  dans tout ordre sériel équivalent par permutation. Réciproquement, s'il n'y a pas de cycle, alors il existe un ordre sériel équivalent. On procède par récurrence sur le nombre de transactions, et on montre que la première action d'une transaction initiale  $T_0$  dans le graphe commute avec toutes les actions qui la précèdent dans le schéma; on peut ainsi déplacer toutes les transactions de  $T_0$  en tête du schéma.

## 2 Les solutions pessimistes et optimistes

Afin de garantir la sérialisabilité, les systèmes de gestion de bases de données proposent plusieurs mécanismes qui en fait garantissent un peu plus, puisqu'ils garantissent la sérialisabilité par permutation.

Les verrous constituent un système pessimiste: ils diffèrent certaines actions parfois sans que cela soit absolument nécessaire, mais ils n'ont jamais à défaire (ROLL BACK) une transaction hormis en cas de blocage (DEADLOCK), ce qui est toujours assez coûteux en temps. Dans ce système l'ordre sériel équivalent est fourni par l'ordre dans lequel les transactions lèvent les verrous.

Une solution plus optimiste est l'estampillage, où chaque transaction reçoit un numéro qu'elle laisse sur les éléments qu'elle lit ou écrit. Ce numéro correspond à l'ordre sériel équivalent. En cas d'ordre des actions incompatible avec l'ordre des transactions il faut différer ou même défaire une transaction.

Une autre solution encore plus optimiste consiste à laisser démarrer les transactions et à vérifier qu'il n'y aura pas de problème avant de lancer les écritures: c'est l'étape de validation dont les instant respectifs correspondent à l'ordre sériel équivalent. Cette méthode appelée validation a l'inconvénient de défaire un peu tardivement les transactions en cas de conflit.

## 3 Une solution pessimiste: les verrous

Les verrous consistent à empêcher toutes les situations conflictuelles pour garantir la sérialisabilité par permutation.

Les transactions comportent, outre les lectures et les écritures, des verrouillages d'éléments  $l_T(X)$  et des déverrouillages d'éléments  $u_T(X)$

Une transaction est dite *cohérente* lorsque:

- Pour écrire ou lire un élément, elle possède un verrou sur cet élément.
- Elle libère tout élément verrouillé.

Les schémas doivent être *légaux*: un élément n'est pas simultanément verrouillé par deux transactions.

### 3.1 2PL Two phase locking (Verrouillage en deux phases)

Cette condition simple est très couramment utilisée pour garantir que le schéma soit sérialisable par permutation: (2PL) dans chaque transaction tous les verrouillages précèdent tous les déverrouillages.

Pour voir que cette condition garantit la sérialisabilité par permutation on procède par récurrence sur le nombre de transactions.

S'il n'y en a qu'une, le schéma est sériel. Sinon, considérons la transaction  $T_i$  qui est la première, dans le schéma, à déverrouiller un élément  $A$  par l'action  $u(A)$ . On va déplacer toutes les actions de  $T_i$  en tête du schéma, soit  $a_i$  la première action de  $T_i$ , appelons  $X$  l'élément sur lequel elle porte. Supposons qu'il y ait un conflit avec  $a_j$  de la transaction  $T_j$ . De fait,  $a_j$  porte sur le même élément  $X$ , et donc  $X$  doit avoir été libéré par  $T_j$ , soit  $u_j(X)$  avant que  $T_i$  ne verrouille  $X$  par  $l_i(X)$ . Mais dans  $T_i$  tous les verrouillages précèdent tous les déverrouillages (2PL), et  $T_i$  est la première transaction du schéma à libérer un élément, on ne peut donc pas avoir à la fois  $u_i(A) < u_j(X)$  ( $u_i(A)$  est la première libération d'un élément) et  $u_j(X) < l_i(X) < u_i(A)$  (2PL). Il ne peut donc pas y avoir de conflit entre  $a_i$  et les  $a_j$  qui la précèdent, et on peut ainsi, de proche en proche, placer en tête toutes les actions de  $T_i$ .

### 3.2 Verrous distincts

On peut affiner le système des verrous en utilisant des verrous partagés qui suffisent pour la lecture ( $sl(X)$ ) et des verrous exclusifs nécessaires à l'écriture ( $xl(X)$ ), l'un comme l'autre étant relâchés par  $u(X)$ . Les transactions sont cohérentes si lors d'une lecture de  $X$  on dispose d'un verrou  $sl(X)$  ou  $sl(X)$  sur  $X$  et d'un verrou  $xl(X)$  pour toute écriture sur  $X$  et que tout verrou posé est levé par la suite.

Les schémas sont dits légaux lorsque aucun  $sl_j(X)$  ou  $xl_j(X)$  ne peut suivre un  $xl_i(X)$  (avec  $i \neq j$ ) sans qu'il y ait eu un  $u_i(X)$  entre les deux et aucun  $sl_j(X)$  ne peut suivre un  $xl_i(X)$  (avec  $i \neq j$ ) sans qu'il y ait eu un  $u_i(X)$  entre les deux. Une transaction peut très bien détenir un  $sl_i(X)$  et un  $xl_i(X)$ .

La condition 2PL est la même: tout déverrouillage doit suivre tout verrouillage et garantit aussi la sérialisabilité par permutation.

### 3.3 Blocage

Quel que soit le type de verrous utilisés, il n'est pas très difficile d'imaginer des situations de blocage (DEAD-LOCK) dans lesquelles chaque transaction attend qu'une autre transaction libère un élément. Pour sortir d'une telle situation, où disons  $U$  détient un verrou que  $T$  attend, le système peut suivre diverses stratégies au choix mais sans les mélanger. Une première stratégie consiste à laisser  $T$  attendre si  $T$  est plus ancienne que  $U$  et à annuler  $T$  sinon. Une autre consiste à dire que si  $T$  est plus ancienne que  $U$ ,  $T$  donne un ultimatum à  $U$ , et si  $U$  n'a pas fini dans ce délai,  $T$  tue  $U$ , et si  $U$  est plus ancienne,  $T$  attend  $U$ .

## 4 Une solution optimiste: l'estampillage

On affecte à chaque transaction  $T$  au moment où elle débute, un numéro  $\#T$  fourni par l'horloge ou le contrôleur. L'ordre entre les numéros est supposé être l'ordre sériel équivalent entre les transactions. Ce numéro sera laissé sur les éléments lorsqu'ils seront lus ou écrits par une transaction. Chaque élément  $X$  sera ainsi doté de deux numéros et un booléen:

- $RT(X)$  numéro de la dernière transaction à avoir lu  $X$
- $WT(X)$  numéro de la dernière transaction à avoir écrit  $X$
- $c(X)$  un booléen indiquant si la dernière instruction d'écrire  $X$ , celle qui correspond à  $WT(X)$ , a été ou non reportée sur le disque.

### 4.1 Les problèmes à éviter pour garantir la sérialisabilité par permutation dans l'ordre des numéros

#### 4.1.1 Le problème de la lecture tardive

On a une demande  $r_T(X)$  de la transaction  $T$ : cette dernière souhaite lire  $X$ . Mais  $\#T < WT(X)$ , c'est-à-dire  $X$  a déjà été écrit par une transaction supposée se passer après  $T$ !

$T$  débute    $U$  débute    $U$  lit  $X$     $T$  écrit  $X$

#### 4.1.2 Le problème de l'écriture tardive

On a une demande  $w_T(X)$  de la transaction  $T$ : cette dernière souhaite écrire  $X$ . Mais  $WT(X) < \#T < RT(X)$ , c'est-à-dire  $X$  a déjà été lu par une transaction de numéro  $RT(X)$  qui a lu la valeur écrite par la transaction de numéro  $WT(X)$  mais qui aurait dû lire la valeur écrite par  $T$ !

$T$  débute    $U$  débute    $U$  écrit  $X$     $T$  lit  $X$

#### 4.1.3 Le problème des données corrompues

Jusqu'à présent nous avons ignoré le  $c(X)$  qui est là pour prendre en compte les transactions annulées. Plusieurs problèmes peuvent se présenter:

$T$  peut lire  $X$ , mais la valeur lue par  $T$  a été écrite par une transaction annulée par la suite: ceci est détectable car  $c(X)$  est faux.

$U$  débute    $U$  écrit  $X$     $T$  débute    $T$  lit  $X$     $U$  est annulée

$T$  veut écrire, mais une valeur a déjà été écrite par une transaction plus tardive  $U$ . On peut très bien ne pas faire cette écriture. (En effet, si une transaction lit cette valeur alors qu'elle devrait lire celle écrite par  $T$ , il s'agit d'une erreur de lecture tardive.) Un problème peut néanmoins se poser si, par la suite,  $U$  est annulée:

$T$  débute    $U$  débute    $U$  écrit  $X$     $T$  écrit  $X$  (pas fait)    $T$  commise    $U$  est annulée

Une solution consiste à considérer que les écritures sont des tentatives qui peuvent être annulées si la transaction est annulée. Tant que la transaction n'est pas commise,  $c(X)$  vaut FAUX, et le contrôleur garde une copie de l'ancienne valeur de  $X$  et son  $WT(X)$  précédent.

## 4.2 Les règles autorisant une demande d'action dans un système par estampillage

Lorsqu'une demande d'écriture ou de lecture d'une transaction  $T$  parvient au contrôleur, trois réponses sont possibles:

- accorder la demande,
- annuler la transaction et la reprendre avec un numéro plus grand
- différer la transaction et décider plus tard si on l'annule ou on l'accorde.

### 1. La demande est $r_T(X)$

(a)  $\#T \geq WT(X)$  la lecture est possible

i. Si  $c(X) = VRAI$  l'autorisation de lecture est accordée.

A. Si  $\#T > RT(X)$  alors on modifie  $RT(X) := \#T$

B. Si  $\#T \leq RT(X)$  on ne modifie pas  $RT(X)$ .

ii. Si  $c(X) = FAUX$  on attend que  $c(X)$  devienne vrai ou que la transaction de numéro  $WT(X)$  soit annulée. (waitr)

iii. Si  $\#T < WT(X)$  alors  $T$  ne peut pas lire  $X$  et  $T$  est annulée. (ar)

### 2. La demande est $w_T(X)$

(a) Si  $\#T \geq RT(X)$  et  $\#T \geq WT(X)$  l'écriture est possible et est lancée. On indique que  $T$  est la dernière transaction à avoir demandé d'écrire  $X$  par  $WT(X) := \#T$  et on prend en compte que cette valeur n'est pas encore reportée sur le disque par  $c(X) := FAUX$

(b) Si  $WT(X) > \#T \geq RT(X)$  L'écriture est possible (du point de vue des lectures) mais une valeur plus tardive de  $X$  est déjà présente!

i. Si  $c(X) = VRAI$ , on ignore la demande d'écriture de  $T$ , et  $T$  poursuit ses demandes.

ii. En revanche, si  $c(X) = FAUX$  on attend comme au cas (waitr).

(c)  $\#T < RT(X)$  Dans ce cas l'écriture est impossible,  $T$  doit être annulée. (aw)

3. La demande est  $commit_T$ : tous les éléments écrits par  $T$  (le contrôleur en possède la liste) doivent être reportés sur le disque et les  $c(X)$  correspondant doivent être mis à VRAI. Si des transactions attendaient  $c(X)$  VRAI pour poursuivre (leur liste est également maintenue par le contrôleur), elles peuvent maintenant poursuivre.

4. La demande est  $roll\_back_T$ , parce qu'il s'agit de l'annulation de  $T$  demandée par le contrôleur en (ar) ou (aw) ou pour une autre raison: toute demande d'écriture ou de lecture de toute transaction qui attend un  $c(X)$  pour l'un des éléments écrits par  $T$  doit être relancée.

## 4.3 Une variante: l'estampillage multiple

L'estampillage multiple permet de dupliquer les éléments pour ne pas empêcher d'écriture des copies. Ces copies sont indexées par le numéro de la transaction  $U$  qui les a créés  $X_{\#U}$ . Bien évidemment lorsqu'une transaction de numéro

$\#T$  souhaite lire  $X$  elle doit lire la version  $X_i$  de  $X$  dont l'indice  $i$  est le plus grand parmi ceux qui sont plus petits que  $\#T$ . La dernière lecture devient dépendante de la version de  $X$ : chaque  $X_i$  a son  $RT(X_i)$ .

Une demande d'écriture de  $T$  est rejetée et  $T$  est annulée lorsque  $\#T$  est moindre que le  $RT(X_i)$  ou  $X_i$  est la version de  $X$  dont l'indice  $i$  est le plus grand possible avec  $i \leq \#T$ : dans ce cas la transaction de numéro  $RT(X_i)$  aurait dû lire la valeur écrite par  $T$ . Sinon, la demande d'écriture est accordée, et on crée  $X_{\#T}$ .

Une demande de lecture  $r_T(X)$  est toujours accordée et la valeur lue est la copie  $X_i$  de  $X$  dont l'indice est le plus grand possible avec  $i \leq \#T$ . On modifie alors l'indice de la dernière lecture de  $X_i$ , s'il est plus petit que  $\#T$  par  $RT(X_i) := \#T$ .

On peut détruire les copies de  $X_i$  d'indice inférieur à  $t$  si toutes les transactions actives ont des numéros plus grands que  $t$ .

## 5 Une solution optimiste: la validation

La validation est une autre technique optimiste qui garantit la sérialisabilité d'une suite d'exécution, l'ordre sériel équivalent entre les transactions étant l'ordre des moments où les transactions valident leur exécution. On suppose connu pour chaque transaction  $T$  les ensembles suivants:

- $RS(T)$  l'ensemble des éléments lus par  $T$ .
- $WS(T)$  l'ensemble des éléments sur lesquels  $T$  écrit.

Chaque transaction  $T$  procède en trois phases:

1. Elle lit toutes les éléments dont elle a besoin, c'est-à-dire ceux de  $RS(T)$ .
2. Le contrôleur valide la transaction en comparant l'ensemble des éléments qu'elle lit et écrit avec ceux des autres transactions (voir la procédure ci-après).
3. La transaction écrit sur la base les éléments qu'elle modifie, c'est-à-dire ceux de  $WS(T)$ .

Le contrôleur général connaît:

- $START$  les transactions qui ont commencé mais pas validé.
- $VAL$  Les transaction qui ont validé mais pas encore reporté sur disques les changements, et pour chacune d'entre elles, l'instant où elle a débuté  $START(T)$  et l'instant où elle a validé  $VAL(T)$ .
- $FIN$  les transaction qui ont complété la phase d'écriture, et pour chacune d'entre elles, les instants  $START(T)$ ,  $VAL(T)$  et  $FIN(T)$  où elles ont respectivement débuté, validé, et écrit. Cet ensemble s'agrandit, mais comme on le verra, si  $FIN(T) < START(U)$  pour toute transaction dans  $START$  ou  $VAL$ , on peut supprimer  $T$  et les informations qui lui sont attenantes.

## 5.1 Les problèmes à éviter pour être équivalent par permutation à l'ordre des validations

### 5.1.1 Ecriture tardive par rapport à une autre lecture

Une transaction  $T$  ne peut valider à  $t$  si une transaction  $U$  ayant précédemment validé écrit peut-être à  $t$  une valeur de  $X$  que  $T$  devrait avoir lu pour  $X$ .

$U$  débute    $T$  débute    $T$  lit  $X$     $U$  valide    $U$  écrit  $X$     $T$  valide

Ce problème peut arriver lorsque:

- $U \in (VAL \cup FIN)$  c'est-à-dire  $U$  a validé
- $FIN(U) > START(T)$  c'est-à-dire  $U$  n'a pas fini avant que  $T$  ne commence.
- $RS(T) \cap WS(U) \neq \emptyset$  c'est-à-dire qu'un même élément est à la fois écrit par  $U$  et lu par  $T$ .

### 5.1.2 Ecriture tardive par rapport à une autre écriture

Une transaction  $T$  ne peut valider à  $t$  si une transaction  $U$  ayant précédemment validé écrit peut-être après que  $T$  ait écrit  $t$  une valeur sur  $X$  – la valeur de  $X$  pour être cohérente avec l'ordre des validations devrait être celle de  $T$ .

$U$  valide    $T$  valide    $T$  écrit  $X$     $U$  écrit  $X$     $U$  finit

Ce problème peut arriver lorsque:

- $U \in VAL$  c'est-à-dire  $U$  a validé
- $FIN(U) > VAL(T)$  c'est-à-dire  $U$  n'a pas fini avant que  $T$  ne cherche à valider.
- $WS(T) \cap WS(U) \neq \emptyset$  c'est-à-dire qu'un même élément est à la fois écrit par  $U$  et par  $T$ .

## 5.2 La procédure de validation

Cette procédure découle des deux cas mentionnés ci-dessus, les deux seuls à poser problème.

Pour que  $T$  soit autorisée à valider il faut que, pour toute autre transaction  $U$  ayant validé:

- Si  $FIN(U) > START(T)$  alors il faut que  $RS(T) \cap WS(U) = \emptyset$
- Si  $FIN(U) > VAL(T)$  alors il faut que  $WS(T) \cap WS(U) = \emptyset$