Transactions et Contrôle de Concurrence

Talel.Abdessalem

Transactions

- L'exécution concurrente des programmes des utilisateurs est essentielle dans un SGBD.
- Les programmes des utilisateurs peuvent contenir plusieurs opérations sur les données obtenues de la BD, mais l'SGBD n'est concerné que par les opérations de lecture/écriture vers/de la base.
- Une *transaction* correspond à une vision d'un programme d'utilisateur du coté du SGBD : une séquence de lectures écritures.

Exemple

- Considérons deux transactions:

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Intuitivement, la première transaction fait un transfert de 100e du compte B vers A. La seconde crédite les deux comptes de 6% d 'intérêts.
- Il n'y a aucune garantie que T1 soit réalisée avant T2 et vice-versa, si elles sont soumises en même temps. L'effet *doit* être équivalent à une exécution en série de ces deux transactions, quelque soit l'ordre.

Exemple (suite)

- Considérons l'ordonnancement suivant:

```
T1:   A=A+100,              B=B-100
T2:             A=1.06*A,             B=1.06*B
```

- Que dire de :

```
T1:   A=A+100,                        B=B-100
T2:             A=1.06*A, B=1.06*B
```

- La vision SGBD du second ordonnancement :

```
T1:   R(A), W(A),                             R(B), W(B)
T2:               R(A), W(A), R(B), W(B)
```

Ordonnancement des Transactions

- *Exécution en série:* une transaction après l'autre.
- *Exécutions équivalentes :* Quelque soit la BD, l'effet de la première exécution est identique à l'effet de la seconde exécution (moyen de vérification : ordre des lectures et écritures conflictuels).
- *Exécution sérialisable:* Equivalente à une exécution en série.

(Rque: Si chaque transaction préserve la cohérence, toute exécution en série préserve la cohérence.)

Anomalies

- Lecture des données non validées (commit non effectué), "dirty reads"):

T1:	R(A), W(A),		R(B), W(B), Abort
T2:		R(A), W(A), Commit	

- Ré-écriture sur une valeur non validée (Uncommited Data):

T1:	W(A),		W(B), C
T2:		W(A), W(B), C	

Propriétés

- Transaction
  - Atomicité, Cohérence, Isolation, Durabilité
- Exécution
  - Recouvrabilité : Possibilité d'annuler l'effet d'une transaction qui abandonne (*abort*).
    - Solution : Ordre des *Commit* effectués par les transactions suit l'ordre de dépendances Lecture(X)/Ecriture(X).
  - Sans Abandons en Cascade (*Cascadeless*) : la lecture d'une valeur écrite par une transaction T ne peut se faire qu'une fois T a réalisé sont *Commit*.
    - Cascadeless ----> Recouvrable
  - Strict : l'écriture d'une valeur déjà affectée par une autre transaction T ne peut se faire qu'une fois T a réalisé sont *Commit*.
    - Solution : Cascadeless + retarder les Write(X) jusqu'à ce que les écritures effectuées par d'autres transactions sur X soient validées (commit).

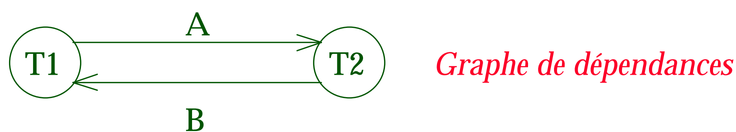
Exécution sérialisable

- Deux exécutions sont équivalentes si:
  - Les mêmes actions (lecture, écriture) se retrouvent dans les mêmes transactions
  - Chaque paire de d'actions conflictuelle (lecture/écriture, écriture/lecture ou écriture/écriture) sont ordonnées de la même façon dans les deux exécutions
- Une exécution S est sérialisable si S est équivalente à une exécution en série des mêmes transactions

Exemple

- Une exécution non sérialisable:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	



- Le cycle dans le graphe révèle le problème. L'effet de T1 dépend de celui de T2, et vice-versa.

Graphe de Dépendance

- *Graphe dépendances* : Un noeud par transaction; liens de *Ti* à *Tj* si *Tj* effectue une lecture/écriture d'un granule précédemment écrit par *Ti*, ou si *Tj* effectue une écriture d'un granule précédemment lut par *Ti*.
- Théorème: Une exécution est sérialisable ssi son graphe de précédence ne comporte pas de cycle

Protocole de verrouillage en 2 phases (2PL)

- *Protocole 2PL*:
  - Chaque transaction doit obtenir un verrou partagé S (*shared*) sur un granule avant de le lire, et un verrou exclusif X (*exclusive*) sur un granule avant d'effectuer une écriture dessus.
  - Tous les verrou émis par une transaction sont libérés à sa terminaison.
  - Si une transaction émet un verrou X sur un granule, aucune transaction ne peut obtenir un verrou (S ou X) sur le même granule.
  - Une transaction ne peut émettre de verrou dès qu'elle commence à libérer ses verrous.

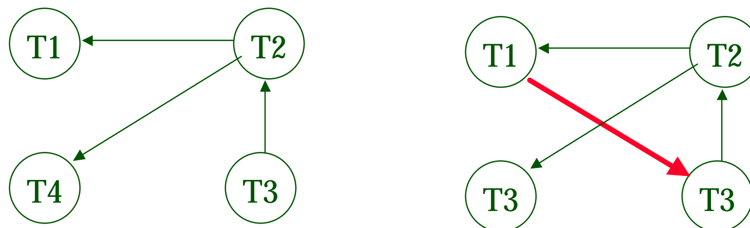
Gestion des verrous

- Les demandes de verrouillage et de déverrouillage sont gérées par le gestionnaire de verrouillage
- Entrées de la table des verrous (pour 1 granule):
  - Nombre de transactions ayant un verrou
  - Type de verrou (S ou X)
  - Pointeur vers la queue de la file des demandes de verrous
- verrouillage et déverrouillage sont des opérations atomiques
- *upgrade d'un verrou*: une transaction qui détient un verrou partagé peut demander à le transformer en verrou exclusif

Deadlocks

Exemple:

T1:	R(A),	R(B)				
T2:		W(B)		W(C)		
T3:			R(C)			W(A)
T4:					R(B)	



Deadlocks

- Deadlock: Des transactions en attente mutuelle.
- Deux manières de gérer les deadlocks:
  - prévention des Deadlocks
  - détection des Deadlocks

Prévention des Deadlocks

- Attribuer des priorités sous forme d'estampilles temporelles (timestamps). Supposons que Ti demande un verrou détenu par Tj. Deux cas de figure :
  - Wait-Die: si Ti a une priorité supérieure, Ti attend Tj; autrement Ti abandonne
  - Wound-wait: si Ti a une priorité supérieure, Tj abandonne; autrement Ti attend

Détection des Deadlocks

- Création d'un graphe d'attente waits-for graph:
  - Les nœuds représentent les transactions
  - Il y a un lien de Ti vers Tj si Ti attend que Tj libère un verrou
- Périodiquement vérifier s 'il y a un cycle dans le graphe waits-for

Estampillage

- **Idée:** attribuer à chaque élément un read-timestamp (RTS) et un write-timestamp (WTS), donner à chaque nouvelle transaction une estampille (TS):
  - Si l'action ai de la transaction Ti est en conflit avec l'action aj de Tj, et TS(Ti) < TS(Tj), alors ai doit être réalisée avant aj. Sinon, relancer la transaction.

T veut lire un granule O

- Si TS(T) < WTS(O),
  - T veut lire une valeur de O déjà recouverte.
  - Alors, la lecture est refusée et T abandonnée et relancée avec une nouvelle estampille.
- Si TS(T) >= WTS(O):
  - Permettre à T de lire O.
  - M-à-J de RTS(O) à max(RTS(O), TS(T))
- Le changement de RTS(O) doit être écrit sur disque!

T veut écrire un granule O

- Si TS(T) < RTS(O),
  - La valeur de O que T est en train de produire a été demandée antérieurement et supposée jamais produite.
    - L 'écriture est refusée et T abandonnée et relancée par la suite.
- SINON
  - Si TS(T) < WTS(O),
    - T tente d 'écrire une valeur périmée
      - L 'écriture est rejetée et T abandonnée.
      - Thomas-Write-Rule : ignore l'écriture.
  - Autrement,
    - L'écriture est acceptée et
    - M-à-J de WTS(O) à max(WTS(O), TS(T))

T1	T2
R(A)	
	W(A)
	Commit
W(A)	
Commit	

Résumé

- Protocole de verrouillage en deux phases (2PL).
- Le gestionnaire de verrous peut soit détecter, soit faire de la prévention des Deadlocks.
- Le fait de gérer soit même les verrous peut aboutir à des problèmes imprévisibles et difficiles à corriger.
- L'estampillage est une alternative au 2PL; permet des ordonnancements sérialisables que le 2PL ne permet pas (le contraire est vrai ?).