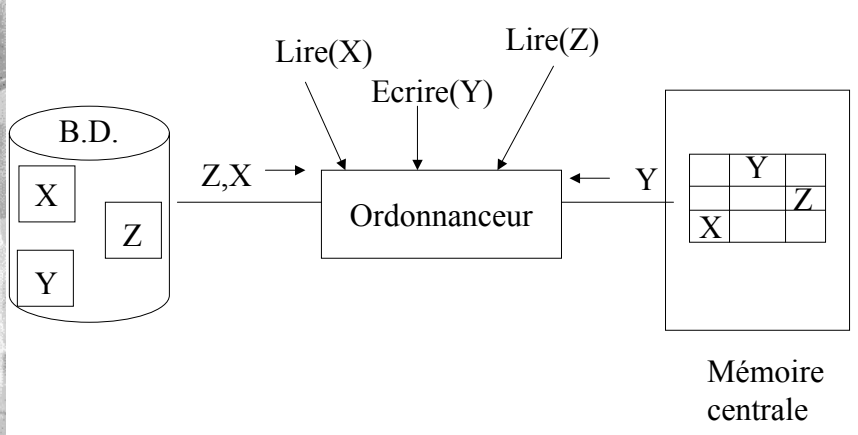
Contrôle de concurrence

J. Akoka - I. Wattiau

1. Introduction

- Charge d ’un SGBD : nb de transactions par seconde (tps)
- Objectif du contrôle de concurrence :
  - permettre la simultanéité des transactions
  - s ’assurer de la bonne exécution simultanée de plusieurs transactions
  - maximiser le nombre de transactions par seconde
  - minimiser le temps de réponse

1. Introduction (suite)



2. Anomalies de concurrence

- Perte de mise à jour

Transaction t1	Transaction t2
bot	
r1(x)	
x=x+1	
	bot
	r2(x)
	x=x+1
	w2(x)
	commit
w1(x)	
commit	

2. Anomalies de concurrence (2)

- Lecture erronée

Transaction t1	Transaction t2
bot	
r1(x)	
x=x+1	
w1(x)	
	bot
	r2(x)
	x=x+1
	w2(x)
	commit
abort	

2. Anomalies de concurrence (3)

- Observation d ’incohérences

Transaction t1	Transaction t2
bot	
r1(x)	
	bot
	r2(x)
	x=x+1
	w2(x)
	commit
r1(x)	
commit	

2. Anomalies de concurrence (4)

- Mise à jour fantôme (introduction d 'incohérences)
  - contrainte d 'intégrité x+y+z=1000

Transaction t1	Transaction t2
bot	
r1(x)	
	bot
	r2(y)
r1(y)	
	y=y-100
	r2(z)
	z=z+100
	w2(y)
	w2(z)
	commit
r1(z)	
s=x+y+z	
commit	

3. Théorie du contrôle de concurrence

- Modèle de transaction :
  - T1 : r1(x)r1(y)w1(x)w1(y)
- Modèle d 'exécution
  - S1 : r1(x)r2(z)w1(x)w2(z)…
- Restrictions :
  - on suppose que toutes les transactions sont validées
  - chaque transaction lit ou écrit un même objet au plus une fois
- Exécution en série :
  - S2 : r0(x)r0(y)w0(x)r1(y)r1(x)w1(y)r2(x)r2(y)r2(z)w2(z)

3. Théorie du contrôle de concurrence

- Une exécution est correcte si elle produit le même résultat que l 'exécution en série correspondante :
  - on dit qu 'elle est sérialisable
- Problème : définition du « même résultat »
- Différentes techniques :
  - équivalence de vue
  - équivalence de conflit
  - verrouillage deux phases
  - estampillage
- Plus une technique reconnaît d 'exécutions équivalentes, plus elle est coûteuse

4. Equivalence de vue

- Une lecture $r_i(x)$ est « suite » d'une écriture $w_j(x)$ dans une exécution si $w_j(x)$ précède $r_i(x)$ et s'il n 'y a pas d'autre écriture $w_k(x)$ entre ces deux opérations
- Une écriture $w_i(x)$ est finale dans une exécution si elle est la dernière écriture de x dans cette exécution
- 2 exécutions sont vue-équivalentes si elles ont les mêmes lectures « suites » et écritures finales
- Une exécution est vue-sérialisable si elle est vue-équivalente à une exécution en série

Equivalence de vue - exemple

S3 : w0(x)r2(x)r1(x)w2(x)w2(z)	vue-équivalente avec S4
S4 : w0(x)r1(x)r2(x)w2(x)w2(z)	exécution série
S5 : w0(x)r1(x)w1(x)r2(x)w1(z)	vue-équivalente avec S6
S6: w0(x)r1(x)w1(x)w1(z)r2(x)	exécution série
S7: r1(x)r2(x)w2(x)w1(x)	perte de mise à jour
S8: r1(x)r2(x)w2(x)r1(x)	lecture incohérente
S9: r1(x)r1(y)r2(z)r2(y)w2(y)w2(z)r1(z)	mise à jour fantôme

- Algorithme de vérification de l 'équivalence de vue complexe (NP-complet)

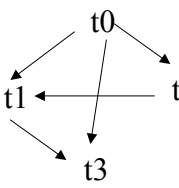
5. Equivalence de conflit

- Une action a_i est en conflit avec a_j si :
  - 1) elles opèrent sur un même objet,
  - 2) l 'une des deux au moins est une écriture
- 2 types de conflits : lecture-écriture et écriture-écriture
- 2 exécutions sont conflit-équivalentes si :
  - elles possèdent les mêmes opérations
  - chaque paire d 'opérations en conflit est dans le même ordre dans les 2 exécutions
- 1 exécution est conflit-sérialisable si elle est conflit-équivalente à une exécution série
- Condition plus restrictive que l 'équivalence de vues

5. Equivalence de conflit

- Graphe de vérification de la conflit-sérialisabilité

S10 : w0(x)r1(x)w0(z)r1(z)r2(x)r3(z)w3(z)w1(x)



Un arc de ti vers tj s 'il y a au moins un conflit entre une action ai et une action aj t.q. ai précède aj

- L 'exécution est conflit-sérialisable si le graphe est acyclique

S11 : w0(x)w0(z)r2(x) r1(x) r1(z)w1(x)r3(z)w3(z) exécution en série conflit-équivalente avec S10

- Plus efficace que la vue-équivalence mais encore trop complexe

6. Verrouillage deux-phases

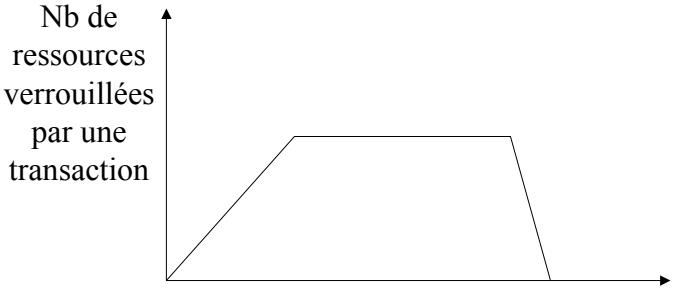
- Utilisé par presque tous les SGBD commerciaux
- Principe :
  - la base de données est découpée en granules
  - chaque granule est géré au moyen de verrous
  - toutes les opérations de lecture et écriture doivent être protégées par l'exécution de primitives de verrouillage : r_lock, w_lock et unlock
  - toute opération de lecture est précédée par un r_lock et suivie par unlock. C 'est un verrou partagé (il peut y en avoir plusieurs au même moment sur le même objet)
  - toute opération d 'écriture est précédée par un w_lock et suivie par unlock. C'est un verrou exclusif.
  - Le gestionnaire de verrou :
    - reçoit les requêtes lock, accorde ou non le verrou, relâche les verrous à unlock et fait attendre les transactions qui posent pb

Table de conflit pour le verrouillage

Requête	Ressource libre	Ressource verrouillée en lecture	Ressource verrouillée en écriture
r_lock	OK Verrouillée en lecture	OK Verrouillée en lecture	Non Verrouillée en écriture
w_lock	OK Verrouillée en écriture	Non Verrouillée en lecture	Non Verrouillée en écriture
unlock	Erreur	OK ?	OK déverrouillée

Verrouillage 2 phases

- Une transaction, après avoir relâché un verrou, ne peut pas obtenir d'autres verrouillages (2PL)
- Permet d'assurer que les transactions sont sérialisables



Verrouillage 2 phases

- Toute exécution qui vérifie 2PL est conflit-sérialisable
- L 'inverse n 'est pas vrai :

  S12 : r1(x) w1(x) r2(x) w2(x) r3(y) w1(y)

  S12 n 'est pas 2PL car :

  t1 relâche un verrou exclusif sur x

  puis demande un verrou sur y

  Pourtant S12 est conflit-sérialisable avec

  S13: r3(y) r1(x) w1(x) w1(y) r2(x) w2(x)

- Verrouillage 2 phases strict : les verrous sur une transaction ne peuvent être relâchés qu 'après l 'ordre commit ou abort

- Le verrouillage 2P résout le problème des m.a.j. fantômes

Transaction 1	Transaction 2	Ressource x	Ressource y	Ressource z
bot		libre	libre	libre
r_lock1(x)		1:read		
r1(x)				
	bot			
	w_lock2(y)		2:write	
	r2(y)			
r_lock1(y)			1:attente	
	y=y-100			
	w_lock2(z)			2:write
	r2(z)			
	z=z+100			
	w2(y)			
	w2(z)			
	commit			
	unlock2(y)		1:read	
r1(y)				
r_lock1(z)				1:attente
	unlock2(z)			1:read
r1(z)				
	eot			
s=x+y+z				
commit				
unlock1(x)		libre		
unlock1(y)			libre	
unlock1(z)				libre
eot				

Problème d ’interblocage

- Principe : T1 attend T2 et T2 attend T1
- 2 techniques :
  - délai d ’attente
  - prévention de l ’interblocage :
    - estampiller les transactions
    - ne faire attendre que s ’il y a une relation de précédence entre les deux transactions (une transaction ne peut attendre une transaction plus jeune)

Problème de famine

- Une transaction est perpétuellement en attente
- exemple :
  - un granule est verrouillé en mode partagé par une transaction T1
  - T2 désire modifier => mise en attente
  - T3 et suivantes veulent un verrou partagé pour exécuter des opérations de lecture => T2 est en situation de famine
- une solution possible : délai d ’attente max

7. Méthode d 'estampillage

- Facile à gérer, moins efficace que le verrouillage 2 phases
- estampille : identifiant qui définit un ordre total des événements à l 'intérieur d 'un système
- Toute transaction reçoit une estampille qui représente le moment où elle commence
- Une exécution est acceptée ssi elle reflète l'ordre en série des transactions basé sur l 'estampille de chaque transaction

7. Méthode d 'estampillage

- Chaque objet x a 2 estampilles RTM(x) et WTM(x), estampilles des dernières transactions ayant accédé à x
- L 'ordonnanceur :
  - reçoit des opérations du types read(x,ts) ou write(x,ts) où ts est l 'estampille de la transaction qui demande

```
Procédure read(x,ts)
Si      ts < WTM(x)
alors   abort
sinon   effectuer la lecture
        RTM(x):=max(ts,RTM(x))
Finsi
```

```
Procédure write(x,ts)
Si      ts < WTM(x)
        ou ts < RTM(x)
alors   abort
sinon   effectuer la lecture
        WTM(x):=ts
Finsi
```

7. Méthode d 'estampillage

Requête	Réponse	Nouvelles valeurs
read(x,6)	OK	
read(x,8)	OK	RTM(x)=8
read(x,9)	OK	RTM(x)=9
write(x,8)	NON	t8 défaite
write(x,11)	OK	WTM(x)=11
read(x,10)	NON	t10 défaite

8. Application à Oracle

- Pour éviter les lectures erronées :
  - mécanisme de versions de données fondé sur un numéro de modification (System Change Number)
  - une requête ne voit que les seules données validées avant le début de son exécution (i.e. dont le SCN est < à celui de la requête) et ses propres modifications

8. Application à Oracle

- Verrouillage :
  - un granule = un tuple ou une table
  - verrous partagés ou exclusifs
  - implicitement : verrouillage niveau tuple
  - on peut placer explicitement des verrous :
    - au niveau table (lock table)
    - pour la durée d 'une session (alter session set isolation_level)
  - déverrouillage implicite en fin de transaction
  - détecte l'interblocage et envoie un message à la transaction : on peut défaire toute la transaction ou différer l'instruction qui pose problème