Transaction and concurrence

Serge Abiteboul

INRIA

April 3, 2009

Contexte

- Plusieurs programmes/utilisateurs manipulent la base de données en même temps,
- Des pannes peuvent surgir
- Ce qu'on veut :

1. Faire cohabiter les différents utilisateurs:
   - La base de données doit paraître cohérente pour chaque utilisateur,
   - les contraintes doivent être vérifiées (Σ débits = Σ crédits).
2. Garantir un comportement cohérent en cas de pannes

Notions importantes

- transaction
- sériabilité

Plan

1. Les transactions
2. Exemples
3. Ordonnancement sériable
4. Verrouillage à deux phases 2PL
5. 2 autres techniques
   - Estampillage
   - Multi-versions
6. Conclusion

Une transaction

- Un seul utilisateur
- contrainte : Σ places vendues = Σ places réservées
- Programme 1 : réserver une place pour utilisateur u
  1. Lire table RESA
  2. Trouve place libre: x
  3. La marquer comme réservée
  4. Écrire RESA

     — état incohérent —
  5. Lire table USER
  6. Écrire x dans la ligne de u

Transaction

- Unité atomique d'exécution
  - Début transaction ... Opérations ... Fin transaction
- Séquence de mises à jour prenant la base dans un état cohérent et la laissant dans un état cohérent (en passant éventuellement par des états incohérent)
- Toute la transaction est réalisée ou rien ne l'est:
  - Validation : toute la transaction est prise en compte
  - Avortement : la transaction n'a aucun effet

En cas de panne ou autre problème:

- Une transaction déjà validée doit être prise en compte
- Le système peut avorter (ou non) une transaction pas encore validée
  - il a le droit de le faire

Vérifications possibles

- à la compilation (indépendant de l'état de la base)
- à l'exécution
  - À priori : pré-conditions minimales
  - À posteriori : la base est modifiée, retour en arrière si nécessaire
    – le plus utilisé

Problèmes générés

1. Retour en arrière : comment “défaire” en cas d’avortement ?
2. Pannes : comment ramener la base dans un état cohérent ?
3. Concurrence : exemple suivant

Exemple 2 : deux utilisateurs

- $P1$: virement$(B \rightarrow A, 100)$
- $P2$: virement$(B \rightarrow A, 200)$
- ordonnancement

$P1$	Lire A	
$P2$	Lire A	
$P1$	Écrire $A + 100$	perdu !
$P2$	Écrire $A + 200$	
$P1$	Lire B	
$P1$	Écrire $B - 100$	
$P2$	Lire B	
$P2$	Écrire $B - 200$	

- état incohérent: Σ débits $\neq \Sigma$ crédits!

Ce que nous allons faire

1. Modèle simple de transaction
2. Définir un ordonnancement correct
3. Trouver des techniques pour n'accepter que des ordonnancements corrects
   - On programme ses transactions
   - On laisse ensuite le système gérer la concurrence

Un modèle de transaction

Trouver le bon niveau d'abstraction pour résoudre le problème

- Le gestionnaire de transactions ne considère dans le programme de chaque transaction que les entité et les ordres de lectures/écritures sur ces entités. Le reste est ignoré.
- Base = $\{E_1, ..., E_n\}$ ensemble fixe d'entités (enregistrements, nuplets, relations, pages)
- Ensemble de transactions: $\{T_1, ..., T_n\}$
- Transaction T_i : séquence d'instructions de la forme (T_i, lire, E_i) ou (T_i, écrire, E_i)

Vers des modèles plus complexes

- Le système ne dispose que d'une partie des transactions
- Les entités ne sont pas indépendantes (un objet est dans une page)
- Distribution: transactions s'exécutant sur plusieurs machines avec chacune des entités
- Ici on se simplifie la vie - modèle simple

Ordonnancement

- Séquence d'instructions (T_i, lire/écrire, E_j)
  dont la projection sur chaque T_i est exactement T_i
- Exemple
  T_1 : (T_1,lire,A) (T_1,écrire,A)
  T_2 : (T_2,lire,A) (T_2,lire,B) (T_2,écrire,A) (T_2,écrire,B)
- Ordonnancement
  (T_1,l,A) (T_2,l,A) (T_1,e,A) (T_2,l,B) (T_2,e,A) (T_2,e,B)

Ordonnancement sériel et sériable

- Parallélisme minimum → Ordo sériel : chaque T_i s'exécute complètement avant qu'une autre débute
- Un ordo sériel est correct (pas de perte de cohérence).
- Plus de parallélisme: un ordo est dit sériable s'il est équivalent à un ordonnancement sériel.
- Définition intuitive: Deux ordonnancements sont équivalents s'ils ont le même effet sur la base.

Exemple 3

T_1 : (T_1,lire,X) (T_1,écrire,X)
T_2 : (T_2,lire,X) (T_2,écrire,X)
O : (T_1,l,X) (T_2,l,X) (T_2,e,X) (T_1,e,X)

- O non sériable
- Le travail de T_2 n'est pas pris en compte: Imaginez que T_1 ne modifie pas X et que T_2 double X. L'effet total d'un ordonnancement sériel ($T_1 T_2$ ou $T_2 T_1$) est de doubler X alors que O le laisse inchangé.

Exemple 4

T_1 : (T_1,l,X) (T_1,e,Y) (T_1,l,X) (T_1,e,Z)
T_2 : (T_2,l,X) (T_2,e,X)
O' : (T_1,l,X) (T_1,e,Y) (T_2,l,X) (T_2,e,X) (T_1,l,X) (T_1,e,Z)

- O non sériable

- P_1 = { lire X; écrire $X \rightarrow Y$; lire X ; écrire $X \rightarrow Z$ }
  P_2 = { lire X ; écrire $2 * X \rightarrow X$ }
- On vérifie que après un ordonnancement sériel les contenus de Y et Z sont identiques, mais pas après O'.

Définition formelle de l'équivalence

- Opérations conflictuelles: des opérations de deux transactions sur la même entité si l'une est une écriture (LE, EL, EE)
- Deux ordonnancements sont équivalents si pour toutes opérations conflictuelles p, q, si p est avant q dans l'un, il est avant q dans l'autre

Exemple 5

- T_1 : (T_1,l,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y)
  T_2 : (T_2,l,X) (T_2,e,X)
  O_1 : (T_1,l,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y) (T_2,l,X) (T_2,e,X)
  O_2 : (T_1,l,X) (T_1,e,X) (T_2,l,X) (T_2,e,X) (T_1,l,Y) (T_1,e,Y)
  O_3 : (T_1,l,X) (T_2,l,X) (T_2,e,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y)

- On vérifiera que
  - O_1 est sériel
  - O_2 est sériable
  - O_3 n'est pas sériable

Graphe de dépendance

Graphe de dépendance $SG(O)$
Pour représenter les conflits

1. les nœuds du graphes sont les transactions de O
2. arc de T_i à T_j s'il existe deux opérations conflictuelles p=(T_i,...) et q=(T_j,...) et p arrive avant q.

Intuition: L'ordonnancement force que p soit avant q, donc que dans un ordonnancement sériel équivalent, T_i soit avant T_j

Exemples

T_1 : (T_1,l,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y)
T_2 : (T_2,l,X) (T_2,e,X)
O_1 : (T_1,l,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y) (T_2,l,X) (T_2,e,X)
O_2 : (T_1,l,X) (T_1,e,X) (T_2,l,X) (T_2,e,X) (T_1,l,Y) (T_1,e,Y)
O_3 : (T_1,l,X) (T_2,l,X) (T_2,e,X) (T_1,e,X) (T_1,l,Y) (T_1,e,Y)

- $SG(O_1)$ est $T_1 \longrightarrow T_2$
- $SG(O_2)$ idem
- $SG(O_3)$ est $T_1 \longleftrightarrow T_2$

Théorème de sériabilité

Théorème Un ordonnancement O est sériable ssi $SG(O)$ est acyclique

- Si le graphe acyclique, on en déduit un ordre $T_{i_1} < ... < T_{i_n}$ qui donne un ordonnancement sériel équivalent à l'ordonnancement $T_{i_1} \dots T_{i_n}$
- Si le graphe est cyclique, supposons que qu'il y ait un ordonnancement sériel équivalent. Prenons le cycle T_1, T_2..., T_i, on en déduit que dans cet ordonnancement sériel, T_1 est avant T_2... est avant T_1 - une contradiction

Concurrence

Solution 1: Construire le graphe SG(O) et détecter les cycles

Trop coûteux $\Rightarrow$ 2PL

Verrouillage à deux phases : 2PL

Technique utilisée : pose de verrous en lecture ou en écriture.

- On ne peut Lire/Écrire que sur une entité verrouillée (avec le bon verrou)
- Deux verrous conflictuels ne peuvent pas être accordés à la fois (L/É ou É/É sur la même entité)
- Déverrouille le tout avant de terminer

Verrouillage à 2 Phases

Loi 2PL: Quand une transaction a déverrouillé une fois, elle n'a plus le droit de poser de verrou

(A, Ver) (B, Ver) ← Phase 1 →	(A, Dever) (B, Dever) ← Phase 2 →	2 Phases
(A, Ver) (A, Dever)	(B, Ver) (B, Dever)	Pas 2 Phases

Opérations: { écriture, lecture, verrouillage, déverrouillage }
Transaction: séquence de ces opérations

Le graphe devient

- Un noeud par transaction
- arcs: (T, Vl, a) ... (T', Ve, a) $\leadsto$ T $\rightarrow$ T'
  (T, Ve, a) ... (T', Ve, a) $\leadsto$ T $\rightarrow$ T'
  (T, Ve, a) ... (T', Vl, a) $\leadsto$ T $\rightarrow$ T'

2PL

- Théorème: Un ordonnancement avec verrou est sériable ssi le graphe n'a pas de cycle
- Théorème 2PL Tout ordo 2PL est sériable
- Preuve: les 2 phases garantissent l'absence de cycle.
  Par contradiction: supposons que O est 2PL mais pas sériable.
  Cela implique qu'il y ait un cycle $T_1 T_2, ..., T_n, T_1$
  - de l'arc (T_1, T_2) on obtient un temps où T_1 a déverrouillé
  - de (T_n, T_1), on obtient un temps plus tard où T_1 a verrouillé
  - donc O n'est pas 2PL - une contradiction

Exemple: ordo sériable pas à deux phases

T_1:	(Ver, A) (Dever, A)		(Ver, C) (Dever, C)
T_2:		(Ver, B) (Dever, B)	

2PL impose trop de contraintes & c'est le prix de la facilité

Deadlocks

- Exemple
  1. T_1 pose un verrou sur E
  2. T_2 sur E'
  3. T_1 demande E'
  4. T_2 demande E
- Détecter les deadlocks: détection de cycle dans un graphe des attentes
  - Beaucoup plus petit que le graphe de dépendance
  - Possible aussi d'utiliser des *timeout*
- Quoi faire: avorter une des transactions qui cause le cycle
- Comment choisir qui avorter: de préférence pas
  - les transactions proches de terminer
  - celles qui ont déjà beaucoup travaillé
  - toujours les mêmes
    - éviter les famines – transaction perpétuellement avortée

Avortement en cascade

- Avortement de T_1
- Si T_2 a lu une valeur écrite par T_1, il faut avorter T_2
- Etc.: Si T_3 a lu une valeur écrite par T_2...
- Problème: on ne peut valider T_2 avant T_1
- Solution: quand une transaction se met à déverrouiller, il faut qu'elle finisse le plus vite possible pour éviter cela
- Les avortements en cascade arrivent dans d'autres contextes aussi

2PL en pratique

- Ordonnanceur agressif: on les met dans la queue au risque d'avoir des conflits plus tard et devoir avorter
  plutôt que frileux: attend et réarrange les transactions
- Verrous de lecture/écriture demandés implicitement avant lecture/écriture
- Déverrouillage en fin de transaction (difficile de prédire quand on n'aura plus besoin de nouveaux verrous)

Autre méthode: l'estampillage

- Chaque transaction T reçoit à sa création une estampille (son heure de début) e_T
- Quand une transaction T veut une entité (lire ou écrire)
  1. elle vérifie si l'entité porte une estampille plus ancienne que e_T (conflit !)
  2. elle marque l'entité avec sa propre estampille
- Théorème: L'estampillage garantit la sériabilité

Parfois plus de parallélisme

Avantage: autorise plus de parallélisme

T_1 (l,X) (é,X) (l,Y) (é,Y)
T_2 (l,X) (é,X) (l,Y) (é,Y)

Accepté par l'estampillage
Refusé par 2PL.

Désavantage: peut entrainer facilement des conflits artificiels (viennent du choix de l'estampille)

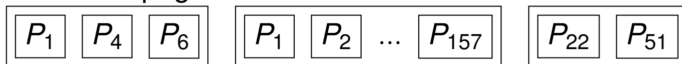
Technique optimiste qui marche bien quand il y a peu de conflits

Autre technique: Multi-versions

- En cas de mises-à-jour, on crée une nouvelle version
- quand quelqu'un veut lire une entité, on lui fait lire une entité dans la version avant mise-à-jour
- avantage: les mises-à-jour ne ralentissent pas les transactions qui ne font que des lectures
- on peut faire du 2PL ou de l'estampillage sur la version qui est mise-à-jour
- Régulièrement on bascule à une nouvelle version pour la lecture
- Intuition: Dans l'ordonnancement sériel équivalent, on regroupe toutes les transactions qui ne font que des lectures en gros paquets (qui correspondent à une version de lecture)
- Techniques sophistiquées pour éviter de dupliquer les données et pour jeter les versions devenues inutiles (GC)

Que trouve-t-on en pratique ?

- Transactions: début, avorter, valider
- Gestion de pages



Zone T_1 *MEMOIRE* *Zone T_2*

- Méthode des pages ombrées
  - Chaque transaction a un ensemble de pages modifiées
  - À la validation : pages copiées sur disque
  - panne mémoire centrale : repartir du disque
- Journal des mises-à-jour

Que trouve-t-on en pratique ? - fin

- 2PL + VL/VE
- Verrous de page
- Déverrouillage en fin de transaction
- Verrouillage plus sophistiqués
  - Verrous hiérarchiques jusqu'au nuplet
  - Grain + fin $\Rightarrow$ Gestion + lourde
  - Méthodes hybrides: e.g., verrous pages et passage au verrou objet en cas de conflits
- Lecture “sale” hors transaction
- BD distribuées
  - Technique peu utilisée : 2 phase commit
  - Très utilisée : réplication de données

Merci