

Gestion de l'accès concurrent

1

ADAPTATION DE

ELMASRI ET NAVATHE, 6TH ED.
CHAPITRE 22PAR
LUC LAVOIEUTILISÉ CONFORMÉMENT AUX CONDITIONS
ÉTABLIES PAR LES AUTEURS

2023-09-05

1

Plan

2

- Introduction
- Problématique
- Modèle
- Solutions par **verrouillage**
- Solutions par **estampillage**
- Synthèse
- Et les autres solutions ?
- Conclusion

2023-09-05

2

Introduction

3

2023-09-05

3

Objectif

4

- Garantir l'**i**solation des transactions sans violer l'**a**tomicit , la **c**oh rence ni la **d**urabilit 
- Corollaires
 - Assurer l'exclusion mutuelle des transactions conflictuelles
 - R soudre les conflits d'acc s (rw et ww)
- Exemple
 - Si deux transactions T1 et T2 sont en conflit relativement   l' l ment X, il faut d terminer laquelle des deux transactions acc de   X et si l'autre doit  tre annul e ou suspendue

2023-09-05

4

Rappels

5

- Concurrency : les processus se partagent un même processeur, leurs exécutions s'entrelacent.
- Parallélisme : les processus sont simultanément exécutés sur des processeurs différents.
- En général, les deux en même temps !

Note sur la concurrence et parallélisme

- En général, un modèle adéquat pour la concurrence l'est aussi pour le parallélisme, mais pas l'inverse.
- Les *Concurrent sequential processes* (CSP) de Hoare demeurent un des modèles les plus pertinents.

2023-09-05

5

Références

6

- ELMASRI, Ramez ; NAVATHE, Shamkant B.;
Fundamentals of database systems ;
Sixth edition, Pearson Addison Wesley, 2011.
ISBN 978-0-13-608620-8.
○ chapitre 22
- George Coulouris, Jean Dollimore, Tim Kindberg ;
Distributed systems : concepts and design ;
Forth edition, Addison-Wesley, 2005.
ISBN 0-321-26354-5.
○ pour un approfondissement, le livre au complet!

2023-09-05

6

Verrouillage

7

2023-09-05

7

Verrouillage (à la recherche de l'isolation)

8

- verrous binaires
- verrous partagés
 - non convertibles
 - convertibles

2023-09-05

8

Verrous binaires

9

Règles

1. verrouillé(x) préalable à lire(x) dans T
2. verrouillé(x) préalable à écrire(x) dans T
3. verrouillé(x) préalable à déverrouiller(x) dans T
4. \neg verrouillé(x) préalable à verrouiller(x) dans T
5. lire(x) et écrire(x) se terminent par déverrouiller(x)

2023-09-05

9

Verrous partagés non convertibles

10

Règles en situation de non-convertibilité

1. verrouiller_ m (x) avant lire(x) dans T,
en spécifiant $m \in \{\text{partagé, exclusif}\}$
2. verrouiller_exclusif(x) avant écrire(x) dans T
3. lire(x) et écrire(x) se terminent par déverrouiller(x)
4. déverrouiller(x) que si verrouillé(x) dans T
5. verrouiller_partagé(x) que si \neg verrouillé(x) dans T
6. verrouiller_exclusif(x) que si \neg verrouillé(x) dans T

2023-09-05

10

Verrous partagés convertibles

11

Règles en situation de convertibilité

- Assouplir les règles 5 et 6
 - 5bis verrouiller_partagé(x) que si \neg verrouillé(x) ou possède verrou exclusif dans T
 - 6bis verrouiller_exclusif(x) que si \neg verrouillé(x) ou possède le seul verrou partagé dans T
- Nécessite le maintien de la liste des détenteurs de verrous (en plus des verrous eux-mêmes).

2023-09-05

11

Interblocage (étreinte fatale – *dead lock*)

12

Définition

- lorsque chacune des transactions d'un ensemble de transactions attend la libération d'un élément verrouillé par une autre transaction du même ensemble

Exemple

T1	T2
v_par(y)	
	v_par(x)
v_ex(x)	
	v_ex(y)

2023-09-05

12

Verrouillage bi-phase

13

- **Modèle 2PL** : garantir a priori l'exécution en séparant les actions en deux phases :
 - acquisition
 - libération
- **Variantes**
 - basique : tel quel;
 - conservatrice : prédéclaration des verrous;
 - stricte : libération des seuls verrous partagés;
 - rigoureuse : libération des verrous à la fin seulement.
- **Limite**
 - certains programmes sérialisables sont refusés.

2023-09-05

13

Verrouillage bi-phase **conservateur**

14

- **Méthode**
 - prédéclaration des verrous
- **Comportement**
 - empêche d'entrer en transaction à moins d'obtenir tous les accès,
 - une transaction est exécutée en entier durant la phase de libération.
- **Corolaires**
 - risque de famine,
 - augmentation de l'écart-type des délais,
 - difficulté voire impossibilité de mise en oeuvre dans un contexte de SGBD.

2023-09-05

14

Verrouillage bi-phase **strict**

15

- Méthode
 - libération des seuls verrous partagés
- Comportement
 - ...
- Corolaires
 - ...

2023-09-05

15

Verrouillage bi-phase **rigoureux**

16

- Méthode
 - libération des verrous à la fin seulement
- Comportement
 - ...
- Corolaires
 - ...

2023-09-05

16

Techniques de gestion des transactions

17

- Prévention 2PL
 - strict, ok
 - autres \Rightarrow ordonnancement strict des attributs
 - non praticable dans un contexte de SGBD (calculabilité, tri des attributs)
- Prévention avec estampilles
 - BTO - WD (wait die)
 - STO - WW (wound wait)
- Prévention conservatrice
 - NW (no wait)
 - CW (cautious wait)
- Détection de l'interblocage
 - Maintien et analyse du graphe des attentes
 - Mise hors délai (solution pratique)
- Famine
 - Mise hors délai (problématique d'équité)

2023-09-05

17

Estampillage

18

2023-09-05

18

Principe

19

- Toutes les transactions sont estampillées en ordre strictement croissant
- Estampille de lecture d'un élément : la plus grande des estampilles des transactions courantes ayant effectivement lu l'élément
- Respectivement pour l'écriture

2023-09-05

19

Algorithme de base

BTO

20

- 1. La transaction T lance l'opération `write_item(X,v)` :
 - ✦ Si `read_TS(X) > TS(T)` ou `write_TS(X) > TS(T)` alors
 - une transaction concurrente a déjà lu ou modifié X, conséquemment : rejeter l'opération `write_item(X,v)` et annuler la transaction T.
 - ✦ Sinon
 - exécuter l'opération `write_item(X,v)` et affecter la valeur de `TS(T)` à l'estampille `write_TS(X)`.
- 2. La transaction T lance l'opération `read_item(X,v)` :
 - ✦ Si `write_TS(X) > TS(T)` alors
 - une transaction concurrente a déjà modifié X, conséquemment : rejeter l'opération `read_item(X,v)` et annuler la transaction T.
 - ✦ Sinon
 - exécuter `read_item(X,v)` et affecter la plus grande des valeurs parmi `TS(T)` et `read_TS(X)` à l'estampille `read_TS(X)`.

2023-09-05

20

Algorithme strict

STO

21

- La transaction T lance l'opération `write_item(X,v)` :
 - ✦ Si $TS(T) > read_TS(X)$ alors
 - reporter T jusqu'à ce que la transaction T' ayant lu ou modifié X soit terminée (complétée ou annulée)
 - ✦ sinon
 - procéder.
- La transaction T lance l'opération `read_item(X,v)` :
 - ✦ Si $TS(T) > write_TS(X)$ alors
 - reporter T jusqu'à ce que la transaction T' ayant lu ou modifié X soit terminée (complétée ou annulée)
 - ✦ sinon
 - procéder.

2023-09-05

21

Règle de Thomas

(TTO : une variante de STO)

22

La règle d'**écriture** est modifiée ainsi

- si $read_TS(X) > TS(T)$ alors
 - ✦ rejeter l'opération et annuler T.
- sinon
 - ✦ si $write_TS(X) > TS(T)$
 - */* Il s'agit d'une deuxième écriture annulant la première */*
 - ignorer l'opération et poursuivre l'exécution.
 - */* s'il devait en résulter un conflit, la règle précédente le détectera */*
 - ✦ sinon
 - */* $read_TS(X) \leq TS(T)$ et $write_TS(X) \leq TS(T)$ */*
 - exécuter `write_item(X)` et affecter la valeur `write_TS(X)` à `TS(T)`.

2023-09-05

22

Synthèse

23

	B2PL	C2PL	S2PL	R2PL	S2PLWD	S2PLWW	S2PLNW	S2PLCW	BTO	STO	TTO
récupérabilité (s-prog)	oui	oui	oui	oui	oui	oui	oui	oui	oui	oui	oui
c-ser garantie	oui	oui	oui	oui	oui	oui	oui	oui	oui	oui	non
c-ser complète	non	non	non	non	non	non	non	non	non	non	non
fermeture a priori	non	oui	non	non	non	non	non	non	non	non	non
annulation et reprise	oui+	oui	oui	oui+	oui	oui	oui	oui-	oui	oui	oui
annulation en cascade	non-	non	non-	non-	non-	non-	non-	non-	oui	oui	oui
interblocage	oui+	non	oui=	oui-	non-	non-	non	non	non	non	non
famine	oui+	non	oui=	oui-	non	non	non	non	non	non	non
facilité de meo	x=	x+	x=	x-	x-	x-	x-	x-	x++	x++	x+
applicable en pratique	non	non	oui	non	oui	oui	oui	oui	oui	oui	oui
contexte								t+f-	t-f+	t-f+	

2023-09-05

23

Et les autres solutions ?

24

- MVCC – en général, voir
 - https://en.wikipedia.org/wiki/Multiversion_concurrency_control
 - https://fr.wikipedia.org/wiki/Multiversion_Concurrency_Control (traduction perfectible de la référence précédente)
 - https://en.wikipedia.org/wiki/List_of_databases_using_MVCC
- MVCC et PostgreSQL, voir
 - <https://momjian.us/main/writings/pgsql/mvcc.pdf>

2023-09-05

24

Fin

25

2023-09-05