Bases de données

Algorithmes de sérialisabilité

Nadime Francis

Université Gustave Eiffel
LIGM - 4B130 Copernic
nadime.francis@univ-eiffel.fr

Transparents inspirés et adaptés du cours de C. Sirangelo à l'Université de Paris

Sérialisabilité

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1	Transaction 2	Exécution
read(A,x) x = x+1 **write**(A,x) **read**(A,x) x = x-1 **write**(A,x)	**read**(A,y) y = y∗2 **write**(A,y)	

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
                read(A,y)
                y = y*2
                write(A,y)
```

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
                    read(A,y)
                    y = y*2
                    write(A,y)
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
                    read(A,y)
read(A,x)
                    y = y*2
                    write(A,y)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
read(A,x)
x = x+1
write(A,x)
                    read(A,y)
read(A,x)
x = x-1
write(A,x)
                    y = y*2
                    write(A,y)
```

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
read(A,x)
x = x+1
write(A,x)
                    read(A,y)
                    y = y*2
                    write(A,y)
read(A,x)
x = x-1
write(A,x)
```

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
read(A,x)
x = x+1
write(A,x)
                    read(A,y)
                    y = y*2
                    write(A,y)
read(A,x)
x = x-1
write(A,x)
```

Un tel choix est appelé ordonnancement (schedule)

Ordonnancement

Lorsque deux transactions s'exécutent en parallèle, le système peut choisir à chaque étape à quelle transaction donner la priorité

Transaction 1

```
read(A,x)
x = x+1
write(A,x)
read(A,x)
x = x-1
write(A,x)
```

Transaction 2

```
read(A,y)
y = y*2
write(A,y)
```

Exécution

```
read(A,x)
x = x+1
write(A,x)
                    read(A,y)
                    y = y*2
                    write(A,y)
read(A,x)
x = x-1
write(A,x)
```

Un tel choix est appelé ordonnancement (schedule)

Objectif : choisir un ordonnancement qui garantit cohérence et isolation

Sérialisabilité

Rappel : les transactions sont supposées correctes

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Conséquence : les exécutions en série sont cohérentes par hypothèse

Sérialisabilité

Rappel : les transactions sont supposées correctes

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Conséquence : les exécutions en série sont cohérentes par hypothèse

Il n'est pas raisonnable de forcer des exécutions en série :

- Services multi-utilisateur : Amazon, AirBnB, Twitter...
- Transaction "en attente" qui ne fera jamais de commit

Sérialisabilité

Rappel : les transactions sont supposées correctes

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Conséquence : les exécutions en série sont cohérentes par hypothèse

Il n'est pas raisonnable de forcer des exécutions en série :

- Services multi-utilisateur : Amazon, AirBnB, Twitter...
- Transaction "en attente" qui ne fera jamais de commit

Sérialisabilité : un ordonnancement est dit sérialisable s'il a le même effet qu'une exécution en série, pour tout état de départ de la base de données

Sérialisabilité

Rappel : les transactions sont supposées correctes

> Une transaction exécutée en entier et en isolation conduit la base de données d'un état cohérent à un état cohérent.

Conséquence : les exécutions en série sont cohérentes par hypothèse

Il n'est pas raisonnable de forcer des exécutions en série :

- Services multi-utilisateur : Amazon, AirBnB, Twitter...
- Transaction "en attente" qui ne fera jamais de commit

Sérialisabilité : un ordonnancement est dit sérialisable s'il a le même effet qu'une exécution en série, pour tout état de départ de la base de données

Nouvel objectif : choisir un ordonnancement sérialisable

Exemple 1 : première exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$

Exemple 1 : première exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x)		
x = x%2		
write(A,x)		$A = 1$

Exemple 1 : première exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$

Exemple 1 : première exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$
	read(A,y) y = y*3 **write**(A,y)	$A = 3$

Exemple 1 : première exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$
	read(A,y) y = y*3 **write**(A,y)	$A = 3$
	read(A,y) y = y*y **write**(A,y)	$A = 9$

Exemple 2 : deuxième exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$

Exemple 2 : deuxième exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$

Exemple 2 : deuxième exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
	read(A,y) y = y∗y **write**(A,y)	$A = 225$

Exemple 2 : deuxième exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
	read(A,y) y = y∗y **write**(A,y)	$A = 225$
read(A,x) x = x%2 **write**(A,x)		$A = 1$

Exemple 2 : deuxième exécution en série

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
	read(A,y) y = y∗y **write**(A,y)	$A = 225$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x)		
x = x%2		$A = 1$
write(A,x)		

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x)		
x = x%2		$A = 1$
write(A,x)		
	read(A,y)	
	y = y*3	$A = 3$
	write(A,y)	

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗3 **write**(A,y)	$A = 3$
	read(A,y) y = y∗y **write**(A,y)	$A = 9$

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y*3 **write**(A,y)	$A = 3$
	read(A,y) y = y*y **write**(A,y)	$A = 9$
read(A,x) x = x%3 **write**(A,x)		$A = 0$

Exemple 3 : exécution non sérialisable

Transaction 1	Transaction 2	État de la BD
		$A = 5$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗3 **write**(A,y)	$A = 3$
	read(A,y) y = y∗y **write**(A,y)	$A = 9$
read(A,x) x = x%3 **write**(A,x)		$A = 0$

Aucune exécution en série ne peut aboutir à ce résultat

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
read(A,x) x = x%2 **write**(A,x)		$A = 1$

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗y **write**(A,y)	$A = 1$

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗y **write**(A,y)	$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$

Exemple 4 : exécution sérialisable “par hasard”

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗y **write**(A,y)	$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$

Le résultat est le même que celui de la deuxième exécution en série

Exemple 4 : exécution sérialisable "par hasard"

Transaction 1	Transaction 2	État de la BD
		$A = 5$
	read(A,y) y = y∗3 **write**(A,y)	$A = 15$
read(A,x) x = x%2 **write**(A,x)		$A = 1$
	read(A,y) y = y∗y **write**(A,y)	$A = 1$
read(A,x) x = x%3 **write**(A,x)		$A = 1$

Le résultat est le même que celui de la deuxième exécution en série
Question : est-ce vrai pour toutes les valeurs de départ de A ?

Sérialisabilité forte

Le système ne peut pas raisonnablement décider la sérialisabilité :

- Il ne voit pas les opérations en local des deux transactions mais seulement une séquence d'accès en lecture et en écriture
- Les interactions des deux transactions peuvent être complexes (comme dans l'Exemple 4)

Sérialisabilité forte

Le système ne peut pas raisonnablement décider la sérialisabilité :

- Il ne voit pas les opérations en local des deux transactions mais seulement une séquence d'accès en lecture et en écriture
- Les interactions des deux transactions peuvent être complexes (comme dans l'Exemple 4)

Sérialisabilité forte : un ordonnancement est fortement sérialisable s'il est sérialisable quel que soit le traitement en local des deux transactions

Sérialisabilité forte

Le système ne peut pas raisonnablement décider la sérialisabilité :

- Il ne voit pas les opérations en local des deux transactions mais seulement une séquence d'accès en lecture et en écriture
- Les interactions des deux transactions peuvent être complexes (comme dans l'Exemple 4)

Sérialisabilité forte : un ordonnancement est fortement sérialisable s'il est sérialisable quel que soit le traitement en local des deux transactions

Ex :

```
read(A,x)
x = x%2
write(A,x)
read(B,y)
y = (x+y)*(x-y)
write(B,y)
-- transaction réelle
```

$\Longrightarrow$

```
read(A)

write(A)
read(B)

write(B)
-- ce que voit l'ordonnanceur
```

Sérialisabilité forte

Le système ne peut pas raisonnablement décider la sérialisabilité :

- Il ne voit pas les opérations en local des deux transactions mais seulement une séquence d'accès en lecture et en écriture
- Les interactions des deux transactions peuvent être complexes (comme dans l'Exemple 4)

Sérialisabilité forte : un ordonnancement est fortement sérialisable s'il est sérialisable quel que soit le traitement en local des deux transactions

Ex :

```
read(A,x)
x = x%2
write(A,x)
read(B,y)
y = (x+y)*(x-y)
write(B,y)
-- transaction réelle
```

$\Longrightarrow$

```
read(A)

write(A)
read(B)

write(B)
-- ce que voit l'ordonnanceur
```

Nouvel objectif : choisir un ordonnancement fortement sérialisable

Exemple : exécution fortement sérialisable

Transaction 1

read(A)
-- opérations en local de T1
write(A)

read(B)
-- opérations en local de T1
write(B)

Transaction 2

read(A)
-- opérations en local de T2
write(A)

Exemple : exécution fortement sérialisable

Transaction 1

read(A)
-- opérations en local de T1
write(A)

read(B)
-- opérations en local de T1
write(B)

Transaction 2

read(A)
-- opérations en local de T2
write(A)

Cette exécution est équivalente à $T_1; T_2$ quelles que soient les opérations effectuées en local par T_1 et T_2

Exemple : exécution fortement sérialisable

Transaction 1

read(A)
-- opérations en local de T1
write(A)

read(B)
-- opérations en local de T1
write(B)

Transaction 2

read(A)
-- opérations en local de T2
write(A)

Cette exécution est équivalente à $T_1; T_2$ quelles que soient les opérations effectuées en local par T_1 et T_2

Syntaxe simplifiée : $R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)\ R_1(B)\ W_1(B)$

Équivalence par conflit

Notation

Un ordonnancement S est une suite d'actions $R_i(E)$ ou $W_i(E)$, avec :

- R : accès en lecture, W : accès en écriture
- i : numéro de la transaction demandant l'accès
- E : élément de la base de données concerné

Équivalence par conflit

Notation

Un ordonnancement S est une suite d'actions $R_i(E)$ ou $W_i(E)$, avec :

- R : accès en lecture, W : accès en écriture
- i : numéro de la transaction demandant l'accès
- E : élément de la base de données concerné

Équivalence par conflit

S et S' sont équivalents par conflit si on peut réécrire S en S' par une séquence d'échanges d'actions non conflictuelles consécutives

Équivalence par conflit

Notation

Un ordonnancement S est une suite d'actions $R_i(E)$ ou $W_i(E)$, avec :

- R : accès en lecture, W : accès en écriture
- i : numéro de la transaction demandant l'accès
- E : élément de la base de données concerné

Équivalence par conflit

S et S' sont équivalents par conflit si on peut réécrire S en S' par une séquence d'échanges d'actions non conflictuelles consécutives

Actions non conflictuelles

Deux actions de transactions différentes i et j sont non conflictuelles si :

- Soit les deux actions sont des lectures

  Ex :$R_i(A)$ n'est pas en conflit avec $R_j(A)$

- Soit les deux actions concernent des éléments différents

  Ex : $R_i(A)$ et $W_i(A)$ ne sont pas en conflit avec $W_j(B)$ ou $R_j(B)$, si $A \neq B$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ R_2(A)\ R_1(A)\ W_2(A)\ W_1(B)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ W_2(A)\ W_1(B)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ \underline{W_2(A)\ W_1(B)}\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ \underline{W_2(A)\ W_1(B)}\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ W_1(B)\ W_2(A)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ \underline{W_2(A)\ W_1(B)}\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ \underline{R_2(A)\ W_1(B)}\ W_2(A)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ \underline{W_2(A)\ W_1(B)}\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ \underline{R_2(A)\ W_1(B)}\ W_2(A)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ W_1(B)\ R_2(A)\ W_2(A)\ W_2(B)$$

Exemple : équivalence par conflit

On considère l'ordonnancement S des opérations des transactions 1 et 2

$$S = W_1(A)\ \underline{R_2(A)\ R_1(A)}\ W_2(A)\ W_1(B)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ R_2(A)\ \underline{W_2(A)\ W_1(B)}\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ \underline{R_2(A)\ W_1(B)}\ W_2(A)\ W_2(B)$$

$$S \sim W_1(A)\ R_1(A)\ W_1(B)\ R_2(A)\ W_2(A)\ W_2(B)$$

S est équivalent par conflit à un ordonnancement en série !

Sérialisabilité par conflit

Propriété : si S est équivalent par conflit à S', alors S est équivalent à S'

Autrement dit, échanger deux actions non conflictuelles ne change pas le comportement de l'ordonnancement

Sérialisabilité par conflit

Propriété : si S est équivalent par conflit à S', alors S est équivalent à S'

Autrement dit, échanger deux actions non conflictuelles ne change pas le comportement de l'ordonnancement

Conséquence : si S est équivalent par conflit à un ordonnancement en série, alors S est fortement sérialisable

Sérialisabilité par conflit

Propriété : si S est équivalent par conflit à S', alors S est équivalent à S'

Autrement dit, échanger deux actions non conflictuelles ne change pas le comportement de l'ordonnancement

Conséquence : si S est équivalent par conflit à un ordonnancement en série, alors S est fortement sérialisable

Sérialisabilité par conflit : un ordonnancement est dit sérialisable par conflit s'il est équivalent par conflit à un ordonnacement en série

Sérialisabilité par conflit

Propriété : si S est équivalent par conflit à S', alors S est équivalent à S'

Autrement dit, échanger deux actions non conflictuelles ne change pas le comportement de l'ordonnancement

Conséquence : si S est équivalent par conflit à un ordonnancement en série, alors S est fortement sérialisable

Sérialisabilité par conflit : un ordonnancement est dit sérialisable par conflit s'il est équivalent par conflit à un ordonnacement en série

Question : Comment peut-on tester automatiquement si un ordonnancement est sérialisable par conflit ?

Exercice : sérialisabilité par conflit

Les ordonnancements suivants sont-ils sérialisables par conflit ?

1. $W_2(A)\ R_1(B)\ W_1(A)\ R_2(B)$
2. $R_1(A)\ R_2(A)\ R_1(B)\ W_2(A)\ W_1(B)\ W_1(A)$
3. $R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)\ R_1(B)\ W_1(B)$
4. $R_2(B)\ R_1(A)\ R_3(C)\ W_2(A)\ W_3(B)\ W_1(C)$

Graphe de précédence

$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Graphe de précédence

$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Graphe de précédence

$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :

Graphe de précédence

$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :

T_1 T_3

T_2 T_4

Graphe de précédence

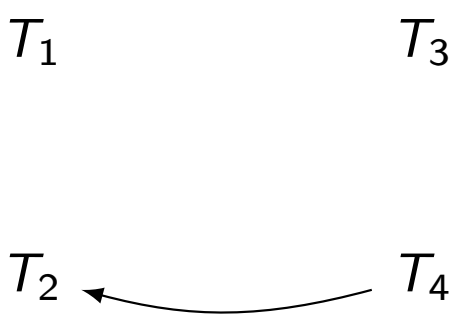
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Graphe de précédence

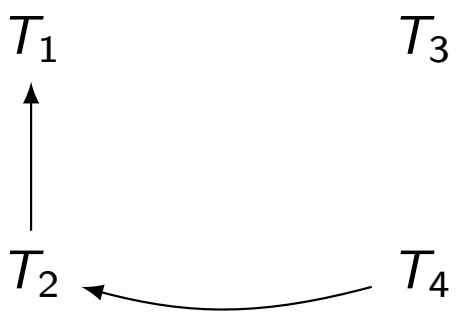
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Graphe de précédence

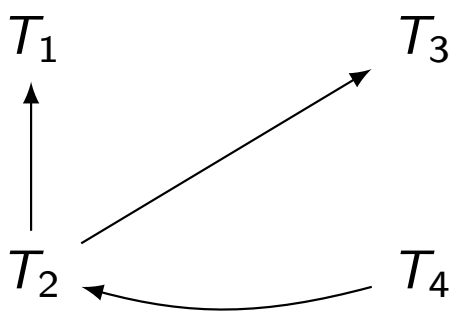
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Graphe de précédence

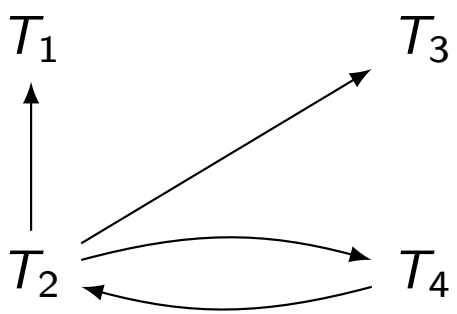
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Graphe de précédence

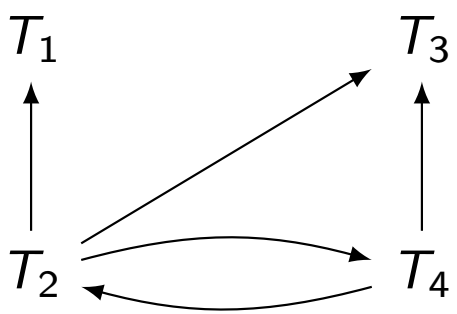
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Graphe de précédence

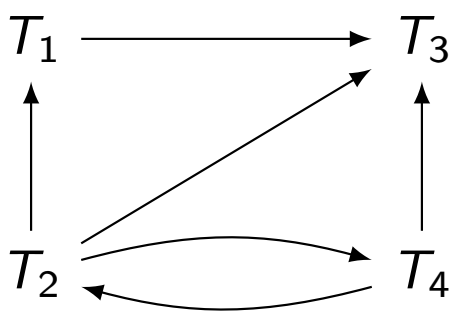
$P(S)$: graphe de précédence d'un ordonnancement S

- Les noeuds de $P(S)$ sont les transactions de S
- Il y a un arc de T_i vers T_j dans $P(S)$ si $i \neq j$ et que S contient une action de T_i qui précède et est en conflit avec une action de T_j

Rappel : deux actions de transactions différentes sont en conflit si elles concernent le même élément et que l'une au moins est une écriture

Ex : Soit $S = R_4(C)\ R_2(A)\ R_2(B)\ W_4(B)\ W_1(A)\ W_2(C)\ W_3(A)\ W_3(B)$

$P(S)$ est le graphe ci-dessous :



Propriétés du graphe de précédence (1)

Propriété : Si $T_i \rightarrow T_j$ dans $P(S)$, alors T_i est exécutée avant T_j dans tout ordonnancement en série équivalent à S par conflit.

Propriétés du graphe de précédence (1)

Propriété : Si $T_i \to T_j$ dans $P(S)$, alors T_i est exécutée avant T_j dans tout ordonnancement en série équivalent à S par conflit.

Éléments de preuve : Supposons que $T_i \to T_j$ dans $P(S)$

Comme $T_i \to T_j$ dans $P(S)$ alors S est de la forme :

$S = \cdots X_i(A) \cdots Y_j(A) \cdots$ (où X et Y sont deux actions conflictuelles)

Propriétés du graphe de précédence (1)

Propriété : Si $T_i \to T_j$ dans $P(S)$, alors T_i est exécutée avant T_j dans tout ordonnancement en série équivalent à S par conflit.

Éléments de preuve : Supposons que $T_i \to T_j$ dans $P(S)$

Comme $T_i \to T_j$ dans $P(S)$ alors S est de la forme :

$S = \cdots X_i(A) \cdots Y_j(A) \cdots$ (où X et Y sont deux actions conflictuelles)

Soit S' un ordonnancement en série qui exécute T_j avant T_i
Alors S' est nécessairement de la forme :

$S' = \cdots Y_j(A) \cdots X_i(A) \cdots$

Propriétés du graphe de précédence (1)

Propriété : Si $T_i \to T_j$ dans $P(S)$, alors T_i est exécutée avant T_j dans tout ordonnancement en série équivalent à S par conflit.

Éléments de preuve : Supposons que $T_i \to T_j$ dans $P(S)$

Comme $T_i \to T_j$ dans $P(S)$ alors S est de la forme :

$S = \cdots X_i(A) \cdots Y_j(A) \cdots$ (où X et Y sont deux actions conflictuelles)

Soit S' un ordonnancement en série qui exécute T_j avant T_i
Alors S' est nécessairement de la forme :

$S' = \cdots Y_j(A) \cdots X_i(A) \cdots$

Toute réécriture de S en S' doit à un moment échanger X_i et Y_j
C'est impossible car X_i et Y_j sont conflictuelles
Donc S et S' ne peuvent pas être équivalents par conflit

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Rightarrow$) :
Soit S un ordonnancement sérialisable par conflit
Alors il existe un ordonnancement en série S' équivalent par conflit à S

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Rightarrow$) :
Soit S un ordonnancement sérialisable par conflit
Alors il existe un ordonnancement en série S' équivalent par conflit à S

Supposons par l'absurde que $P(S)$ a un circuit de la forme
$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Rightarrow$) :
Soit S un ordonnancement sérialisable par conflit
Alors il existe un ordonnancement en série S' équivalent par conflit à S

Supposons par l'absurde que $P(S)$ a un circuit de la forme
$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$

Donc d'après la propriété 1 :
T_1 est exécutée avant T_2 dans S'
T_2 est exécutée avant T_3 dans S'
$\ldots$
T_n est exécutée avant T_1 dans S'

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Rightarrow$) :
Soit S un ordonnancement sérialisable par conflit
Alors il existe un ordonnancement en série S' équivalent par conflit à S

Supposons par l'absurde que $P(S)$ a un circuit de la forme
$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$

Donc d'après la propriété 1 :
T_1 est exécutée avant T_2 dans S'
T_2 est exécutée avant T_3 dans S'
$\ldots$
T_n est exécutée avant T_1 dans S'

T_1 est exécutée à la fois avant et après T_n dans S', ce qui est absurde

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Donc S peut être réécrite pour placer au début toutes les actions de T_i

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Donc S peut être réécrite pour placer au début toutes les actions de T_i

S est alors équivalent à un ordonnancement de la forme $S' = T_i; S''$
où S'' ne contient aucune action de T_i

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Donc S peut être réécrite pour placer au début toutes les actions de T_i

S est alors équivalent à un ordonnancement de la forme $S' = T_i; S''$
où S'' ne contient aucune action de T_i

On remarque alors que $P(S'')$ est exactement $P(S)$ sans T_i et ses arcs
Donc $P(S'')$ n'a pas de circuit

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Donc S peut être réécrite pour placer au début toutes les actions de T_i

S est alors équivalent à un ordonnancement de la forme $S' = T_i; S''$
où S'' ne contient aucune action de T_i

On remarque alors que $P(S'')$ est exactement $P(S)$ sans T_i et ses arcs
Donc $P(S'')$ n'a pas de circuit

Et on recommence alors avec S''

Propriétés du graphe de précédence (2)

Propriété : S est sérialisable par conflit **ssi** $P(S)$ n'a pas de circuit.

Éléments de preuve ($\Leftarrow$) :
Soit S un ordonnancement tel que $P(S)$ n'a pas de circuit
Alors $P(S)$ contient nécessairement un noeud T_i qui n'a pas d'arc entrant

Donc S peut être réécrite pour placer au début toutes les actions de T_i

S est alors équivalent à un ordonnancement de la forme $S' = T_i; S''$
où S'' ne contient aucune action de T_i

On remarque alors que $P(S'')$ est exactement $P(S)$ sans T_i et ses arcs
Donc $P(S'')$ n'a pas de circuit

Et on recommence alors avec S''

Exercice : formalisez ce raisonnement par une jolie preuve par induction !

Test de sérialisabilité par conflit

Comment tester la sérialisabilité par conflit d'un ordonnancement S ?

En lien avec le cours d'algorithmique des graphes

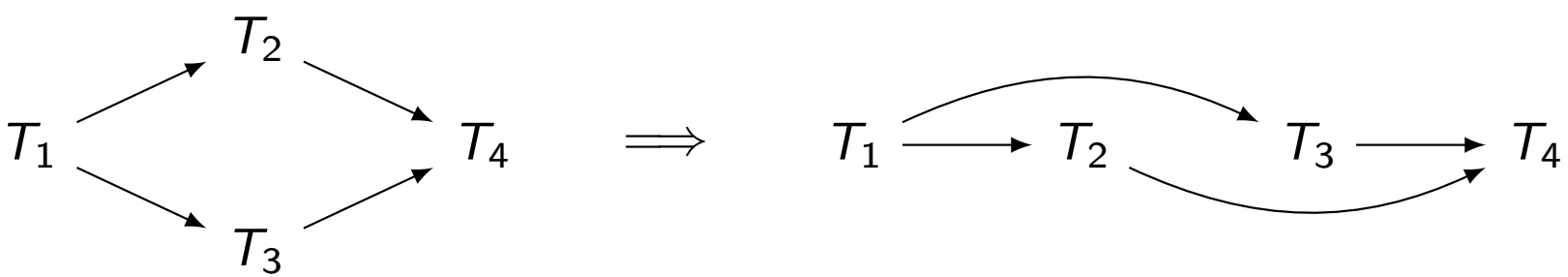
- Construire le graphe de précédence $P(S)$ de S (*en* $O(|S|^2)$)
- Tester si $P(S)$ contient un circuit
  En $O(n^2)$ *ou* $O(n+e)$ *selon la représentation choisie pour* $P(S)$
- Si $P(S)$ a un circuit, alors S n'est pas sérialisable par conflit
- Sinon, S est sérialisable par conflit
  Un tri topologique de $P(S)$ *donne un ordonnancement en série équivalent*

Test de sérialisabilité par conflit

Comment tester la sérialisabilité par conflit d'un ordonnancement S ?

En lien avec le cours d'algorithmique des graphes

- Construire le graphe de précédence $P(S)$ de S (*en* $O(|S|^2)$)
- Tester si $P(S)$ contient un circuit
  En $O(n^2)$ *ou* $O(n+e)$ *selon la représentation choisie pour* $P(S)$
- Si $P(S)$ a un circuit, alors S n'est pas sérialisable par conflit
- Sinon, S est sérialisable par conflit
  Un tri topologique de $P(S)$ *donne un ordonnancement en série équivalent*



Exercice : sérialisabilité et précédence

Construire les graphes de précédence des ordonnancements suivants.

En déduire les ordonnancements qui sont sérialisables par conflit, et pour chacun un ordonnancement en série équivalent.

1. $R_2(B)\ R_1(A)\ R_3(C)\ W_2(A)\ W_3(B)\ W_1(C)$
2. $R_2(A)\ R_1(A)\ W_2(A)\ R_3(C)\ W_2(B)\ R_4(B)\ R_3(B)\ W_4(C)$
3. $W_3(A)\ W_2(C)\ R_1(A)\ R_1(B)\ R_1(C)\ W_2(A)\ R_4(A)\ W_4(D)$
4. $R_1(A)\ R_2(B)\ W_3(B)\ W_4(A)\ R_3(A)\ W_3(C)\ W_1(C)$

Contrôle de la concurrence

Contrôle de la concurrence

Rappel : le système ne contrôle pas l'ordre d'arrivée des demandes de lecture et d'écriture, mais peut choisir dans quel ordre les autoriser

Ordonnanceur (scheduler) : module du système chargé de coordonner les accès concurrents pour garantir un ordonnancement sérialisable

Contrôle de la concurrence

Rappel : le système ne contrôle pas l'ordre d'arrivée des demandes de lecture et d'écriture, mais peut choisir dans quel ordre les autoriser

Ordonnanceur (scheduler) : module du système chargé de coordonner les accès concurrents pour garantir un ordonnancement sérialisable

Plusieurs approches :

- Estampilles (timestamps)
- Verrous (locks)
- Isolation par instantanés (snapshot isolation)

Contrôle de la concurrence

Rappel : le système ne contrôle pas l'ordre d'arrivée des demandes de lecture et d'écriture, mais peut choisir dans quel ordre les autoriser

Ordonnanceur (scheduler) : module du système chargé de coordonner les accès concurrents pour garantir un ordonnancement sérialisable

Plusieurs approches :

- Estampilles (timestamps)
- Verrous (locks)
- Isolation par instantanés (snapshot isolation)

En pratique : méthodes reposant sur des idées issues des trois approches

Estampillage

Principe général :

- Chaque transaction est estampillée par le moment où elle a débuté
- Chaque élément de la base de données est estampillé par :
  - la plus grande des estampilles des transactions qui ont lu l'élément
  - la plus grande des estampilles des transactions qui ont écrit l'élément
- L'ordonnanceur s'assure que les actions conflictuelles ont lieu dans l'ordre des estampilles
- L'ordre des estampilles garantit une exécution en série équivalente

Mise en œuvre de l'estampillage

Le système mémorise, pour chaque transaction T et élément A :

- $E(T)$: estampille de T, date à laquelle T a débuté
- $E_R(A)$: maximum des estampilles des transactions qui ont lu A
- $E_W(A)$: maximum des estampilles des transactions qui ont écrit A

Règle d'estampillage : une transaction T peut :

- lire A seulement si $E(T) \geq E_W(A)$
- écrire A seulement si $E(T) \geq E_R(A)$ et $E(T) \geq E_W(A)$

Mise en œuvre de l'estampillage

Le système mémorise, pour chaque transaction T et élément A :

- $E(T)$: estampille de T, date à laquelle T a débuté
- $E_R(A)$: maximum des estampilles des transactions qui ont lu A
- $E_W(A)$: maximum des estampilles des transactions qui ont écrit A

Règle d'estampillage : une transaction T peut :

- lire A seulement si $E(T) \geq E_W(A)$
- écrire A seulement si $E(T) \geq E_R(A)$ et $E(T) \geq E_W(A)$

Si la transaction T tente une action qui enfreint la règle d'estampillage :

- La transaction T est annulée (rollback)
- Les transactions qui ont lu des valeurs écrites par T sont annulées
- Et ainsi de suite... jusqu'à revenir à un ordonnancement sérialisable

Estampillage : remarques

L'estampillage est une approche curative :

- On assure la sérialisabilité des demandes par un ordre préétabli
- Si une erreur survient, on traite en annulant les transactions fautives

Inconvénients :

- Les abandons (notamment en cascade) sont lourds à gérer
- Les transactions annulées doivent être relancées jusqu'à aboutir

L'estampillage est une méthode correcte : la sérialisabilité est garantie en forçant un tri topologique du graphe de précédence

Estampillage : remarques

L'estampillage est une approche curative :

- On assure la sérialisabilité des demandes par un ordre préétabli
- Si une erreur survient, on traite en annulant les transactions fautives

Inconvénients :

- Les abandons (notamment en cascade) sont lourds à gérer
- Les transactions annulées doivent être relancées jusqu'à aboutir

L'estampillage est une méthode correcte : la sérialisabilité est garantie en forçant un tri topologique du graphe de précédence

Question : l'estampillage est-il complet ? Peut-on trouver un ordonnancement sérialisable qui ne passent pas les règles d'estampillage ?

Verrouillage

Principe général :

- Les transactions doivent demander un verrou sur un élément avant de pouvoir le lire ou l'écrire
- L'ordonnanceur accorde le verrou seulement si aucune autre transaction n'a un verrou actif sur l'élément concerné
- Les demandes qui n'ont pas été accordées sont mises en attente
- Les verrous sont relâchés quand la transaction n'utilise plus l'élément
- Le verrouillage force donc un ordre des actions conflictuelles

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A)	
read(A)	

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*
lock(B)	

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*
lock(B)	
lock(A) *-- transaction 1 mise en attente*	

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*
lock(B)	
lock(A) *-- transaction 1 mise en attente*	
	read(A) **unlock**(A)

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*
lock(B)	
lock(A) *-- transaction 1 mise en attente*	
	read(A) **unlock**(A)
lock(A) *-- transaction 1 reprise*	

Exemple : verrouillage

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
write(A) **unlock**(A)	
	lock(A) *-- transaction 2 reprise*
lock(B)	
lock(A) *-- transaction 1 mise en attente*	
	read(A) **unlock**(A)
lock(A) *-- transaction 1 reprise*	
write(A) **unlock**(A)	

Exemple : poser des verrous ne suffit pas !

Transaction 1

```
lock(A)
read(A)
unlock(A)
```

```
lock(B)
write(B)
unlock(B)
```

Transaction 2

```
lock(A)
write(A)
lock(B)
write(B)
unlock(A)
unlock(B)
```

Exemple : poser des verrous ne suffit pas !

Transaction 1

lock(A)
read(A)
unlock(A)

lock(B)
write(B)
unlock(B)

Transaction 2

lock(A)
write(A)
lock(B)
write(B)
unlock(A)
unlock(B)

- Cette exécution n'est pas sérialisable par conflit
  (et pas du tout selon les opérations locales)
- Le verrouillage ne suffit pas même si les verrous sont correctement demandés et relâchés

Verrouillage à deux phases

Principe général :

- Une version plus stricte de verrouillage qui garantit la sérialisabilité
- Les transactions doivent se décomposer en deux phases :
  - Phase 1 : la transaction peut demander des verrous
  - Phase 2 : la transaction peut relâcher ses verrous
- L'ordonnanceur refuse tout nouveau verrou à une transaction qui a déjà relâché un verrou
- Les transactions doivent déclarer en phase 1 tous les verrous dont elles auront besoin

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A)	
read(A)	

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)
	write(A)

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)
	write(A)
	lock(B) *-- transaction 2 mise en attente*

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)
	write(A)
	lock(B) *-- transaction 2 mise en attente*
unlock(B)	

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)
	write(A)
	lock(B) *-- transaction 2 mise en attente*
unlock(B)	
	lock(B)

Exemple : verrouillage à deux phases

Réécriture de l'exemple non sérialisable avec verrous en deux phases

Transaction 1	Transaction 2
lock(A) **read**(A)	
	lock(A) *-- transaction 2 mise en attente*
lock(B) **write**(B) **unlock**(A)	
	lock(A)
	write(A)
	lock(B) *-- transaction 2 mise en attente*
unlock(B)	
	lock(B)
	write(B) **unlock**(A) **unlock**(B)

Niveaux de verrouillage

On peut améliorer les performances (diminuer le nombre de blocages) en définissant plusieurs niveaux de verrous plus souples

Niveaux de verrouillage

On peut améliorer les performances (diminuer le nombre de blocages) en définissant plusieurs niveaux de verrous plus souples

Selon l'intention, parce que deux lectures ne sont jamais conflictuelles :

- Verrou partagé en lecture
- Verrou exclusif en écriture
- Et d'autres : verrou de mise-à-jour, promotion de verrou...

	verrou actif	
	lecture	écriture
lecture	accordé	en attente
écriture	en attente	en attente

Niveaux de verrouillage

On peut améliorer les performances (diminuer le nombre de blocages) en définissant plusieurs niveaux de verrous plus souples

Selon l'intention, parce que deux lectures ne sont jamais conflictuelles :

- Verrou partagé en lecture
- Verrou exclusif en écriture
- Et d'autres : verrou de mise-à-jour, promotion de verrou...

	verrou actif	
	lecture	écriture
lecture	accordé	en attente
écriture	en attente	en attente

Selon la granularité, de la plus fine à la plus grossière :

- Verrou sur un enregistrement
- Verrou sur un bloc du disque
- Verrou sur une table

Verouillage : remarques

Le verrouillage est une approche préventive :

- On évite les conflits en faisant attendre les opérations conflictuelles
- Il est rare de devoir annuler une transaction

Inconvénients :

- Deadlocks lorsque deux transactions s'attendent l'une l'autre
- Approche coûteuse et complexe : quoi verrouiller et quand ?

Le verrouillage à deux phases est une méthode correcte : l'ordre des débuts des phases 2 est un tri topologique du graphe de précédence

Verouillage : remarques

Le verrouillage est une approche préventive :

- On évite les conflits en faisant attendre les opérations conflictuelles
- Il est rare de devoir annuler une transaction

Inconvénients :

- Deadlocks lorsque deux transactions s'attendent l'une l'autre
- Approche coûteuse et complexe : quoi verrouiller et quand ?

Le verrouillage à deux phases est une méthode correcte : l'ordre des débuts des phases 2 est un tri topologique du graphe de précédence

Question : le verrouillage est-il complet ? Peut-on trouver un ordonnancement sérialisable qu'on ne peut pas exécuter avec un verrouillage à deux phases ?

Isolation par instantanés

Principe général :

- Chaque transaction prend un instantané (snapshot) de la base de données quand elle commence
- Les transactions lisent et écrivent uniquement dans leur snapshot
- Les mises-à-jour concurrentes sont invisibles aux autres transactions
- Lorsqu'une transaction termine par un commit :
  - Les modifications du snasphot sont reportées dans la base de données
  - En cas de conflit, la transaction est annulée et doit être relancée

Isolation par instantanés : remarques

L'isolation par instantanés est une méthode performante :

- Les demandes des transactions ne sont jamais bloquantes
- Les abandons sont possibles, mais ne cascadent pas
- Évite les lectures sales, non-reproductives et fantômes

Inconvénients :

- L'isolation par instantanés ne garantit pas la sérialisabilité !
- Les transactions annulées doivent être relancées jusqu'à aboutir

En pratique :

- La plupart des systèmes utilisent une variante d'isolation par instantanées qui ne garantit pas toujours la sérialisabilité
- Le niveau **SERIALIZABLE** de Postgres utilise une variante correcte, enrichie avec des verrous et des estampilles

Isolation par instantanés : remarques

L'isolation par instantanés est une méthode performante :

- Les demandes des transactions ne sont jamais bloquantes
- Les abandons sont possibles, mais ne cascadent pas
- Évite les lectures sales, non-reproductives et fantômes

Inconvénients :

- L'isolation par instantanés ne garantit pas la sérialisabilité !
- Les transactions annulées doivent être relancées jusqu'à aboutir

En pratique :

- La plupart des systèmes utilisent une variante d'isolation par instantanées qui ne garantit pas toujours la sérialisabilité
- Le niveau **SERIALIZABLE** de Postgres utilise une variante correcte, enrichie avec des verrous et des estampilles

Exercice : exhiber une anomalie malgré l'isolation par instantanés