Cours de bases de données, aspects systèmes, http://sys.bdpedia.fr

Le contrÃţle de concurrence multi-versions

Philippe Rigaux

le cnam

Algorithme de contrôle de concurrence multi-versions

Connu sous le nom de contrôle de concurrence *isolation snapshot*. Très utilisé.

Tire parti du versionnement, qui limite les opportunités de conflits

Très léger : aucun verrouillage en lecture, un contrôle sur les écritures.

Mais : ne garantit pas la sérialisabilité stricte

Souvenons-nous des versions

Un système qui fournit un niveau `repeatable read` doit s'appuyer sur un système de versions estampillées (voir la session).

Dans ce cas les lectures se font sur l'état de la base **avant** le début de la transaction.

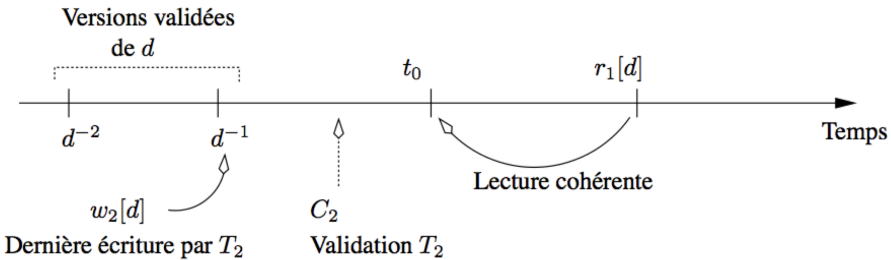
Deux possibilités de conflit entre une **lecture** de T_1 et T_2.

- $r_1[d]$ est en conflit avec une écriture $w_2[d]$ qui a eu lieu **avant** t_0 ;
- $r_1[d]$ est en conflit avec une écriture $w_2[d]$ qui a eu lieu **après** t_0.

Approfondissons.

Premier cas de conflit

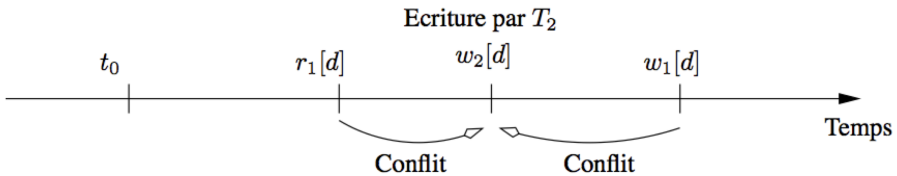
Cas 1 : $r_1[d]$ en conflit avec $w_2[d]$ **avant** t_0. Alors T_2 a validé. Pas de risque.



En fait, indique que T_2 et T_1 sont en série.

Second cas de conflit

Cas 2 : $r_1[d]$ en conflit avec $w_2[d]$ **après** t_0.



Si T_1 cherche à écrire d après l'écriture $w_2[d]$, un conflit cyclique apparaît. **L'algorithme doit surveiller ce cas.**

Le contrôle de concurrence multi-versions

Le principe

On vérifie, au moment d'exécuter $w_1[e]$, qu'aucune transaction T_2 n'a modifié e entre le début de T_1 et l'instant présent.

En cas d'écriture $w_1[e]$,

- si $\tau_e \leq \tau_1$ et e non verrouillé : T_1 verrouille (exclusif) e, et effectue $w_1[e]$;
- si $\tau_e \leq \tau_1$ et e verrouillé : T_1 est mise en attente ;
- si $\tau_e > \tau_1$, T_1 est rejetée.

Estampillage. Au moment de C_1 tous les tuples modifiés par T_1 obtiennent une nouvelle version avec pour estampille l'instant du `commit`.

Exemple

Maj perdues : $r_1(s)r_1(c_1)r_2(s)r_2(c_2)w_2(s)w_2(c_2)C_2w_1(s)w_1(c_1)C_1$.
On prend $\tau_1 = 100$, $\tau_2 = 120$.

- T_1 lit s, T_1 lit c_1, T_2 lit s, T_2 lit c_2, sans verrouiller ; **aucun verrou**.
- T_2 veut modifier s : l'estampille de s est inférieure à $\tau_2 = 120$: s n'a pas été modifié ; on pose un verrou exclusif sur s et on effectue $w_2[s]$:
- T_2 modifie c_2, avec pose d'un verrou exclusif ;
- T_2 valide, relâche les verrous ; versions de s et c_2 avec l'estampille 150 ;
- T_1 veut à son tour modifier s : version de s avec $\tau_s > \tau_1 = 100$. T_1 est rejetée.

Un cas de non-sérialisabilité

L'algorithme *snapshot isolation* dans certaines exécutions non sérialisables.

Exemple : une simple copie d'une ligne à une autre :

```
function copie (id1, id2)
begin
  val := select valeur from T where id = id1
  update T set valeur = :val where id = id2
  commit
end
```

Exécution concurrente de $copie(A, B)$ et $copie(b, A)$:

$$r_1[x]r_2[y]w_1[y]w_2[x]$$

Non sérialisable, mais autorisée par le contrôle multi-versions.

À retenir

Algorithme très simple à mettre en œuvre, utilisé par exemple dans des contexte distribués (applications web)

Très fluide, très peu de verrouillage, contrôle bien le cas des mises à jour perdues

Ne garantit pas la sérialisabilité stricte (des améliorations ont été proposées)