SGBD-R : Les transactions

Reprise après panne

Guillaume Raschia — Nantes Université

originaux de Philippe Rigaux, CNAM

Dernière mise-à-jour : 31 mars 2025

Panne (S11.1)

Plan de la session

La notion de panne

Panne = tout dysfonctionnement affectant le comportement du système

Par souci de simplicité on va distinguer deux types de panne (quelle que soit la cause).

- Panne légère : affecte la RAM du serveur de données, pas les disques
- Panne lourde : affecte un disque

Dans un premier temps, on se concentre sur les pannes légères.

Garantie en cas de panne

Souvenons-nous des propriétés des transactions.

Durabilité et atomicité : quand le système rend la main après un `commit`, toutes les modifications de la transaction deviennent permanentes.

Réparabilité et atomicité : tant qu'un `commit` n'a pas eu lieu, toutes les modifications de la transaction doivent pouvoir être annulées par un `rollback`.

Garantir un `commit`

Pour garantir le `commit`, la condition suivante doit être respectée.

Les données modifiées par une transaction approuvée doivent être sur le disque

On désigne ces données par le terme « image après ».

Autrement dit

L'image après doit être sur le disque avant l'acquittement du `commit`.

Garantir un `rollback`

Pour garantir le `rollback`, la condition suivante doit être respectée.

Les données **avant** modification doivent être sur le disque

On désigne ces données par le terme « image avant ».

Autrement dit

L'image avant doit être sur le disque jusqu'à l'acquittement du `commit`.

L'état de la base

État résultant de l'ensemble des transactions approuvées depuis l'origine.

On résume ce qui précède par :

L'état de la base doit **toujours** être sur le disque.

La difficulté

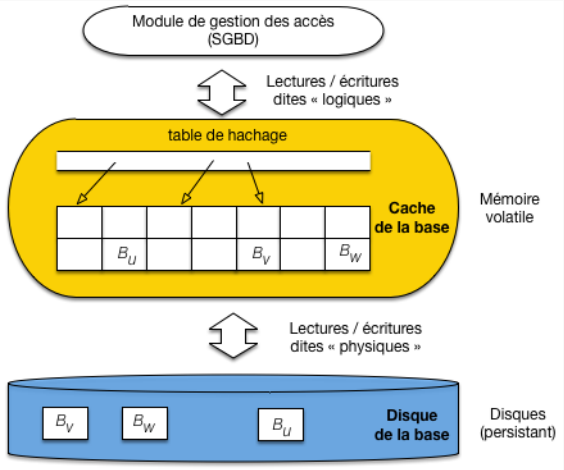
On peut énoncer la difficulté ainsi :

- Jusqu'au `commit`, c'est l'image avant qui fait partie de l'état de la base.
- au `commit`, l'image après remplace l'image avant dans l'état de la base.

Problème : Comment assurer que ce remplacement s'effectue de manière atomique (« tout ou rien ») ?

Ce n'est pas facile…

La hiérarchie des mémoires



Cache et disque (S11.2)

Opération de lecture

Toute opération de lecture contient l'adresse du bloc à lire.

Si le bloc est dans le cache, le système accède à la donnée dans le bloc, et la retourne.

Sinon il faut d'abord lire un bloc du disque, et le placer dans le cache.

Le SGBD garde en mémoire les blocs après lecture.

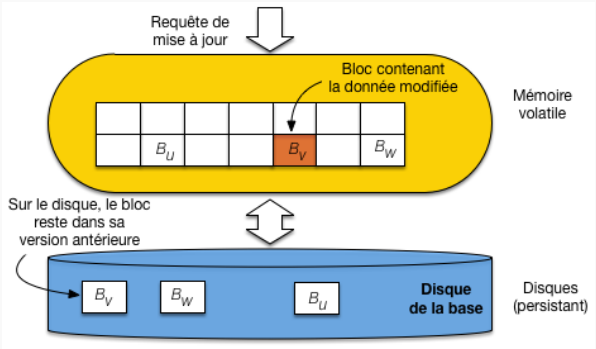
Stratégie de remplacement

Que faire quand la mémoire est pleine ?

L'algorithme le plus courant est dit *Least Recently Used* (LRU).

1. La « victime » est le bloc dont la dernière date de lecture logique est la plus ancienne.
2. Ce bloc est alors soustrait de la mémoire centrale.
3. Le nouveau bloc vient le remplacer.

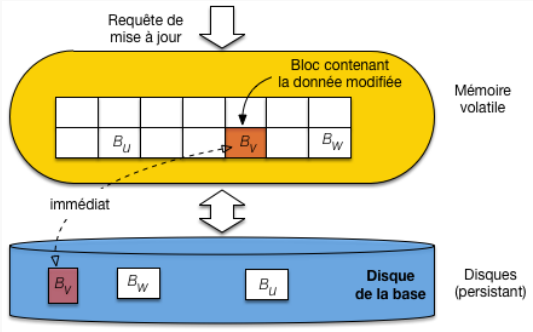
Opération de mise à jour



On trouve le bloc (comme pour une lecture), on modifie la donnée.

Faut-il écrire le bloc ou non ?

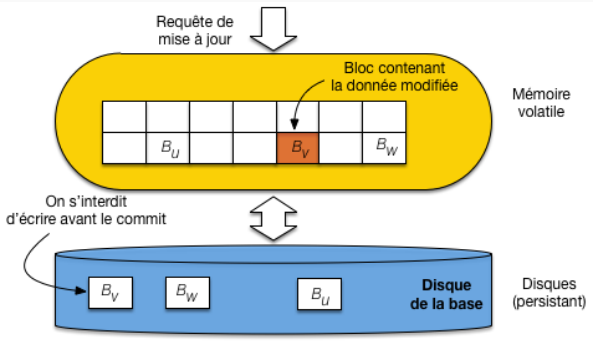
Mise à jour immédiate



Dès qu'un bloc est modifié, on écrit sur le disque pour synchroniser.

- Peu performant : écritures aléatoires
- On écrase l'image avant

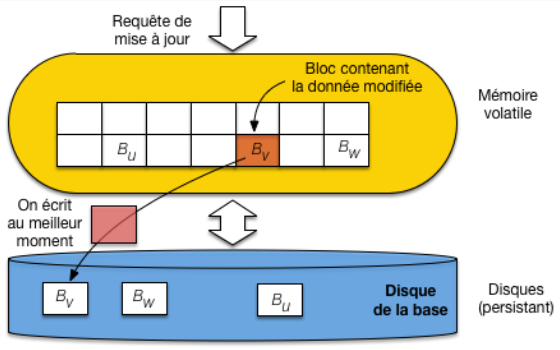
Mise à jour différée



On interdit l'écriture d'un bloc modifié avant le `commit`.

- Risque de surcharge du cache
- Préserve l'image avant

Mise à jour opportuniste



C'est la méthode la plus souple : on attend la meilleure opportunité pour synchroniser le cache et le disque.

Le moins pénalisant ; permet des écritures successives dans le cache.

Reprise : première approche (S11.3)

Résumé : cache et disque

Retenir :

- Une partie de la base est copiée dans le cache.
- Toute mise à jour s'effectue d'abord dans le cache.
- La synchronisation avec le disque peut s'effectuer de manière immédiate, différée, ou opportuniste.

La méthode de synchronisation a un impact fort sur la reprise sur panne.

Comment reprendre sur panne légère

Une transaction s'exécute ; l'image avant est sur le disque ; l'image après est dans le cache.

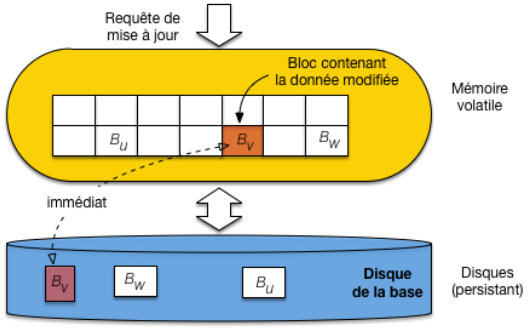
Nous disposons de plusieurs options d'écriture : immédiate, différée, opportuniste.

Si une panne légère survient : le contenu de la RAM est perdu.

Existe-t-il une stratégie possible de reprise sur panne ?

Scénario 1 : avec des écritures opportunistes

Supposons que les écritures fonctionnent en mode opportuniste.



Si un bloc contenant des données non validées est écrit, l'image avant est écrasée. Pas de reprise possible.

Vrai, à plus forte raison, avec des écritures immédiates.

Scénario 2 : avec des écritures différées

Tentons une approche plus raisonnée.

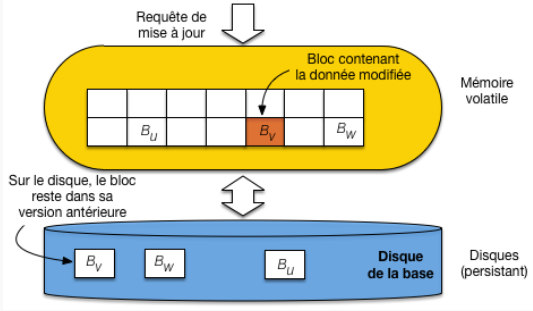
- ne jamais écrire un nuplet modifié avant le `commit` ;
- au moment du `commit` de T, forcer l'écriture de tous les blocs modifiés par T.

Semble assurer le `commit` et le `rollback` :

- Le `commit` transfère l'image après du cache vers le disque
- Le `rollback` supprime les blocs modifiés en cache : on en revient à l'image avant.

Pourquoi ça ne marche pas ?

Une panne survient après les écritures mais avant l'acquittement du `commit`.



`rollback` impossible si je ne garde pas sur disque les données après et avant modification.

Résumé : cache, disque et reprise sur panne

Au moment de l'exécution de l'ordre `commit`

- l'image avant doit être sur disque : permet un `rollback` jusqu'à la complétion.
- l'image après doit être sur disque : assure la reprise sur panne (légère).

De plus, le mode d'écriture opportuniste doit être privilégié car

- le mode différé encombre le cache
- le mode immédiat induit des écritures aléatoires.

La solution s'appelle le journal des transactions.

Le journal des transactions (S11.4)

Le journal des transactions (*log*)

Le fichier des tx, ou *log*, est un fichier complémentaire à la base de données.

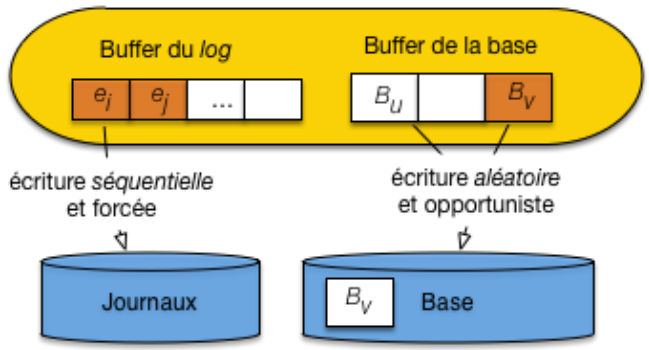
- le système y écrit séquentiellement
- le *log* dispose de sa propre mémoire cache
- on n'efface jamais un enregistrement du *log*

La reprise sur panne repose sur la stratégie suivante :

- le fichier de données et son cache sont en écriture opportuniste
- le *log* et son cache sont en écriture immédiate

Le *log* contient l'historique des mises à jour.

Le système d'entrées/sorties avec *log*



Contenu du *log*

Toutes les instructions de mise à jour.

Chaque ligne contient un enregistrement d'une des formes suivantes.

- `start(T)` : début de la tx T
- `write(T, x, old_val, new_val)`
- `commit(T)`
- `rollback(T)`
- `checkpoint` : point de sauvegarde (ou point de reprise)

Le `write` peut être représenté sous forme logique (instructions) ou physique (enregistrements).

Le *log* et la gestion des `commit`

La règle suivante est dite du point de commit[1] :

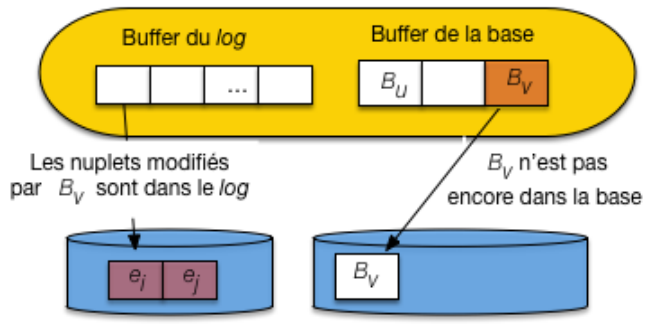
- au `commit`, on force l'écriture des modifications effectuées dans le *log*

Conséquences

- L'état de la base peut être reconstitué à partir du *log*
- Une tx est approuvée quand l'instruction `commit` est écrite dans le *log*

1. En anglais, *Force Logging at Commit* ou FLaC.

Illustration : après un `commit`



Le *log* et la gestion des `rollback`

Un bloc modifié mais pas encore validé peut être écrit dans la base.

En cas de panne, il faudra reprendre l'image avant.

La règle suivante est dite du log préalable[2] :

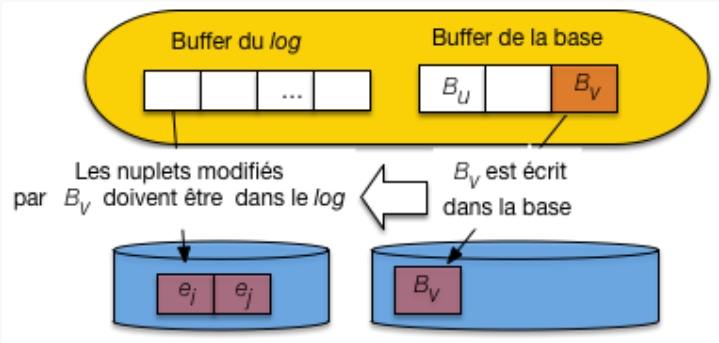
- quand un bloc modifié est écrit dans la base, il faut d'abord écrire les modifications dans le *log*.

Conséquence

- Le *log* contient des modifications qui n'ont pas été validées.

2. En anglais, *Write-ahead Logging* ou WaL.

Illustration du *WaL*



Résumé : le *log*

Retenir :

- Le *log* est un fichier séquentiel dans lequel on enregistre toutes les mises à jour
- un `commit` est effectif quand l'instruction est écrite dans le *log*
  $\Rightarrow$ l'état de la base est dans le *log*
- un `rollback` peut s'effectuer en reprenant l'image avant dans le *log*
- performances : on laisse la base et son cache en mode d'écriture opportuniste

Prochaine étape : l'algorithme de reprise sur panne

Refaire et défaire

En cas de panne légère, l'écriture opportuniste a pour conséquences :

- Des modifications validées qui ne sont pas encore dans la base.
- Inversement, des modifications non validées qui sont dans la base.

À la reprise, il faut donc

- Refaire (REDO) les transactions approuvées ;
- Défaire (UNDO) les transactions en cours ou rejetées.

Ces deux opérations sont basées sur le journal.

Algorithmes de reprise (S11.5)

Les informations du *log*

On peut reconstituer, à partir du log :

- la liste L_V des transactions approuvées : on trouve un `commit` dans le *log*
- la liste L_A des transactions actives ou rejetées : pas de `commit` dans le *log*
- l'image après (`new_val`) et l'image avant (`old_val`) pour chaque nuplet modifié par une tx.

Ces informations sont nécessaires et suffisantes pour la reprise sur panne

Défaire

Les transactions en cours ou rejetées, L_A, celles qui n'ont pas de `commit` dans le journal

Fondé sur des écritures immédiates, l'algorithme de UNDO considère chaque opération des tx de L_A dans l'ordre inverse d'exécution.

Pour chaque `write(T, x, old_val, new_val)`, il écrit `old_val` dans `x`.

Refaire

Les transactions validées, L_V : pour lesquelles on trouve un `commit(T)` dans le *log*

Fondé sur des écritures différées, l'algorithme de REDO considère chaque tx de L_V dans l'ordre d'exécution.

Pour chaque `write(T, x, old_val, new_val)`, il écrit `new_val` dans `x`.

✎ Et si la reprise subit une panne ?

Il faut recommencer : le REDO est idempotent.

L'algorithme de reprise UNDO/REDO

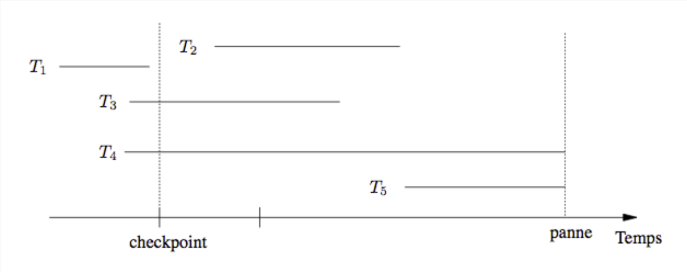
Le cas général fondé sur des écritures opportunistes du fichier de données

1. Parcours dans l'ordre inverse d'exécution pour :
   - constituer la liste des transactions approuvées L_V ;
   - défaire les écritures des tx actives ou rejetées au moment de la panne. La liste L_A n'est en fait pas nécessaire.
2. Parcours dans l'ordre d'exécution pour :
   - rejouer les écritures de L_V à partir des enregistrements du journal.

À propos des *checkpoints*

En cas de panne, il faudrait en principe refaire toutes les transactions du journal, depuis l'origine de la création de la base.

Un checkpoint écrit sur disque tous les blocs modifiés (les « pages sales »), ce qui garantit que les données validées par `commit` sont dans la base.



Point de sauvegarde glissant

Évite l'arrêt du service pendant la sauvegarde !

1. Le système écrit un enregistrement `start checkpoint` au moment de la sauvegarde.
2. On marque et on commence à écrire les pages sales au moment du point de sauvegarde, tout en continuant à traiter les tx.
3. À la fin de l'écriture des pages sales, un enregistrement `end checkpoint` est écrit dans le *log*.

Le point de reprise est défini par l'enregistrement `start checkpoint`.

✎ À la reprise, faut-il remonter au-delà du `start checkpoint` ?

Pannes de disque (S11.6)

Résumé : reprise sur panne UNDO/REDO

Retenir :

- Le *log* contient la liste des tx validées ; la liste des tx annulées/en cours.
- Une reprise ne prend en compte que les tx depuis le dernier *checkpoint*.
- La reprise consiste à parcourir le *log* et à défaire les transactions annulées/en cours, puis refaire les transactions validées.

Panne d'un disque

Panne la plus grave, car on risque de perdre l'état de la base.

Solution : la réplication. Réfléchissons.

- L'état de la base est dans le fichier de journal.
- L'état de la base est aussi dans le fichier de données complété par le cache !

Constat : pour qu'il y ait réplication, il est impératif de stocker la base et le *log* sur deux disques distincts.

Cas 1 : panne du disque de la base

Solution triviale :

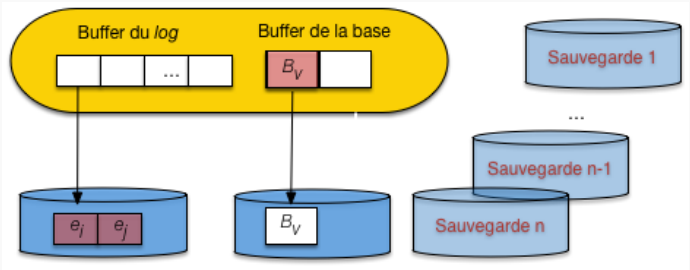
- On réinstalle un disque pour la base.
- On ré-exécute toutes les transactions du *log*.

Inconvénients :

- Phase de latence longue au moment du redémarrage.
- Implique de conserver tous les *log* depuis l'origine.

Panne du disque de la base : avec sauvegarde

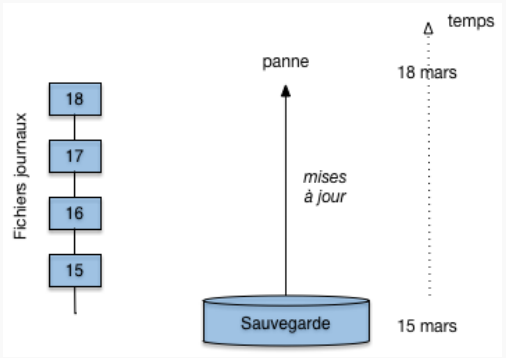
Une sauvegarde S_t est une copie de l'état de la base à un instant t.



Avec S_t, il suffit de réappliquer le *log* depuis t.

Illustration

Avec une sauvegarde datant du 15 mars, et un fichier *log* par jour.



On applique à la sauvegarde les transactions depuis le 15 mars.

Cas 2 : panne du disque du *log*

Rappel : l'état de la base est dans les fichiers et le cache.

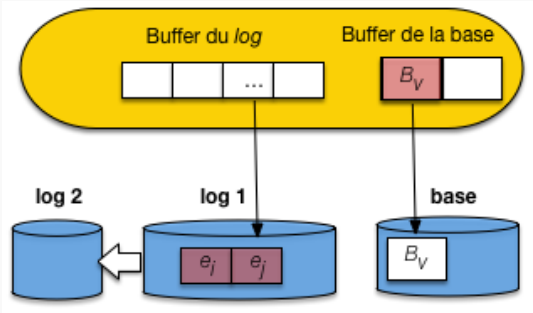
On gère la panne en effectuant un *checkpoint*

- on écrit les blocs modifiés sur le disque de la base,
- on fait les réparations nécessaires et on redémarre.

Assez fragile : *quid* en cas de panne électrique et panne du disque de *log* ?

Cas 2, avec réplication du *log*

On peut répliquer le *log* pour plus de sûreté.



Et pourquoi ne pas répliquer la base ? On obtient un système distribué.

Résumé : les pannes de disque

Retenir :

- Le *log* et la base doivent être sur des disques distincts.
- Il faut mettre en place une politique de sauvegarde.
- La reprise combine choix de la sauvegarde et application du *log*.

En principe, on garantit la durabilité.

Sûreté totale ?

- Obtenue (asymptotiquement) par réplication.
- Une des motivations des systèmes distribués.