

Module BDR Master d'Informatique (SAR)

Cours 9- bases de données parallèles
Stéphane Gançarski
Stephane.Gancarski@lip6.fr

1

Plan

Introduction
Architectures
Placement des données
Parallélisme dans les requêtes
Optimisation de requêtes

2

Introduction

- Les machines parallèles deviennent courantes et abordables
 - Les prix des microprocesseurs, de la mémoire et des disques ont fortement diminué
- Les BD sont de plus en plus volumineuses (ex. entrepôts de données, multimedia, ...)
- Principe des BD parallèles :
 - utiliser les machines parallèles (systèmes distribués) pour exploiter le parallélisme dans la gestion de données.

3

Parallélisme dans les BD

- Les données peuvent être partitionnées sur plusieurs disques pour faire des entrées/sorties en parallèle.
- Les opérations relationnelles (tri, jointure, agrégations) peuvent être exécutées en parallèle
 - Les données peuvent être partitionnées, et chaque processeur peut travailler indépendamment sur sa partition
- Les requêtes sont exprimées dans un langage de haut niveau (SQL), et traduites en opérateurs algébriques.
 - Facilite le parallélisme
- Les requêtes peuvent être exécutées en parallèle

4

Objectifs des BD parallèles

- Améliorer les performances
- Améliorer la disponibilité
- Réduire les coûts (utiliser plusieurs petites machines est moins cher qu'un gros ordinateur).

Approche cluster : réduire les coûts (utiliser 100 petites machines)

• Les SGBD parallèles sont utilisés pour :

- Stocker de très grandes quantités de données
- Effectuer des requêtes d'aide à la décision coûteuses
- Permettre le traitement de transactions à haut débit

• Approche Cloud : "Pay as you go" / le changement de modèle économique

5

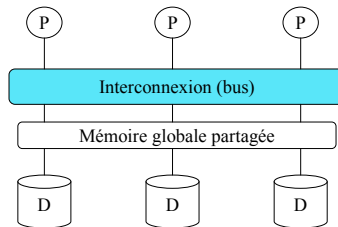
Architectures de BD parallèles

- Se classent selon ce qui est partagé :
 - Shared Memory Systems (partage de la mémoire)
 - Shared Disk Systems (partage du disque)
 - Shared Nothing Systems (pas de partage) – *sauf bus rapide*
 - Hybrid Systems (systèmes hybrides)

6

Mémoire partagée

- Tous les processeurs partagent le disque et la mémoire



7

Mémoire partagée

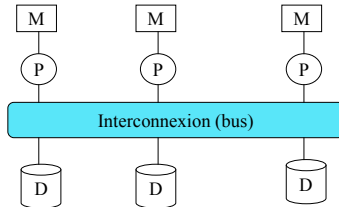
- + simplicité (*1 annule si une seule mem.*)
- + équilibrage de charge (*n'importe quel proc.*)
- + parallélisme inter-requête (ajout de processeur), direct, intra-requête possible
- Peu extensible : limité à une dizaine de processeurs → *accès concurrents à mem*
- Cher : chaque processeur est lié à chaque module de mémoire
- Conflits d'accès à la mémoire (dégrade les performances)
- Pb de disponibilité des données en cas de défaut de mémoire

Utilisé dans plusieurs systèmes commerciaux : Oracle, DB2, Ingres

8

Disques partagés

- Ensemble de noeuds (processeurs et mémoire) ayant accès via un réseau d'interconnexion à tous les disques (les mémoires ne sont pas partagées).



Chaque noeud a sa mémoire et ils se partagent les disques.

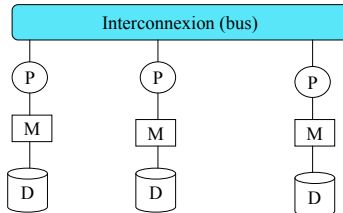
Disques partagés

- + moins cher que les systèmes à mémoire partagée.
- + extensibilité (centaines de processeurs) : les processeurs ont suffisamment de mémoire cache pour minimiser les interférences sur le disque.
- + bonne disponibilité (les défauts de mémoire sont locaux à un processeur)
- + migration facile de systèmes existants (pas de réorganisation des données sur le disque)
- Assez complexe
- Pbs de performance :
 - Verrous distribués pour éviter les accès conflictuels aux mêmes pages,
 - maintien de la cohérence des copies,
 - l'accès aux disques partagés peut devenir un goulot d'étranglement
- Utilisé dans IMS/VS(IBM), VAX (DEC)

10

Aucun partage

- Les différents noeuds sont reliés par un réseau d'interconnexion. Chaque processeur a un accès exclusif à la mémoire et au disque (idem BD réparties).



11

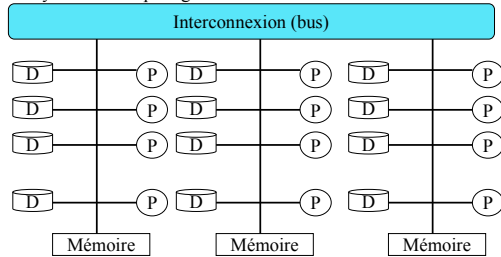
Aucun partage

- + Coût semblable à celui des disques partagés
- + Bonne extensibilité (milliers de noeuds)
- + Disponibilité par réplication des données
- Bien adapté au parallélisme intra-requête
- Plus complexe que les systèmes à mémoire partagée (mêmes fct. qu'en BD réparties)
- Répartition de charge difficile à mettre en oeuvre, à cause de la répartition statique des données
- Ajouter des noeuds entraîne une réorganisation des données (pour permettre la répartition de charge)
- Utilisé dans ICL Goldrush, Teradata DBC, EDS, ...

12

Systèmes hybrides

- Combinaison de systèmes à mémoire partagée et de systèmes sans partage.



13

Systèmes hybrides

- Chaque noeud individuel est un système à mémoire partagée.
- + Combine la souplesse et les performances des systèmes à mémoire partagée avec les capacités d'extensibilité des systèmes sans partage.
- + La communication au sein de chaque noeud est efficace (mémoire partagée).
- + Bon compromis entre répartition de charge et passage à l'échelle.

14

Partitionnement des données

Compromis entre coût du partitionnement et gain du partitionnement.

- Répartir les données sur les disques permet de réduire le temps d'accès aux données.
- Partitionnement horizontal : les n-uplets d'une relation sont répartis sur les différents disques (pas de partitionnement de n-uplet). *→ Seul pour OLAP*
- Différentes techniques de partitionnement (nb de disques = n) :
 - Round-robin** : le $i^{\text{ème}}$ n-uplet inséré dans la relation est stocké sur le disque $i \bmod n$
 - Partitionnement par hachage (hash partitioning)** : *→ la fonction renvoie le processeur sur lequel envoyer les données*
 - Choisir un ou plusieurs attributs comme attributs de partitionnement
 - Appliquer une fonction de hachage (renvoyant un entier de 0 à n-1) à ces attributs
 - Stocker le n-uplet sur le disque correspondant au résultat de la fonction de hachage

15

Partitionnement des données (suite)

- Partitionnement par intervalles (range partitionning)** : répartit les n-uplets en fonction des intervalles de valeurs d'un attribut
 - Choisir un attribut pour le partitionnement
 - Choisir un vecteur de partitionnement $[v_0, v_1, \dots, v_{n-2}]$
 - Soit v la valeur de l'attribut de partitionnement.
 - Les n-uplets tq $v < v_0$ vont sur le disque 0
 - Les n-uplets tq $v \geq v_{n-2}$ vont sur le disque n-1
 - Les n-uplets tq $v_i < v \leq v_{i+1}$ vont sur le disque $i+1$

16

Comparaison

- Comparer sur les opérations :
 - Parcours de la relation (scan)
 - Sélection sur égalité (ex: $R.A=15$)
 - Sélection sur intervalles (ex: $10 \leq R.A < 25$)
- **Round-robin** :
 - Convient bien au parcours de relations
 - Bonne répartition des n-uplets sur les disques (bonne répartition de charge)
 - Mal adapté aux requêtes avec sélection (pas de regroupement, les données sont dispersées)

17

Comparaison (suite)

Partitionnement par hachage

- Efficace pour les accès séquentiels
 - Si on a une bonne fonction de hachage, et si le(s) attribut(s) de partitionnement est (sont) clé, il y a une bonne répartition des n-uplets sur les disques, et une bonne répartition de la charge pour rechercher les données
- Efficace pour les sélections par égalité sur l'attribut de partitionnement
 - La recherche peut se limiter à un seul disque
 - Possibilité d'avoir un index local sur l'attribut de partitionnement
- Mal adapté aux sélections sur intervalles, car il n'y a pas de regroupement.

18

Comparaison (suite)

Partitionnement par intervalle

- Les données sont regroupées par valeurs de l'attribut de partitionnement
- Efficace pour les accès séquentiels
- Efficace pour les sélections par égalité sur l'attribut de partitionnement
- Bien adapté aux sélections sur intervalles (seul un nombre limité de disques est concerné)
 - Efficace si les n-uplets du résultat se trouvent sur peu de disques
- Risque de répartition inégale sur les disques

19

Partitionnement d'une relation

- Ne pas partitionner une relation qui tient sur un seul disque
- Utiliser l'ensemble des disques pour un partitionnement
- Ne pas partitionner plus que nécessaire : si une relation tient sur m blocs et qu'il y a n disques, partitionner sur $\min(n, m)$ disques

20

Gestion des répartitions non uniformes

- Certains algorithmes de répartition risquent de donner lieu à une répartition inégale des données (bcp sur un disque, peu sur un autre), ce qui peut dégrader les performances:
 - Mauvais choix de vecteur de partition
 - Mauvais choix de fonction de hachage
 - Répartition inégale des valeurs d'un attribut de partitionnement
- Solutions (partitionnement par intervalles) :
 - Créer un vecteur de partitionnement équilibré (trier la relation sur l'attribut de partitionnement et diviser en sous-ensembles de même taille)
 - Utiliser des histogrammes

21

Parallélisme inter-requêtes

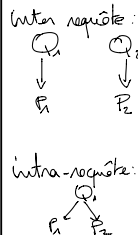
- Forme la plus simple du parallélisme
- Les requêtes (et les transactions) s'exécutent en parallèle
- Augmente le débit de transactions (utilisé pour augmenter le nombre de transactions par seconde)
- Plus difficile à implémenter sur les architectures disque partagé et sans partage
 - La coordination des verrouillages et des journaux s'effectue par envois de messages entre processeurs
 - Les données d'un buffer local peuvent avoir été mises à jour sur un autre processeur
 - Nécessité de maintenir la cohérence des caches (les lectures et écritures dans le buffer doivent concerner la version la plus récente.)

22

Cohérence du cache

- Protocole pour les architectures à disques partagés :
 - Avant de lire/écrire une page, la verrouiller en mode partagé/exclusif
 - Une page verrouillée doit être lue du disque
 - Avant de déverrouiller une page, elle doit être écrite sur le disque (si elle a été modifiée)
- Des protocoles plus complexes existent, limitant les lectures/écritures sur disque.
- Des protocoles semblables existent pour les architectures sans partage.

23



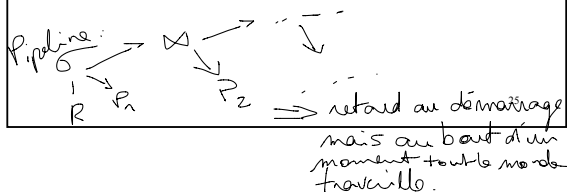
Parallélisme intra-requêtes

- Exécuter une seule requête en parallèle sur plusieurs processeurs (important pour les requêtes longues)
- Deux formes de parallélisme :
 - **Inter-opérateurs** : exécuter plusieurs opérations en parallèle
 - **Intra-opérateurs** : paralléliser l'exécution de chaque opération dans la requête

24

Parallélisme inter-opérateurs

- Deux formes :
 - **Pipeline** : plusieurs opérateurs sont exécutés en parallèle, le résultat intermédiaire n'est pas matérialisé. Gain de place mémoire et minimise les accès disque
 - **Parallélisme indépendant** : les opérateurs sont indépendants, pas d'interférence entre les processeurs. Le résultat est matérialisé.



Parallélisme intra-opérateurs

- Décomposer un opérateur en un ensemble de sous-opérateurs, chacun s'exécutant sur une partition de la relation. *la partition juste pour exécuter la req*
- Exemple pour la sélection :

$$\sigma(R) \equiv \sigma_{S_1}(R_1) \sigma_{S_2}(R_2) \dots \sigma_n(R_n)$$
- Pour les jointures, on utilise les propriétés du partitionnement : par ex. si R et S sont partitionnées par hachage sur l'attribut de jointure, et s'il s'agit d'une équi-jointure, on peut partitionner la jointure en deux jointures indépendantes.

26

Traitement parallèle des opérateurs relationnels

- Suppositions :
 - Requêtes en lecture seule
 - Architecture sans partage
 - N processeurs, et N disques (D_i est associé à P_i)
- Les architectures sans partage peuvent être simulées efficacement sur des architectures à mémoire partagées ou à disques partagés. Les algorithmes peuvent être appliqués, avec différentes optimisations, sur ces architectures.

27

Tri parallèle

Range-partitioning sort :

- Choisir les processeurs P_0, \dots, P_m , avec $m < n$, pour faire le tri.
- Créer des vecteurs de partition par intervalle avec m entrées, sur l'attribut de jointure
- Redistribuer la relation selon cette partition :
 - tous les n-uplets du $i_{\text{ème}}$ intervalle sont envoyés au processeur P_i
 - P_i stocke temporairement les n-uplets sur son disque D_i
- Chaque processeur trie sa partition de la relation localement (en parallèle, sans interaction avec les autres processeurs)
- On fusionne les différents résultats.

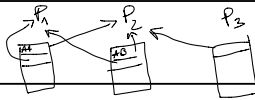
28

Tri parallèle

Trifusion parallèle externe :

- La relation est partitionnée sur les n disques. (Rond de bois)
- Chaque processeur trie localement ses données.
- La fusion est parallélisée :
 - Les partitions triées sur chaque processeur sont partitionnées par intervalle sur les différents processeurs (utiliser le même vecteur de partition)
 - Chaque processeur fusionne les données qu'il reçoit à la volée.
 - Les différents résultats sont concaténés

29



Jointure par partitionnement

attention : courbes

- Partitionner les deux relations (par hachage ou par intervalle) en n partitions, sur l'attribut de jointure, en utilisant la même fonction de hachage ou le même vecteur de partitionnement.
- Les partitions r_i et s_i sont envoyées au processeur P_i .
- Chaque processeur calcule la jointure entre r_i et s_i (avec n'importe quel algorithme)
- Les résultats sont concaténés.

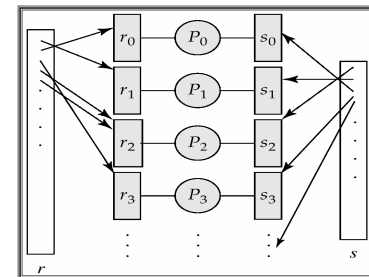
31

Jointures parallèles

- Principes : Partitionner puis regrouper les rs.
 - La jointure consiste à tester les n -uplets deux par deux et à comparer la valeur des attributs de jointure.
 - Le parallélisme consiste à répartir ces tests sur des processeurs différents, chacun calculant une partie de la jointure localement.
 - Les résultats sont ensuite rassemblés.

30

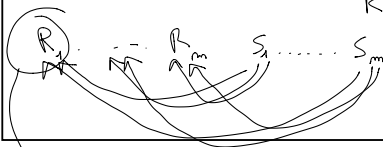
Jointure par partitionnement



32

Boucles imbriquées en parallèle

- Les relations R et S sont fragmentées en m et n fragments respectivement.
- Envoyer les m fragments de R sur les n nœuds où se trouvent des fragments de S (en parallèle)
- Faire la jointure sur ces nœuds (en parallèle)
- Concaténer les résultats.



$$R = \bigcup_i R_i$$

$$\Rightarrow R_1 \bowtie S, R_2 \bowtie S, \dots, R_m \bowtie S = R \bowtie S$$

33

Jointure parallèle par hachage

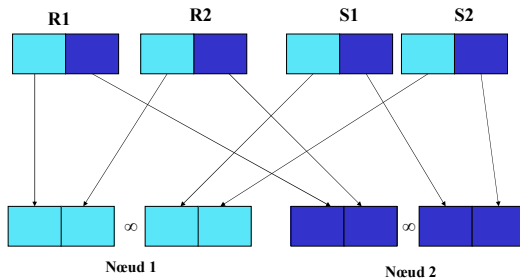
- Partitionner (en parallèle) les relations R et S en k partitions à l'aide d'une fonction de hachage h appliquée à l'attribut de jointure tq

$$R \bowtie S = \bigcup_{(i=1 \dots k)} (R_i \bowtie S_i)$$

- Les partitions de R et de S ayant même valeur de hachage sont envoyées sur le même processeur.
- Faire les jointures sur chaque processeur en parallèle
- Concaténer les résultats

34

Jointure parallèle par hachage



35

Optimisation de requêtes

- Semblable à l'optimisation de requêtes dans les BD réparties.
- Espace de recherche
 - Les arbres algébriques sont annotés pour permettre de déterminer les opérations pouvant être exécutées à la volée (pipeline). Génération d'arbres en profondeur gauche, droite, en zig-zag, touffus.
- Modèle de coût
 - Dépend en partie de l'architecture
- Stratégie de recherche
 - Les mêmes qu'en environnement centralisé (les stratégies aléatoires sont mieux adaptées en raison de la taille de l'espace de recherche)

36

Conclusion

- Les BD parallèles améliorent
 - Les performances,
 - La disponibilité
 - L'extensibilité

Tout en réduisant le ratio prix/performance.

Plusieurs systèmes commercialisés.

Pbs :

- choisir la meilleure architecture
- améliorer les techniques de placement de données (éviter les répartitions inégales)