Module BDR
Master d'Informatique

Cours 6- Interrogation de bases de données réparties

Stephane Gançarski
Stephane.Gancarski@lip6.fr

Interrogation de Bases de Données réparties

- Transparence des requêtes
- Evaluation des requêtes
- Fragmentation de requêtes
- Optimisation de requêtes

Transparence de la répartition

- Transparence de la répartition : degré d'intégration du schéma
  - haut niveau de transparence : pas d'information sur la fragmentation
  - bas niveau de transparence : l'utilisateur doit spécifier les fragments qu'il manipule => pb de nommage non ambigu.
- Chaque item de la BD doit avoir un nom unique
  - 1. serveur de noms : attributs, relations, utilisateurs, …
    - goulot d'étranglement, pb en cas de panne, peu d'autonomie locale
  - 2. Préfixer par le site + serveur de nom local

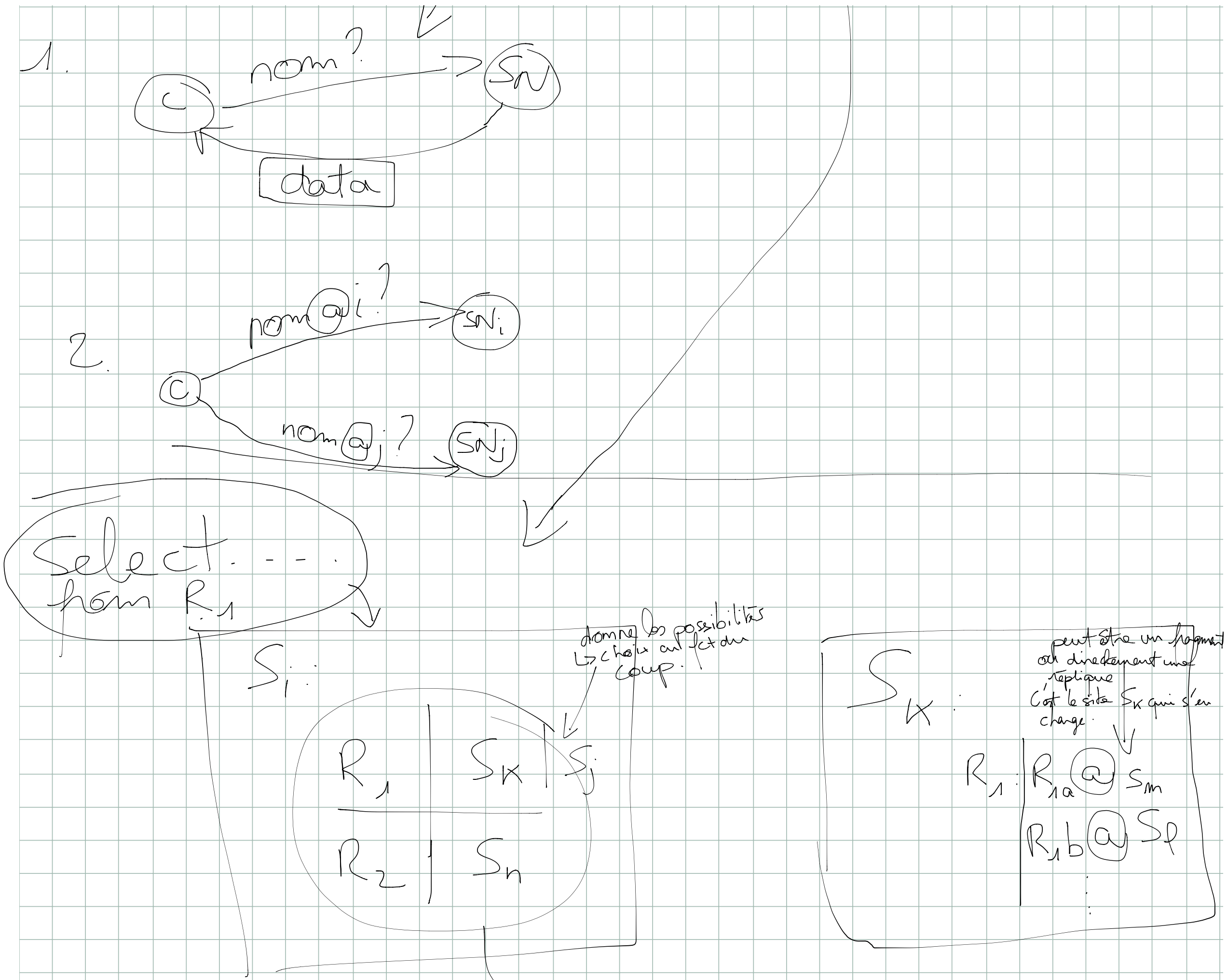
peu de maj → réplication

Transparence des requêtes

- Le système doit optimiser les lectures/écritures, effectuer automatiquement les mises à jour des répliques, optimiser les requêtes.
- Où se trouve le schéma de répartition (table des fragments, des répliques)?
  1. Chaque site a une copie de tout le schéma :
     + : lectures rapides
     - : modifications de schémas

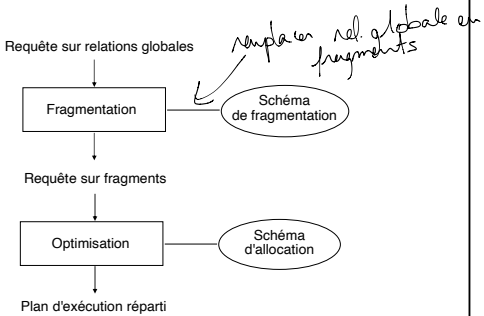
  2. Chaque site a un schéma local
     - : il faut retrouver les informations sur les autres sites

  3. Solutions mixtes : seuls certains sites ont une copie du schéma, certains sites conservent les informations concernant un item, etc.

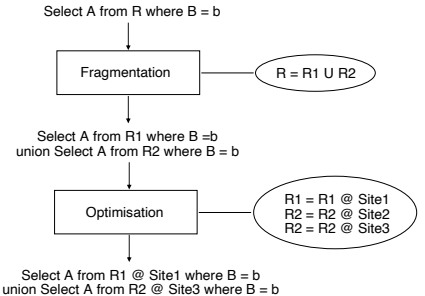


On réplique ça sur plusieurs sites.

Evaluation de Requêtes Réparties



Exemple d'évaluation simple



Fragmentation

- Réécriture
  - mettre la requête sous forme d'un arbre algébrique (feuille = relation, noeud = op. relationnel)
- Reconstruction
  - remplacer chaque feuille par le programme de reconstruction de la relation globale
- Transformation
  - appliquer des techniques de réduction pour éliminer les opérations inutiles

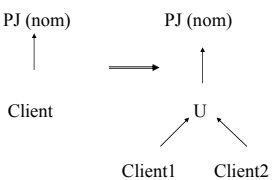
- Notations utilisées :
  - SL : select
  - JN : join
  - PJ : project

Reconstruction

Select nom from Client

Client1 = Client where ville = Paris

Client2 = Client where ville not= Paris



Réduction pour la fragmentation horizontale

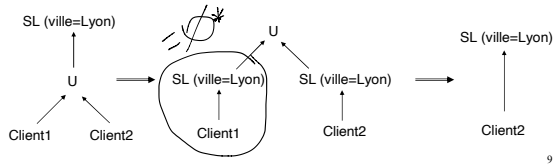
Règle : éliminer l'accès aux fragments inutiles

Exemple :

Client1 = Client where ville = Paris

Client2 = Client where ville not= Paris

Select * from Client where ville=Lyon



$\sigma_{ville='Lyon'}(\sigma_{ville='Paris'}(Client))$

$\equiv \sigma_{ville='Lyon' \wedge ville='Paris'}(Client)$

faux !

Réduction pour la Fragmentation Verticale

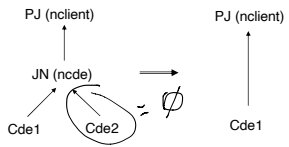
Règle : éliminer les accès aux relations de base qui n'ont pas d'attributs utiles pour le résultat final

Exemple $Cde = Cde_1 \bowtie Cde_2$

Cde1 = Cde (ncode, nclient) $= \Pi_{ncode, client}(Client)$

Cde2 = Cde (ncode, produit, qté) $= \Pi_{ncode, pd, qte}(Client)$

Select nclient from Cde



Réduction pour la Fragmentation Horizontale Dérivée

Règle : distribuer les jointures par rapport aux unions et appliquer les réductions pour la fragmentation horizontale

Exemple

Client1 = Client where ville=Paris

Client2 = Client where ville not Paris

Cde1 = Cde where Cde.nclient=Client1.nclient

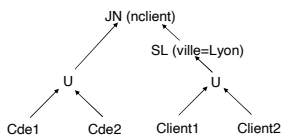
Cde2 = Cde where Cde.nclient=Client2.nclient

Select all from Client, Cde

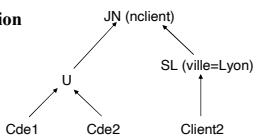
where Client.nclient = Cde.nclient and ville=Lyon

Exemple

Requête canonique

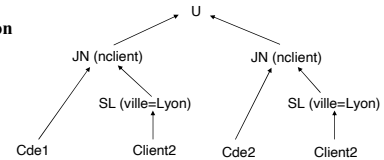


Après 1ère réduction

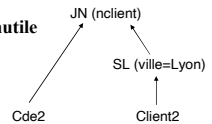


Exemple (suite)

Distribution



Elimination du sous-arbre inutile



Optimisation de Requêtes Réparties

entrée : une requête simplifiée exprimée sur des fragments
sortie : un plan d'exécution réparti optimal

- Objectifs
  - choisir la meilleure localisation des fragments
  - minimiser une fonction de coût: f (I/O, CPU, Comm.)
  - exploiter le parallélisme → tps de rep
  - exprimer les transferts inter-sites → occupat° des ressources
- Solution
  - examiner le coût de chaque plan possible (par transformation) et prendre le meilleur

+ rapide ✗ – gourmande

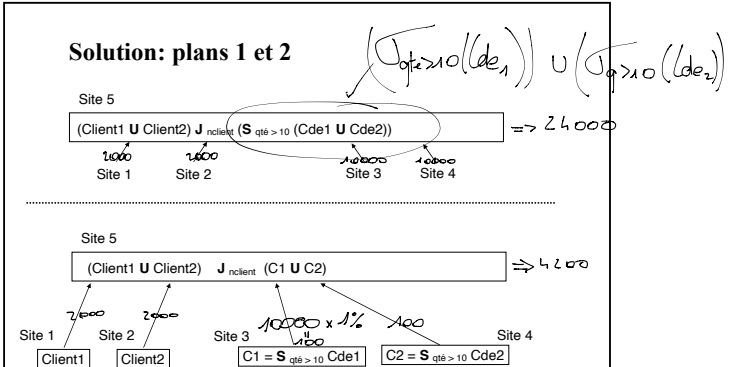
Map Reduce : Métrique qui fait payer les accès selon les nœuds utilisés.

Exemple

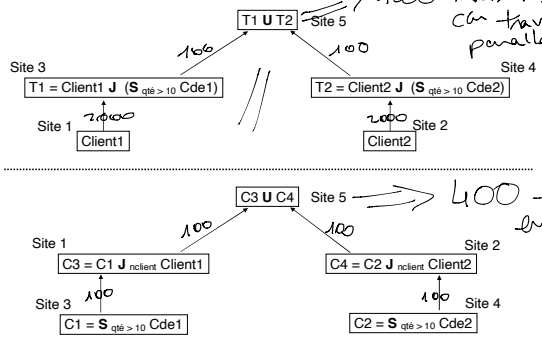
Site 1 : Client1 = Client where ville=Paris
Site 2 : Client2 = Client where ville not Paris
Site 3 : Cde1 = Cde where Cde.nclient=Client1.nclient
Site 4 : Cde2 = Cde where Cde.nclient=Client2.nclient
Site 5 : résultat

Select all from Client, Cde
where Client.nclient = Cde.nclient
and qté > 10

Solution: plans 1 et 2



Solutions: plans 3 et 4



Coût des Solutions

Supposons
- taille (Cde1) = taille (Cde2) = 10 000
- taille (Client1) = taille (Client2) = 2 000
- coût de transfert de 1n-uplet = 1
- Sélectivité(*qté>10*) = 1%

Stratégie 1
- transfert de Cde1 + Cde2 = 20 000
- transfert de Client1 + Client2 = 4000

Stratégie 2
- transfert de Client1 + Client2 = 4000
- transfert de C1 + C2 = 200

Stratégie 3
- transfert de Client1 + Client2 = 4000
- Transfert de T1 + T2 = 200

Stratégie 4
- transfert de C1 + C2 = 200
- transfert de C3 + C4 = 200

Jointure

R sur S1, S sur S2, T sur S3.
Requête demandée sur le site S0 : R ⋈ S ⋈ T

Plusieurs possibilités :
a) Copier tout sur S0 et faire les jointures sur S0
b) Copier R sur S2, et joindre R et S sur S2
   Copier le résultat sur S3, et faire la jointure avec T sur S3
   Copier le résultat sur S0

Tenir compte des index, de la taille des relations à transférer, de la taille des relations intermédiaires, de la charge des sites, de la vitesse de transmission, etc.
On peut paralléliser les jointures : grand nombre de stratégies

Semi-jointure

Traiter la jointure entre deux relations réparties sur 2 sites
R1 sur S1, R2 sur S2 Rappel: $R1 \ltimes R2 = \Pi_{R1}(R1 \bowtie R2)$

Requête $R1 \bowtie R2 = R1 \bowtie (R2 \ltimes R1)$ et résultat sur S1
- $T1 = \Pi_A(R1)$ sur S1, puis envoi de T1 sur S2
- $T2 = R2 \ltimes T1$ sur S2, puis envoi de T2 sur S1
- Calcul de $R1 \bowtie T2$ sur S1

Requête $R1 \bowtie R2 = (R1 \ltimes R2) \bowtie (R2 \ltimes R1)$ et résultat sur S0
- Transférer $T1 = \Pi_A(R1)$ de S1 sur S2
- Transférer $T2 = \Pi_A(R2)$ de S2 sur S1
- Transférer $T3 = R1 \ltimes R2$ de S1 sur S0
- Transférer $T4 = R2 \ltimes R1$ de S2 sur S0
- Sur S0, $T3 \bowtie T4$

Algorithme d'optimisation R* (System R)

- Choix de la meilleure stratégie dans la liste exhaustive des stratégies possibles.
- Le site maître (où la requête est posée) prend toutes les décisions globales :
  - sélection des sites où exécuter les requêtes
  - choix des fragments
  - choix des méthodes de transferts
- Les autres sites intervenant dans la requête prennent les décisions locales :
  - ordre des jointures locales
  - génération des plans d'accès locaux
- Fonction de coût totale incluant le coût des traitements locaux, et les coûts de transmission.

Algorithme R*

Input : arbre de requête, localisation des relations, statistiques

Output : stratégie d'exécution la moins chère

Pour chacune des relations
- déterminer le coût de chacun des accès possibles (index, accès séquentiel, etc.)
- prendre le moins cher

Calculer le coût de chacune des combinaisons possibles et prendre le plus faible.

Pour chaque site intervenant dans la requête, calculer le plan d'exécution local le moins cher.

Optimisation

L'optimiseur choisit (à l'aide de statistiques et de fonctions de coût)
- l'ordre des jointures
- l'algorithme de jointure
- le chemin d'accès
- les sites des résultats intermédiaires
- la méthode de transfert de données
  - Tout envoyer (bien pour les petites relations)
  - Envoyer uniquement les n-uplets utiles : on envoie la valeur de jointure sur le site de la relation interne, et on renvoie uniquement les n-uplets correspondants. (bien si bonne sélectivité)

Stratégies de jointure

4 stratégies possibles pour effectuer R ⋈ S sur l'attribut A, avec l'algorithme des boucles imbriquées (R relation externe, S relation interne).

On note le nombre moyen de n-uplets de S joignant un n-uplet de R.

$$s = \frac{Card\ (S \ltimes_A R)}{Card\ (R)}$$

On note LC le coût du traitement local, et CC le coût de communication.

1. Envoyer R (relation externe) sur le site de S (relation interne)

Les n-uplets de R sont joints au fur et à mesure de leur arrivée sur le site de S

Coût total = LC (retrouver *card* (R) n-uplets de R)
+ CC (*size* (R))
+ LC (retrouver *s* n-uplets de S) * *card* (R)

Stratégies de jointure (2)

2. Envoyer S (relation interne) sur le site de R (relation externe).

(les n-uplets internes ne peuvent pas être joints au fur et à mesure de leur arrivée sur le site, et sont stockés dans la relation temporaire T).

Coût total = LC (retrouver *card(S)* n-uplets de S)
+ CC (*size*(R))
+ LC (stocker *card(S)* n-uplets dans T)
+ LC (retrouver *card(R)* n-uplets de R)
+ LC (retrouver *s* n-uplets de T) * *card(R)*

Stratégies de jointure (3)

3. Chercher les n-uplets de S (relation interne) nécessaires pour chaque n-uplet de R (relation externe).

La valeur de l'attribut de jointure de chaque n-uplet de R est envoyée sur S. Les *s* n-uplets de S correspondant à cette valeur sont envoyés sur le site de R et joints.

Coût total = LC (retrouver *card* (R) n-uplets de R)
+ CC (*length(A)*) * *card(R)*
+ LC (retrouver *s* n-uplets de S) * *card* (R)
+ CC(*s * length(S))* * *card (R)*

Stratégie de jointure (4)

4. Transférer les deux relations sur un troisième site et calculer la jointure sur ce site.

S (relation interne) est transférée en premier, et stockée dans T. Les n-uplets de R sont joints au fur et à mesure de leur arrivée.

Coût total = LC (retrouver *card(S)* n-uplets de S)
+ CC (*size*(S))
+ LC (stocker *card(S)* n-uplets dans T)
+ LC (retrouver *card* (R) n-uplets de R)
+ CC (*size* (R))
+ LC (retrouver *s* n-uplets de T) * *card(R)*

Conclusion

- Importance de la fragmentation horizontale pour augmenter le parallélisme inter- et intra-requête
- Utiliser la fragmentation verticale si forte affinité d'attributs et beaucoup d'attributs
- L'optimisation reposant sur un modèle de coût est critique s'il y a beaucoup de sites