SIABD : Conception physique d'une base de données relationnelle

Clément AMIOT

October 19, 2011

Contents

Part I

Exercice sur les opérations de tri

1 Question 1

Soit une relation R(S,T,U,V,W), contenant n tuples, stockée sur p pages, ayant un index (B+) plaçant sur S et deux index (B+) non plaçant sur U et V. Le nombre de pages de chaque index est respectivement : nbpi(S), nbpi(U), nbpi(V). Le nombre de pages feuilles de chacun des index est respectivement nbf(S), nbf(U), nbf(V).
Quel est le cout respectif en E/S pour lire tous les tuples de R triés dans l'ordre de S, dans l'ordre de U et dans l'ordre de T ?

Réponses :

- dans l'ordre de S : nombre de pages P
- dans l'ordre de U : nombre de feuilles de l'index sur U + n
- dans l'ordre de T : 2P * log_b P + P_{seq}

2 Question 2

Soit les données suivantes : b = 10

	Petite relation	*Moyenne relation*
p	100	1000
n	1000	10000

Calcul du nombre de feuilles de l'index non plaçant sur U Au niveau des feuilles de l'index, pour un index non plaçant, on a une entrée d'index par n-uplet. C'est le cas si U est une clé de la relation. C'est le cas aussi si U est un attribut dupliqué (plusieurs n-uplets peuvent avoir la même valeur de U), mais que l'on gère l'index de façon très simple avec des entrées d'index de longueur fixe (longueur clé +longueur numéro de page). L'autre façon de gérer l'index, dans le cas d'un attribut clé d'accès dupliqué est d'avoir dans les feuilles de l'index des entrées d'index de longueur variable contenant (valeur de clé, suite de numéro de page) avec un numéro de page pour chaque n-uplet ayant la même valeur de clé. Dans la suite pour simplifier les calculs on suppose que l'index contient des entrées de longueur fixe.

- Pour les petites relations :
  - Pour une clé de 4 caractères :
    Chaque entrée d'index contient la valeur de la clé sur 4 caractères et l'adresse de la page sur 4 caratères. Une entrée fait donc 8 caractères.
    Une page = 2000

Nombre d'entrées d'index par page : 2000 / 8 = 250. On peut donc mettre 250 entrées par page, mais c'est un index B+ Donc il y a de la place utilisée en plus au début de la page, et toutes les pages ne sont pas remplies complètement à cause des suppressions et des éclatements.

On peut dire que : 1000 / 250 = 4 pages et on ajoute environ 20% pour le fait que c'est un B+, soit environ 5 pages pour le nombre de feuilles de l'index sur U avec U petite clé.

- Pour une clé de 36 caractères :
  Chaque entrée contient la valeur de la clé sur 36 caractères et l'adresse de la page sur 4 caratères. Une entrée fait donc 40 caractères.
  Une page = 2000
  Nombre d'entrées d'index par page : 2000 / 40 = 50. On peut donc mettre 50 entrées par page, mais c'est un index B+ Donc il y a de la place utilisée en plus au début de la page, et toutes les pages ne sont pas remplies complètement.

  On peut dire que : 1000 / 50 = 20 pages et on ajoute environ 20% pour le fait que c'est un B+, soit environ 24 pages pour le nombre de feuilles de l'index sur U, avec U grandes clé.

- Pour les moyennes relations : Même raisonnement mais en changeant les valeurs de p et n

Tableau récapitulatif :

	Petite relation		Moyenne relation	
	Petite clé	Grande clé	Petite clé	Grande clé
Lecture de tous les tuples de S (1)	100	100	1000	1000
Lecture dans l'ordre de U (2)	1005	1024	10048	10240
Lecture dans l'ordre de T (3)	480	480	6800	6800

Avec (1) : P, (2) : nbf(U) + n et (3) : 2P log_b P + lect P_{seq}

3 Question 3 - Relation placée en aléatoire

Soit une relation R (S, T, U, V, W) placée en aléatoire sur S dans un segment de p pages.
Quel est le coût respectif en E/S pour lire tous les tuples de R triés dans l'ordre de S, dans l'ordre de T ?

Correction :

Pour lire les tuples de R triés dans l'ordre de S ou dans l'ordre de T, il faudra, dans les deux cas trier la relation dans l'orde de S (respectivement dans l'ordre de T), puis lire le résultat du tri. La formule est la formule (3) du tableau de la question 2, mais les

chiffres devraient être un peu plus élevés car une relation placée en aléatoire occupe plus de pages qu'une relation placée avec un index plaçant (en général de l'ordre de 20% de plus).

Part II

TD Algorithmes de jointure

4 Question 1 - Relations séquentielles

On considère deux relations R et S implantées séquentiellement et ne possédent aucun chemin d'accès privilégié. L'algorithme le plus simple pour exécuter la jointure de R et S sur les attributs R.a et S.b consiste à comparer chaque n-uplet de R à tous les n-uplets de S puis à concaténer les couples de n-uplets des deux relations satisfaisant le prédicat R.a=S.b afin de constituer le résultat. Cet algorithme est communément appelé jointure par produit cartésien ou encore jointure par boucle imbriquées.

- On suppose que les deux relations à joindre ont une taille supérieure à la taille de l'espace mémoire aloué à l'opérateur de jointure. Donner l'algorithme du produit cartésien tirant partie du tampon MC de b + 2 pages
- Calculer le nombre d'E/S et le nombre de comparaisons générées par cet algorithme. Les résultats obtenus sont-ils équivalents suivant que l'on exécute la jointure RxS ou la jointure SxR ? Que faut il conclure ?

Correction :

- Idée de l'algorithme de jointure : On utilise un tampon d'une page pour écrire le résultat. Dans le tampon de b pages, on met b pages de la relation 1 et on compare les tuples s'y trouvant avec les tuples de la relation 2 que l'on lira page à page dans un autre tampon d'une page

Algorithme de jointure

```
Tant que non fin de relation 1 faire :
        /*remplir le tampon de b pages */
        tant que non fin tampon et non fin relation 1
                lire page suivante (relation 1, fin relation1) dans T1
        fin tant que
        tant que non fin de relation 2
                lire page suivante (relation 2, fin relation 2) dans T2
                /*jointure des tuples dans T1 et T2*/
                tant que non fin T1
                        lire tuple suivant de T1
                        tant que non fin de T2
                                lire tuple suivant de T2
                                comparer
                                si ils joignent ecrire dans T3 le résultat
                        fin tant que
                fin tant que
        fin tant que
fin tant que
vider le tampon d'écriture T3
```

- Performances : nombre de lectures et de comparaisons.

$$\boxed{P_{rel1} + \lceil \frac{P_{rel1}}{b} \rceil * P_{rel2}}$$

Lire la plus petite relation dans le tampon de b pages est donc plus intéressant. Le nombre de comparaison est le même quelle que soit la relation lue dans le tampon de b pages. On compare toujours chacun des tuples de rel1 avec tous ceux de rel2, soit $n_{rel1} * n_{rel2} = 10000000$ de comparaisons.
Quand $P_{rel1} < b$, le coût en nombre de lectures devient $P_{rel1} + P_{rel2}$, ce qui est tout à fait comparable au coût des autres algorithmes de jointure. Mais cet algotithme reste toujours beaucoup plus coûteux, et de très loin, en nombre de comparaisons et donc en temps CPU.

5 Question 2 - Relations triées

On considère à présent que les relations R et S sont maintenues triées sur leur attribut de jointure respectif de R.a et S.b, par exemple par une organisation arborescente de type Arbre-B. L'algorithme de jointure optimal dans ce cas consiste à parcourir séquentiellement les deux relations en parallèle et à profiter de l'ordre du tri pour déterminer lors du parcours tous les couples de n-uplets des deux relations satisfaisant le prédicat R.a=S.b.

- Détailler le principe de cet algorithme, communément appelé *Jointure par tri-fusion.*
- Calculer le nombre d'E/S et le nombre de comparaisons générées par cet algorithme.

Correction :
On ne traite pas dans l'algorithme le cas des retours arrières nécessaires dans le cas où plusieurs tuples de R sont susceptibles de joindre avec les mêmes tuples de S (cas rare; ce n'est jamais le cas pour les equi-jointures entre une clé étrangère et la clé référencée qui constituent la majorité des jointures en pratique).

- **Algorithme de jointure par tri-fusion**

```
/*Lire la première page de chacune des deux relations*/
  lire page suivante (R, T1, fin R)
  lire page suivante (S, T2, fin S)
/*récupérer les deux premiers tuples de chaque tampon*/
  lire tuple suivant (R, T1, fin R)
  lire tuple suivant (S, T2, fin S)
  tant que non fin de R et non fin de S faire
        Si tuple courant T1.a < tuple courant T2.B
                lire tuple suivant (R, T1, fin R)
        Sinon
                Si tuple courant T1.a > tuple courant T2.b alors
                        lire tuple suivant (S, T2, fin S)
                Sinon /*a = b*/
                        ecrire resultat (tuple courant T1, tuple courant T2)
                        lire tuple suivant (S, T2, fin S)
                fin si
        fin si
  fin tant que
  vider le tampon d'écriture T3
```

- – **Nombre d'E/S** : On lit simplement chaque page de chaque relation (dans le cas ou il n'y a pas de retours arrière). Le nombre de lectures est donc $\boxed{P_{Rel_1} + P_{Rel_2}}$
  – **Nombre de comparaisons** : au maximum $\boxed{a_r + a_s - 1}$

  Attention ceci n'est valable que dans le cas d'équi-jointures !

Complément :

- La formule ci-dessus est valable si les deux relations à joindre ont un index plaçant sur l'attribut de jointure (souvent clé d'un côté et clè étrangère de l'autre).

- Si la relation 1 avait un index plaçant sur l'attribut de jointure et la relation 2 un index non plaçant sur l'attribut de jointure on aurait alors un nombre d'E/S égal à : $\boxed{P_{Rel_1} \ + \ NF(index \ Rel2.b) \ + \ nombre \ de \ tuples \ de \ Rel2}$
- Si la relation Rel1 a un index plaçant sur son attribut de jointure et Rel2 n'a aucun index sur son attribut de jointure, on peut la trier, et le nombre de lectures devient alors : $\boxed{P_{Rel_1} \ + \ 2P_{Rel_2} * log_b(P_{Rel_2}) \ + \ P_{seq-Rel_2}}$
- Si la relation 1 n'a pas d'index sur l'attribut de jointure, donc n'est pas triée selon l'attribut de jointure et la relation 2 a un index sur l'attribut de jointure, on peut concevoir un autre algorithme de jointure (BI-AD) tirant partie du fait que c'est la petite relation qui est non indexée sur l'attribut de jointure et la grosse relation qui a index sur l'attribut de jointure:
  - Parcourir séquentiellement la relation non indexée (R)
  - Pour chaque tuple de R, faire un accès direct à S pour trouver tous les tuples qui joignent.

  On aurait donc un nombre d'E/S de : $\boxed{P_R \ + \ a_R * nombre \ de \ niveaux \ de \ l'index}$

6 Question 3 - Relations hachées

On considère enfin que les relations R et S sont hachées (avec la même fonction de hachage sur leur attribut de jointure respectif R.a et S.b, par l'intermédiaire d'une organisation aléatoire quelconque. L'algorithme de jointure optima dans ce cas consiste à effectuer la jointure deux à deux des paquets de R et de S qui correspondent au même résultat de hachage de leur attribut de jointure respectif. La complexité de l'algorithme de jointure est ainsi diminué en remplaçant une jointure de deux relations par q jointures de paquets de faible taille, où q est le nombre de paquets de hachage commun aux deux relations. Pour chaque tuple d'une relation, il suffit alors de rechercher dans un seul paquet les tuples de l'autre relation joignant avec lui.

- Détailler le principe de cet algorithme, communément appelé *Jointure par hachage*
- Calculer le nombre d'E/S et le nombre de comparaisons générées par cet algorithme. Pour simplifier les évaluations, on supposera que la taille d'un paquet de hachage est toujours inférieure à la taille du tampon mémoire alloué à l'opérateur de jointure.

Correction :

- Principe de l'algorithme : faire une jointure par boucle imbriquée des paquets de R et de S de même numéro et ce de 0 à q-1
- – Nombre de lectures : $\boxed{P_R \ + \ P_S}$

– Nombre de comparaison : $\boxed{(\frac{a_R}{q} * \frac{a_S}{q}) \rightarrow \frac{a_R * a_S}{q}}$

Cette algorithme n'est pas très souvent applicable car les deux relations sont rarement placées avec la même fonction de hachage, surtout si elles ont des volumes disparates.

Complément :
Dans le cas où sur les deux relations à joindre l'une est petite (R) et l'autre (grosse) est placée par hachage sur l'attribut de jointure, on peut utiliser l'algorithme (BI-AD) qui donne comme nombre de lectures: $\boxed{PR + aR * 1,2}$
(1,2 est le coût moyen d'un accès direct dans un fichier aléatoire si très peu de paquets font plus d'une page).