COM S 461: ASSIGNMENT 4

Date Given: Nov. 3, 2004
Due: Nov. 15, 2004
Percentage in your final grade: 4%
Maximum score for this assignment: 100 points

Objectives:

1. To enhance your understanding of estimation of query cost.

Questions

1. (40 points) Consider a relation with this schema:

   Employee(eid:integer, ename:string, sal:integer, title:string, age:integer)

   Suppose that the following indexes, all using Alternative (2) for data entries, exist: a hash index on *eid*, a dense, unclustered B+tree index on *sal*, a hash index on *age*, and a dense, clustered B+tree index on (age, sal). Each Employee record is 100 bytes long, and each index data entry is 20 bytes long. The Employee relation occupies 10,000 pages.

   Consider each of the following selection conditions and, assuming that the reduction factor (RF) for each term that matches an index is 0.1, compute the I/O cost of the most selective access path for retrieving all Employee tuples that satisfy the condition. Use the following assumptions in your calculation.

   - Each data page can contain as many as 20 Employee tuples. Each page of Employee relation contains as many tuples as possible. The Employee relation is stored as a heap file.
   - One disk I/O is needed to retrieve a page.
   - The cost for retrieving all relevant internal nodes of a B+tree from a root to a desirable leaf node is 2 disk I/Os.
   - A page size is 2048 bytes where 48 bytes are reserved and cannot be used to store data records or data entries.
   - 1.2 I/O is needed to use a hash index to find a data entry that satisfies the selection criterion.
   - As many data entries as possible are stored in a page.

   (a) $sal > 100$
   (b) $age = 20$
   (c) $sal > 200$ and $age > 30$ and title='CFO'

   Answer:

   Table 1 summarizes storage statistics. The cost estimates are given below.

   (a) Cost of query $sal > 100$

      i. **Dense, unclustered B+ tree index on** *sal*:
      Cost of using the B+ tree index = (1) Cost of traversing from root to leaf + (2) Cost of traversing fetching matching data entries from the index + (3) Cost of fetching tuples from the relation.
      Cost (1) = 2 (given).

No. of pages	10,000 pages
No. of bytes for storage in a page	2,048 - 48 = 2,000 bytes
No. of bytes a tuple occupies	100 bytes
No. of tuples/page	$\lfloor\frac{2{,}000 \text{ bytes/page}}{100 \text{ bytes/tuple}}\rfloor$ = 20 tuples/page
No. of tuples	20 tuples/page * 10,000 pages = 200,000 tuples
No. of bytes a data entry occupies	20 bytes
No. of data entries/page in a	$\lfloor\frac{2{,}000 \text{ bytes/page}}{20 \text{ bytes/data entry}}\rfloor$ = 100 data entries/page
No. of matching tuples	0.1 * 200,000 tuples = 20,000 tuples

Table 1: Storage statistics.

Cost (2) = No. of pages containing matching data entries = $\lceil\frac{\text{No. of matching data entries}}{\text{No. of data entries/page}}\rceil$
= $\lceil\frac{20{,}000 \text{ data entries}}{100 \text{ data entries/page}}\rceil$ = 200 I/O's.
Cost (3): Since the index is dense and unclustered, each matching data entry could point to a different page on disk. Hence, the worst-case I/O cost of this operation would be equal to the no. of matching data entries, which is 20,000 I/O's.
Hence, the total cost using the B+ tree index = **2 + 200 + 20,000 = 20,202 I/O's**.

ii. **Filescan**: Cost of filescan = No. of pages in relation = **10,000 I/O's**.

Therefore, **most selective access path is Filescan**.

(b) Cost of query $age = 20$

i. **Hash index on** age:
Cost of using the hash index = (1) Cost of retreiving matching data entries from the index + (2) Cost of retreiving matching tuples from the relation.
Cost (1) = 1.2 * No. of matching tuples = 1.2 * 20,000 = 24,000 I/O's.
Cost (2): Since each matching data entry could point to a different page, the worst-case cost of retreving matching tuples is equal to the no. of matching data entries = 20,000 I/O's.
Hence, the total cost = 24,000 + 20,000 = **44,000 I/O's**.

ii. **Dense, clustered B+ tree index on** (age, sal):
Cost of using the B+ tree index = (1) Cost of traversing from root to leaf + (2) Cost of traversing fetching matching data entries from the index + (3) Cost of fetching tuples from the relation.
Cost (1) = 2 (given).
Cost (2) = No. of pages containing matching data entries = $\lceil\frac{\text{No. of matching data entries}}{\text{No. of data entries/page}}\rceil$
= $\lceil\frac{20{,}000 \text{ data entries}}{100 \text{ data entries/page}}\rceil$ = 200 I/O's.
Cost (3): Since the index is dense and clustered, all matching tuples would reside on contiguous pages on disk. Hence, the I/O cost of this operation would be equal to the no. of pages that the matching tuples occupy, which is $\lceil\frac{20{,}000 \text{ tuples}}{20 \text{ tuples/page}}\rceil = 1,000$ I/O's.
Hence, the total cost using the B+ Tree index = **2 + 200 + 1,000 = 1,202 I/O's**.

iii. **Filescan**: Cost of filescan = No. of pages in relation = **10,000 I/O's**.

Therefore, **most selective access path is the dense, clustered B+ tree index**.

(c) Cost of query $sal > 200$ and $age > 30$ and $title$='CFO'
For this query, two kinds of access paths are possible — one which uses just one index (single-index access path), and the other which uses multiple indexes (multiple-index access path).
Single-index access paths:

i. **Dense, unclustered B+ tree index on** *sal*:
Delete tuples not matching $age > 30$ and *title*='CFO' *on the fly.*
Cost: **20,202 I/O's** (this, we know from a(i)).

ii. **Dense, clustered B+ tree index on** (age, sal):
Delete tuples not matching $sal > 200$ and *title*='CFO' *on the fly.*
Cost: **1,202 I/O's** (from b(ii)).

Multiple-index access path:

i. **Use both**
(1) Dense, unclustered B+ tree index on *sal*, **and**
(2) Dense, clustered B+ tree index on (age, sal)
The idea is to first retrieve matching data entries that satisfy the query $sal > 200$ using index (1). Next, we retrieve matching data entries that satisfy the query $age > 30$ using index (2). The intersection of these two sets of data entries is taken to determine all those entries that satisfy both the conditions — $sal > 200$ and $age > 30$. This set is used to fetch the actual tuples. Tuples not matching *title*='CFO' are deleted on the fly.

Cost of this process:
(1) Cost of fetching matching data entries using index (1) +
(2) Cost of fetching matching data entries using index (2) +
(3) Cost of fetching the actual tuples. (The cost of performing the intersection is neglible.)

Cost (1): From a(i) we see that this cost equals 2 + 200 = 202 I/O's. Cost (2): From b(ii) we see that this cost equals 2 + 200 = 202 I/O's. Cost (3): The no. of matching data entries retrieved by each of the above indexes = 20,000 data entries. If we assume that the reduction factor in doing the intersection operation is 0.1, then the no. of data entries that are present in the intersection = 0.1 * 20,000 = 2,000 data entries. Cost of retreiving matching tuples = $\lceil \frac{2{,}000 \text{ tuples}}{20 \text{ tuples/page}} \rceil = 100$ I/O's.
Hence, the total cost of the multiple-index access path is 202 + 202 + 100 = **504 I/O's**.

Therefore, **most selective access path is the multiple-index access path**.

2. (60 points) Consider the join of R and S where $R.a = S.b$ given the following information about the relations to be joined. The cost metric is the number of disk I/Os and the cost of writing out the result is ignored.

- Relation R contains 10,000 tuples; each tuple is 400 bytes long.
- Relation S contains 2,000 tuples; each tuple is 400 bytes long.
- A page size is 4096 bytes and the unpacked, bitmap page format is used. For each page, 96 bytes are reserved and cannot be used to store data. The rest of the page is used to store as many tuples as possible.
- Attribute *b* of relation S is the primary key for S.
- Both relations are stored as simple heap files. Neither relation has any indexes built on it.
- The available memory buyer has 52 pages.
- The fudge factor is 1.1.

Answer the following questions and explain why:

(a) What is the cheapest cost of joining R and S using a block nested loops join for the given amount of memory buyer space? What should the number of buyer pages be so that the cost of the join is the minimum?

(b) What is the cheapest cost of joining R and S using a GRACE hash join?

(c) What is the cheapest cost of joining R and S using a sort-merge join?

Number of tuples in relation R = 10,000
Number of tuples per page in relation R = 10
Number of total number of tuples in R = 1000

Number of tuples in relation S = 2,000
Number of tuples per page in relation S = 10
Number of total number of tuples in S = 200

- **Block nested loops join**
  There are 2 options
  1. R is outer, S is inner relation
  2. S is outer, R is inner relation

  Buffer utilization
  Output: 1 page
  Inner relation: 1 page
  Outer relation: 50 pages

  1. R is outer, S is inner relation

  Cost to read outer = 1,000 I/Os
  Join = 4,000 I/Os
  Total cost = (1,000 + 4,000) = 5,000 I/Os

  2. S is outer, R is inner relation
  Cost to read outer = 200 I/Os
  Join = 4,000 I/Os
  Total cost = (200 + 4,000) = 4,200 I/Os

  Best case is the first option
  Therefore cost is 4,200 I/Os

  To minimize the cost of the join, the buffer size should be 202 pages so that we can store entire smaller relation in the buffer.

- **Grace hash join**
  Partitioning Phase
  I/O cost = 2*1000(read and write) + 2*200(read and write) = 2,400 I/Os

  Probing Phase
  Need to check whether the buffer size is enough to hold all buckets. It should satisfy the following condition

  $$B > f * M/(B - 1)$$

  Since $52 > 1.1 * 1,000/(52 - 1) = 21.5$, the buffer size is adequate.

  Therefore, I/O cost = 1000(read) + 200(read) = 1,200 I/Os

  Total cost = 2,400 + 1,200 = 3,600 I/Os

- **Sort-Merge join**
  Sorting Phase
  Since we have a 52 page buffer, we have enough space to hold sorted sublists = 21 sublists < 52

Cost to sort R = 2*(1,000 + 1,000) = 4,000 I/Os

Cost to sort S = 2* (200 + 200) = 800 I/Os

Merging phase
Since b is the primary key for S, we do not have to worry about duplicates.

Cost for merging = 1,000 + 200 = 1,200 I/Os

Total Cost = (4,000 + 800 + 1,200) = 6,000 I/Os