CS 143 Homework 7

Charles Zhang

November 28, 2021

Problem 1

Consider the relation $R(A, B, C)$ with the following properties:

- **R contains 8,000 tuples**
- **The size of a disk block is 4,000B**
- **The size of each R tuple is 40B**
- **Each disk block holding R tuples is full**

a) How many blocks are required to store the relation R?

$$8,000 \text{ tuples} \times \frac{40\text{B}}{1 \text{ tuple}} \times \frac{1 \text{ block}}{4,000\text{B}} = \boxed{80 \text{ blocks}}$$

b) We would like to sort the tuples of R on attribute A using the sort-merge algorithm in two passes. That is, after the initial sorting pass, we want to generate the completely sorted relation by one more merging pass. What is the minimum number of main memory blocks required?

$$\lceil \log_{M-1}(\frac{b_R}{M}) \rceil = 1$$

$$\log_{M-1}(\frac{b_R}{M}) \leq 1$$

$$\frac{b_R}{M} \leq M - 1$$

$$\frac{80}{M} \leq M - 1$$

$$80 \leq M^2 - M$$

$$\boxed{M = 10 \text{ blocks}}$$

Problem 2

Consider two relations $R(A, B)$ and $S(A, C)$. Suppose relation R occupies 1,000 blocks and relation S occupies 100 blocks. When we performed $R \bowtie S$ using the hash join algorithm that we learned in the class, the algorithm incurred (approximately) 3,300 disk block I/Os (either read or write), excluding the I/Os for writing the final join result. Write the minimum number of main memory buffers (each of the size of a disk block) that we must have had when we performed the join.

$$\lceil \log_{M-1} \frac{b_R}{M-2} \rceil \times 2(b_R + b_S) + b_R + b_S = 3,300$$

$$\lceil \log_{M-1} \frac{1,000}{M-2} \rceil \times 2,200 + 1,100 = 3,300$$

$$\lceil \log_{M-1} \frac{1,000}{M-2} \rceil = 1$$

$$\log_{M-1} \frac{1,000}{M-2} \leq 1$$

$$\frac{1,000}{M-2} \leq M - 1$$

$$M^2 - 3M - 998 \geq 0$$

$$\boxed{M = 34 \text{ blocks}}$$

Problem 3

Suppose you have 2 relations, $R(A, B, C)$ and $S(B, C, D, E)$.

You have a clustering unique (no duplicate keys) B+-tree index on attribute A for relation R. Assume this index is kept entirely in memory (i.e., you do not need to read it from disk).

For relation S, you have a non-clustering non-unique B+-tree index for attribute B. Furthermore, assume that the index is kept in memory (i.e., you do not need to read it from disk).

Other relevant data:

- **500 tuples of R are stored per block on disk**
- $|R| = 750,000$ **(number of tuples of R)**
- **100 tuples of S are stored per block on disk**
- $|S| = 250,000$ **(number of tuples of S)**
- **For every R tuple, there are roughly 5 tuples in S with $R.B = S.B$**

You want to execute the following query:

```
SELECT * FROM R, S WHERE R.B=S.B AND R.C=S.C
```

We present you with the following query plans:

```
For every block Bi of R, retrieved using the clustered index on A for R
    For every tuple r of Bi
        Use the index on B for S to retrieve all of the tuples s of S
        such that s.B=r.B
            For each of these tuples s, if r.C=s.C, output R and S
```

How many disk I/Os are needed to execute this query plan?

$$b_R = \frac{750,000 \text{ tuples}}{500 \text{ tuples/block}} = 1,500 \text{ blocks}$$

$$b_S = \frac{250,000 \text{ tuples}}{100 \text{ tuples/block}} = 2,500 \text{ blocks}$$

$$b_R + |R|(C + J)$$

$$C = 0$$

$$J = 5$$

$$1,500 + 750,000 \times 5$$

$$\boxed{3,751,500 \text{ disk I/Os}}$$

Now assume that we have a clustering non-unique B+-tree index for attribute C of S. For every R tuple, there are roughly 5,000 tuples in S with $R.C = S.C$. Assume that all of the tuples of S that agree on attribute C are stored in sequentially adjacent blocks on disk (that is, if more than one block is needed to store all of the tuples with some value of C, then these blocks will be sequentially located on the disk). The index is kept in main memory.

Using this new index, we came up with a second query plan:

```
For every block Bi of R, retrieved using the clustered index on A for R
    For every tuple r of Bi
        Use the index on C for S to retrieve all of the tuples s of S
        such that s.C=r.C
        For each of these tuples s, if s.B=r.B, output R and S
```

Now analyze each of the two plans more carefully in terms of their behavior regarding accesses to disk. Your analysis should consider the behavior of the number of I/Os and access time. Explain which of the two plans is therefore better under what circumstances. For the analysis of access time, you do not need to compute a concrete number. Just include in your analysis what access to disk are sequential accesses and which ones are random accesses and discuss its consequence on overall access time.

$$b_R + |R|(C + J)$$

$$C = 0$$

$$J = 5,000$$

$$1,500 + 750,000 \times 5,000$$

$$3,750,001,500 \text{ disk I/Os}$$

Since the number of matching S tuples per R tuple goes up by 1,000x in the second query plan, the number of required disk I/Os does as well. However, due to the clustering nature of the new index, the 5,000 disk I/Os per R tuple are sequential accesses, while the 5 disk I/Os per R tuple in the first query plan would be random accesses. Therefore, if the random access time of the disk was more than 1,000x longer than the sequential access time, the speed of each disk I/O would offset the increased number of disk I/Os, resulting in the second query plan being more efficient. If this was not the case, then it would be more beneficial to use the first query plan and trade off the random I/Os for the decreased number of I/Os.