Lecture 22:

Friday, November 22, 2002

1

Outline

- Query execution: 15.1 – 15.5

2

Two-Pass Algorithms Based on Sorting

- Recall: multi-way merge sort needs only two passes !
- Assumption: $B(R) <= M^2$
- Cost for sorting: 3B(R)

3

Two-Pass Algorithms Based on Sorting

Duplicate elimination $\delta(R)$

- Trivial idea: sort first, then eliminate duplicates
- Step 1: sort chunks of size M, write
  - cost 2B(R)
- Step 2: merge M-1 runs, but include each tuple only once
  - cost B(R)
- Total cost: 3B(R), Assumption: $B(R) <= M^2$

4

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{a,\ sum(b)}$ (R)

- Same as before: sort, then compute the sum(b) for each group of a's
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

5

Two-Pass Algorithms Based on Sorting

$R \cup S$

Complete the program in class:

```
x = first(R)
y = first(S)

While (_______________) do
{    case x < y:
        output(x)
        x = next(R)
     case x=y:


     case x > y;
}
```

6

Two-Pass Algorithms Based on Sorting

$R \cap S$

Complete the program in class:

```
x = first(R)
y = first(S)

While (______________) do
{    case x < y:


     case x=y:


    case x > y;
}
```

7

Two-Pass Algorithms Based on Sorting

R - S

Complete the program in class:

```
x = first(R)
y = first(S)

While (______________) do
{    case x < y:


     case x=y:


    case x > y;

}
```

8

Two-Pass Algorithms Based on Sorting

Binary operations: $R \cup S$, $R \cap S$, $R - S$

- Idea: sort R, sort S, then do the right thing
- A closer look:
  - Step 1: split R into runs of size M, then split S into runs of size M. Cost: 2B(R) + 2B(S)
  - Step 2: merge M/2 runs from R; merge M/2 runs from S; ouput a tuple on a case by cases basis
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R)+B(S) \leq M^2$

9

Two-Pass Algorithms Based on Sorting

R(A,C), S(B,D)

$R \bowtie_{R.A = S.B} S$

Complete the program in class:

```
x = first(R)
y = first(S)

While (______________) do
{    case x.A < y.B:


     case x.A=y.B:


    case x.A > y.B;

}
```

10

Two-Pass Algorithms Based on Sorting

Join $R \bowtie S$

- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Difficulty: many tuples in R may match many in S
  - If at least one set of tuples fits in M, we are OK
  - Otherwise need nested loop, higher cost
- Total cost: 5B(R)+5B(S)
- Assumption: $B(R) \leq M^2$, $B(S) \leq M^2$

11

Two-Pass Algorithms Based on Sorting

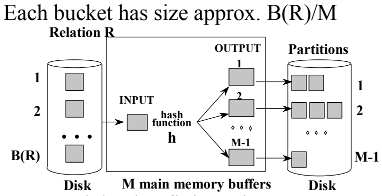
Join $R \bowtie S$

- If the number of tuples in R matching those in S is small (or vice versa) we can compute the join during the merge phase
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R) + B(S) \leq M^2$

12

Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M, i.e. B(R) <= M^2

Hash Based Algorithms for δ

- Recall: δ(R) = duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption:B(R) <= M^2

Hash Based Algorithms for γ

- Recall: γ(R) = grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply γ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption:B(R) <= M^2

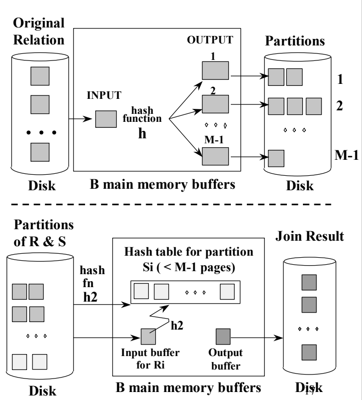
Partitioned Hash Join

R ⋈ S
- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

Hash-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.

- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.



Partitioned Hash Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

Hybrid Hash Join Algorithm

- Partition S into k buckets
- But keep first bucket S_1 in memory, k-1 buckets to disk
- Partition R into k buckets
  - First bucket R_1 is joined immediately with S_1
  - Other k-1 buckets go to disk
- Finally, join k-1 pairs of buckets:
  - (R_2,S_2), (R_3,S_3), …, (R_k,S_k)

19

Hybrid Join Algorithm

- How big should we choose k ?
- Average bucket size for S is B(S)/k
- Need to fit B(S)/k + (k-1) blocks in memory
  - B(S)/k + (k-1) <= M
  - k slightly smaller than B(S)/M

20

Hybrid Join Algorithm

- How many I/Os ?
- Recall: cost of partitioned hash join:
  - 3B(R) + 3B(S)
- Now we save 2 disk operations for one bucket
- Recall there are k buckets
- Hence we save 2/k(B(R) + B(S))
- Cost: (3-2/k)(B(R) + B(S)) =
  (3-2M/B(S))(B(R) + B(S))

21

Hybrid Join Algorithm

- Question in class: what is the real advantage of the hybrid algorithm ?

22

Indexed Based Algorithms

- Recall that in a clustered index all tuples with the same value of the key are clustered on as few blocks as possible

a a a	a a a a a	a a

- Note: book uses another term: "clustering index". Difference is minor…

23

Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)

24

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20, compute the cost of $\sigma_{a=v}(R)$
- Cost of table scan:
  - If R is clustered: B(R) = 2000 I/Os
  - If R is unclustered: T(R) = 100,000 I/Os
- Cost of index based selection:
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5000
- Notice: when V(R,a) is small, then unclustered index is useless

25

Index Based Join

- $R \bowtie S$
- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Assume R is clustered. Cost:
  - If index is clustered: B(R) + T(R)B(S)/V(S,a)
  - If index is unclustered: B(R) + T(R)T(S)/V(S,a)

26

Index Based Join

- Assume both R and S have a sorted index (B+ tree) on the join attribute
- Then perform a merge join (called zig-zag join)
- Cost: B(R) + B(S)

27

Questions in Class

- B(Product), B(Company) are large
- Which join method would you use ?
- Consider:
  - 10 bozos v.s. 100…0
  - 10 cool companies v.s. 100…00

```
SELECT Product.name,
       Company.city
FROM   Product, Company
WHERE  Product.maker = Company.name
       and Product.category = 'bozo' and
       Company.rating = 'cool'
```

28