Lecture 21:

Wednesday, November 20, 2002

1

Outline

- Query execution: 15.1 – 15.5

2

One-pass Algorithms

Hash join: $R \bowtie S$

- Scan S, build buckets in main memory
- Then scan R and join

- Cost: B(R) + B(S)
- Assumption: B(S) <= M

3

One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep tuples in memory
- When new tuple arrives, need to compare it with previously seen tuples
- Balanced search tree, or hash table
- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

4

Question in Class

Grouping:

Product(name, department, quantity)

$\gamma_{department,\ sum(quantity)}$ (Product) → Answer(department, sum)

Question: how do you compute it in main memory ?

Answer:

5

One-pass Algorithms

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

- Need to store all a's in memory
- Also store the sum(b) for each a
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: number of cities fits in memory

6

One-pass Algorithms

Binary operations: R ∩ S, R ∪ S, R – S

- Assumption: min(B(R), B(S)) <= M
- Scan one table first, then the next, eliminate duplicates
- Cost: B(R)+B(S)

7

Question in Class

Fill in missing lines to compute R ∪ S

```
H ← emptyHashTable

/* scan R */
For each x in R do
      insert(H, _________ )

/* scan S */
For each y in S do
      _____________________

/* collect result */
for each z in H do
      _____________________
```

8

Question in Class

Fill in missing lines to compute R – S

```
H ← emptyHashTable

/* scan R */
For each x in R do
      insert(H, _________ )

/* scan S */
For each y in S do
      _____________________

/* collect result */
for each z in H do
      _____________________
```

9

Question in Class

Fill in missing lines to compute R ∩ S

```
H ← emptyHashTable

/* scan R */
For each x in R do
      insert(H, _________ )

/* scan S */
For each y in S do
      _____________________

/* collect result */
for each z in H do
      _____________________
```

10

Nested Loop Joins

- Tuple-based nested loop R ⋈ S

```
for each tuple r in R do
   for each tuple s in S do
      if r and s join then output (r,s)
```

- Cost: T(R) T(S), sometimes T(R) B(S)

11

Nested Loop Joins

- We can be much more clever
- *Question*: how would you compute the join in the following cases ? What is the cost ?
  - B(R) = 1000, B(S) = 2, M = 4
  - B(R) = 1000, B(S) = 4, M = 4
  - B(R) = 1000, B(S) = 6, M = 4

12

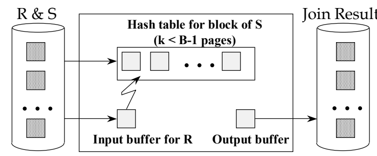
Nested Loop Joins

- Block-based Nested Loop Join

<u>for</u> each (M-1) blocks bs of S <u>do</u>
  <u>for</u> each block br of R <u>do</u>
    <u>for</u> each tuple s in bs
      <u>for</u> each tuple r in br <u>do</u>
        <u>if</u> r and s join <u>then</u> output(r,s)

13

Nested Loop Joins



14

Nested Loop Joins

- Block-based Nested Loop Join
- Cost:
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-1) times, and each time need to read R: costs B(S)B(R)/(M-1)
  - Total cost: B(S) + B(S)B(R)/(M-1)
- Notice: it is better to iterate over the smaller relation first
- R ⋈ S: R=outer relation, S=inner relation

15

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)

16

Two-Pass Algorithms Based on Sorting

Duplicate elimination δ(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

17

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

18

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: $B(R)+B(S) <= M^2$

19

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: $B(R) <= M^2$, $B(S) <= M^2$

20

Two-Pass Algorithms Based on Sorting

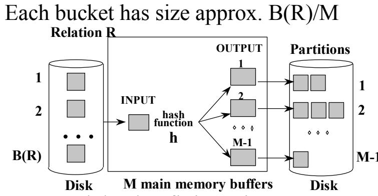
Join $R \bowtie S$

- 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: $B(R) + B(S) <= M^2$

21

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. $B(R) <= M^2$

22

Hash Based Algorithms for δ

- Recall: $\delta(R)$ = duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket (may read in main memory)
- Cost: 3B(R)
- Assumption:$B(R) <= M^2$

23

Hash Based Algorithms for γ

- Recall: $\gamma(R)$ = 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:$B(R) <= M^2$

24

Partitioned Hash Join

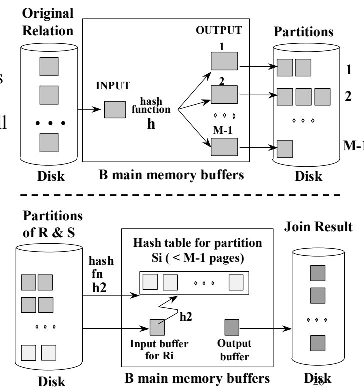
$R \bowtie S$

- 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

25

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.

26

Partitioned Hash Join

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

27

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:
  - $(R_2,S_2), (R_3,S_3), \ldots, (R_k,S_k)$

28

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

29

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))

30

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…

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)

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

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)

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)