Lecture 17: Query Execution

Tuesday, February 28, 2001

Outline

- Logical/physical algebra revisited
- Cost parameters and sorting
- One-pass algorithms
- Nested-loop joins
- Two-pass algorithms

Textbook: Chapter 12
Slides follow closer Ullman's book

Logical v.s. Physical Algebra

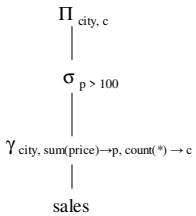
- Logical operators
  - *what* they do
- Physical operators
  - *how* they do it

Algebra for Bags

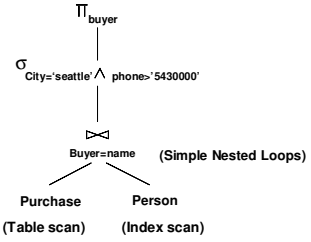
- Union, intersection, difference
- Selection σ
- Projection Π
- Join $\asymp$
- Duplicate elimination δ
- Grouping γ
- Sorting τ

Example

```
Select city, count(*)
From sales
Group by city
Having sum(price) > 100
```



Physical Operators



- each logical operator may have one or more physical operators associated (e.g. join → nested loop join or hash join or...)
- some physical operators don't correspond to logical (e.g. sort, scan)

Cost Parameters

- Cost parameters (note: book uses different notations):
  - M = number of blocks that fit in main memory
  - B(R) = number of blocks holding R
  - T(R) = number of tuples in R
  - V(R,a) = number of distinct values of the attribute a
- Estimating the cost:
  - Important in optimization (next lecture)
  - Compute I/O cost only
  - We compute the cost to *read* the tables
  - We don't compute the cost to *write* the result (because pipelining)

7

Sorting

- Two pass multi-way merge sort
- Step 1:
  - Read M blocks at a time, sort, write
  - Result: have runs of length M on disk
- Step 2:
  - Merge M-1 at a time, write to disk
  - Result: have runs of length $M(M-1) \approx M^2$
- Cost: 3B(R), Assumption: $B(R) \leq M^2$

8

Scanning Tables

- The table is *clustered* (I.e. blocks consists only of records from this table):
  - Table-scan: if we know where the blocks are
  - Index scan: if we have a sparse index to find the blocks
- The table is unclustered (e.g. its records are placed on blocks with other tables)
  - May need one read for each record

9

Cost of the Scan Operator

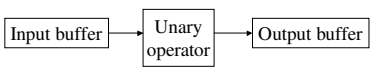
- Clustered relation:
  - Table scan: B(R); to sort: 3B(R)
  - Index scan: B(R); to sort: B(R) or 3B(R)
- Unclustered relation
  - T(R); to sort: T(R) + 2B(R)

10

One-pass Algorithms

Selection $\sigma(R)$, projection $\Pi(R)$

- Both are *tuple-at-a-Time* algorithms
- Cost: B(R)



11

One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep a dictionary in memory:
  - balanced search tree
  - hash table
  - etc
- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

12

One-pass Algorithms

Grouping: $\gamma_{city,\ sum(price)}$ (R)

- Need to keep a dictionary in memory
- Also store the sum(price) for each city
- Cost: B(R)
- Assumption: number of cities fits in memory

13

One-pass Algorithms

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

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

14

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)

15

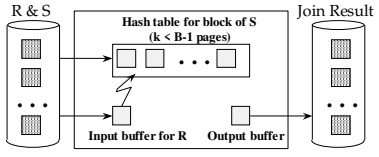
Nested Loop Joins

- Block-based Nested Loop Join

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

16

Nested Loop Joins



17

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

18

Two-Pass Algorithms Based on Sorting

Duplicate elimination $\delta(R)$

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

19

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{city,\ sum(price)}$ (R)

- Same as before: sort, then compute the sum(price) for each group
- As before: compute sum(price) during the merge phase.
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

20

Two-Pass Algorithms Based on Sorting

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

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

21

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

22

Two-Pass Algorithms Based on Sorting

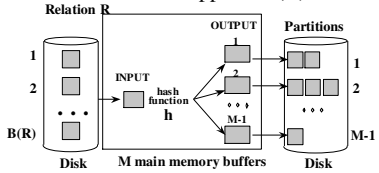
Join $R \bowtie S$

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

23

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

24

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

25

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

26

Hash-based Join

- R ⋈ S
- Recall the *main memory hash-based join*:
  - Scan S, build buckets in main memory
  - Then scan R and join

27

Partitioned Hash Join

R ⋈ S

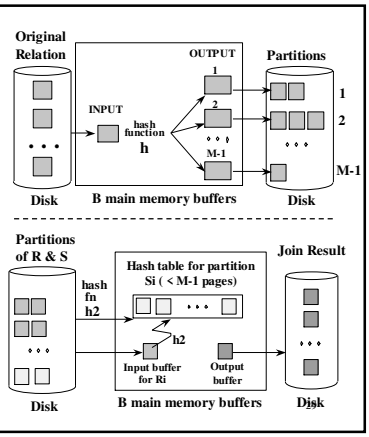
- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

28

Partitioned 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.



29

Partitioned Hash Join

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

30

Hybrid Hash Join Algorithm

- When we have more memory: B(S) << M^2
- 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:
  - $(R_2,S_2), (R_3,S_3), \ldots, (R_k,S_k)$

31

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

32

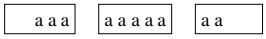
Hybrid Join Algorithm

- 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))

33

Indexed Based Algorithms

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



34

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)

35

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

36

Index Based Join

- $R \rtimes S$
- 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)

37

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)

38