CSE 544: Optimizations

Wednesday, 5/19/2004

1

The three components of an optimizer

We need three things in an optimizer:

- Algebraic laws
- An optimization algorithm
- A cost estimator

2

Algebraic Laws

- Commutative and Associative Laws
  - $R \cup S = S \cup R$, $R \cup (S \cup T) = (R \cup S) \cup T$
  - $R |\times| S = S |\times| R$, $R |\times| (S |\times| T) = (R |\times| S) |\times| T$
  - $R |\times| S = S |\times| R$, $R |\times| (S |\times| T) = (R |\times| S) |\times| T$
- Distributive Laws
  - $R |\times| (S \cup T) = (R |\times| S) \cup (R |\times| T)$

3

Algebraic Laws

- Laws involving selection:
  - $\sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R)$
  - $\sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \text{ U } \sigma_{C'}(R)$
  - $\sigma_C(R |\times| S) = \sigma_C(R) |\times| S$
- When C involves only attributes of R
  - $\sigma_C(R - S) = \sigma_C(R) - S$
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
  - $\sigma_C(R |\times| S) = \sigma_C(R) |\times| S$

4

Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)
  - $\sigma_{F=3}(R |\times|_{D=E} S) =$?
  - $\sigma_{A=5 \text{ AND } G=9}(R |\times|_{D=E} S) =$?

5

Algebraic Laws

- Laws involving projections
  - $\Pi_M(R |\times| S) = \Pi_M(\Pi_P(R) |\times| \Pi_Q(S))$
  - $\Pi_M(\Pi_N(R)) = \Pi_{M,N}(R)$
- Example R(A,B,C,D), S(E, F, G)
  - $\Pi_{A,B,G}(R |\times|_{D=E} S) = \Pi_?(\Pi_?(R) |\times|_{D=E} \Pi_?(S))$

6

Algebraic Laws

- Laws involving grouping and aggregation:
  $\delta(\gamma_{A,\ agg(B)}(R)) = \gamma_{A,\ agg(B)}(R)$
  $\gamma_{A,\ agg(B)}(\delta(R)) = \gamma_{A,\ agg(B)}(R)$ if agg is "duplicate insensitive"

- Which of the following are "duplicate insensitive" ? sum, count, avg, min, max

$\gamma_{A,\ agg(D)}(R(A,B) \mid\times\mid_{B=C} S(C,D)) =$
$\gamma_{A,\ agg(D)}(R(A,B) \mid\times\mid_{B=C} (\gamma_{C,\ agg(D)}S(C,D)))$

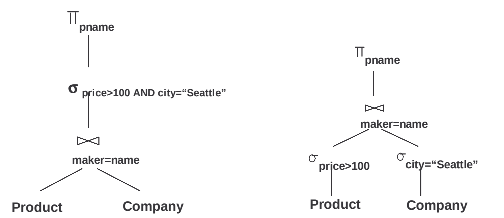
Optimization Algorithms

- Heuristic based
- Cost based
  - Dynamic programming: System R
  - Rule-based optimizations: DB2, SQL-Server

Heuristic Based Optimizations

- Query rewriting based on algebraic laws
- Result in better queries most of the time
- Heuristics number 1:
  - Push selections down
- Heuristics number 2:
  - Sometimes push selections up, then down

Predicate Pushdown



The earlier we process selections, less tuples we need to manipulate higher up in the tree (but may cause us to loose an important ordering of the tuples, if we use indexes).

Predicate Pushdown

```
Select   y.name, Max(x.price)
From     product x, company y
Where    x.maker = y.name
GroupBy  y.name
Having   Max(x.price) > 100
```

```
Select y.name, Max(x.price)
From     product x, company y
Where    x.maker=y.name  and
         x.price > 100
GroupBy  y.name
Having Max(x.price) > 100
```

- *For each company, find the maximal price of its products.*
- Advantage: the size of the join will be smaller.
- Requires transformation rules specific to the grouping/aggregation operators.
- **Won't work if we replace Max by Min.**

Dynamic Programming

Originally proposed in System R

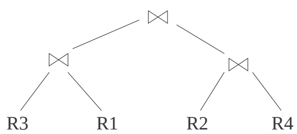
- Only handles single block queries:

SELECT list
FROM list
WHERE cond_1 AND cond_2 AND . . . AND cond_k

- Heuristics: selections down, projections up
- Dynamic programming: ***join reordering***

Join Trees

- R1 $\bowtie$ R2 $\bowtie$ …. $\bowtie$ Rn
- Join tree:

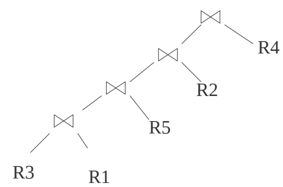


- A plan = a join tree
- A partial plan = a subtree of a join tree

13

Types of Join Trees

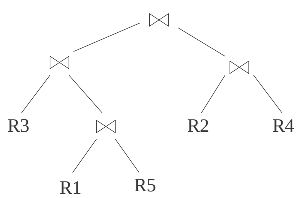
- Left deep:



14

Types of Join Trees

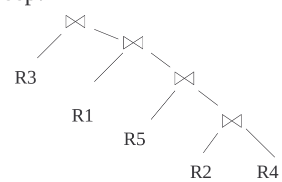
- Bushy:



15

Types of Join Trees

- Right deep:



16

Dynamic Programming

- Given: a query R1 $\bowtie$ R2 $\bowtie$ … $\bowtie$ Rn
- Assume we have a function cost() that gives us the cost of every join tree
- Find the best join tree for the query

17

Dynamic Programming

- Idea: for each subset of {R1, …, Rn}, compute the best plan for that subset
- In increasing order of set cardinality:
  - Step 1: for {R1}, {R2}, …, {Rn}
  - Step 2: for {R1,R2}, {R1,R3}, …, {Rn-1, Rn}
  - …
  - Step n: for {R1, …, Rn}
- It is a bottom-up strategy
- A subset of {R1, …, Rn} is also called a *subquery*

18

Dynamic Programming

- For each subquery Q $\subseteq$ {R1, …, Rn} compute the following:
  - Size(Q)
  - A best plan for Q: Plan(Q)
  - The cost of that plan: Cost(Q)

19

Dynamic Programming

- **Step 1**: For each $\{R_i\}$ do:
  - Size($\{R_i\}$) = B(R_i)
  - Plan($\{R_i\}$) = R_i
  - Cost($\{R_i\}$) = (cost of scanning R_i)

20

Dynamic Programming

- **Step i**: For each Q $\subseteq \{R_1, \ldots, R_n\}$ of cardinality i do:
  - Compute Size(Q)   (later…)
  - For every pair of subqueries Q', Q''
    s.t. Q = Q' $\cup$ Q''
    compute cost(Plan(Q') |⋉| Plan(Q''))
  - Cost(Q) = the smallest such cost
  - Plan(Q) = the corresponding plan

21

Dynamic Programming

- Return Plan($\{R_1, \ldots, R_n\}$)

22

Dynamic Programming

To illustrate, we will make the following simplifications:

- Cost(P_1 |⋉| P_2) = Cost(P_1) + Cost(P_2) + size(intermediate result(s))
- Intermediate results:
  - If P_1 = a join, then the size of the intermediate result is size(P_1), otherwise the size is 0
  - Similarly for P_2
- Cost of a scan = 0

23

Dynamic Programming

- Example:
- Cost(R5 |⋉| R7) = 0   (no intermediate results)
- Cost((R2 |⋉| R1) |⋉| R7)
  = Cost(R2 |⋉| R1) + Cost(R7) + size(R2 |⋉| R1)
  = size(R2 |⋉| R1)

24

Dynamic Programming

- Relations: R, S, T, U
- Number of tuples: 2000, 5000, 3000, 1000
- Size estimation: T(A |×| B) = 0.01*T(A)*T(B)

25

Subquery	Size	Cost	Plan
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

26

Subquery	Size	Cost	Plan
RS	100k	0	RS
RT	60k	0	RT
RU	20k	0	RU
ST	150k	0	ST
SU	50k	0	SU
TU	30k	0	TU
RST	3M	60k	(RT)S
RSU	1M	20k	(RU)S
RTU	0.6M	20k	(RU)T
STU	1.5M	30k	(TU)S
RSTU	30M	60k+50k=110k	(RT)(SU)

27

Reducing the Search Space

- Left-linear trees v.s. Bushy trees
- Trees without cartesian product

Example: R(A,B) |×| S(B,C) |×| T(C,D)

Plan: (R(A,B) |×| T(C,D)) |×| S(B,C) has a cartesian product – most query optimizers will not consider it

28

Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)
- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further

29

Rule-Based Optimizers

- ***Extensible*** collection of rules
  Rule = Algebraic law with a direction
- Algorithm for firing these rules
  Generate many alternative plans, in some order
  Prune by cost
- Volcano (later SQL Sever)
- Starburst (later DB2)

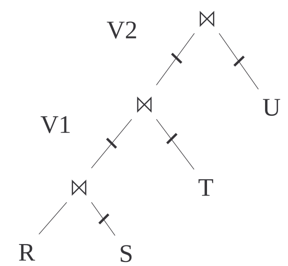
30

Completing the Physical Query Plan

- Choose algorithm to implement each operator
  - Need to account for more than cost:
    - How much memory do we have ?
    - Are the input operand(s) sorted ?
- Decide for each intermediate result:
  - To materialize
  - To pipeline

31

Materialize Intermediate Results Between Operators




```
HashTable ß  S
repeat    read(R, x)
          y ß  join(HashTable, x)
          write(V1, y)

HashTable ß  T
repeat    read(V1, y)
          z ß  join(HashTable, y)
          write(V2, z)

HashTable ß  U
repeat    read(V2, z)
          u ß  join(HashTable, z)
          write(Answer, u)
```

32

Materialize Intermediate Results Between Operators

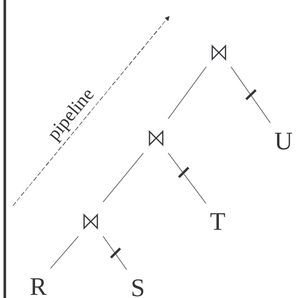
Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

33

Pipeline Between Operators




```
HashTable1 ß  S
HashTable2 ß  T
HashTable3 ß  U
repeat    read(R, x)
          y ß  join(HashTable1, x)
          z ß  join(HashTable2, y)
          u ß  join(HashTable3, z)
          write(Answer, u)
```

34

Pipeline Between Operators

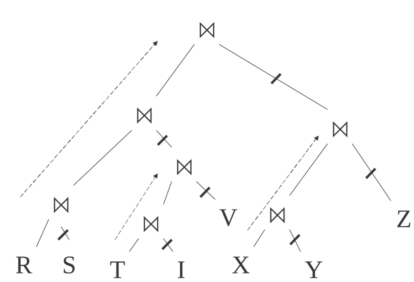
Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

35

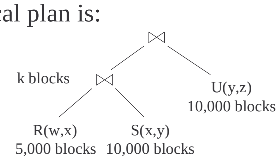
Pipeline in Bushy Trees



36

Example

- Logical plan is:

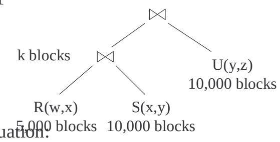


- Main memory M = 101 buffers

37

Example

M = 101



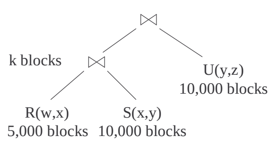
Naïve evaluation:

- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

38

Example

M = 101



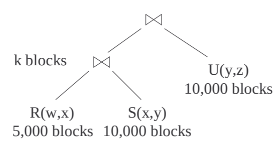
Smarter:

- Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
- Step 2: hash S on x into 100 buckets; to disk
- Step 3: read each R_i in memory (50 buffer) join with S_i (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we *pipeline*
- Cost so far: 3B(R) + 3B(S)

39

Example

M = 101



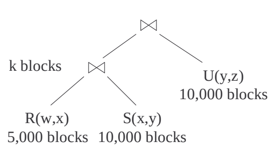
Continuing:

- How large are the 50 buckets on y ? Answer: k/50.
- If k <= 50 then keep all 50 buckets in Step 3 in memory, then:
- Step 4: read U from disk, hash on y and join with memory
- Total cost: 3B(R) + 3B(S) + B(U) = 55,000

40

Example

M = 101



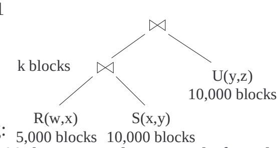
Continuing:

- If 50 < k <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size k/50 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: 3B(R) + 3B(S) + 2k + 3B(U) = 75,000 + 2k

41

Example

M = 101



Continuing:

- If k > 5000 then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4k

42

Example

Summary:

- If $k <= 50$, cost = 55,000
- If $50 < k <= 5000$, cost = 75,000 + 2k
- If $k > 5000$, cost = 75,000 + 4k

43