CSE 444: Database Internals

Lecture 12
Query Optimization (part 3)

Reminders

- I'm not Magda
- Lab 2 is due on Friday by 11pm
- Lab 3 released this Friday (transactions, yay!)
- HW 5 due next week
- Quiz on 4/25 (next Monday)

Acknowledgments

Today's lecture focuses on how to actually implement the Selinger optimizer

(Many slides from Sam Madden at MIT)

Selinger Optimizer

Goal:
- How to order a series joins over N tables A,B,C,…

E.g. A.a = B.b AND A.c = D.d AND B.e = C.f

Selinger Optimizer

Goal:
- How to order a series joins over N tables A,B,C,…

E.g. A.a = B.b AND A.c = D.d AND B.e = C.f

Problem:
- … too … many … plans …

Selinger Optimizer

Goal:
- How to order a series joins over N tables A,B,C,…

E.g. A.a = B.b AND A.c = D.d AND B.e = C.f

Problem:
- N! ways to order joins; e.g. ABCD, ACBD, ….
- $C_{N-1} = \frac{1}{N}\binom{2(N-1)}{N-1}$ plans/ordering; e.g. (((AB)C)D), ((AB)(CD)))
- Multiple implementations (hash, nested loops)
- Naïve approach does not scale
  - E.g. N = 20, #join orders 20! = 2.4 x 10^{18} ; many more plans

Selinger Optimizer

- Only left-deep plans: (((AB)C)D) – eliminate C_{N-1}.
- Push down selections
- Don't consider Cartesian products
- Dynamic programming algorithm

Dynamic Programming

OrderJoins:
R = set of relations to join
For d = 1 to |R|:
  For S in {all size-d subsets of R}:
    Pick a∈S with lowest cost (S-a)⋈a

Dynamic Programming

OrderJoins:
R = set of relations to join
For d = 1 to |R|:
  For S in {all size-d subsets of R}:
    Pick a∈S with lowest cost (S-a)⋈a

↑
What is the cost?
* Cost to scan a
* Cost to produce S-a
* Cost to join (S-a) with a

Dynamic Programming

OrderJoins:
R = set of relations to join
For d = 1 to |R|:
  For S in {all size-d subsets of R}:
    Pick a∈S with lowest cost (S-a)⋈a

↑
What is the cost?
* Cost to scan a
* Cost to produce S-a ← Calculated in previous iteration
* Cost to join (S-a) with a

Dynamic Programming

OrderJoins:
R = set of relations to join
For d = 1 to |R|:
  For S in {all size-d subsets of R}:
    optjoin(S) = (S – a) join a,
      where a is the single relation that minimizes:
        cost(**optjoin**(S – a)) +
        min.cost to join (S – a) with a +
        min.access cost for a

Note: **optjoin**(S-a) is cached from previous iterations

Example

- **orderJoins(A, B, C, D)**
- Assume all joins are NL

Subplan S	optJoin(S)	Cost(OptJoin(S))
A		

Example

- **orderJoins(A, B, C, D)**
- Assume all joins are NL

- d = 1
- A = best way to access A
  - (sequential scan, predicate-pushdown on index, etc)
  - B = best way to access B
  - C = best way to access C
  - D = best way to access D

- Total number of steps: choose(N, 1)

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
C	Seq scan	120
D	B+tree scan	400

Example

- **orderJoins(A, B, C, D)**

- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		

Example

- **orderJoins(A, B, C, D)**

- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		
{A, B}	BA	156

Example

- **orderJoins(A, B, C, D)**

- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B
  - {B,C} = BC or CB

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		
{A, B}	BA	156
{B, C}	BC	98

Example

- **orderJoins(A, B, C, D)**

- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B
  - {B,C} = BC or CB

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		
{A, B}	BA	156
{B, C}	BC	98

Example

- **orderJoins(A, B, C, D)**

- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B
  - {B,C} = BC or CB
  - {C,D} = CD or DC
  - {A,C} = AC or CA
  - {B,D} = BD or DB
  - {A,D} = AD or DA

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		
{A, B}	BA	156
{B, C}	BC	98
……..		

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
…		
{A, B}	BA	156
{B, C}	BC	98
……..		

- **orderJoins(A, B, C, D)**
- d = 2
- {A,B} = AB or BA
- use previously computed
  - best way to access A and B
  - {B,C} = BC or CB
  - {C,D} = CD or DC
  - {A,C} = AC or CA
  - {B,D} = BD or DB
  - {A,D} = AD or DA
- Total number of steps: choose(N, 2) × 2

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
….		
{A, B}	BA	156
{B, C}	BC	98
….		

- **orderJoins(A, B, C, D)**
- d = 3

{A,B,C} =
Remove A: compare A({B,C}) to ({B,C})A
–

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
….		
{A, B}	BA	156
{B, C}	BC	98
….		

- **orderJoins(A, B, C, D)**
- d = 3

{A,B,C} =
Remove A: compare A({B,C}) to ({B,C})A
–

optJoin(B,C) and its cost are already cached in table

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
….		
{A, B}	BA	156
{B, C}	BC	98
….		
{A, B, C}	BAC	500

- **orderJoins(A, B, C, D)**
- d = 3

{A,B,C} =
Remove A: compare A({B,C}) to ({B,C})A
Remove B: compare B({A,C}) to ({A,C})B
– Remove C: compare C({A,B}) to ({A,B})C

optJoin(B,C) and its cost are already cached in table

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
….		
{A, B}	BA	156
{B, C}	BC	98
{A, B, C}	BAC	500

- **orderJoins(A, B, C, D)**
- d = 3

{A,B,C} =
Remove A: compare A({B,C}) to ({B,C})A
Remove B: compare B({A,C}) to ({A,C})B
– Remove C: compare C({A,B}) to ({A,B})C

optJoin(B,C) and its cost are already cached in table

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
….		
{A, B}	BA	156
{B, C}	BC	98
{A, B, C}	BAC	500

- **orderJoins(A, B, C, D)**
- d = 3

{A,B,C} =
Remove A: compare A({B,C}) to ({B,C})A
Remove B: compare B({A,C}) to ({A,C})B
– Remove C: compare C({A,B}) to ({A,B})C
– {A,B,D} =
– Remove A: compare A({B,D}) to ({B,D})A
– …
– {A,C,D} =…
– {B,C,D} =…

- Total number of steps: choose(N, 3) × 3 × 2

optJoin(B,C) and its cost are already cached in table

Example

Subplan S	optJoin(S)	Cost(OptJoin(S))
A	Index scan	100
B	Seq. scan	50
{A, B}	BA	156
{B, C}	BC	98
{A, B, C}	BAC	500
{B, C, D}	DBC	150
……..		

optJoin(B, C, D) and its cost are already cached in table

- **orderJoins(A, B, C, D)**

- d = 4
- {A,B,C,D} =
-
-
- Remove A: compare A({B,C,D}) to ({B,C,D})A
- Remove B: compare B({A,C,D}) to ({A,C,D})B
- Remove C: compare C({A,B,D}) to ({A,B,D})C
  - Remove D: compare D({A,B,C}) to ({A,B,C})D

- Total number of steps: choose(N, 4) × 4 × 2

Complexity

- Total #subsets considered
  - Choose(N, 1) + Choose(N, 2) + ….. + Choose (N, N)
  - All nonempty subsets of a size N set: $2^N - 1$
  - Equivalently: number of binary strings of size N, except 00…0:
  - ~~000~~, 001, 010, 011, 100, 101, 110, 111

Complexity

- Total #subsets considered
  - Choose(N, 1) + Choose(N, 2) + ….. + Choose (N, N)
  - All nonempty subsets of a size N set: $2^N - 1$
  - Equivalently: number of binary strings of size N, except 00…0:
  - ~~000~~, 001, 010, 011, 100, 101, 110, 111

- For each subset of size d:
  - d ways to remove one element
  - 2 ways for compute AB or BA

Complexity

- Total #subsets considered
  - Choose(N, 1) + Choose(N, 2) + ….. + Choose (N, N)
  - All nonempty subsets of a size N set: $2^N - 1$
  - Equivalently: number of binary strings of size N, except 00…0:
  - ~~000~~, 001, 010, 011, 100, 101, 110, 111
- For each subset of size d:
  - d ways to remove one element
  - 2 ways for compute AB or BA
- Total #plans considered
  - Choose(N, 1) + 2 Choose(N, 2) + ….. + N Choose (N, N)
  - Equivalently: total number of 1's in all strings of size N
  - $N\,2^{N-1}$ because every 1 occurs 2^{N-1} times
  - Need to further multiply by 2, to account for AB or BA

Why Left-Deep

Asymmetric, cost depends on the order

- Left: Outer relation Right: Inner relation

- For nested-loop-join, we try to load the outer (typically smaller) relation in memory, then read the inner relation one page at a time
  B(R) + B(R)*B(S) or B(R) + B(R)/M * B(S)

- For index-join,
  we assume right (inner) relation has index

Why Left-Deep

- Advantages of left-deep trees?
  1. Fits well with standard join algorithms (nested loop, one-pass), more efficient

  2. One pass join: Uses smaller memory
     1. ((R, S), T), can reuse the space for R while joining (R, S) with T
     2. (R, (S, T)): Need to hold R, compute (S, T), then join with R, worse if more relations

  3. Nested loop join, consider top-down iterator next()
     1. ((R, S), T), Reads the chunks of (R, S) once, reads stored base relation T multiple times
     2. (R, (S, T)): Reads the chunks of R once, reads computed relation (S, T) multiple times, either more time or more space

Excruciatingly Detailed Optimization Example

Interesting Orders

- Some query plans produce data in sorted order
  - E.g scan over a primary index, merge-join
  - Called *interesting order*
- Next operator may use this order
  - E.g. can be another merge-join
- For each subset of relations, compute multiple optimal plans, one for each interesting order
- Increases complexity by factor k+1, where k=number of interesting orders