Query Optimization

It is safer to accept any chance that offers itself, and extemporize a procedure to fit it, than to get a good plan matured, and wait for a chance of using it.

Thomas Hardy (1874) in *Far from the Madding Crowd*

---

**Query optimization**

Generate query plans
Estimate size of intermediate results
Estimate cost of plan ($, time, ...)

\[ Q \xrightarrow{P_1} C_1 \]
\[ Q \xrightarrow{P_2} C_2 \]
\[ Q \xrightarrow{P_3} C_3 \]
\[ Q \xrightarrow{P_n} C_n \]

pick minimum
Query Optimization

- Query: R1 \( \bowtie \) R2 \( \bowtie \) R3
- R1@Site1, R2@Site2, R3@Site3
- Result@Site1
- Possible plans
  - Plan 1: Send R2 and R3 to Site1. Perform query at Site 1
  - Plan 2: Send R3 to Site2; Evaluate I = R2 \( \bowtie \) R3; send I to Site1; Evaluate result = I \( \bowtie \) R1
  - Many other plans …(including types of joins, number of sites, semijoins, etc)

Cost estimation

- As in centralized system: estimate result sizes
- But: # IOs may not be best metric

\[\text{e.g., Transmission time may dominate}\]

\[
\begin{array}{c}
\text{work at site} & \text{work at site} & \text{answer} \\
T1 & \text{TIME} & T2 \\
\end{array}
\]

\[\text{or $}\]

\[\text{or $}\]
Another reason why plain IOs not enough: Parallelism

<table>
<thead>
<tr>
<th>Plan A</th>
<th>Plan B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>site 1</td>
</tr>
<tr>
<td>100 IOs</td>
<td>site 2</td>
</tr>
<tr>
<td></td>
<td>site 3</td>
</tr>
</tbody>
</table>

- **Cost metrics**
  - IOs, Bytes transmitted, $, ...
  - Can add together

- **Response time metric**
  - cannot add
  - need scheduling and dependency info
  - skew important
Take into account:
(in parallel/distributed system)

- Start up costs (for parallel operation)
- Data distribution costs/time
- Contention
  - memory, disk, network,…
- Assembling result

Example: Response time
Query Optimizer

- Cost model
- Plan space
  - Deep tree vs bushy tree
- Enumeration/Search strategy
  - Exhaustive (with pruning)
  - Hill climbing (greedy)
  - Query separation
  - SDD-1 (semi-join based)

(1) **Exhaustive**
  - consider “all” query plans with a set of techniques
  - prune some plans (heuristics)
Example:

```
| R | | S | | T |
---|---|---|---|
R  | S  | T  |
  | R  |  |
  |  | S  |
  |  | T  |
```

Heuristics:
1. Prune because cross-product not necessary
2. Prune because larger relation first

In generating plans, keep goal in mind:

- e.g.: Goal is parallelism in system with fast net, consider partitioning relation(s) first
- e.g.: Goal is reduction of net traffic, consider semi-joins
(2) Hill climbing

Better plans

Worse plans

Step 1: Do initial processing
Step 2: Select initial feasible solution ($P_0$)
  2.1 Determine the candidate result sites - sites where a relation referenced in the query exist
  2.2 Compute the cost of transferring all the other referenced relations to each candidate site
  2.3 $P_0 = \text{candidate site with minimum cost}$
Step 3: Determine candidate splits of $P_0$ into
  $P_1 = \{P_{1a}, P_{1b}\}$
  3.1 $P_{1a}$ consists of sending one of the relations to the other relation’s site
  3.2 $P_{1b}$ consists of sending the join of the relations to the final result site
Hill Climbing Algorithm

Step 4: Replace $P_0$ with $P_1$ that gives

$$cost(P_{1a}) + cost(\text{local join}) + cost(P_{1b}) < cost(P_0)$$

Step 5: Recursively apply steps 3–4 on $P_1$ until no such plans can be found

Step 6: Check for redundant transmissions in the final plan and eliminate them.

Example  $R \bowtie S \bowtie \sigma(U) \bowtie V$

<table>
<thead>
<tr>
<th>Rel</th>
<th>Site</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>U</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>V</td>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>

tuple size = 1

Goal: minimize data transmission
Step 1: \( R \bowtie S \bowtie T \bowtie V \)

<table>
<thead>
<tr>
<th>Rel</th>
<th>Site</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>V</td>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>

tuple size = 1

\( T = \sigma(U) \)

Selectivity is 1/3

Step 2: Initial plan - send relations to one site

What site do we send all relations to?
To site 1: cost=20+30+40=90
To site 2: cost=10+30+40=80
To site 3: cost=10+20+40=70
To site 4: cost=10+20+30=60
\[ \text{P0: } R (1 \rightarrow 4) \]
\[ S (2 \rightarrow 4) \]
\[ T (3 \rightarrow 4) \]
\[ \text{Compute } R \bowtie S \bowtie T \bowtie V \text{ at site 4} \]

\[ \begin{align*}
\text{CS5225} & \quad \text{Query Optimization} \\
19 & \quad \quad \\
\end{align*} \]

---

**Steps 3 & 4**

- Consider sending each relation to neighbor:

\[ \begin{align*}
\text{e.g.: } & \quad \begin{array}{c}
\text{1} \\
\text{R}
\end{array} \quad \begin{array}{c}
\text{2} \\
\text{S}
\end{array} \quad \begin{array}{c}
\text{4}
\end{array} \\
\end{align*} \]

\[ \begin{align*}
\text{1} & \quad \text{R} \\
\text{4} & \quad \text{S} \\
\end{align*} \]

\[ \begin{align*}
\text{1} & \quad \text{R} \\
\text{2} & \quad \text{S} \\
\end{align*} \]

\[ \begin{align*}
\text{CS5225} & \quad \text{Query Optimization} \\
20 & \quad \quad \\
\end{align*} \]
Assume: Size

\[ R \bowtie S = 20 \]
\[ S \bowtie T = 5 \]
\[ T \bowtie V = 1 \]

\[
\begin{array}{c}
1 \xrightarrow{10} R \xrightarrow{4} S \xrightarrow{20} 2 \\
\text{cost} = 30 \\
\end{array}
\]
No savings

\[
\begin{array}{c}
1 \xrightarrow{10} R \xrightarrow{20} S \\
\text{cost} = 30 \\
\end{array}
\]

\[
\begin{array}{c}
1 \xrightarrow{20} S \xrightarrow{R} 2 \\
\text{cost} = 40 \\
\end{array}
\]
Worse!

\[
\begin{array}{c}
2 \xrightarrow{20} S \xrightarrow{T} 3 \\
\text{cost} = 50 \\
\end{array}
\]

\[
\begin{array}{c}
2 \xrightarrow{30} T \xrightarrow{S} 3 \\
\text{cost} = 35 \\
\end{array}
\]
A Win!

\[
\begin{array}{c}
2 \xrightarrow{30} T \xrightarrow{S} 3 \\
\text{cost} = 25 \\
\end{array}
\]
A Bigger Win
P1: P1a: S (2 → 3)  
   α = S ◦ T  
P1b: R (1 → 4)  
   α (3 → 4)  
   compute answer at site 4  

Step 5: Repeat Steps 3 & 4  
- Treat α = S ◦ T as relation
Hill climbing may miss best plan!

Example: best plan could be:

\[ \text{PB: } T \rightarrow 4 \quad 30 \]

\[ \beta = T \bowtie V \]

\[ \beta (4 \rightarrow 2) \quad 1 \]

\[ \beta' = \beta \bowtie S \]

\[ \beta' (2 \rightarrow 1) \quad 1 \]

\[ \beta'' = \beta' \bowtie R \]

[optional] \[ \beta'' (1 \rightarrow 4) \quad 1 \]

Costs could be low because \( \beta \) is very selective

Compute answer \[ 33 = \text{total} \]

(3) Query separation

- separate query into 2 or more steps
- optimize each step independently
Example: simple queries

1. Compute $R' = \Pi_A[\sigma_{c_2} R]$  
2. Compute $J = R' \bowtie S'$
3. Compute

\[\text{Ans} = \sigma_{c_1}\{[J \bowtie \sigma_{c_2} R] \bowtie [J \bowtie \sigma_{c_3} S]\}\]

In other words:
(a) Compute $A$ values in answer  
     (steps 1,2)  
(b) Get tuples from sites with matching  
     $A$ values and compute answer  
     (step 3)
Simple query

- Relations have a single attribute
- Output has a single attribute
  e.g., \( J \leftarrow R' \bowtie S' \)

Idea

- Decompose query into
  - Local processing
  - Simple query (or queries)
  - Final processing
- Optimize simple query
- Philosophy
  - Hard part is distributed join
  - Do this part with only keys; get rest of data later
  - Simpler to optimize simple queries
SDD-1 Algorithm

Step 1: In the execution strategy (call it ES), include all the local processing.

Step 2: Reflect the effects of local processing on the database profile.

Step 3: Construct a set of beneficial semijoin operations (BS) as follows:

\[ BS = \emptyset \]

For each semijoin \( SJ_i \),

\[ BS \leftarrow BS \cup SJ_i \text{ if } \text{cost}(SJ_i) < \text{benefit}(SJ_i) \]

SDD-1 Algorithm – Example

Consider the following query:

```
SELECT R3.C
FROM R1, R2, R3
WHERE R1.A = R2.A
AND R2.B = R3.B
```

which has the following query graph and statistics:

<table>
<thead>
<tr>
<th>relation</th>
<th>card</th>
<th>tuple size</th>
<th>relation size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>30</td>
<td>50</td>
<td>1500</td>
</tr>
<tr>
<td>R2</td>
<td>100</td>
<td>30</td>
<td>3000</td>
</tr>
<tr>
<td>R3</td>
<td>50</td>
<td>40</td>
<td>2000</td>
</tr>
</tbody>
</table>

SF: selectivity factor
Result = SF*R
SDD-1 Algorithm – Example

- Beneficial semijoins:
  - $SJ_1 = R2 \bowtie R1$, whose benefit is $2100 = (1 - 0.3) \times 3000$ and cost is 36
  - $SJ_2 = R2 \bowtie R3$, whose benefit is $1800 = (1 - 0.4) \times 3000$ and cost is 80

- Nonbeneficial semijoins:
  - $SJ_3 = R1 \bowtie R2$, whose benefit is $300 = (1 - 0.8) \times 1500$ and cost is 320
  - $SJ_4 = R3 \bowtie R2$, whose benefit is 0 and cost is 400

SDD-1 Algorithm

Iterative Process

Step 4: Remove the most beneficial $SJ_i$ from $BS$ and append it to $ES$

Step 5: Modify the database profile accordingly

Step 6: Modify $BS$ appropriately
  - compute new benefit/cost values
  - check if any new semijoin need to be included in $BS$

Step 7: If $BS \neq \emptyset$, go back to Step 4.
SDD-1 Algorithm – Example

• Iteration 1:
  – Remove SJ₁ from BS and add it to ES.
  – Update statistics
    \[ R₂' = \text{size}(R₂) = 900 = 3000 \times 0.3 \]
    \[ \text{size}(R₂'.A) = 320 \times 0.3 = 96 \]
    \[ SFₙ(R₂'.A) = 0.8 \times 0.3 = 0.24 \]

• Iteration 2:
  – Two beneficial semijoins:
    \[ SJ₂ = R₂' \bowtie R₃, \text{ whose benefit is } 540 = (1 - 0.4) \times 900 \text{ and cost is 80} \]
    \[ SJ₃ = R₁ \bowtie R₂', \text{ whose benefit is } 300 = (1 - 0.8) \times 1500 \text{ and cost is 96} \]
  – Add SJ₂ to ES
  – Update statistics
    \[ \text{size}(R₂') = 360 = 900 \times 0.4 \]
    \[ SFₙ(R₂'.A) = 0.3 \times 0.8 = 0.24 \]

Note: selectivity of R₂' may also change, but not important in this example.

<table>
<thead>
<tr>
<th>attribute</th>
<th>SFₙ</th>
<th>size(Π_attribute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁.A</td>
<td>0.3</td>
<td>36</td>
</tr>
<tr>
<td>R₂'.A</td>
<td>0.24</td>
<td>96</td>
</tr>
<tr>
<td>R₂'.B</td>
<td>1.0</td>
<td>120</td>
</tr>
<tr>
<td>R₃.B</td>
<td>0.4</td>
<td>80</td>
</tr>
</tbody>
</table>

SDD-1 Algorithm – Example

• Iteration 3:
  – No new beneficial semijoins.
  – Remove remaining beneficial semijoin SJ₃ from BS and add it to ES.
  – Update statistics
    \[ \text{size}(R₁) = 1200 = 1500 \times 0.8 \]
    \[ SFₙ(R₁.A) = 0.3 \times 0.8 = 0.24 \]
SDD-1 Algorithm

Assembly Site Selection

**Step 8:** Find the site where the largest amount of data resides and select it as the assembly site.

Example:

Amount of data stored at sites:
- Site 1: 1200
- Site 2: 360
- Site 3: 2000

Therefore, Site 3 will be chosen as the assembly site.

SDD-1 Algorithm

Postprocessing

**Step 9:** For each \( R_i \) at the assembly site, find the semijoins of the type \( R_i \bowtie R_j \) where the total cost of \( ES \) without this semijoin is smaller than the cost with it and remove the semijoin from \( ES \).

**Step 10:** Permute the order of semijoins if doing so would improve the total cost of \( ES \).
- Example: Final strategy:
  - Send \( (R2 \bowtie R1) \bowtie R3 \) to Site 3
  - Send \( (R1 \bowtie R2) \) to Site 3
Summary: Query Optimization

• Cost/result estimation
• Three key components in optimizer
  – Cost model, search space, enumeration algorithm
• In practice, avoid bad plans (rather than find optimal)
• Strategies
  – Exhaustive
  – Hill climbing
  – Separation
  – SDD-1 (semi-join based)