CS 347: Distributed Databases and Transaction Processing

Notes04: Query Optimization
Query optimization

- Cost estimation
- Strategies for exploring plans

\[ \min Q \]
Cost estimation

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

e.g., Transmission time may dominate

work at site  work at site  answer

T1 T2

>>>---------TIME--------->

or $
Example: AWS

http://calculator.s3.amazonaws.com/index.html
Another reason why plain IOs not enough:  

![Parallelism](image)

**Plan A**

- 100 IOs

**Plan B**

- Site 1: 50 IOs
- Site 2: 70 IOs
- Site 3: 50 IOs
• 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

<table>
<thead>
<tr>
<th>Site</th>
<th>Startup</th>
<th>Distribution</th>
<th>Searching + send results</th>
<th>Final proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Site 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Searching strategies

(1) Exhaustive (with pruning)
(2) Hill climbing (greedy)
(3) Query separation
(1) Exhaustive
  - consider “all” query plans
    with a set of techniques
  - prune some plans
    - heuristics
Example: join \[ R \bowtie S \bowtie T \]

\[ |R| > |S| > |T| \]

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

Initial plan

```
1  X
```

Note: The diagram illustrates the concept of Hill climbing, where an initial plan is evaluated, and if it is worse than the current best plan, a new plan is generated. This process repeats until a better plan is found.
(2) **Hill climbing**

Better plans

Worse plans

Initial plan

1

2
Example \( R \bowtie S \bowtie T \bowtie V \)

<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

Goal: minimize data transmission
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 ✔
P₀: R (1 → 4)
    S (2 → 4)
    T (3 → 4)
Compute R ⊙ S ⊙ T ⊙ V at site 4
Local search

- Consider sending each relation to neighbor:

e.g.:
Assume: Size

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

Option (a)

No savings

\[
\begin{align*}
10 & \quad R & 4 & \quad S & 20 \\
\text{cost} = 30 & & & & \\
\end{align*}
\]

\[
\begin{align*}
4 & \quad 20 & \quad R & \times S \\
\text{cost} = 30 & & & \\
\end{align*}
\]
Option (b)

\[
\begin{align*}
\text{cost} &= 30 \\
\text{Worse off!}
\end{align*}
\]

\[
\begin{align*}
\text{cost} &= 40
\end{align*}
\]
Option (c)

cost = 50

A Win!

cost = 35
Option (d)

\[
\begin{array}{c}
\text{S} \\
2
\end{array}
\xrightarrow{20}
\begin{array}{c}
4 \quad T \\
3
\end{array}
\xrightarrow{30}
\begin{array}{c}
2 \\
\text{S}
\end{array}
\xrightarrow{20}
\begin{array}{c}
3
\end{array}
\]

\[
\begin{array}{c}
4
\end{array}
\xrightarrow{5}
\begin{array}{c}
\text{S} \quad \text{T}
\end{array}
\xrightarrow{20}
\begin{array}{c}
3
\end{array}
\]

\text{cost = 50}

A Bigger Win!

\text{cost = 25}
P₁:  P₁a:  S (2 → 3)  
\[ \alpha = S \bowtie T \]

P₁b:  R (1 → 4)  
\[ \alpha (3 \rightarrow 4) \]

compute answer at site 4
Repeat local search

- Treat $\alpha = S \bowtie T$ as relation
Hill climbing may miss best plan!

Example: best plan could be:

\[ P_B: \quad T \quad (3 \rightarrow 4) \]

\[ \beta = T \otimes V \]

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

\[ \beta' = \beta \otimes S \]

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

\[ \beta'' = \beta' \otimes R \]

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

Compute answer
Hill climbing may miss best plan!

Example: best plan could be:

\[ P_B: \quad T \quad (3 \rightarrow 4) \quad \Rightarrow 30 \]

\[ \beta = T \otimes V \]

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

\[ \beta' = \beta \otimes S \]

\[ \beta'' = \beta' \otimes R \]

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

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

\[ \text{Compute answer} \quad 33 = \text{total} \]
(3) **Query separation**
- separate query into 2 or more steps
- optimize each step independently
Example: simple queries

e.g.: \( \sigma_{c_1} \)

1. Compute \( R' = \Pi_A[\sigma_{c_2} R] \)
   \( S' = \Pi_A[\sigma_{c_3} S] \)

2. Compute \( J = R' \bowtie S' \)
1. Compute $R' = \Pi_A[\sigma_{c_2} R]$  
   $S' = \Pi_A[\sigma_{c_3} S]$
2. Compute $J = R' \Join S'$
3. Compute

   $\text{Ans} = \sigma_{c_1}\{[J \Join \sigma_{c_2} R] \Join [J \Join \sigma_{c_3} S]\}$
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
Summary: Query Optimization

- Cost estimation
- Strategies
  - Exhaustive
  - Hill climbing
  - Separation
Words of wisdom

“Optimization is like chess playing”

i.e., May have to make sacrifices

(move data, partition relations, build indexes)

for later gains!