Cost Estimation

Based on estimating **result sizes**

Like in centralized databases

Cost Estimation

But # of IOs may not be the best metric

E.g., *transmission time* may dominate
Cost Estimation

Another reason why # of IOs is not enough: parallelism

Plan A

<table>
<thead>
<tr>
<th>Site</th>
<th>IOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>100 I/Os</td>
</tr>
<tr>
<td>Site 2</td>
<td>50 I/Os</td>
</tr>
<tr>
<td>Site 3</td>
<td>50 I/Os</td>
</tr>
</tbody>
</table>

Plan B

<table>
<thead>
<tr>
<th>Site</th>
<th>IOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>70 I/Os</td>
</tr>
<tr>
<td>Site 2</td>
<td>50 I/Os</td>
</tr>
</tbody>
</table>

Cost Estimation

Cost metrics

*E.g., IOs, bytes transmitted, $,...*

Additive

Response time metric

Not additive

Need scheduling and dependency info

Skew is important

Cost Estimation

Also take into account

- Start up cost
- Data distribution cost/time
- Resource contention (for memory, disk, network)
- Cost of assembling results

Cost Estimation

Response time example

<table>
<thead>
<tr>
<th>Site</th>
<th>工序</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>start up</td>
</tr>
<tr>
<td>Site 2</td>
<td>distribution</td>
</tr>
<tr>
<td>Site 3</td>
<td>searching &amp; sending results</td>
</tr>
<tr>
<td>Site 4</td>
<td>final processing</td>
</tr>
</tbody>
</table>
Search Strategies

- Exhaustive (with pruning)
- Hill climbing (greedy)
- Query separation

Exhaustive Search

Consider all query plans (given a set of techniques for operators)
Prune some plans
Heuristics

Exhaustive Search

Example
\[ R \bowtie S \bowtie T \]

| R | > | S | > | T |

- Prune because cross-product not necessary
- Prune because larger relation first

Search Strategies

In generating plans, keep goal in mind

E.g., if goal is *parallelism* (in system with fast network)
→ Consider partitioning relations first

E.g., if goal is reduction of *network* traffic
→ Consider semi-joins
Hill Climbing

Better plans

Worse plans

Better plans

Worse plans

Hill Climbing

Example

<table>
<thead>
<tr>
<th>relation</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

Hill Climbing

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 ✔
Hill Climbing

\(P_0\)
- \(R (1 \rightarrow 4)\)
- \(S (2 \rightarrow 4)\)
- \(T (3 \rightarrow 4)\)

Compute \(R \bowtie S \bowtie T \bowtie V\) at site 4

Hill Climbing

Local search
Consider sending each relation to neighbor

Hill Climbing

Assume
- size \(R \bowtie S = 20\)
- size \(S \bowtie T = 5\)
- size \(T \bowtie V = 1\)

Option A

Option B

Worse off
Hill Climbing

Option C

```
4  3  2  1
\rightarrow
2  3  2  1
```

\text{cost} = 50

```
4  3  2  1
\rightarrow
2  3  2  1
```

\text{cost} = 35

Win

Hill Climbing

Option D

```
4  3  2  1
\rightarrow
2  3  2  1
```

\text{cost} = 50

```
4  3  2  1
\rightarrow
2  3  2  1
```

\text{cost} = 25

Trigger win

Hill Climbing

\( P_1 \)

\( S (2 \rightarrow 3) \)
\( a = S \bowtie T \)
\( R (1 \rightarrow 4) \)
\( T (3 \rightarrow 4) \)

Compute answer at site 4

Hill Climbing

Repeat local search

Treat \( a = S \bowtie T \) as relation

\[ \begin{array}{c}
\text{vs.}
\end{array} \]
Hill Climbing

Hill climbing may miss best plan
E.g., best plan could be
\[ P_{\text{best}} \]
\[ T \rightarrow 3 \rightarrow 4 \]
\[ \beta = T \bowtie V \]
\[ \beta (4 \rightarrow 2) \]
\[ \beta' = \beta \bowtie S \]
\[ \beta'' = \beta' \bowtie R \]
\[ \beta'' (1 \rightarrow 4) \text{ (optional)} \]
Compute answer

Search Strategies

Exhaustive (with pruning) ✔
Hill climbing (greedy) ✔
Query separation

Query Separation

Separate query into 2 or more steps
Optimize each step independently

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

Total cost: 33
Query Separation

Example
Simple queries technique

\[ \sigma_{c1} \bowtie \sigma_{c2} \sigma_{c3} R \]

Query Separation

1. Compute \[ R' = \Pi_A [ \sigma_{c2} R ] \]
   \[ S' = \Pi_A [ \sigma_{c3} S ] \]
2. Compute \[ J = R' \bowtie S' \]
3. Compute answer \[ \sigma_{c1} \{ [ J \bowtie \sigma_{c2} R ] \bowtie [ J \bowtie \sigma_{c3} S ] \} \]

Query Separation

Simple query
Relations have a single attribute
Output has a single attribute
E.g., \[ J = R' \bowtie S' \]

Query Separation

1. Compute \[ R' = \Pi_A [ \sigma_{c2} R ] \]
   \[ S' = \Pi_A [ \sigma_{c3} S ] \]
2. Compute \[ J = R' \bowtie S' \]
3. Compute answer \[ \sigma_{c1} \{ [ J \bowtie \sigma_{c2} R ] \bowtie [ J \bowtie \sigma_{c3} S ] \} \]

Query Separation

Simple query

Get tuples from sites matching A and compute answer next
Query Separation

Idea

1. **Decompose** query
   - Local processing
     - Simple query (or queries)
   - Final processing

2. **Optimize** simple query

Query Separation

Philosophy

Hard part is distributed join
→ Do this part with only keys; get the rest of the data later
Simpler to optimize simple queries

Summary

Cost estimation
Optimization strategies
   - Exhaustive (with pruning)
   - Hill climbing (greedy)
   - Query separation

Words of Wisdom

*Optimization is like chess playing*

May have to make sacrifices for later gains

Move data, partition relations, build indexes