**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**

- 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

E.g., Transmission time may dominate

![Diagram showing time and cost metrics](image)

Another reason why plain IOs not enough: Parallelism

**Plan A**

- 100 IOs
- site 1: 50 IOs
- site 2: 70 IOs
- site 3: 50 IOs

**Plan B**

- 50 IOs
- site 1: 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 1</th>
<th>Site 2</th>
<th>Site 3</th>
<th>Site 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Startup</td>
<td>Distribution</td>
<td>Searching</td>
<td>Final proc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

\[
\begin{array}{ccc}
R \times S & S \times T & T \\
2 & 1 & 1 \\
\end{array}
\]

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

In generating plans, keep goal in mind:

**Example:**

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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
Hill Climbing Algorithm

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 $P_0$ = 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
Step 4: Replace $P_0$ with $P_1$ that gives
\[
\text{cost}(P_{1a}) + \text{cost(local join)} + \text{cost}(P_{1b}) < \text{cost}(P_0)
\]
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>
<th>tuple size = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>3</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>4</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

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>
<th>tuple size = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>4</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

$T = \sigma(U)$

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
P0: R (1 → 4)  
S (2 → 4)  
T (3 → 4)  
Compute R \bowtie S \bowtie T \bowtie V at site 4

Steps 3 & 4

- Consider sending each relation to neighbor:
  e.g.:

Assume: Size  
R \bowtie S = 20  
S \bowtie T = 5  
T \bowtie V = 1

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>S</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>cost = 30</td>
<td>cost = 30</td>
<td>No savings</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>R \bowtie S</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>cost = 40</td>
<td>A Bigger Win</td>
</tr>
</tbody>
</table>

P1: P1a: S (2 → 3)  
α = S \bowtie T  
P1b: R (1 → 4)  
α (3 → 4)  
compute answer at site 4

Step 5: Repeat Steps 3 & 4

- Treat \( \alpha = S \bowtie T \) as relation

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>α</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>vs</td>
<td></td>
</tr>
</tbody>
</table>

Step 5: Repeat Steps 3 & 4

- Treat \( \alpha = S \bowtie T \) as relation

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>α</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>vs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>α</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>vs</td>
<td></td>
</tr>
</tbody>
</table>
Hill climbing may miss best plan!

Example: best plan could be:

\[ \beta_{T \Rightarrow V} \]
\[ \beta_{(4 \rightarrow 2)} \]
\[ \beta'_{(2 \rightarrow 1)} \]
\[ \beta''_{(1 \rightarrow 4)} \]

Compute answer: \( 33 = \text{total} \)

(3) Query separation
- separate query into 2 or more steps
- optimize each step independently

Example: simple queries

1. Compute \( R' = \Pi_A[\sigma_{c_1} R] \)
2. Compute \( J = R' \bowtie S' \)
3. Compute \( \text{Ans} = \sigma_{c_2} \{[J \bowtie \sigma_{c_2} R] \bowtie [J \bowtie \sigma_{c_2} S]\} \)

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

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>Site</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site 1</td>
<td>30</td>
<td>50</td>
<td>1500</td>
</tr>
<tr>
<td>Site 2</td>
<td>100</td>
<td>30</td>
<td>3000</td>
</tr>
<tr>
<td>Site 3</td>
<td>50</td>
<td>40</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>SF</th>
<th>Size</th>
<th>Size(Rrelief)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.A</td>
<td>0.3</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>R2.A</td>
<td>0.6</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>R2.B</td>
<td>1.0</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>R3.B</td>
<td>0.4</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

Cost = transfer semijoin attribute
\[
= TMSG + TTR \cdot \text{size}
\]
Benefit = cost of transferring irrelevant tuples (avoided by the semijoin)
\[
= (1 - SF) \cdot \text{size} \cdot TTR
\]

Assume: $TMSG = 0$

Iterative Process

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

Step 5: Modify the database profile accordingly

Step 6: Modify $BS$ appropriately
\[
- \text{compute new benefit/cost values}
- \text{check if any new semijoin need to be included in $BS$}
\]

Step 7: If $BS \neq \emptyset$, go back to Step 4.

SDD-1 Algorithm – Example

Iteration 1:
- Remove $SJ_1$ from $BS$ and add it to $ES$.
- Update 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>50</td>
<td>40</td>
<td>2000</td>
</tr>
<tr>
<td>R3</td>
<td>50</td>
<td>40</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>SF</th>
<th>Size</th>
<th>Size(Rrelief)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.A</td>
<td>0.3</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>R2.A</td>
<td>0.6</td>
<td>320</td>
<td></td>
</tr>
<tr>
<td>R2.B</td>
<td>1.0</td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>R3.B</td>
<td>0.4</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

Iteration 2:
- Two beneficial semijoins:
  - $SJ_2 = R2 \bowtie R3$, whose benefit is $540 = (1 - 0.3) \times 900$ and cost is 80
  - $SJ_3 = R1 \bowtie R2$, whose benefit is $300 \times (1 - 0.8) \times 1500$ and cost is 96
- Add $SJ_2$ to $ES$.
- Update statistics

| Size(R2) | 360 (900*0.4) |

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

Iteration 3:
- No new beneficial semijoins.
- Remove remaining beneficial semijoin $SJ_3$ from $BS$ and add it to $ES$.
- Update statistics

| Size(R1) | 1200 (= 1500 * 0.8) |
| SF$_{R1}$ (R1.A) | -0.3 * 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.

---

**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)