Distributed Query Processing

(Chapters 7, 8, 9)

DISTRIBUTED QUERY PROCESSING PROBLEM

Besides the choice of ordering relational algebra operations, we must also select the best sites to process data.

Example:
```
SELECT *
FROM E, G
WHERE E.ENO=G.ENO AND
RESP='Manager'
```

G or ASG (ENO, PNO, RESP, DUR)
E or EMP(ENO, ENAME, TITLE)

Bad algebra: $\sigma_{RESP='Manager' \wedge E.ENO=G.ENO}(E \times G)$

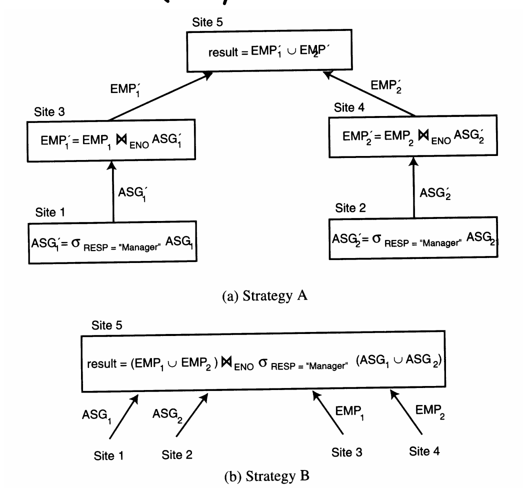
Good algebra: $E \rhd\lhd_{ENO} (\sigma_{RESP='Manager'}(G))$

$$EMP \rhd\lhd_{ENO} (\sigma_{RESP='Manager'}(ASG))$$

Fragmentation Schema

$$EMP_1 = \sigma_{ENO \le "E3"}(EMP)$$
$$EMP_2 = \sigma_{ENO > "E3"}(EMP)$$
$$ASG_1 = \sigma_{ENO \le "E3"}(EMP_1)$$
$$ASG_2 = \sigma_{ENO > "E3"}(EMP_2)$$

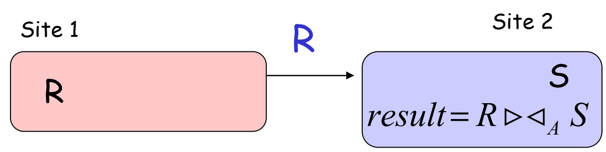
Query Plans



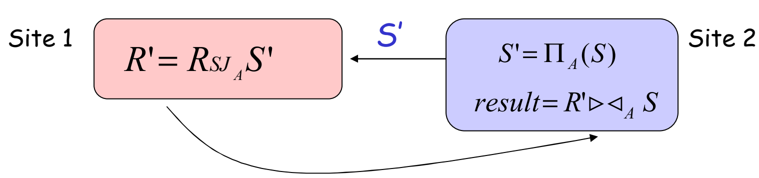
Allocation Schema

Site 1: ASG_1
Site 2: ASG_2
Site 3: EMP_1
Site 4: EMP_2

Regular Join



Semijoin: $R \rhd\lhd_A S \Leftrightarrow (R \ltimes_A S) \rhd\lhd_A S$



Typically $|S'|+|R'| \ll |R|$

More processing overhead to do a join using semi-join

Operation	Complexity
Select Project (without duplicate elimination)	$O(n)$
Project (with duplicate elimination) Group	$O(n*\log n)$
Join Semijoin Division Set Operators	$O(n*\log n)$
Cartesian Product	$O(n^2)$

Figure 7.2. Complexity of Relational Algebra Operations

Assume that tuples of each relation must be sorted.

n denotes the relation cardinality.

PRINCIPLES:

- The most selective operations that reduce cardinalities should be performed first.
- Operations should be ordered by increasing complexity so that expensive operations can be avoided or delayed.

OBJECTIVES OF QUERY OPTIMIZATION

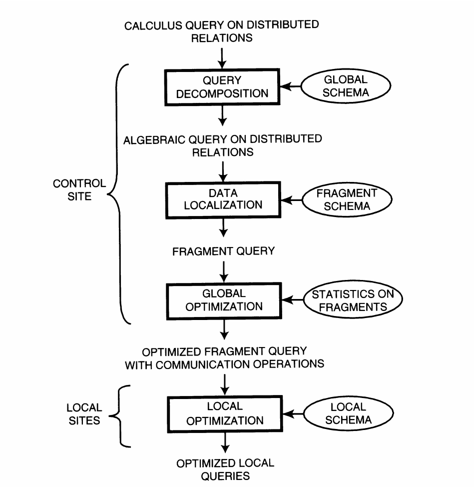
1. Good measurement parameters:
   1. Total cost: The sum of all time incurred in processing the operations at various sites and in intersite communication.
   2. Response Time: The time elapsed for executing the query.

2. **Cost=f(CPU_cost, IO_cost, Communication_Cost)**

   Minimize CPU_cost: use less expensive strategies
   Minimize IO_cost: use fast access methods and good buffer management strategies
   Minimize communication_cost: choose the execution sites intelligently.

   Very slow communication networks:
   Minimize communication costs generally at the expense of local processing.

Distributed Query Processing Steps



Query Decomposition:

- Normalization
- Semantically analyze the normalized query to eliminate incorrect queries.
- Simplify the correct query by removing redundant predicates.
- Restructure the algebraic query into a "better" algebraic specification.

This step is the same as standalone DBMS.

Distributed Query Processing Steps

Data Localization:

- Determine which fragments are involved and transform the global query into fragment queries.

Global Query Optimization:

- Find the "best" ordering of fragment queries and specifies the communication operations.

Local Query Optimization:

- Each site determines the access methods for the local fragment queries using the local schema.

Query Decomposition and Data Localization

Query Decomposition

- Normalization
- Semantic Analysis
- Simplification
- Rewriting

Data Localization: Reduction techniques for different types of fragmentation.

- Horizontal
- Vertical
- Derived
- Hybrid

Normalization

- The input query may be arbitrary complex.
- Normalization transforms the query into a normalized form to facilitate further processing.

Two possible normal forms:

Conjunctive Normal Form:

$$(p_{11} \vee p_{12} \vee \ldots \vee p_{1n}) \wedge (p_{m1} \vee p_{m2} \vee \ldots \vee p_{mn})$$

Disjunctive Normal Form:

$$(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \vee (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$$

where p_{ij} is a simple predicate.

Simple Predicates

Given a relation $R(A_1, A_2, \ldots, A_n)$ where A_i has domain D_i, a simple predicate p_j defined on R has the form

p_j: A_i θ *Value*

where $\theta \in \{=,<,\neq,\leq,>,\geq\}$ and *Value* $\in D_i$

Example:

J

JNO	JNAME	BUDGET	LOC
J1	Instrumental	150,000	Montreal
J2	Database Dev.	135,000	New York
J3	CAD/CAM	250,000	New York
J4	Maintenance	350,000	Orlando

Simple predicates:

p_1: JNAME= 'Maintenance'

P_2: BUDGET= '200,000'

Examples:

E(ENO, ENAME, TITLE)
G(ENO, JNO, RESP, DUR)

Query: SELECT ENAME
FROM E, G
WHERE E.ENO=G.ENO AND G.JNO='J1' AND
(DUR=12 OR DUR=24)

Query qualification

Qualification in conjunctive normal form:

$$(E.ENO = G.ENO) \wedge (G.JNO = 'J1') \wedge (DUR = 12 \vee DUR = 24)$$

Qualification in disjunctive normal form:

$$(E.ENO = G.ENO \wedge G.JNO = 'J1' \wedge DUR = 12) \vee$$
$$(\underline{E.ENO = G.ENO} \wedge \underline{G.JNO = 'J1'} \wedge DUR = 24)$$

Replicated join

Replicated select

Disjunctive normal form can lead to replicated join and select predicates.

Semantic Analysis

Semantic analysis enables rejection of incorrect queries.

- Type checking detects type incorrect problems.

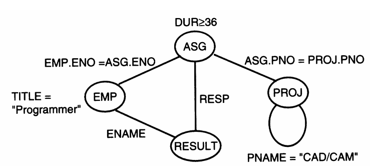
```
SELECT E#              <- Undefined attribute!
FROM E
WHERE ENAME>200;       <- Incompatible operation!
```

- **Query graph**: Determine the semantic correctness of a conjunctive multivariable (multi-table) query without negation.

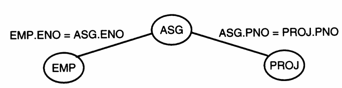
```
SELECT ENAME, RESP
FROM EMP,ASG,PROJ
WHERE EMP.ENO=ASG.ENO AND PNAME='CAD/CAM' AND
DUR>=36 AND TITLE='Programmer'
```

```
SELECT ENAME, RESP
FROM EMP,ASG,PROJ
WHERE EMP.ENO=ASG.ENO AND
PNAME='CAD/CAM' AND DUR>=36 AND
TITLE='Programmer'
```

EMP(<u>ENO</u>, ENAME, TITLE)
AGG(<u>ENO, JNO</u>, RESP, DUR)
PROJ(<u>PNO</u>,PNAME,LOC)



(a) Query graph



(b) Corresponding join graph

Systems should reject queries with the unconnected join graphs.

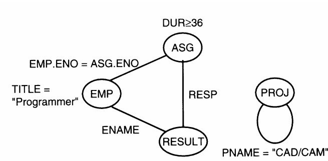


Figure 8.2. Disconnected Query Graph

Elimination of Redundancy

Redundant predicates are likely to arise when a query is the result of system transformations applied to the user query.

Such redundancy and thus redundant work may be eliminated by simplifying the qualification using well-known idempotency rules.

Example:

$$
\begin{aligned}
&(\neg p_1 \wedge (p_1 \vee p_2) \wedge \neg p_2) \vee p_3 \\
&= (\neg p_1 \wedge p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2 \wedge \neg p_2) \vee p_3 \\
&= (F \wedge \neg p_2) \vee (\neg p_1 \wedge F) \vee p_3 \\
&= F \vee F \vee p_3 = p_3
\end{aligned}
$$

Rewriting

Two Steps:

1. Transforming the query into an algebraic relational query tree.
2. Restructuring the algebraic tree to improve performance