STEP 3 – GLOBAL QUERY OPTIMIZATION

Input: Fragment query

- find the $best$ (not necessarily optimal) global schedule
  - minimize a cost function
  - distributed join processing
    - bushy vs linear trees
    - which relation to ship where?
    - ship-whole vs ship-as-needed
  - decide on the use of semijoins
    - semijoin saves on communication at the expense of more local processing
  - join methods
    - nested loops vs ordered joins (merge join or hash join)

COST-BASED OPTIMIZATION

- solution space
  - the set of equivalent algebra expressions (query trees)
- cost function (in terms of time)
  - I/O cost + CPU cost + communication cost
  - these might have different weights in different distributed enviornments (LAN vs WAN)
  - can also maximize throughput
- search algorithm
  - how do we move inside the solution space?
    * exhaustive search
    * heuristic algorithms
    * randomized algorithms

COST FUNCTIONS

- total time (or total cost)
  - reduce each cost component individually
  - do as little of each cost component as possible
  - optimizes the utilization of the resources (increases the system throughput)
- response time
  - do as many things as possible in parallel
  - may increase total time because of increased total activity

Total Cost

summation of all cost factors

- total cost = CPU cost + I/O cost + communication cost
  - CPU cost = unit instruction cost $\times$ no. of instructions
  - I/O cost = unit disk I/O cost $\times$ no. of disk I/Os
  - communication cost = message initiation + transmission
- wide area network
  - message initiation and transmission costs high
  - local processing cost is low (fast mainframes or minicomputers)
  - ratio of communication to I/O costs = 20:1
- local area network
  - communication and local processing costs are more or less equal
  - ratio = 1:1.6

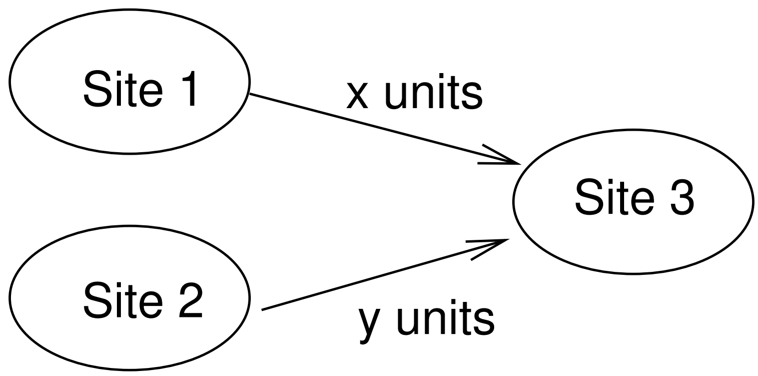
Response Time

Elapsed time between the initiation and the completion of a query

- response time = max $\{$ CPU time + I/O time + communication time $\}$
  - CPU time = unit instruction time $\times$ seq_no. of instructions
  - I/O time = unit I/O time $\times$ seq_no. of I/Os
  - communication time =
    unit msg initiation time $\times$ seq_no. of msg
    + unit transmission time $\times$ seq_no. of bytes

  where "seq_no. of U" is the maximum number of U which must be done sequentially for the execution of the query.

EXAMPLE



Assume that only the communication cost is considered

- total time = 2 $\times$ msg initialization time + unit transmission time $\times$ (x + y)
- response time = max $\{$ time to send x from 1 to 3, time to send y from 2 to 3 $\}$
  - time to send x from 1 to 3 = msg initiation time + unit transmission time $\times$ x
  - time to send y from 2 to 3 = msg initiation time + unit transmission time $\times$ y

OPTIMIZATION STATISTICS

- primary cost factor: size of intermediate relations
- make them precise (more costly to maintain)
  - for each relation $R[A_1, \ldots, A_n]$ fragmented as $R_1, \ldots, R_r$
    - length of each attribute: $length(A_i)$
    - the number of distinct values for each attribute in each fragment: $card(\pi_{A_i} R_j)$
    - minimum and maximum values in the domain of each attribute: $min(A_i)$, $max(A_i)$
    - the cardinality of each domain: $card(dom[A_i])$
    - the cardinality of each fragment: $card(R_j)$
  - selectivity factor of each operation for relations
    - for joins

$$SF_{\bowtie}(R, S) = \frac{card(R \bowtie S)}{card(R) \times card(S)}$$

INTERMEDIATE RELATION SIZES

Selection

$$size(R) = card(R) \times length(R)$$

where length(R) is the record length of R

$$card(\sigma_F(R)) = SF_\sigma(F) \times card(R)$$

where

$$SF_\sigma(A = value) = \frac{1}{card(\pi_A(R))}$$

Assume that the values of A is uniformly distributed in the relation R

$$SF_\sigma(A > value) = \frac{max(A) - value}{max(A) - min(A)}$$

$$SF_\sigma(A < value) = \frac{value - min(A)}{max(A) - min(A)}$$

$$SF_\sigma(p(A_i) \wedge p(A_j)) = SF_\sigma(p(A_i)) \times SF_\sigma(p(A_j))$$

$$\begin{aligned} SF_\sigma(p(A_i) \vee p(A_j)) \;=\; & SF_\sigma(p(A_i)) + SF_\sigma(p(A_j)) - \\ & (SF_\sigma(p(A_i)) \times SF_\sigma(p(A_j))) \end{aligned}$$

$$SF_\sigma(A = values) = SF_\sigma(A = value) \times card(values)$$

where values is a set of values and value $\in$ values

INTERMEDIATE RELATION SIZES (CONT)

Projection

$$card(\pi_A(R)) = card(R)$$

if A contains a key of R

Cartesian Product

$$card(R \times S) = card(R) * card(S)$$

Union

$$upperbound : card(R \cup S) = card(R) + card(S)$$

$$lowerbound : card(R \cup S) = max\{card(R), card(S)\}$$

Set Difference

$$upperbound : card(R - S) = card(R)$$

$$lowerbound : card(R - S) = 0$$

INTERMEDIATE RELATION SIZES (CONT)

Join

- special case: A is a key of R and B is a foreign key of S

$$card(R \bowtie_{A=B} S) = card(S)$$

- more general

$$card(R \bowtie_{A=B} S) = SF_{\bowtie}(R \bowtie_{A=B} S) * card(R) * card(S)$$

Semijoin

$$card(R \ltimes_A S) = SF_{\ltimes}(R \ltimes_A S) * card(R)$$

where

$$SF_{\ltimes}(R \ltimes_A S) = \frac{card(\pi_A(S))}{card(dom[A])}$$

Note that $SF_{\ltimes}(R \ltimes_A S)$ depends only on attribute A of S. It is often called the selectivity factor of attribute A of S, denoted $SF_{\ltimes}(S.A)$.

CENTRALIZED QUERY OPTIMIZATION

- INGRES
  - use a dynamic query optimization algorithm
  - interpretive
- System R
  - use a static query optimization algorithm
  - exhaustive search using statistics about the database
  - query compilation

INGRES ALGORITHM

no statistical information is maintained

- decompose each multi-variable query into a sequence of mono-variable queries with a common variable
- a multi-variable query is one that refers to more than one relation or refers to a relation more than once
- process each mono-variable query by a one variable query processor (OVQP)
  - choose an initial execution plan (heuristics)
  - order the rest by considering intermediate relation sizes
  - choose the best access method to that relation (e.g. index or sequential scan)
    e.g. if the predicate is of the form <A=value>, an index available on attribute A would be used. However, if the predicate is <A $\neq$ value>, an index on A would not help, and sequential scan should be used.

INGRES ALGORITHM – DECOMPOSITION

- replace an n variable query q by a series of n queries
$$q_1 \rightarrow q_2 \rightarrow \ldots \rightarrow q_n$$
where q_i uses the result of q_{i-1}
- detachment
  - query q decomposed into $q' \rightarrow q''$ where q' and q'' have a common variable which is the result of q'
    where $q' \rightarrow q''$ indicates query q' executes before query q" which uses the result of q'
- tuple substitution
  - replace the value of each tuple with actual values and simplify the query
$$q(V_1, \ldots, V_n) \rightarrow \bigcup_{t_1 \in R} q'(t_1, V_2, \ldots, V_n)$$
    where the range of the variable V_1 is the relation R

DETACHMENT

q: **SELECT** $V_2.A_2$, $V_3.A_3$, ..., $V_n.A.n$
FROM R_1 V_1, ..., R_n V_n
WHERE $P_1(V_1.A_1')$ **AND** $P_2(V_1.A_1, V_2.A_2, \ldots, V_n.A.n)$

query q is decomposed into 2 subqueries q_1 followed by q_2, by detachment of the common variable V_1

q': **SELECT** $V_1.A_1$ **INTO** R_1'
FROM R_1 V_1
WHERE $P_1(V_1.A_1')$

R_1' is a temporary relation

q'': **SELECT** $V_2.A_2$, ..., $V_n.A.n$
FROM R_1' V_1, R_2 V_2, ..., R_n V_n
WHERE $P_2(V_1.A_1, V_2.A_2, \ldots, V_n.A.n)$

DETACHMENT EXAMPLE

Query: Get names of employees working on CAD/CAM project

q_1: **SELECT** E.ENAME
FROM E, G, J
WHERE E.ENO = G.ENO
AND G.JNO = J.JNO
AND J.JNAME = 'CAD/CAM'

q_1 is decomposed into $q_{11} \rightarrow q'$

q_{11}: **SELECT** J.JNO **INTO** JVAR
FROM J
WHERE J.JNAME = 'CAD/CAM'

q': **SELECT** E.ENAME
FROM E, G, JVAR
WHERE E.ENO = G.ENO
AND G.JNO = JVAR.JNO

q' is decomposed into $q_{12} \rightarrow q_{13}$

q_{12}: **SELECT** G.ENO **INTO** GVAR
FROM G, JVAR
WHERE G.JNO = JVAR.JNO

q_{13}: **SELECT** E.ENAME
FROM E, GVAR
WHERE E.ENO = GVAR.ENO

Hence q_1 is decomposed into $q_{11} \rightarrow q_{12} \rightarrow q_{13}$.
q_{11} is a mono-variable query and can be performed by OVQP. However, q_{12} and q_{13} are not mono-variable queries and cannot be reduced by detachment, use tuple substitution instead.

TUPLE SUBSTITUTION

- q_{11} is a mono-variable query
- q_{12} and q_{13} are subject to tuple substitution
- assume GVAR has two tuples only: $<$E1$>$ and $<$E2$>$
- Then q_{13} becomes

```
q131: SELECT  E.ENAME
      FROM    E
      WHERE   E.ENO = 'E1'

q132: SELECT  E.ENAME
      FROM    E
      WHERE   E.ENO = 'E2'
```

Note that q_{131} and q_{132} are mono-variable queries. The result of q_{13} is the union of the results of q_{131} and q_{132}. It is also the result of the original query q_1.

SYSTEM R ALGORITHM

- simple (i.e., mono-relation) queries are executed according to the best access path
- execute joins
  - determine the possible ordering of joins
  - determine the cost of each ordering
  - choose the join ordering with minimal cost

SYSTEM R ALGORITHM (CONT)

for joins, two alternative algroithms:

- nested loops

  foreach tuple of $outer$ relation (cardinality n_1)
      foreach tuple of $inner$ relation (cardinality n_2)
          join two tuples if the join predicate is true
      end
  end

  complexity: $n_1 \times n_2$

  For the join $R \bowtie S$, we call R the outter relation and S the inner relation.

- merge join

  sort relations
  merge relations

  complexity: $n_1 + n_2$ for equijoin if relations are already sorted

Exercise: read some hash join algorithms

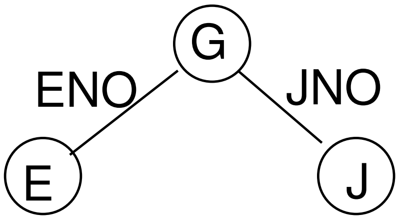
SYSTEM R ALGORITHM – EXAMPLE

Recall the query: Get names of employees working on the CAD/CAM project

q_1: **SELECT** E.ENAME
FROM E, G, J
WHERE E.ENO = G.ENO
AND G.JNO = J.JNO
AND J.JNAME = 'CAD/CAM'

assume

- E has an index on ENO
- G has an index on JNO
- J has an index on JNO and an index on JNAME

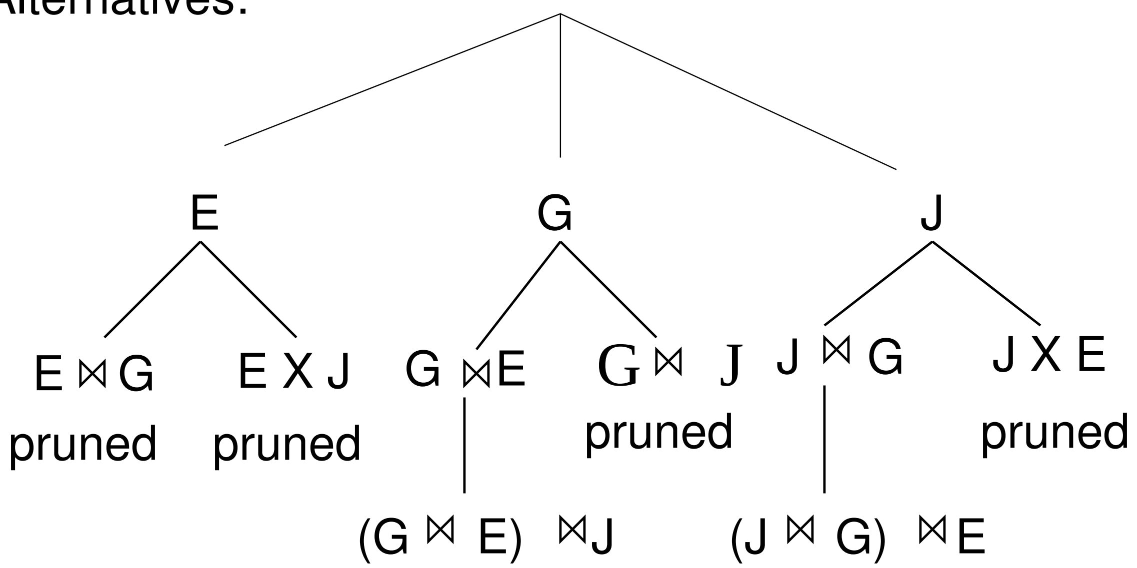


SYSTEM R EXAMPLE (CONT)

- choose the best access paths to each relation
  - E: sequential scan (no selection on E)
  - G: sequential scan (no selection on G)
  - J: index on JNAME (there is a selection on J based on JNAME)
- determine the best join ordering
  - $E \bowtie G \bowtie J$
  - $G \bowtie E \bowtie J$
  - $G \bowtie J \bowtie E$
  - $J \bowtie G \bowtie E$
  - $E \bowtie J \bowtie G$
  - $J \bowtie E \bowtie G$
  - the maximum number of join orders is 3!
  - select the best ordering based on the join costs evaluated according to the two methods

SYSTEM R EXAMPLE (CONT)

Alternatives:



The cost of Cartesian products is high, try to prune them whenever possible.
Assume that $(E \bowtie G)$ and $(G \bowtie J)$ have a cost higher than $(G \bowtie E)$ and $(J \bowtie G)$, resp, they can be pruned.

Best total join order is one of

- $((G \bowtie E) \bowtie J)$
- $((J \bowtie G) \bowtie E)$

SYSTEM R ALGORITHM

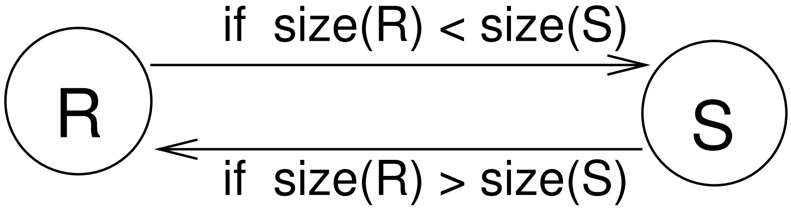
- $((J \bowtie G) \bowtie E)$ has a useful index on the select attribute and direct access to the join attributes of G and E
- thus, choose it with the following access methods:
  - select J using index on JNAME
  - then join with G using index on JNO
  - then join with E using index on ENO

JOIN ORDERING IN FRAGMENT QUERIES

- ordering joins
  - Distributed INGRES
  - System R*
- semijoin ordering
  - SDD-1
  - Apers-Hevner-Yao Algorithms

JOIN ORDERING

- consider two relations only: computer the join of R and S, where R and S are stored at different sites



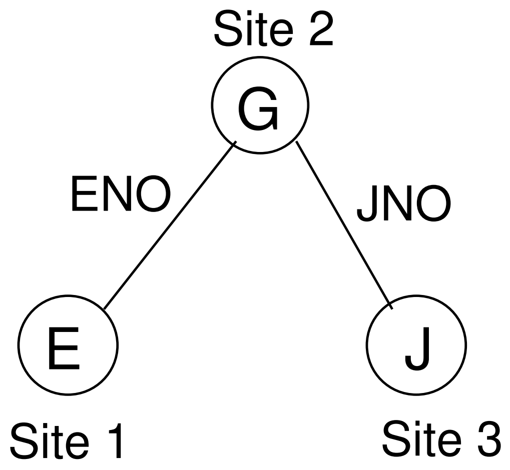
  Obvious choice: transfer the smaller relation to the site of the larger one

- multiple relations more difficult because too many alternatives
  - compute the cost of all alternatives and select the best one
    - necessary to compute the size of intermediate relations which is difficult
  - use heuristics

JOIN ORDERING – EXAMPLE

Consider

$$E \bowtie_{ENO} G \bowtie_{JNO} J$$



JOIN ORDERING – EXAMPLE (CONT)

execution alternatives:

(E $\longrightarrow$ Site i) stands for "relation R is transferred to Site i

- E $\rightarrow$ Site 2
  Site 2 computes E' = E $\bowtie$ G
  E' $\rightarrow$ Site 3
  Site 3 computes E' $\bowtie$ J
- G $\rightarrow$ Site 1
  Site 1 computes E' = E $\bowtie$ G
  E' $\rightarrow$ Site 3
  Site 3 computes E' $\bowtie$ J
- G $\rightarrow$ Site 3
  Site 3 computes G' = G $\bowtie$ J
  G' $\rightarrow$ Site 1
  Site 1 computes G' $\bowtie$ E
- J $\rightarrow$ Site 2
  Site 2 computes J' = J $\bowtie$ G
  J' $\rightarrow$ Site 1
  Site 1 computes J' $\bowtie$ E
- E $\rightarrow$ Site 2
  J $\rightarrow$ Site 2
  Site 2 computes E $\bowtie$ J $\bowtie$ G

To select one of these, sizes of relations E, G, J, $E \bowtie G$, $G \bowtie J$ must be known/predicted.

Note that the transfers can be in parallel in strategy 5.

SEMIJOIN ALGORITHMS

- consider the join of two relations:
  - R[A] located at site 1
  - S[A] located at site 2
- alternatives
  - do the join $R \bowtie_A S$
  - perform one of the semijoin equivalents

$$\begin{aligned} R \bowtie_A S &\leftrightarrow (R \ltimes_A S) \bowtie_A S \\ &\leftrightarrow R \bowtie_A (S \ltimes_A R) \\ &\leftrightarrow (R \ltimes_A S) \bowtie (S \ltimes_A R) \end{aligned}$$

  - Which semijoin strategy is the best?

SEMIJOIN ALGORITHMS (CONT)

assume R at Site 1 and S at Site 2

- perform the join $R \bowtie_A S$
  - send R to Site 2
  - site 2 computes $R \bowtie_A S$
- consider semijoin $(R \rtimes_A S) \bowtie_A S$
  - $S' \leftarrow \pi_A(S)$
  - $S' \rightarrow$ site 1
  - site 1 computes $R' = R \rtimes_A S'$
  - $R' \rightarrow$ site 2
  - site 2 computes $R' \bowtie_A S$

semijoin is better if

$$size(\pi_A(S)) + size(R \rtimes_A S)) < size(R)$$