DISTRIBUTED QUERY PROCESSING

	Distributed INGRES	R*	SDD-1	AHY
Optm. Timing	Dynamic	Static	Static	Static
Objective Ftn.	Response Time/ Total Cost	Total Cost	Total Cost	Respose/ Total
Optm. Factors	Msg Size, Processing	#Msg, Msg size, IO, CPU	Msg Size	#Msg, Msg Size
Network Topology	General/ Broadcast	General/ Local	General	General
Semijoin	no	no	yes	yes
Statistics$^+$	1	1, 2	1, 3, 4, 5	1, 3, 5
Fragments	Horizontal	no	no	no

+: 1 – relation cardinality;
2 – number of unique values per attribute
3 – join selectivity factor
4 – size of projection on each join attribute
5 – attribute size and tuple size

DISTRIBUTED INGRES ALGORITHM

same as the centralized version except

- movement of relations (and fragments) need to be considered
- optimization with respect to communication cost or response time possible

R* ALGORITHM

- cost function includes local processing as well as transmission
- considers only joins
- exhaustive search
- compilation
- published papers provide solutions to handling horizontal and vertical fragmentations but the implemented prototype does not

R* ALGORITHM (CONT)

performing joins

- ship whole
  - larger data transfer
  - smaller number of messages
  - better if relations are small
- fetch as needed
  - number of messages = O(cardinality of external relation)
  - data transfer per message is minimal
  - better if relations are large and the selectivity is good

R* ALGORITHM – VERTICAL PARTITIONING & JOINS

Strategy 1. move outer relation tuples to the site of the inner relation

- retrieve outer tuples
- send them to the inner relation site
- join them as they arrive

$$
\begin{aligned}
total\ cost\ =\ & cost(retrieving\ qualified\ outer\ tuples) \\
& +\ \#outer\ tuples\ fetched \times \\
& cost(retrieving\ qualified\ inner\ tuples) \\
& +\ msg.\ cost \times (\#outer\ tuples\ fetched \times \\
& avg.\ outer\ tuple\ size)/(msg.\ size)
\end{aligned}
$$

R* ALGORITHM – VERTICAL PARTITIONING & JOINS (CONT)

Strategy 2. move inner relation to the site of outer relation

- cannot join as they arrive; they need to be stored first
- perform a uniprocessor join

$$
\begin{aligned}
total\ cost\ =\ & cost(retrieving\ qualified\ outer\ tuples)\\
& + \#outer\ tuples\ fetched \times\\
& cost(retrieving\ matching\ inner\ tuples from\\
& temporary storage)\\
& + cost(retrieving\ qualified\ inner\ tuples)\\
& + cost(storing\ all\ qualified\ inner\ tuples\\
& in\ temporary\ storage)\\
& + msg.\ cost\ \times\ (\#inner\ tuples\ fetched\ \times\\
& avg.\ inner\ tuple\ size)/(msg.\ size)
\end{aligned}
$$

R* ALGORITHM – VERTICAL PARTITIONING & JOINS (CONT)

Strategy 3. move both relations to another site

$$
\begin{aligned}
total\ cost \ = \ & cost(retrieving\ qualified\ outer\ tuples) \\
& + cost(retrieving\ qualified\ inner\ tuples) \\
& + cost(storing\ inner\ tuples\ in\ storage) \\
& + msg.\ cost \times (\#outer\ tuples\ fetched \times \\
& avg.\ outertuplesize)/(msg.\ size) \\
& + msg.\ cost \times (\#inner\ tuples\ fetched \times \\
& avg.\ inner\ tuple\ size)/(msg.\ size) \\
& + \#outer\ tuples\ fetched \times \\
& cost(retrieving\ inner\ tuples\ from \\
& temporary\ storage)
\end{aligned}
$$

R* ALGORITHM – VERTICAL PARTITIONING & JOINS (CONT)

Strategy 4. fetch inner tuples as needed

- retrieve qualified tuples at outer relation site
- send request containing join column value(s) for outer tuples to inner relation site
- retrieve matching inner tuples at inner relation site
- send the matching inner tuples to outer relation site
- join as they arrive

$$
\begin{aligned}
total\ cost \ = \ & cost(retrieving\ qualified\ outer\ tuples) \\
& + msg.\ cost \times (\#outer\ tuples\ fetched) \\
& + \#outer\ tuples\ fetched \times \\
& (\#inner\ tuples\ fetched \times avg.\ inner\ tuple\ size \\
& \times msg.\ cost)/(msg.\ size) \\
& + \#outer\ tuples\ fetched \times \\
& cost(retrieving\ matching\ inner\ tuples\ for \\
& one\ outer\ value)
\end{aligned}
$$

SDD-1 ALGORITHM

based on Hill Climbing Algorithm

- no semijoins
- no replication
- no fragmentation
- cost of transferring the result to the user site from the final result site is not considered
- can minimize either total time or response time

HILL CLIMBING ALGORITHM

Assume join is between three relations

- step 1: do initial processing
- step 2: select initial feasible solution (ES_0)
  - determine the candidate result sites – sites where a relation referenced in the query exist
  - compute the cost of transferring all the other referenced relations to each candidate site
  - ES_0 = candidate site with minimum cost
- step 3: determine candidate splits of ES_0 into $\{ES_1, ES_2\}$
  - ES_1 consists of sending one of the relations to the other relations's site
  - ES_2 consists of sending the join of the relations to the final result site
- step 4: replace ES_0 with the split schedule which gives

  $cost(ES_1) + cost(local\ join) + cost(ES_2) < cost(ES_0)$

HILL CLIMBING ALGORITHM

- step 5: recursively apply steps 3 and 4 on ES_1 and ES_2 until no such plans can be found
- step 6: check for redundant transmissions in the final plan and eliminate them

Note: If n-way joins are involved, ES_0 will be divided into n subschedules instead of just 2.

HILL CLIMBING ALGORITHM – EXAMPLE

what are the salaries of engineers who work on the CAD/CAM project?

$$\pi_{SAL}(S \bowtie_{TITLE} (E \bowtie_{ENO} (G \bowtie_{JNO} (\sigma_{JNAME=CAD/CAM}(J)))))$$

Relation	Size	Site
E	8	1
S	4	2
J	4	3
G	10	4

Assume:

- size of relations is defined as their cardinality
- minimize total cost
- transmission cost between two sites is 1
- ignore local processing cost
- Based on join selectivities, assume that
  $size(E \bowtie S) = size(E)$
  $size(J \bowtie G) = 2 * size(J)$
  $size(G \bowtie E) = size(G)$

HILL CLIMBING ALGORITHM – EXAMPLE (CONT)

step 1:

- selection on J
  assume that there are 25% of projects with name "CAD/CAM"
- result has cardinality 1

Relation	Size	Site
E	8	1
S	4	2
J	1	3
G	10	4

HILL CLIMBING ALGORITHM – EXAMPLE (CONT)

step 2: initial feasible solution

- alternative 1: resulting site is Site 1

$$\begin{aligned} total\ cost &= cost(S \rightarrow Site\ 1) \\ &\quad + cost(G \rightarrow Site\ 1) \\ &\quad + cost(J \rightarrow Site\ 1) \\ &= 4 + 10 + 1 = 15 \end{aligned}$$

- alternative 2: resulting site is Site 2

$$total\ cost = 8 + 10 + 1 = 19$$

- alternative 3: resulting site is Site 3

$$total\ cost = 8 + 4 + 10 = 22$$

- alternative 4: resulting site is Site 4

$$total\ cost = 8 + 4 + 1 = 13$$

thus, alternative 4 is the initial feasible solution
$ES_0 = \{E \rightarrow Site4;\ S \rightarrow Site4;\ J \rightarrow Site4\}$

HILL CLIMBING ALGORITHM – EXAMPLE (CONT)

step 3: determine candidate splits

- alternative 1: $\{ES_1, ES_2, ES_3\}$ where
  - $ES_1 : E \rightarrow Site\ 2$
  - $ES_2 : (E \bowtie S) \rightarrow Site\ 4$
  - $ES_3 : J \rightarrow Site\ 4$
- alternative 2: $\{ES_1, ES_2, ES_3\}$ where
  - $ES_1 : S \rightarrow Site\ 1$
  - $ES_2 : (S \bowtie E) \rightarrow Site\ 4$
  - $ES_3 : J \rightarrow Site\ 4$

  Question: Any other alternatives?

HILL CLIMBING ALGORITHM – EXAMPLE (CONT)

step 4: determine costs of each split alternative

$$\begin{aligned} cost(alternative\ 1) &= cost(E \rightarrow Site\ 2) \\ &+ cost((E \bowtie S) \rightarrow Site\ 4) \\ &+ cost(J \rightarrow Site\ 4) \\ &= 8 + 8 + 1 = 17 \end{aligned}$$

Note that $size(S \bowtie E) = size(E) = 8$

$$\begin{aligned} cost(alternative\ 2) &= cost(S \rightarrow Site\ 1) \\ &+ cost((S \bowtie E) \rightarrow Site\ 4) \\ &+ cost(J \rightarrow Site\ 4) \\ &= 4 + 8 + 1 = 13 \end{aligned}$$

Decision: DO NOT SPLIT

step 5: ES_0 is the “best”

step 6: No redundant transmissions

HILL CLIMBING ALGORITHM

problems

- greedy algorithm $\rightarrow$ determines an initial feasible solution and interatively tries to improve it
- if there are local minimas, it may not find global minima
- if the optimal schedule has a high initial cost, it won't find it since it won't choose it as the initial feasible solution

Example: A better schedule is

- $J \rightarrow Site\ 4$
- $G' = (J \bowtie G) \rightarrow Site\ 1$
- $(G' \bowtie E) \rightarrow Site\ 2$
- total cost = 1 + 2 + 2 = 5
  since
  $size(G') = size(J \bowtie G) = 2 * size(J) = 1 * 2 = 2$
  $size(G' \bowtie E) = size(G') = 2$

SDD-1 ALGORITHM

initialization

- 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

    $$BS \leftarrow BS \cup SJ_i \;\; if \;\; cost(SJ_i) < benefit(SJ_i)$$

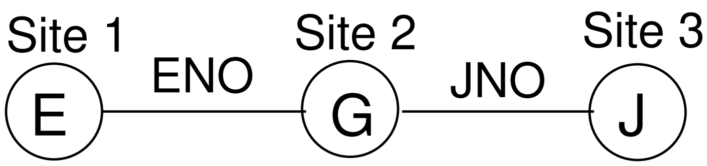
    where $cost(SJ_i)$ is the communication cost of a semijoin for transferring the semijoin attributes A,
    $cost(R \rhd\!\!< _A S) = C_{MSG} + C_{TR} * size(\pi_A(S))$
    while its benefit is not to transfer the irrelevant tuples of R
    $benefit(R \rhd\!\!< _A S) = (1 - SF_{\ltimes}(S.A)) * size(R) * C_{TR}$

SDD-1 ALGORITHM – EXAMPLE

Consider the following query

```
SELECT   *
FROM     E, G, J
WHERE    E.ENO = G.ENO
AND      G.JNO = J.JNO
```

which has the following query graph, statistics and assume $C_{MSG} = 0$ and $C_{TR} = 1$



relation	card	tuple size	relation size
E	30	50	1500
G	100	30	3000
J	50	40	2000

attribute	$SF_{\ltimes}$	$size(\pi_{attribute})$
E.ENO	0.3	120
G.ENO	0.8	400
G.JNO	1.0	400
J.JNO	0.4	200

SDD-1 ALGORITHM – EXAMPLE (CONT)

- beneficial semijoins:
  - $SJ_1 = G \rhd\!< E$, whose benefit is $(1 - SJ_{\rhd<}(E.ENO) * size(G) * C_{TR} = (1 - 0.3) * 3000 * 1 = 2100$
    and cost is $C_{MSG} + C_{TR} * size(\pi_{ENO}(E)) = 0 + 1 * 120 = 120$

  - $SJ_2 = G \rhd\!< J$, whose benefit is (1 - 0.4)*3000 = 1800 and cost is 200
- nonbeneficial semijoins:
  - $SJ_3 = E \rhd\!< G$, whose benefit is (1 - 0.8)*1500 = 300 and cost is 400
  - $SJ_4 = J \rhd\!< G$, whose benefit is 0 and cost is 400

SDD-1 ALGORITHM

Iterative Process

- step 4: remove the most beneficial SJ_i from BS and append it to ES
- step 5: modify the database profile accordingly
- step 6: modify BS appropriately
  - compute new benefit/cost values
  - 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

Iterative Process

- Iteration 1:
  - remove SJ_1 from BS and add it to ES
  - update statistics of G
    - $size(G) = 3000 \times 0.3 = 900$
      the size of Relation G is change to $size(G) * SJ_{\ltimes}(E.ENO)$ because of the semi-join $G \ltimes_{ENO} E$
    - $SF_{\ltimes}(G.ENO) = 0.8 \times 0.3 = 0.24$
      the semijoin selectivity factor of attribute G.ENO is reduced becuase $card(\pi_{ENO}(G))$ is reduced.
- Iteration 2:
  - two beneficial semijoins:
    - $SJ_2 = G' \ltimes J$, whose benefit is $(1 - 0.4) \times 900$ = 540 and cost is 200
      where $size(G') = size(G \ltimes E) = 900$ from SJ_1.
    - $SJ_3 = E \ltimes G'$, whose benefit is $(1-0.24) \times 1500$ = 1400 and cost is 400
  - add SJ_3 to ES
  - update statistics of E
    - $size(E) = 1500 \times 0.24 = 360$
    - $SF_{\ltimes}(E.ENO) = 0.3 \times 0.24 = 0.072$

SDD-1 ALGORITHM – EXAMPLE (CONT)

Iterative Process

- Iteration 3:
  - no new beneficial semijoins
  - remove remaining beneficial semijoin SJ_2 from BS and add it to ES
  - update statistics of G
    - $size(G) = 900 \times 0.4 = 360$

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: 360 (size of new E)
Site 2: 360 (size of new G)
Site 3: 2000 (no change on J)

Therefore, Site 3 will be chosen as the assembly site

SDD-1 ALGORITHM

Postprocessing

- step 9: for each R_i at the assembly site, find the semijoins of the type

  $$R_i \rtimes R_j$$

  where the total cost of ES without this semijoin is smaller than the cost with it and remove the semijoin from ES
  Note: There might be indirect benefits, e.g. no semijoins are removed

  For this example, no semijoins are removed since they are all beneficial.

- step 10: premute the order of semijoins if doing so would improve the total of ES, e.g., final strategy:
  Send $(G \rtimes E) \rtimes J$ to Site 3
  Send $E \rtimes G'$ to Site 3
  Final result is computed at Site 3.

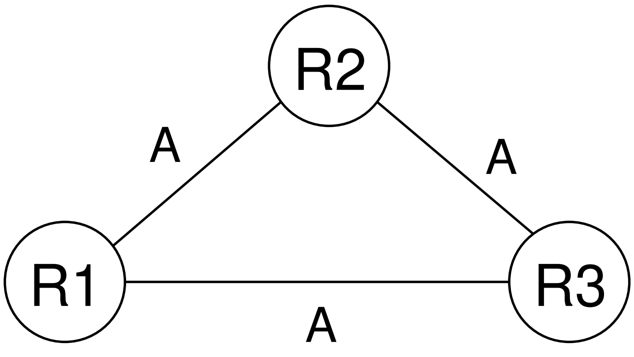
Note: This algorithm may not be able to select the global minimum cost solution.

APERS-HEVNER-YAO ALGORITHM

- user specified result site
- makes use of semijoins
- can be used for total time minimization or response time minimization
- considers transmission cost only
- simple queries
  - those queries where, after initial local processing, each relation in the query contains only the common join attribute, which is also the only output of the query
  - simple queries can be optimally solved.
- general queries

AHY ALGORITHM – REPRESENTATION

consider the following simple query and its statistics

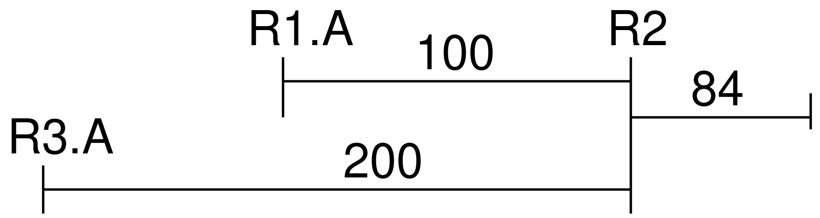


attribute	$SF_{\ltimes}$	size($\pi_{attribute}$)
$R_1.A$	0.3	100
$R_2.A$	1.0	400
$R_3.A$	0.7	200

It can be represented by the following schedule:

- (1) send attribute R_1.A and attribute R_3.A to the site of R_2
- (2) perform $R_2' = R_2 \ltimes_A R_1$ and $R_2'' = R_2' \ltimes_A R_3$ at the site of R_2 and send the relevant tuples of R_2 (after reduction by semijoins, i.e. R_2'') to the result sites.

The size of R_2 after semijoins is 400*0.3*0.7=84



SIMPLE QUERY OPTIMIZATION

total time minimization

- move smaller relations to the larger ones

Algorithm SERIAL

- step 1: order relations such that

$$size(R_1) \leq size(R_2) \leq \ldots \leq size(R_n)$$

- step 2: assume R_r is the relation at the result site and R_i is stored at Site i. compare the cost

$$R_1 \rightarrow R_2 \ldots \rightarrow R_r \ldots \rightarrow R_n \rightarrow R_r$$

  i.e.
  $R_1 \rightarrow$ Site 2
  $T_2 = (R_2 \rtimes R_1) \rightarrow$ Site 3
  ..
  $T_i = (R_i \rtimes T_{i-1}) \rightarrow$ Site (i+1)
  ..
  $T_n = (R_n \rtimes T_{n-1}) \rightarrow$ Site r

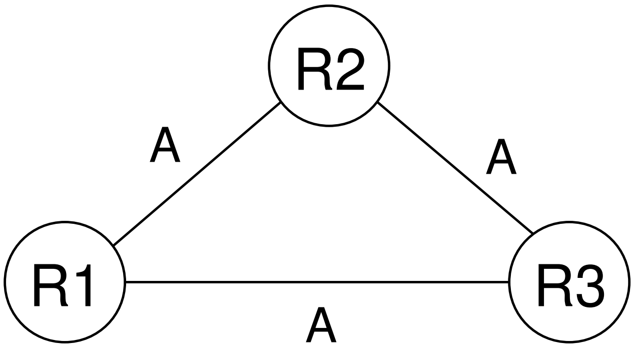
  versus that of

$$R_1 \rightarrow R_2 \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \ldots \rightarrow R_n \rightarrow R_r$$

- step 3: select the one with the smaller cost

SIMPLE QUERY OPTIMIZATION – EXAMPLE

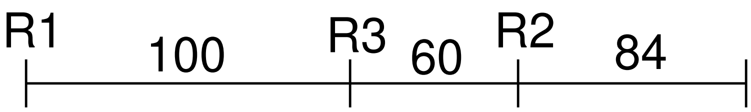
consider the same example query and assume that message initiation cost is 0, and unit message transmission cost is 1



attribute	$SF_{\ltimes}$	size($\pi_{attribute}$)
$R_1.A$	0.3	100
$R_2.A$	1.0	400
$R_3.A$	0.7	200

The optimal execution schedule is:

- $R_1 \rightarrow$ Site 3 (cost = $size(\pi_A(R_1))$)
- At site 3, compute $T_3 = R_3 \ltimes R_1$ and then $T_3 \rightarrow$ Site 2 (cost = $size(T_3)$ = 200 * 0.3 = 60
- At Site 2, compute $T_2 = R_2 \ltimes T_3$ and then
  $T_2 \rightarrow$ Site of result (not Site 1, 2, or 3)
  cost = $size(T_2)$ = 400* 0.3 * 0.7 = 84
- The toal response time of this schedule is 244.



SIMPLE QUERY OPTIMIZATION

response time minimization

- start with an initial feasible solution (IFS) and improve on it

Algorithm PARALLEL

- step 1: order relations such that

$$size(R_1) \leq size(R_2) \leq \ldots \leq size(R_n)$$

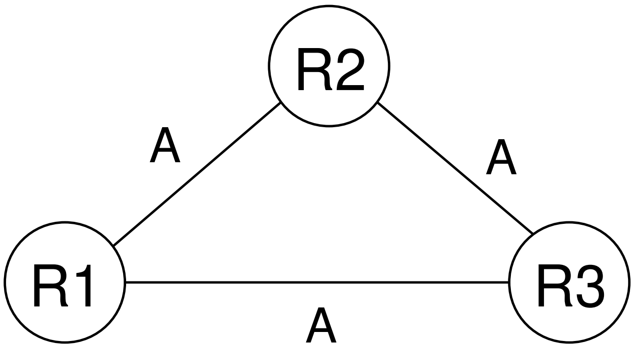
- step 2: IFS: transmit all relations *in parallel* to the result site. response time is

$$max\{transmission\ cost(R_i),\ 1 \leq i \leq n\}$$

- step 3: set $i = 1$
- step 4: pick R_i. compute the cost of transmitting $R_j, \forall j < i$, and all other $R_k(k < j)$ to R_i in parallel. Let this cost be cost(R_{ji})
- step 5: repeat step 4 for all $i \leq n$
- step 6: select the schedule with minimum cost(R_{ji}) as the new feasible solution
- step 7: eliminate redundant schedules

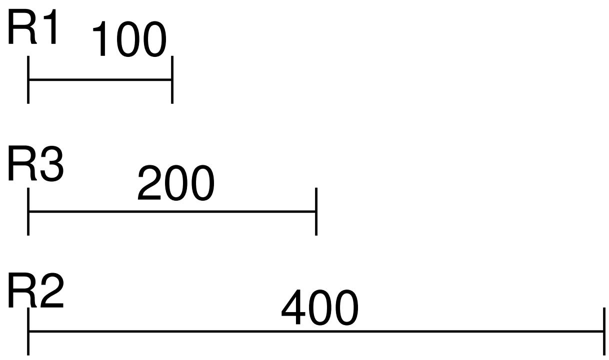
SIMPLE QUERY OPTIMIZATION – EXAMPLE (CONT)

again, consider the same example query:



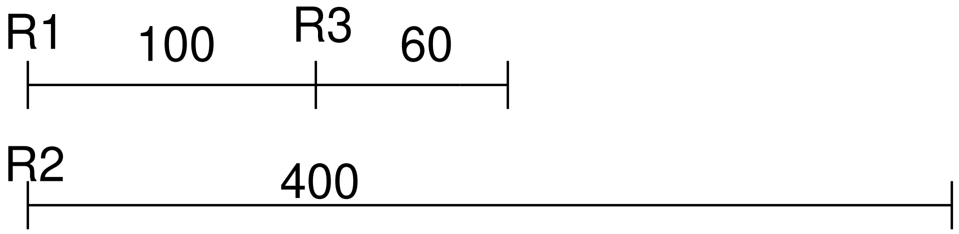
attribute	$SF_{\ltimes}$	size($\pi_{attribute}$)
$R_1.A$	0.3	100
$R_2.A$	1.0	400
$R_3.A$	0.7	200

The initial feasible schedule is:

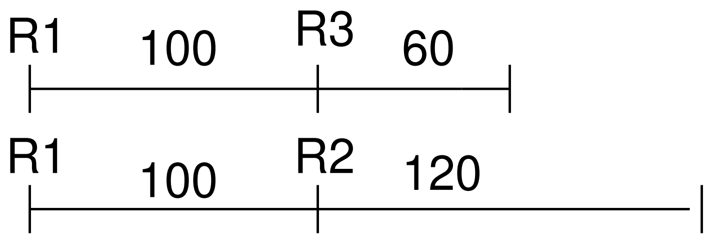


SIMPLE QUERY OPTIMIZATION – EXAMPLE (CONT)

schedule for R_1 cannot be improved (it is the smallest).
after improving the schedule of R_3:
$(size(R_3 \ltimes R_1) = 200 * 0.3 = 60)$



after improving the schedule of R_2:
$(size(R_2 \ltimes R_1) = 400 * 0.3 = 120)$



The response time of this schedule is 220, compared to 244 for the serial algorithm. However, its total cost is 380.

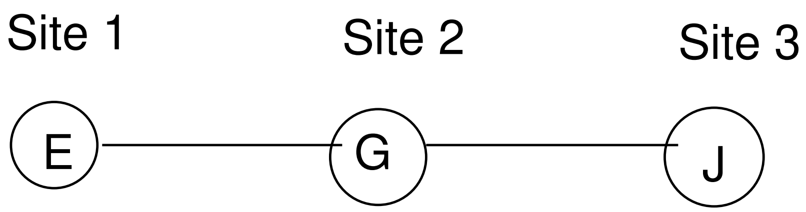
GENERAL QUERY OPTIMIZATION

- assume n join attributes are present in query
- treat each join attribute in isolation
  - decompose into n simple queries
  - apply SERIAL or PARALLEL to each simple query
- integrate results

AHY – GENERAL QUERY OPTIMIZATION

- step 1: local processing at each site
- step 2: for each join attribute, generate the set of candidate strategies
  - isolate the simple query
  - apply SERIAL or PARALLEL
- step 3: order candidate strategies for each relation on the join attribute in increasing order of cost
- step 4:
  - response time
    - for each candidate ES_{ij} (for each relation R_i)
      - perform ES_{ij}
      - move all ES_{ik} $(k < j)$ to R_i
      - reduce R_i
      - move R_i to the result site
    - pick alternative with minimum cost
  - total time – similar
- step 5: eliminate redundancies

AHY – GENERAL QUERY EXAMPLE



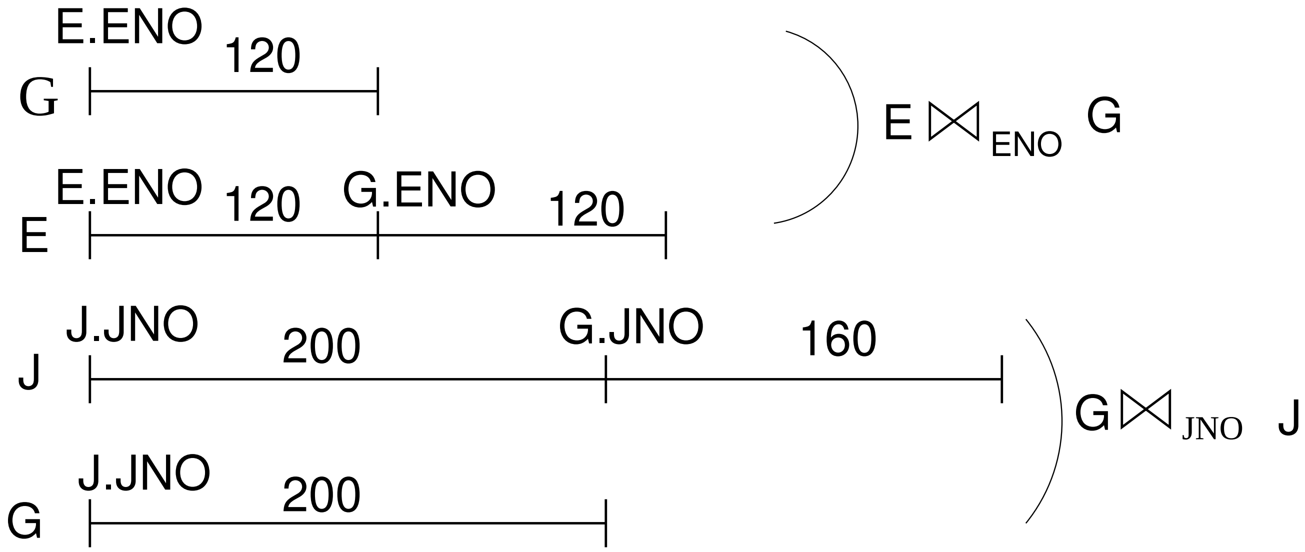
relation	card	tuple size	rel. size
E	30	50	1500
G	100	30	3000
J	50	40	2000

attribute	$SF_{\ltimes}$	size($\pi_{attribute}$)
E.ENO	0.3	120
G.ENO	0.8	400
G.JNO	0.1	400
J.JNO	0.4	200

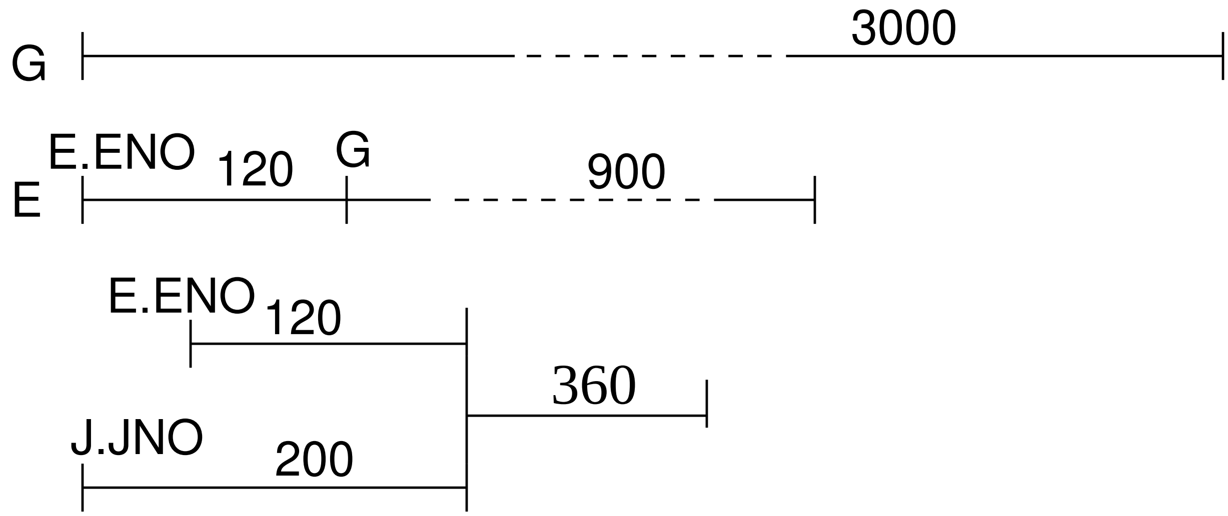
AHY – GENERAL QUERY EXAMPLE (CONT)

best strategies for response time minimization

There are only two simple queries, $E \bowtie_{ENO} G$ and $G \bowtie_{JNO} J$, whose best strtegies are given below:



common relation is G to reduce. alternatives:



The third strategy (response time 560) is selected.

STEP 4 – LOCAL OPTIMIZATION

Input: Best global execution schedule

- select the best access path
- use the centralized optimization techniques

DISTRIBUTED QUERY OPTIMIZATION PROBLEMS

- cost model
  - multiple query optimization
  - heuristics to cut down on alternatives
- larger set of queries
  - optimization only on select-project-join queries
  - also need to handle complex queries (e.g., unions, disjunctions, aggregations and sorting)
- optimization cost vs execution cost tradeoff
  - heuristics to cut down on alternatives
  - controllable search strategies
- optimization/reoptimization interval
  - extent of changes in database profile before re-optimization is necessary