Co-funded by the
Erasmus+ Programme
of the European Union
MASTIS

Distributed Database Systems and Data Warehouses

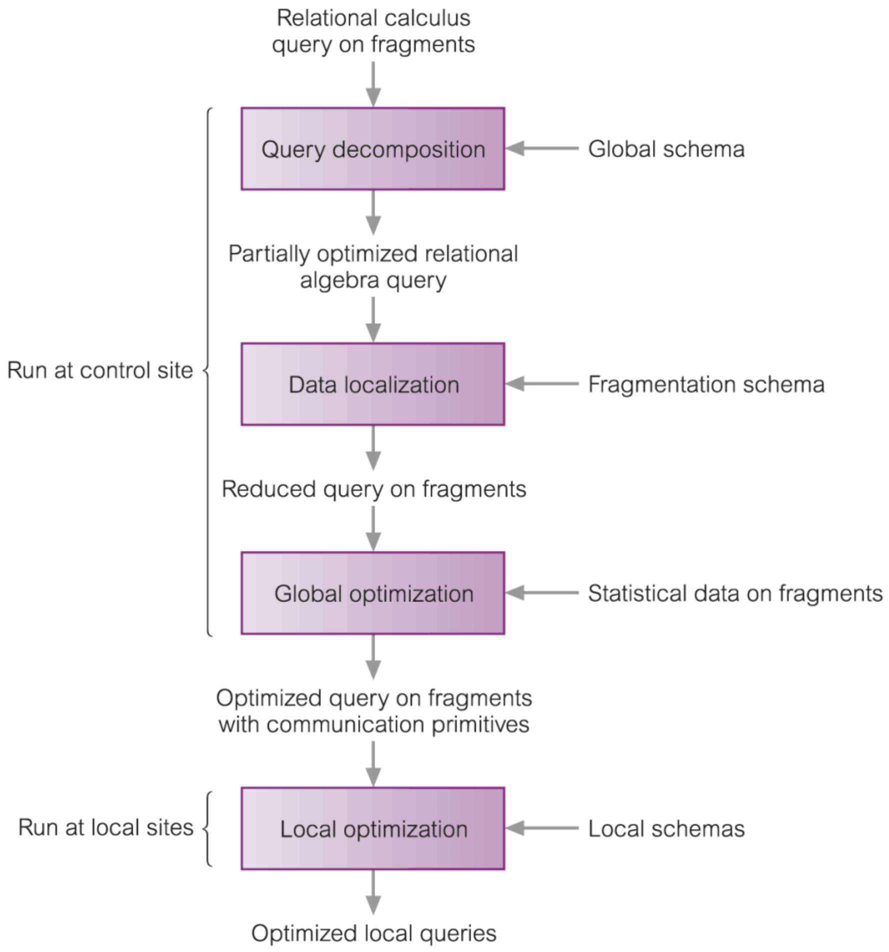
Dr. Volodymyr Sokol
(vlad.sokol@gmail.com)

Co-funded by the
Erasmus+ Programme
of the European Union

MASTIS

LECTION 7

Distributed Query Optimization


Relational calculus query on fragments
Query decomposition
Global schema
Partially optimized relational algebra query
Run at control site
Data localization
Fragmentation schema
Reduced query on fragments
Global optimization
Statistical data on fragments
Optimized query on fragments with communication primitives
Run at local sites
Local optimization
Local schemas
Optimized local queries

Distributed Query Optimization

- Query decomposition: takes query expressed on global relations and performs partial optimization using centralized QO techniques. Output is some form of RAT based on global relations.
- Data localization: takes into account how data has been distributed. Replace global relations at leaves of RAT with their *reconstruction algorithms*.

Distributed Query Optimization

- Global optimization: uses statistical information to find a near-optimal execution plan. Output is execution strategy based on fragments with communication primitives added.
- Local optimization: Each local DBMS performs its own local optimization using centralized QO techniques.

Data Localization

- In QP, represent query as R.A.T. and, using transformation rules, restructure tree into equivalent form that improves processing.
- In DQP, need to consider data distribution.
- Replace global relations at leaves of tree with their reconstruction algorithms - RA operations that reconstruct global relations from fragments:
  - For horizontal fragmentation, reconstruction algorithm is Union;
  - For vertical fragmentation, it is Join.

Data Localization

- Then use reduction techniques to generate simpler and optimized query.
- Consider reduction techniques for following types of fragmentation:
  - Primary horizontal fragmentation.
  - Vertical fragmentation.
  - Derived fragmentation.

Reduction for Primary Horizontal Fragmentation

- If selection predicate contradicts definition of fragment, this produces empty intermediate relation and operations can be eliminated.
- For join, commute join with union.
- Then examine each individual join to determine whether there are any useless joins that can be eliminated from result.
- A useless join exists if fragment predicates do not overlap.

Reduction for PHF

```
SELECT *
FROM Branch b, PropertyForRent p
WHERE b.branchNo = p.branchNo AND p.type = 'Flat';
```

P_1: $\sigma_{branchNo='B003' \wedge type='House'}$ (PropertyForRent)

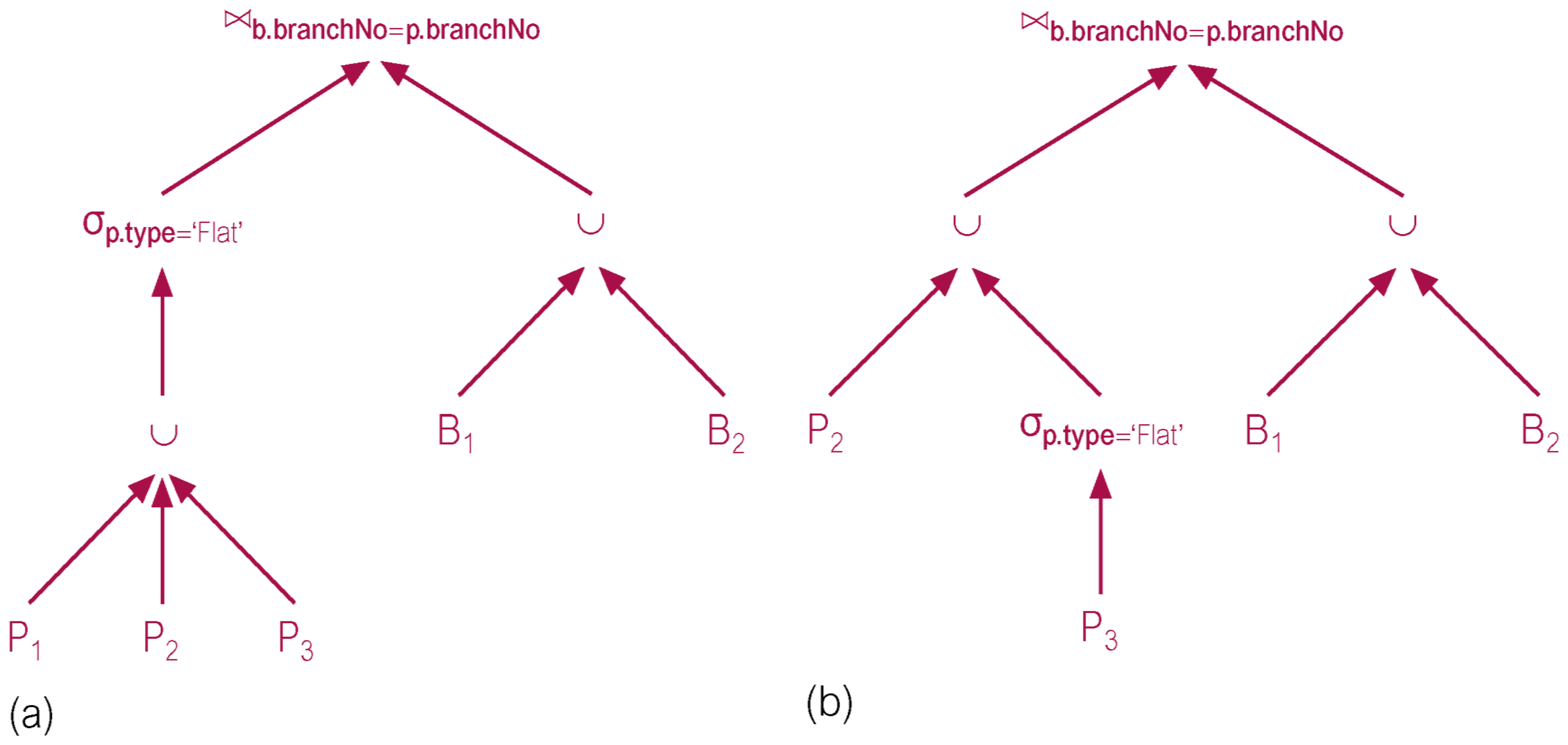
P_2: $\sigma_{branchNo='B003' \wedge type='Flat'}$ (PropertyForRent)

P_3: $\sigma_{branchNo!='B003'}$ (PropertyForRent)

B_1: $\sigma_{branchNo='B003'}$ (Branch)

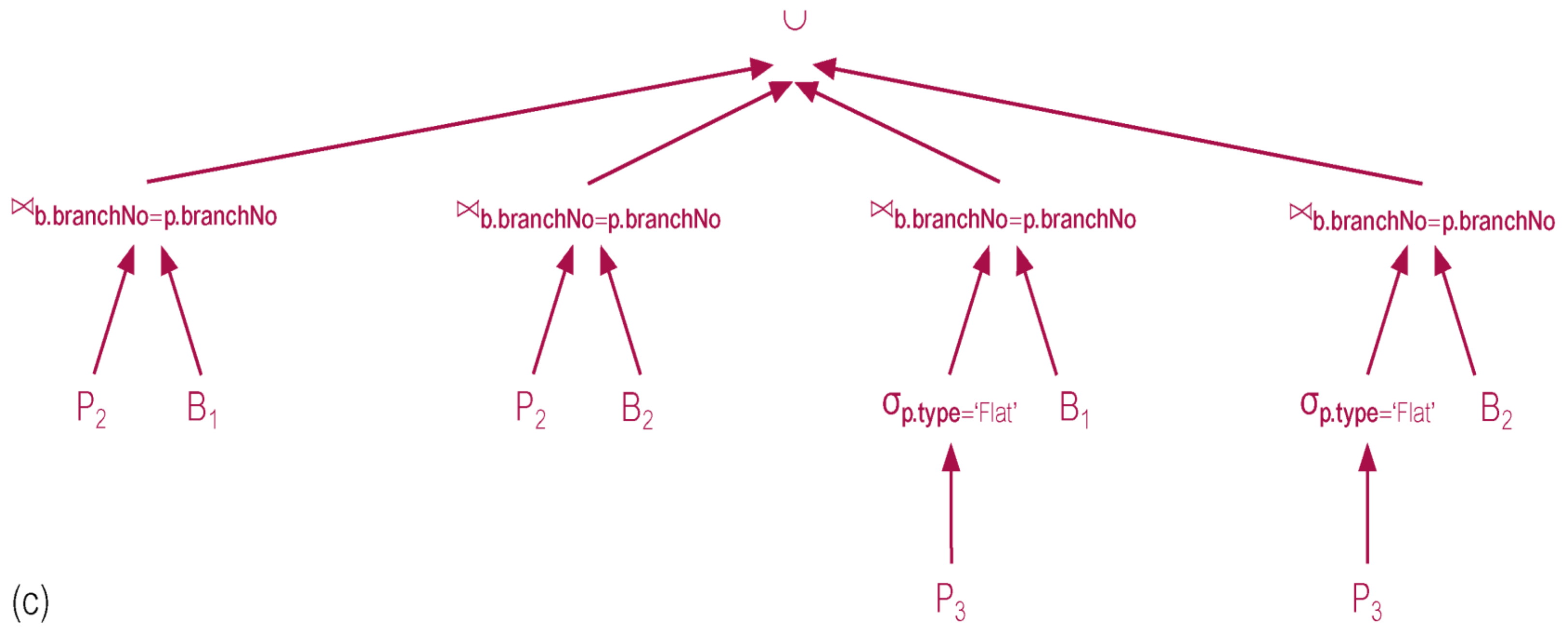
B_2: $\sigma_{branchNo!='B003'}$ (Branch)

Reduction for PHF



Co-funded by the
Erasmus+ Programme
of the European Union
MASTIS

Reduction for PHF



Reduction for PHF

$$\begin{tikzcd}
 & & \cup & & \\
\bowtie_{b.branchNo=p.branchNo} \arrow[urr] & & & \bowtie_{b.branchNo=p.branchNo} \arrow[ul] & \\
P_2 \arrow[u] & B_1 \arrow[ul] & & \sigma_{p.type='Flat'} \arrow[u] & B_2 \arrow[ul] \\
 & & & P_3 \arrow[u] &
\end{tikzcd}$$

(d)

Co-funded by the
Erasmus+ Programme
of the European Union
MASTIS

Reduction for Vertical Fragmentation

- Reduction for vertical fragmentation involves removing those vertical fragments that have no attributes in common with projection attributes, except the key of the relation.

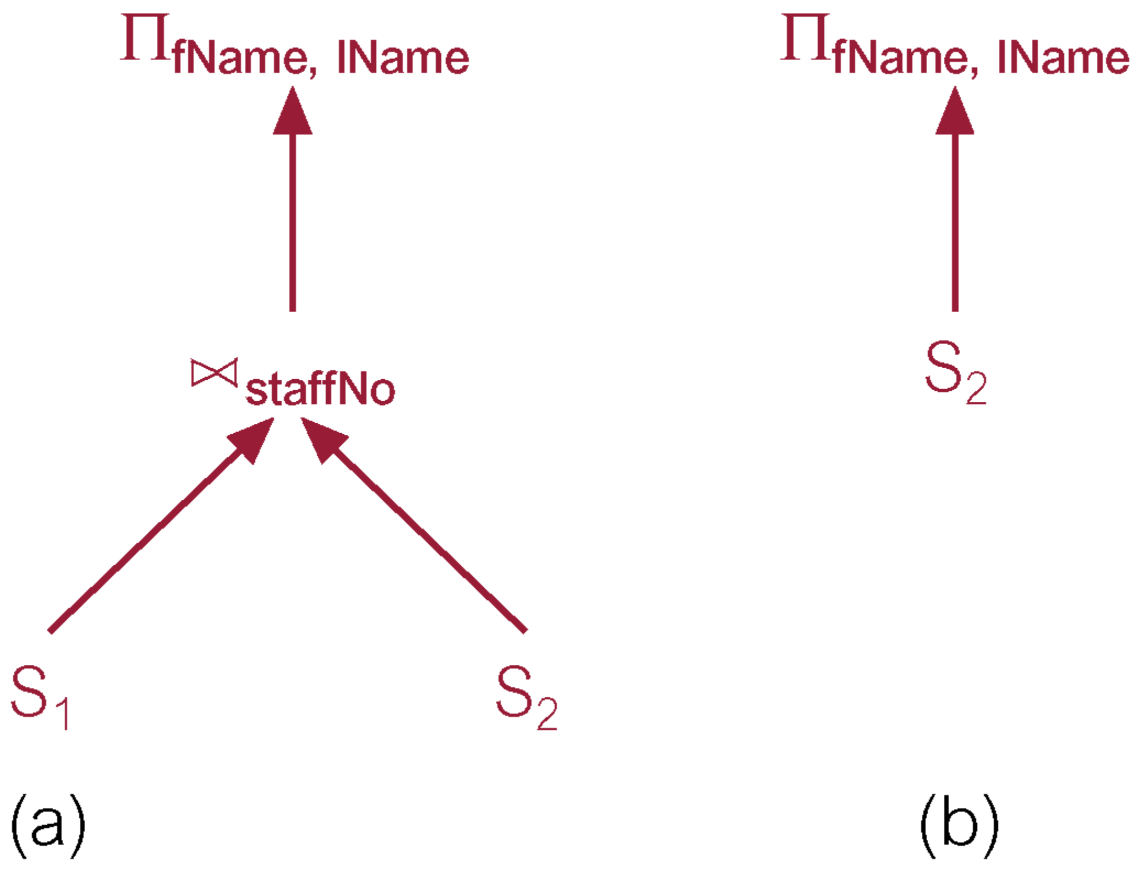
Reduction for Vertical Fragmentation

```
SELECT fName, lName
FROM Staff;
```

S$_1$: $\Pi_{\text{staffNo, position, sex, DOB, salary}}(\text{Staff})$

S$_2$: $\Pi_{\text{staffNo, fName, lName, branchNo}}(\text{Staff})$

Reduction for Vertical Fragmentation



Reduction for Derived Fragmentation

- Use transformation rule that allows join and union to be commuted.
- Using knowledge that fragmentation for one relation is based on the other and, in commuting, some of the partial joins should be redundant.

Reduction for Derived Fragmentation

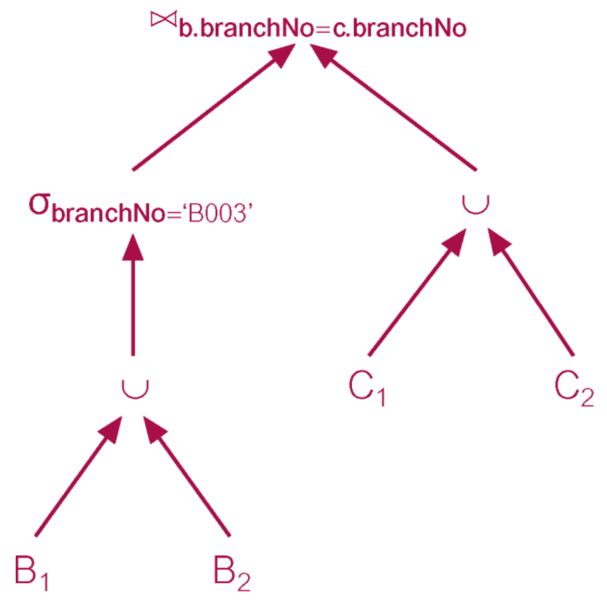
```
SELECT *
FROM Branch b, Client c
WHERE b.branchNo = c.branchNo AND
                  b.branchNo = 'B003';
```

$B_1 = \sigma_{\text{branchNo='B003'}} (\text{Branch})$

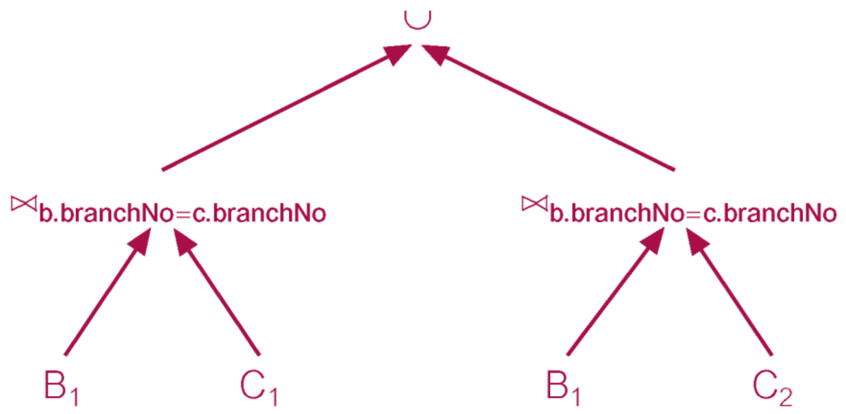
$B_2 = \sigma_{\text{branchNo!='B003'}} (\text{Branch})$

$C_i = \text{Client} \triangleright_{\text{branchNo}} B_i \qquad i = 1, 2$

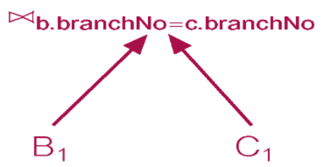
Reduction for Derived Fragmentation



(a)



(b)



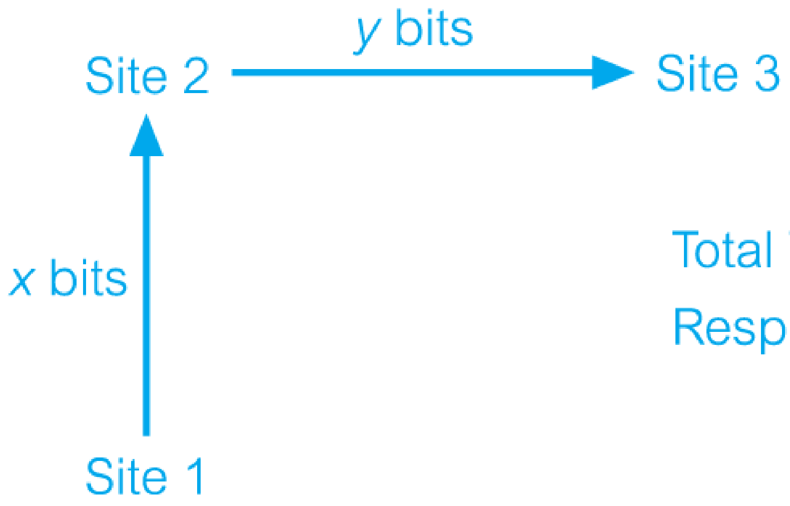
(c)

Global Optimization

- Objective of this layer is to take the reduced query plan for the data localization layer and find a near-optimal execution strategy.
- In distributed environment, speed of network has to be considered when comparing strategies.
- If know topology is that of WAN, could ignore all costs other than network costs.
- LAN typically much faster than WAN, but still slower than disk access.

Global Optimization

- Cost model could be based on total cost (time), as in centralized DBMS, or response time. Latter uses parallelism inherent in DDBMS.



$$\text{Total Time} = 2{*}C_0 + (x + y)/\text{transmission\_rate}$$

$$\text{Response Time} = \max\{C_0 + (x/\text{transmission\_rate}),\ C_0 + (y/\text{transmission\_rate})\}$$

Global Optimization – R*

- R* uses a cost model based on total cost and static query optimization.
- Like centralized System R optimizer, algorithm is based on an exhaustive search of all join orderings, join methods (nested loop or sort-merge join), and various access paths for each relation.
- When Join is required involving relations at different sites, R* selects the sites to perform Join and method of transferring data between sites.

Global Optimization – R*

- For a Join of R and S with R at site 1 and S at site 2, there are three candidate sites:
  - site 1, where R is located;
  - site 2, where S is located;
  - some other site (e.g., site of relation T, which is to be joined with join of R and S).

Global Optimization – R*

- In R*, there are 2 methods for transferring data:
  1. Ship whole relation
  2. Fetch tuples as needed.
- First method incurs a larger data transfer but fewer message then second.
- R* considers only the following methods:
  1. Nested loop, ship whole outer relation to site of inner.
  2. Sort-merge, ship whole inner relation to site of outer.
  3. Nested loop, fetch tuples of inner relation as needed for each tuple of outer relation.
  4. Sort-merge, fetch tuples of inner relation as needed for each tuple of outer relation.
  5. Ship both relations to third site.

Global Optimization – SDD-1

- Based on an earlier method known as "hill climbing", a greedy algorithm that starts with an initial feasible solution that is then iteratively improved.
- Modified to make use of Semijoin to reduce cardinality of join operands.
- Like R*, SDD-1 optimizer minimizes total cost, although unlike R* it concentrates on communication message size.
- Like R*, query processing timing used is static.

Global Optimization – SDD-1

- Based on concept of "beneficial Semijoins".
- Communication cost of Semijoin is simply cost of transferring join attribute of first operand to site of second operand.
- "Benefit" of Semijoin is taken as cost of transferring irrelevant tuples of first operand, which Semijoin avoids.

Global Optimization – SDD-1

- Phase 1 – Initialization: Perform all local reductions using Selection and Projection. Execute Semijoins within same site to reduce sizes of relations. Generate set of all beneficial Semijoins across sites (Semijoin is beneficial if its cost is less than its benefit).
- Phase 2 – Selection of beneficial Semijoins: Iteratively select most beneficial Semijoin from set generated and add it to execution strategy. After each iteration, update database statistics to reflect incorporation of the Semijoin and update the set with new beneficial Semijoins.

Global Optimization – SDD-1

- Phase 3 – Assembly site selection: Select, among all sites, site to which transmission of all relations incurs a minimum cost. Choose site containing largest amount of data after reduction phase so that sum of the amount of data transferred from other sites will be minimum.
- Phase 4 – Postoptimization: Discard useless Semijoins; e.g. if R resides in assembly site and R is due to be reduced by Semijoin, but is not used to reduce other relations after Semijoin, then since R need not be moved to another site during assembly phase, Semijoin on R is useless and can be discarded.