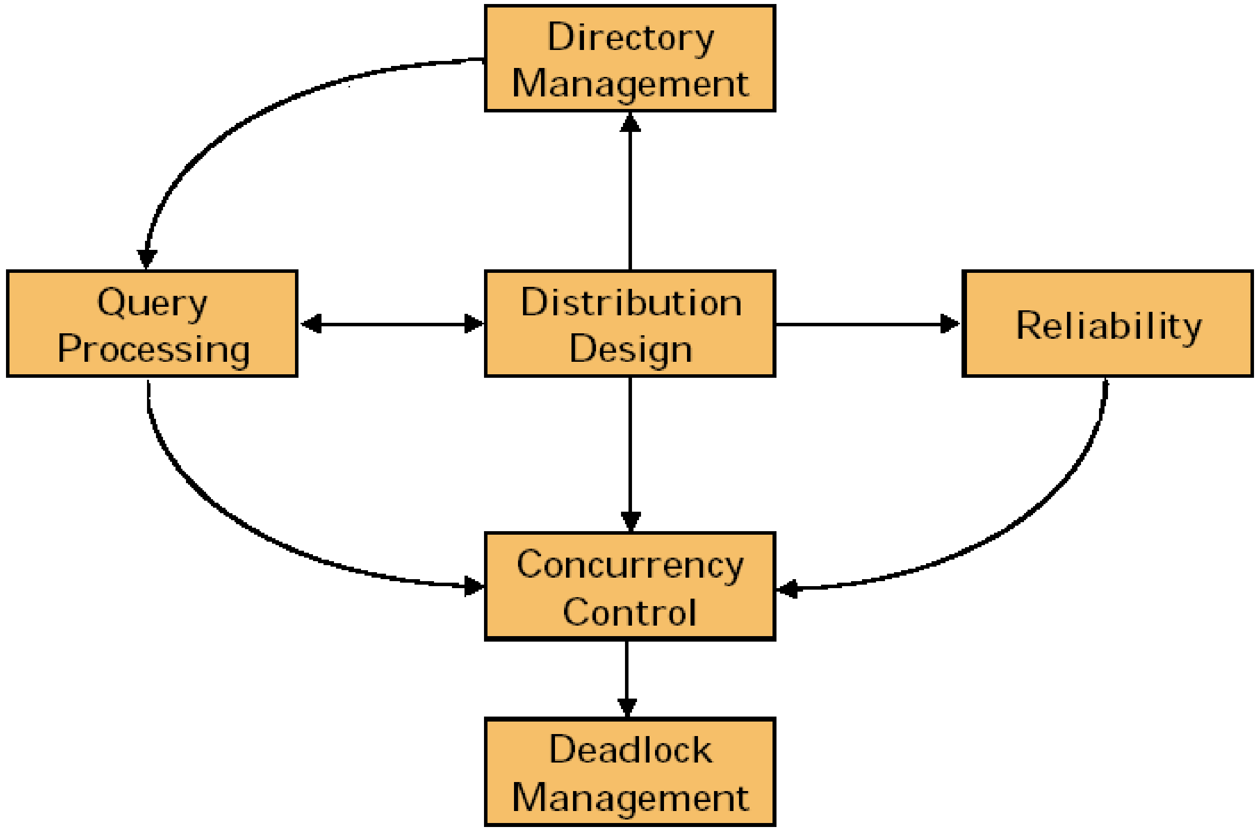
Distributed Data Processing

- Introduction
- Distributed DBMS Architecture
- **Distributed DB Design**
- Query Processing
- Transaction Management

Review:
Relationship between issues



Design Problem

- In general Distributed System

  Making decisions about the placement of *data* and *programs* across the sites of a computer network as well as possibly designing the network itself.

- In Distributed DBMS
  - Placement of programs entails
    - placement of the distributed DBMS software;
    - placement of the applications that run on the database
  - Concentrate on distribution of data

3.Distributed Database Design

- Alternative Design Strategies
- Distribution Design Issues
- Fragmentation
- Allocation
- Conclusion

3.1 Alternative Design Strategies

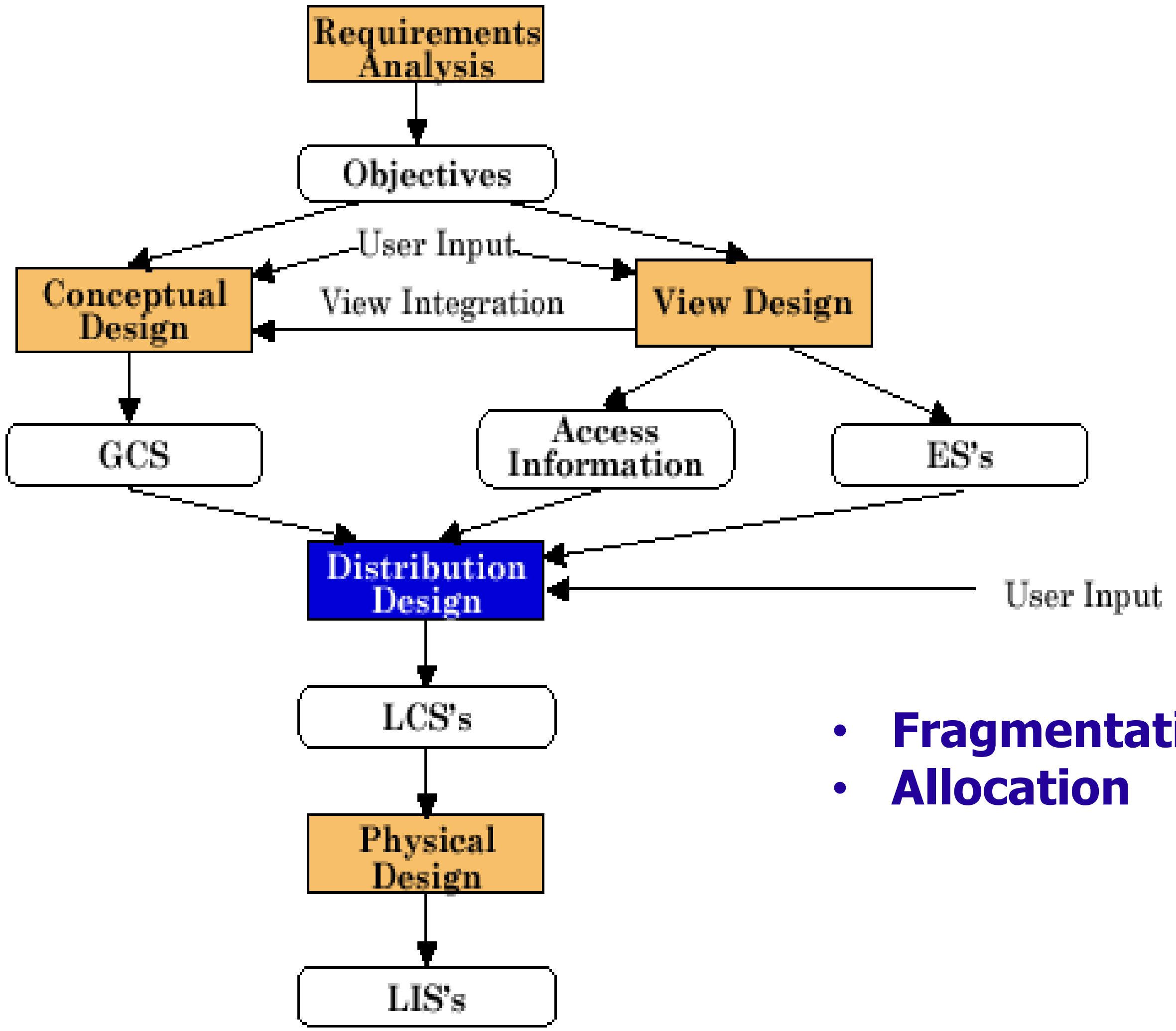
Alternative Design Strategies

- Top-down
  - Mostly in designing systems from scratch
  - Mostly in homogeneous systems
- Bottom-up
  - When the databases already exist at a number of sites
  - In general, integrating local schemas into the global conceptual schema

Alternative Design Strategies

- Top-down
  - mostly in designing systems from scratch
  - mostly in homogeneous systems
- Bottom-up
  - when the databases already exist at a number of sites
  - In general, integrating local schemas into the global conceptual schema

Top-Down Design



- **Fragmentation**
- **Allocation**

3.2 Distribution Design Issues

Distribution Design Issues

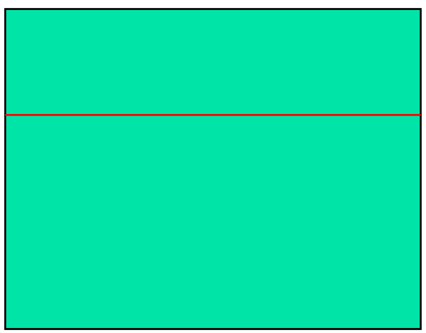
- Why fragment at all?
- How to fragment?
- How much to fragment?
- How to test correctness?
- How to allocate?
- Information requirements?

Reasons for Fragmentation

- Can't we just distribute relations?
- What is a reasonable unit of distribution?
  - relation
    - views are subsets of relations
    - extra communication/storage
  - fragments of relations (sub-relations)
    - concurrent execution
      - Not only for a number of transactions
      - But also for a single query
    - **Disadvantage**
      - views that cannot be defined on a single fragment will require extra processing
      - semantic data control (especially integrity enforcement) more difficult

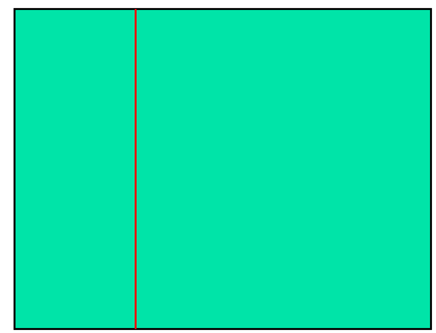
Fragmentation Alternatives

1.Horizontal



tuples

2.Vertical



attributes

3.Hybrid

Example of Horizontal Fragmentation

$PROJ_1$: projects with budgets less than \$200,000

$PROJ_2$: projects with budgets greater than or equal to \$200,000

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York

$PROJ_2$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

Example of Vertical Fragmentation

$PROJ_1$: information about project budgets

$PROJ_2$: information about project names and locations

PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris
P5	CAD/CAM	500000	Boston

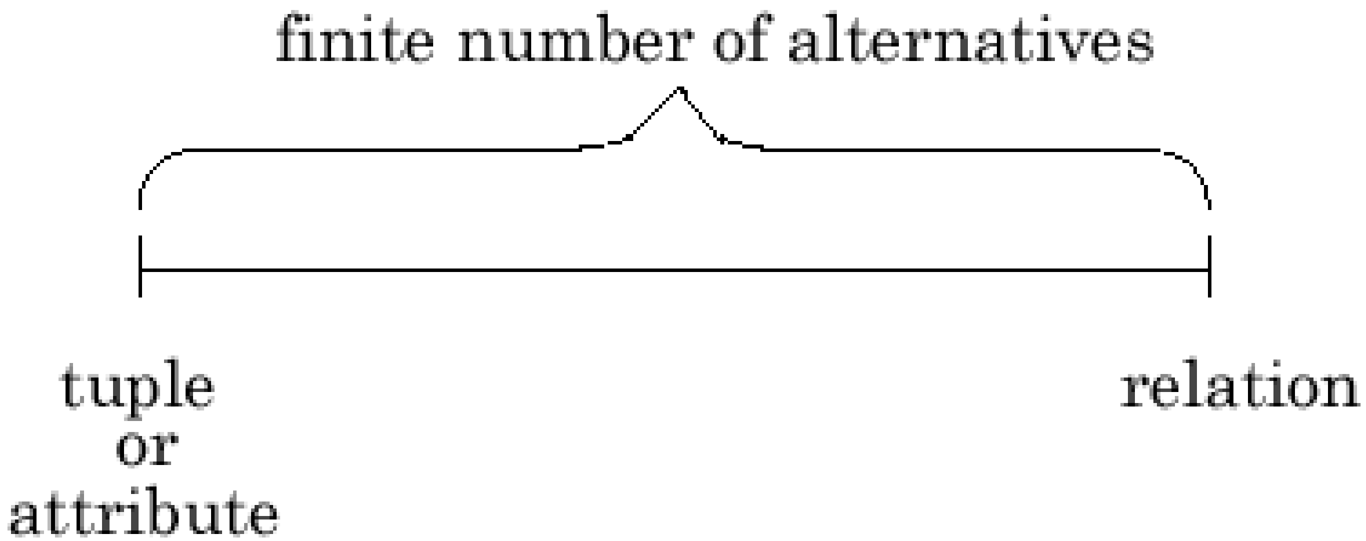
$PROJ_1$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

$PROJ_2$

PNO	PNAME	LOC
P1	Instrumentation	Montreal
P2	Database Develop.	New York
P3	CAD/CAM	New York
P4	Maintenance	Paris
P5	CAD/CAM	Boston

Degree of Fragmentation



Finding the suitable level of partitioning within this range

-- That can only be defined with respect to applications

Correctness Rules of Fragmentation

- Completeness
  - Decomposition of relation R into fragments R_1, R_2, ..., R_n is complete if and only if each data item in R can also be found in some R_i
- Reconstruction
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n , then there should exist some relational operator $\triangledown$ such that $R = \triangledown_{1 \le i \le n} R_i$
- Disjointness
  - If relation R is decomposed into fragments R_1, R_2, ..., R_n , and data item d_i is in R_j , then d_i should not be in any other fragment R_k $(k \neq j)$.

Allocation Alternatives

- Non-replicated
  - partitioned: each fragment resides at only one site
- Replicated
  - fully replicated: each fragment at each site
  - partially replicated: each fragment at some of the sites
- Rule of thumb:

If $\underline{\text{(read-only queries)} / \text{(update queries)} \geqslant 1}$,

replication is advantageous,

otherwise replication may cause problems

Comparison of Replication Alternatives

	Full-replication	Partial-replication	Partitioning
QUERY PROCESSING	Easy	Same Difficulty	
DIRECTORY MANAGEMENT	Easy or Non-existant	Same Difficulty	
CONCURRENCY CONTROL	Moderate	Difficult	Easy
RELIABILITY	Very high	High	Low
REALITY	Possible application	Realistic	Possible application

Information Requirements

Four categories:

- Database information
- Application information
- Communication network information
- Computer system information

3.3 Fragmentation

Fragmentation

- Horizontal Fragmentation
  - Primary Horizontal Fragmentation (PHF)
  - Derived Horizontal Fragmentation (DHF)
- Vertical Fragmentation (VF)
- Hybrid Fragmentation (HF)

3.3.1 Horizontal Fragmentation

Horizontal Fragmentation

Partition a relation along its tuples

SELECT

- Primary Horizontal Fragmentation (PHF)
  - Using predicates that are defined on that relation
- Derived Horizontal Fragmentation (DHF)
  - Using predicates that are defined on another relation

Primary Horizontal Fragmentation

Primary Horizontal Fragmentation

- Definition :
  - $R_j = \delta_{Fj}(R), 1 \leq j \leq w$
    where F_j is a **selection** formula, which is (preferably) a minterm predicate.
- Therefore

  A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i .

- *Primary horizontal fragmentation is defined by a selection operation on the owner/member relations of a database schema.*

Simple predicates

- Given $R[A_1, A_2, \ldots, A_n]$, a <u>simple predicate p_j</u> is

  $$\boldsymbol{p_j : A_i \ \theta \ Value}$$

  where $\theta \in \{=, <, >, \leq, \geq, \neq\}$, $Value \in D_i$ and D_i is the domain of A_i.
- For relation R we define $\boldsymbol{P_r = \{p_1, p_2, \ldots, p_m\}}$ - the set of all simple predicates defined on R

- **Example**
  - PNAME = "Maintenance"
  - BUDGET ≤200000

Minterm predicates

- Given R and $P_r = \{p_1, p_2, \ldots, p_m\}$, define
  M = {$\boldsymbol{m_1, m_2, \ldots, m_r}$} as

  $$\mathbf{M=\{ \mathit{m}_i \,/\, \mathit{m}_i = \bigwedge_{\mathit{p_j} \in \mathit{P_r}} \mathit{p}_j^{\,*} \}, 1 \leq \mathit{j} \leq m, 1 \leq \mathit{i} \leq \mathit{z}}$$

  where $p_j^{\,*} = p_j$ or $p_j^{\,*} = \neg(p_j)$.

- **Example**
  - m_1 : PNAME="Maintenance" $\wedge$ BUDGET ≤200000
  - m_2 : NOT(PNAME="Maintenance") $\wedge$ BUDGET ≤200000
  - m_3 : PNAME= "Maintenance" $\wedge$ NOT(BUDGET ≤200000)
  - m_4 : NOT(PNAME="Maintenance") $\wedge$ NOT(BUDGET ≤200000)

Primary Horizontal Fragmentation

- Given a set of minterm predicates *M*, there are as many horizontal fragments of relation *R* as there are minterm predicates.
  - Set of horizontal fragments also referred to as ***minterm fragments***.

- ***Miniterm Fragments -> Miniterm Predicates -> Simple Predicates***

*Therefore, the first step of any fragmentation algorithm is to determine **a set of simple predicate** that will form the minterm predicates*

Application information

- Predicate – used in user queries
  - 80/20 rule
- Simple predicates
- Minterm predicates

PHORIZONTAL Algorithm

1. $P_{r'} \leftarrow$ COM_MIN (R, P_r)
2. determine the set M of minterm predicates
3. determine the set I of implications among $p_i \in P_r$
4. eliminate the contradictory minterms from M

Completeness of Simple Predicates

- A set of simple predicates *Pr* is said to be *complete* if and only if the accesses to the tuples of the minterm fragments defined on *Pr* requires that *two **tuples** of the same minterm fragment have the same probability of being accessed by any application.*

Example of Completeness

- Assume

  PROJ[PNO,PNAME,BUDGET,LOC] has two applications defined on it.

  - Find the budgets of projects at each location. (1)
  - Find projects with budgets less than $200000. (2)

Example of Completeness (Cont.)

According to (1),

Pr={LOC="Montreal",LOC="New York",LOC="Paris"}

which is not complete with respect to (2).

Modify

Pr ={LOC="Montreal",LOC="New York",LOC="Paris",

BUDGET ≤200000,BUDGET>200000}

which is complete.

Minimality of Simple Predicates

- If a predicate influences how fragmentation is performed, (i.e., causes a fragment f to be further fragmented into, say, f_i and f_j) then *there should be at least one application that accesses* ***fragmentation*** *f_i and f_j differently.*
- In other words, the simple predicate should be ***relevant*** in determining a fragmentation.

$$acc(m_i) / card(f_i) \neq acc(m_j) / card(f_j)$$

- If all the predicates of a set P_r are relevant, then P_r is ***minima*****l**.

Example of Minimality

Pr ={LOC="Montreal",LOC="New York", LOC="Paris",
BUDGET ≤200000,BUDGET>200000}

is minimal (in addition to being complete).

However, if we add

PNAME = "Instrumentation"

then *Pr* is not minimal.

Example of PHF

- **Fragmentation of relation PROJ**
  - Application1: check the project info given their location
  - Application2: project management based on budget

Pr ={LOC="Montreal",LOC="New York", LOC="Paris", BUDGET ≤200000,BUDGET>200000}

is complete and minimal

Example of PHF

$i_1 : p_1 \Rightarrow \neg p_2 \wedge \neg p_3$

$i_2 : p_2 \Rightarrow \neg p_1 \wedge \neg p_3$

$i_3 : p_3 \Rightarrow \neg p_1 \wedge \neg p_2$

$i_4 : p_4 \Rightarrow \neg p_5$

$i_5 : p_5 \Rightarrow \neg p_4$

$i_6 : \neg p_4 \Rightarrow p_5$

$i_7 : \neg p_5 \Rightarrow p_4$

- **Minterm fragments** left after elimination
  - m_1 : **(LOC = "Montreal") $\wedge$ (BUDGET ≤200000)**
  - m_2 : **(LOC = "Montreal") $\wedge$ (BUDGET > 200000)**
  - m_3 : **(LOC = "New York") $\wedge$ (BUDGET ≤200000)**
  - m_4 : **(LOC = "New York") $\wedge$ (BUDGET > 200000)**
  - m_5 : **(LOC = "Paris") $\wedge$ (BUDGET ≤200000)**
  - m_6 : **(LOC = "Paris") $\wedge$ (BUDGET > 200000)**

Example of PHF

$PROJ_1$

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal

$PROJ_3$

PNO	PNAME	BUDGET	LOC
P2	Database Develop.	135000	New York

$PROJ_4$

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

$PROJ_6$

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

Correctness for PHF

- Completeness
  - Guaranteed by the PHORIZONTAL algorithm - Minterm predicate is defined by $\bigwedge \boldsymbol{p}_j^*$ (p_j or $\neg p_j$).
- Reconstruction
  - If relation R is fragmentead into $F_R = \{R_1, R_2, \ldots, R_r\}$, $R = \bigcup_{\text{all } R_i \in F_R} R_i$
- Disjointness
  - Minterm predicates that form the basis of fragmentation should be mutually exclusive.

Derived Horizontal Fragmentation

Derived Horizontal Fragmentation

- Defined on a **member relation** of a link according to a selection operation specified on its owner.
  - Each link is an equijoin.
  - Equijoin can be implemented by means of semijoins.

Database information

- **Global Conceptual Schema**
  - How are the database relations connected to one another, esp. with joins
  - Link: between relations that are related to each other by an equijoin operation
    - owner -> member (one-to-many relationship)
  - example

Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

ASG

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

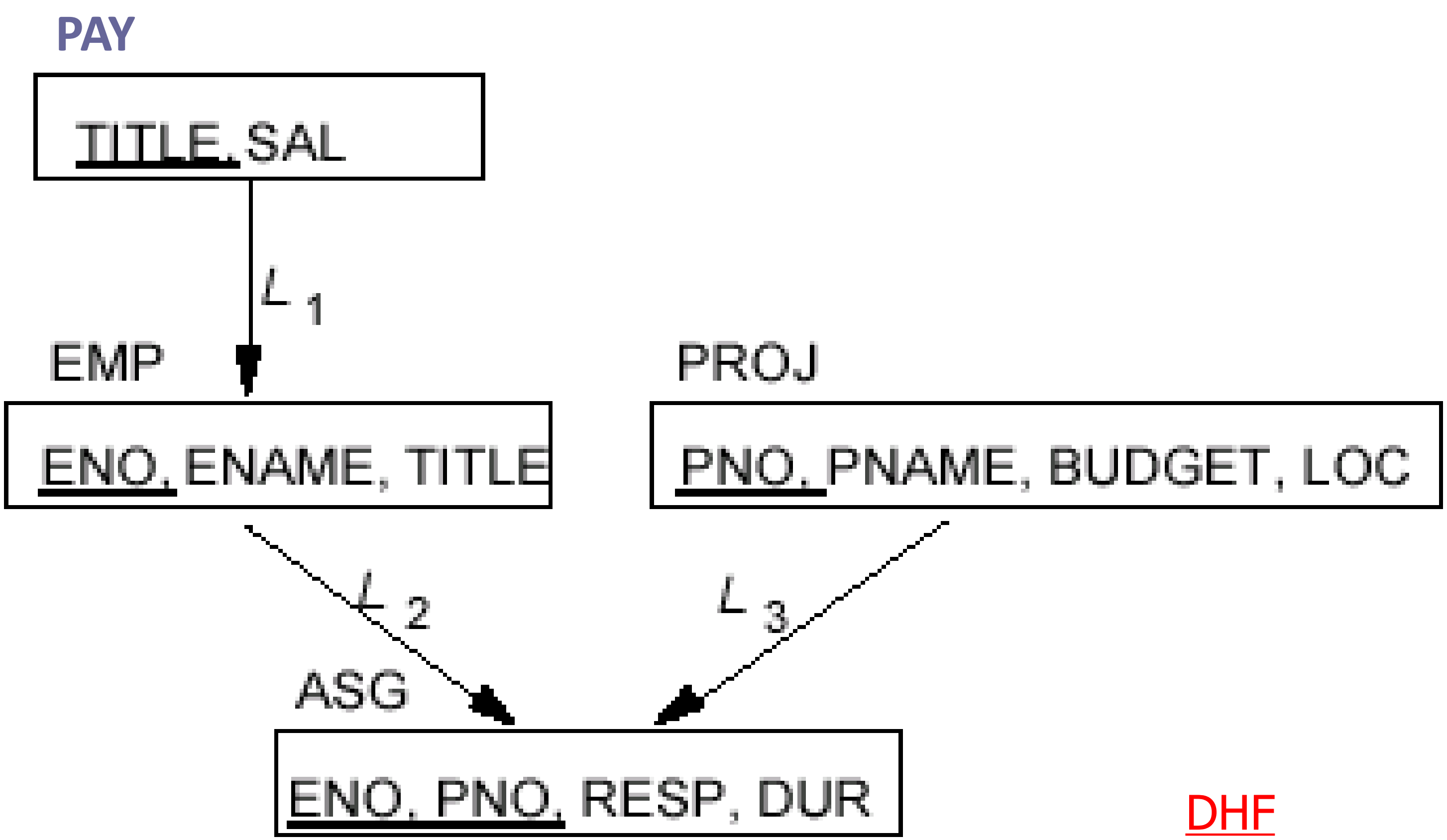
PROJ

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Montreal
P2	Database Develop.	135000	New York
P3	CAD/CAM	250000	New York
P4	Maintenance	310000	Paris

PAY

TITLE	SAL
Elect. Eng.	40000
Syst. Anal.	34000
Mech. Eng.	27000
Programmer	24000

Example of Links



DHF

Example: ASG

Definition of DHF

Given a link *L* where *owner*(L)=*S* and *member*(L)=R, the derived horizontal fragments of *R* are defined as

$$\boldsymbol{R_i = R \propto_F S_i}, \ 1 \leq i \leq w$$

where *w* is the maximum number of fragments that will be defined on *R* and

$$S_i = \delta_{F\,i}\,(S)$$

where F_i is the formula according to which the primary horizontal fragment S_i is defined.

Inputs of DHF

- The set of partitions of the owner relation
- The member relation
- The set of semijoin predicates between the owner and the member

Example of DHF – simple join graph

Given link L_1

where owner(L_1)=PAY and member(L_1)=EMP

$$EMP_1 = EMP \propto PAY_1$$

$$EMP_2 = EMP \propto PAY_2$$

where

$$PAY_1 = \delta_{SAL \leq 30000}(PAY)$$

$$PAY_2 = \delta_{SAL > 30000}(PAY)$$

Example: DHF for EMP

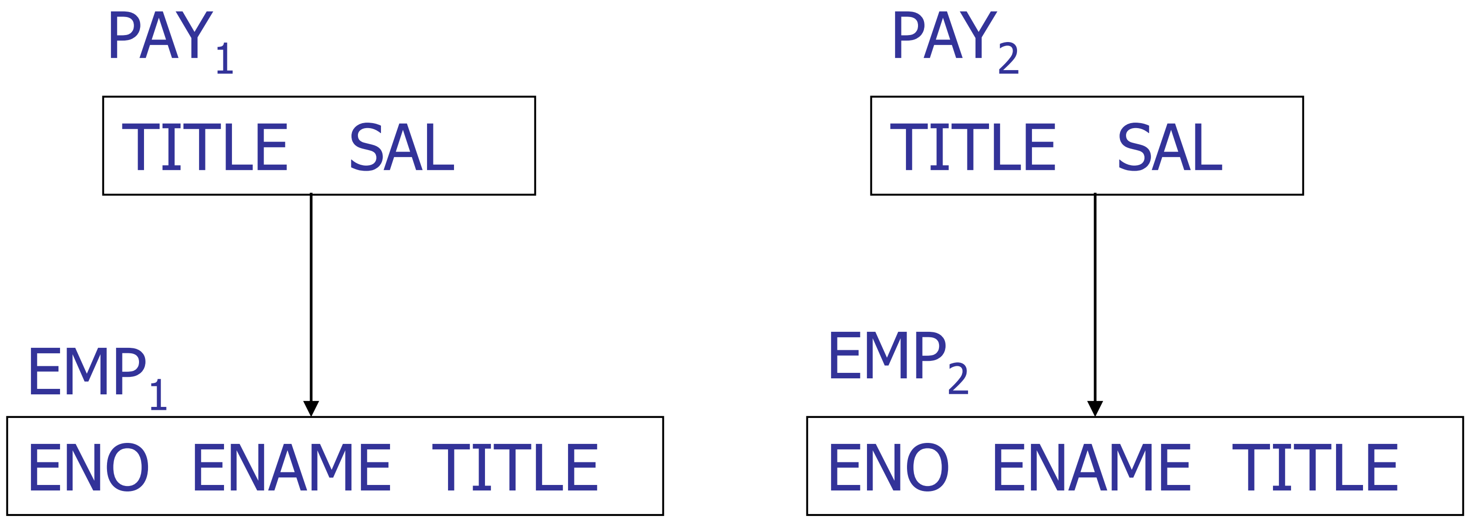
EMP_1

ENO	ENAME	TITLE
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E7	R. Davis	Mech. Eng.

EMP_2

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E8	J. Jones	Syst. Anal.

Simple join graph



Example: DHF for ASG

Given link L_2 ,L_3

where

owner(L_2)=EMP and member(L_2)=ASG;

owner(L_3)=PROJ and member(L_3)=ASG

Example: DHF for ASG

- Applications
  - Finding the information of engineers who work at certain places
    - $ASG_i = ASG \propto PROJ_i$
  - At each administrative site where employee records are managed, users would like to access the responsibilities on the projects
    - $ASG_i = ASG \propto EMP_i$

DHF

- DHF may follow a chain (PAY-EMP-ASG)
- Typically, there will be more than one candidate fragmentation for a relation (ASG). The final choice may be a decision problem addressed during allocation

Correctness of DHF

- Completeness
  - Referential integrity
    - Let *R* be the member relation of a link whose owner is relation *S* which is fragmented as $F_S = \{S_1, S_2, ..., S_n\}$. Furthermore, let *A* be the join attribute between *R* and S. Then, for each tuple *t* of *R*, there should be a tuple *t'* of *S* such that t[A]=*t*'[A]
- Reconstruction
  - Same as primary horizontal fragmentation.
- Disjointness
  - Semijoin involved – complexity; maybe necessary to investigate actual tuple values
  - Simple join graphs

3.3.2 Vertical Fragmentation

Vertical Fragmentation

- Contains a subset of R's **attributes** as well as the **primary key** of R.
  - **Objective**: to partition a relation into a set of smaller relations so that many of the user applications will run on only one fragment.
  - An "optimal" fragmentation will **minimizes the execution time** of user applications that run on these fragments.

Replication of the key attribute

- Advantage:
  - Allows the reconstruction
  - Easier to enforce semantic integrity checking
    - Reduces the chances of communication among sites, although it doesn’t eliminate it (for those integrity constraints without primary key, and concurrency control)
- Another alternative:
  - tuple identifier (TID)

Alternatives

- More difficult than horizontal, because more alternatives exist.

  Two heuristic approaches:
  - **Grouping** attributes to fragments
  - **Splitting** relation to fragments
    - Fits more naturally within the top-down design methodology
    - The “optimal” solution is probably closer to the full relation
    - Non-overlap (except the key attribute)

Information Requirements

- Application Information
  - **Attribute affinities**
    - A measure that indicates how closely related the attributes are
    - This is obtained from more primitive usage data

Information Requirements

- Attribute usage values
  - Given a set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ that will run on the relation $R[A_1, A_2, \ldots, A_n]$,

    $use(q_i, A_j)$ =

    1 if attribute A_j is referenced by query q_i

    0 otherwise

    $use(q_i, \bullet)$ vectors for each application can be defined accordingly

Example of use(q_i,A_j)

Consider the following 4 queries for relation PROJ

q_1: **SELECT** BUDGET
FROM PROJ
WHERE PNO=Value

q_2: **SELECT** PNAME,BUDGET
FROM PROJ

q_3: **SELECT** PNAME
FROM PROJ
WHERE LOC=Value

q_4: **SELECT** **SUM**(BUDGET)
FROM PROJ
WHERE LOC=Value

Let A_1= PNO, A_2= PNAME, A_3= BUDGET, A_4= LOC

$$\begin{array}{cc} & \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} & \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

Affinity Measure *aff*(A_i, A_j)

- The **attribute affinity measure** between two attributes A_i and A_j of a relation R[A_1 , A_2 , …, A_n] with respect to the set of applications $Q = (q_1 , q_2 , …, q_q)$ is defined as follows :

  aff $(A_i , A_j) =_{\text{all queries that access } Ai \text{ and } Aj}$ **(query access)**

  query access =

  $_{\text{all sites}}$ **access frequency of a query *(access/execution)**

aff(A_i,A_j) = $\sum_{k|\text{use}(qk, Ai)=1 \wedge \text{use}(qk, Aj)=1} \sum_{s1} \text{ref}_1(q_k)\,\text{acc}_1(q_k)$

Example of *aff*(*Ai* ,*Aj*)

Assume each query in the previous example accesses the attributes once during each execution.

Also assume the access frequencies

$$\begin{array}{c|ccc} & S_1 & S_2 & S_3 \\ \hline q_1 & 15 & 20 & 10 \\ q_2 & 5 & 0 & 0 \\ q_3 & 25 & 25 & 25 \\ q_4 & 3 & 0 & 0 \end{array}$$

Then

$$\begin{array}{c|cccc} & A_1 & A_2 & A_3 & A_4 \\ \hline q_1 & 1 & 0 & 1 & 0 \\ q_2 & 0 & 1 & 1 & 0 \\ q_3 & 0 & 1 & 0 & 1 \\ q_4 & 0 & 0 & 1 & 1 \end{array}$$

$$\begin{aligned} aff(A_1, A_3) &= 15*1 + 20*1+10*1 \\ &= 45 \end{aligned}$$

and the attribute affinity matrix AA is

$$\begin{array}{c|cccc} & A_1 & A_2 & A_3 & A_4 \\ \hline A_1 & 45 & 0 & 45 & 0 \\ A_2 & 0 & 80 & 5 & 75 \\ A_3 & 45 & 5 & 53 & 3 \\ A_4 & 0 & 75 & 3 & 78 \end{array}$$

Clustering Algorithm

- Take the **attribute affinity matrix *AA*** and **reorganize** the attribute orders to form clusters where the attributes in each cluster demonstrate high affinity to one another.
- **Bond Energy Algorithm (BEA)** has been used for clustering of entities. BEA finds an ordering of entities (in our case attributes) such that the global affinity measure

  $AM = {}_{j}\,{}_{i}$**(affinity of A_i and A_j with their neighbors)**

  is maximized.

Bond Energy Algorithm

- **Input:** The *AA* matrix
- **Output:** The clustered affinity matrix *CA* which is a perturbation of *AA*

1. ***Initialization*:** Place and fix one of the columns of *AA* in *C*A.
2. ***Iteration*:** Place the remaining *n-i* columns in the remaining i+1 positions in the *CA* matrix. For each column, *choose the best placement that makes the most contribution to the global affinity measure*.
3. ***Row order*:**Order the rows according to the column ordering.

“Best” placement?

- The global affinity measure AM is maximized

  $AM = {}_{j}{}_{i}$(affinity of A_i and A_j with their neighbors)

  AM $= \Sigma_{i=1..n}\Sigma_{j=1..n}\ aff(A_i, A_j)[aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]$

  $= \Sigma_{j=1..n}\ [\Sigma_{i=1..n}\ aff(A_i, A_j)aff(A_i, A_{j-1}) +$

  $\Sigma_{i=1..n}\ aff(A_i, A_j)aff(A_i, A_{j+1})]$

  $= \Sigma_{j=1..n}[bond(A_j, A_{j-1}) + bond(A_j, A_{j+l})]$

$bond(A_x, A_y) = \sum_{z=1..n} aff(A_z, A_x)aff(A_z, A_y)$

Where $aff(A_0, A_i) = aff(A_i, A_{n+1}) = 0$

"Best" placement?

$$\underbrace{A_1\ A_2\ \ldots\ldots\ A_{i-1}\ A_i}_{AM'}\quad \mathbf{A_k}\quad \underbrace{A_j\ A_{j+1}\ \ldots\ldots\ A_n}_{AM''}$$

$$AM_{old} = AM' + AM'' + bond(A_i, A_j) + bond(A_j, A_i)$$

$$AM_{new} = AM' + AM'' +$$

$$bond(A_i, A_k) + bond(A_k, A_i) + bond(A_k, A_j) + bond(A_j, A_k)$$

cont = $AM_{new} - AM_{old}$

$$cont(A_i, A_k, A_j) = 2bond(A_i, A_k) + 2bond(A_k, A_j) - 2bond(A_i, A_j)$$

“Best” placement?

- Define contribution of a placement:
  - $cont(A_i, A_k, A_j) =$

    $2bond(A_i, A_k) + 2bond(A_k, A_l) - 2bond(A_i, A_j)$

    where $bond(A_x, A_y) = \sum_{z=1..n} aff(A_z, A_x) aff(A_z, A_y)$

Example of BEA

Consider the following AA matrix and the corresponding CA matrix where A_1 and A_2 have been placed. Place A_3:

$$AA = \begin{array}{cc} & \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{array}{c} A_1 \\ A_2 \\ A_3 \\ A_4 \end{array} & \begin{bmatrix} 45 & 0 & 45 & 0 \\ 0 & 80 & 5 & 75 \\ 45 & 5 & 53 & 3 \\ 0 & 75 & 3 & 78 \end{bmatrix} \end{array} \qquad CA = \begin{array}{c} \begin{array}{cccc} A_1 & A_2 & & \end{array} \\ \begin{bmatrix} 45 & 0 & & \\ 0 & 80 & & \\ 45 & 5 & & \\ 0 & 75 & & \end{bmatrix} \end{array}$$

Ordering (0-3-1) :

$$\begin{aligned} cont(A_0,A_3,A_1) &= 2bond(A_0 , A_3)+2bond(A_3 , A_1)–2bond(A_0 , A_1) \\ &= 2* 0 + 2* 4410 – 2*0 = 8820 \end{aligned}$$

Ordering (1-3-2) :

$$\begin{aligned} cont(A_1,A_3,A_2) &= 2bond(A_1 , A_3)+2bond(A_3 , A_2)–2bond(A_1,A_2) \\ &= 2* 4410 + 2* 890 – 2*225 = 10150 \end{aligned}$$

Ordering (2-3-4) :

$$cont\ (A_2,A_3,A_4) = 1780$$

Example of BEA

Therefore, the *CA* matrix has to form

$$\begin{array}{c} \begin{array}{ccc} A_1 & A_3 & A_2 \end{array} \\ \begin{bmatrix} 45 & 45 & 0 \\ 0 & 5 & 80 \\ 45 & 53 & 5 \\ 0 & 3 & 75 \end{bmatrix} \end{array}$$

Example of BEA

When A4 is placed, the final form of the *CA* matrix (after row organization) is

$$
\begin{array}{c|cccc|}
 & A_1 & A_3 & A_2 & A_4 \\
\hline
A_1 & 45 & 45 & 0 & 0 \\
A_3 & 45 & 53 & 5 & 3 \\
A_2 & 0 & 5 & 80 & 75 \\
A_4 & 0 & 3 & 75 & 78
\end{array}
$$

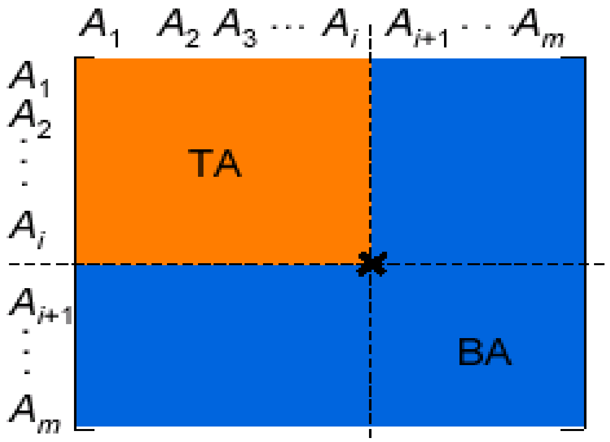
Example of BEA

$$
\begin{array}{c|cc|}
 & A1 & A2 \\
\hline
A1 & 45 & 0 \\
A2 & 0 & 80 \\
A3 & 45 & 5 \\
A4 & 0 & 75
\end{array}
\longrightarrow
\begin{array}{c|ccc|}
 & A1 & A3 & A2 \\
\hline
A1 & 45 & 45 & 0 \\
A2 & 0 & 5 & 80 \\
A3 & 45 & 53 & 5 \\
A4 & 0 & 3 & 75
\end{array}
\longrightarrow
$$

$$
\longrightarrow
\begin{array}{c|cccc|}
 & A1 & A3 & A2 & A4 \\
\hline
A1 & 45 & 45 & 0 & 0 \\
A2 & 0 & 5 & 80 & 75 \\
A3 & 45 & 53 & 5 & 3 \\
A4 & 0 & 3 & 75 & 78
\end{array}
\longrightarrow
\begin{array}{c|cccc|}
 & A1 & A3 & A2 & A4 \\
\hline
A1 & 45 & 45 & 0 & 0 \\
A3 & 45 & 53 & 5 & 3 \\
A2 & 0 & 5 & 80 & 75 \\
A4 & 0 & 3 & 75 & 78
\end{array}
$$

Partitioning

How can you divide a set of clustered attributes $\{A_1, A_2, \ldots, A_n\}$ into two (or more) sets $\{A_1, A_2, \ldots, A_i\}$ and $\{A_{i+1}, \ldots, A_n\}$ such that *there are no (or minimal) applications that access both (or more than one) of the sets.*



Algorithm of VF

Define

TQ = set of applications that access only *TA*

BQ = set of applications that access only *BA*

OQ = set of applications that access both *TA* and *BA*

and

CTQ = total number of accesses to attributes by applications that access only *TA*

CBQ = total number of accesses to attributes by applications that access only *BA*

COQ = total number of accesses to attributes by applications that access both *TA* and *BA*

Then find the point along the diagonal that maximizes

$$CTQ * CBQ - COQ^2$$

That means that the total accesses to only one fragment are maximized while the total accesses to both fragments are minimized.

Example of VF

FPROJ = {PROJ1,PROJ2}

PROJ1 = {A1,A3}

= {PNO,BUDGET}

PROJ2 = {A1,A2,A4}

= {PNO,PNAME,LOC}

$$\begin{array}{cc} & \begin{array}{cccc} A_1 & A_2 & A_3 & A_4 \end{array} \\ \begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} & \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \begin{array}{ccc} S_1 & S_2 & S_3 \end{array} \\ \begin{array}{c} q_1 \\ q_2 \\ q_3 \\ q_4 \end{array} & \begin{bmatrix} 15 & 20 & 10 \\ 5 & 0 & 0 \\ 25 & 25 & 25 \\ 3 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \begin{array}{cccc} A_1 & A_3 & A_2 & A_4 \end{array} \\ \begin{array}{c} A_1 \\ A_3 \\ A_2 \\ A_4 \end{array} & \begin{bmatrix} 45 & 45 & 0 & 0 \\ 45 & 53 & 5 & 3 \\ 0 & 5 & 80 & 75 \\ 0 & 3 & 75 & 78 \end{bmatrix} \end{array}$$

Two problems for partitioning

- **Cluster forming in the middle of the *CA* matrix**
  - Shift a row up and a column left and apply the algorithm to find the "best" partitioning point
  - Do this for all possible shifts
- **More than two clusters: m-way partitioning**
  - try 1, 2, …, m–1 split points along diagonal and try to find the best point for each of these
  - Recursively apply the binary partitioning algorithm to each of the fragments obtained during the previous iteration

Correctness of VF

A relation R, defined over attribute set *A* and key K, generates the vertical partitioning $F_R = \{R_1 , R_2 , \ldots, Rr \}$.

- Completeness
  - Guaranteed by the PARTITION algorithm since each attribute of the global relation is assigned to one of the fragments. The following should be true for A:

$$A = \cup R_i$$

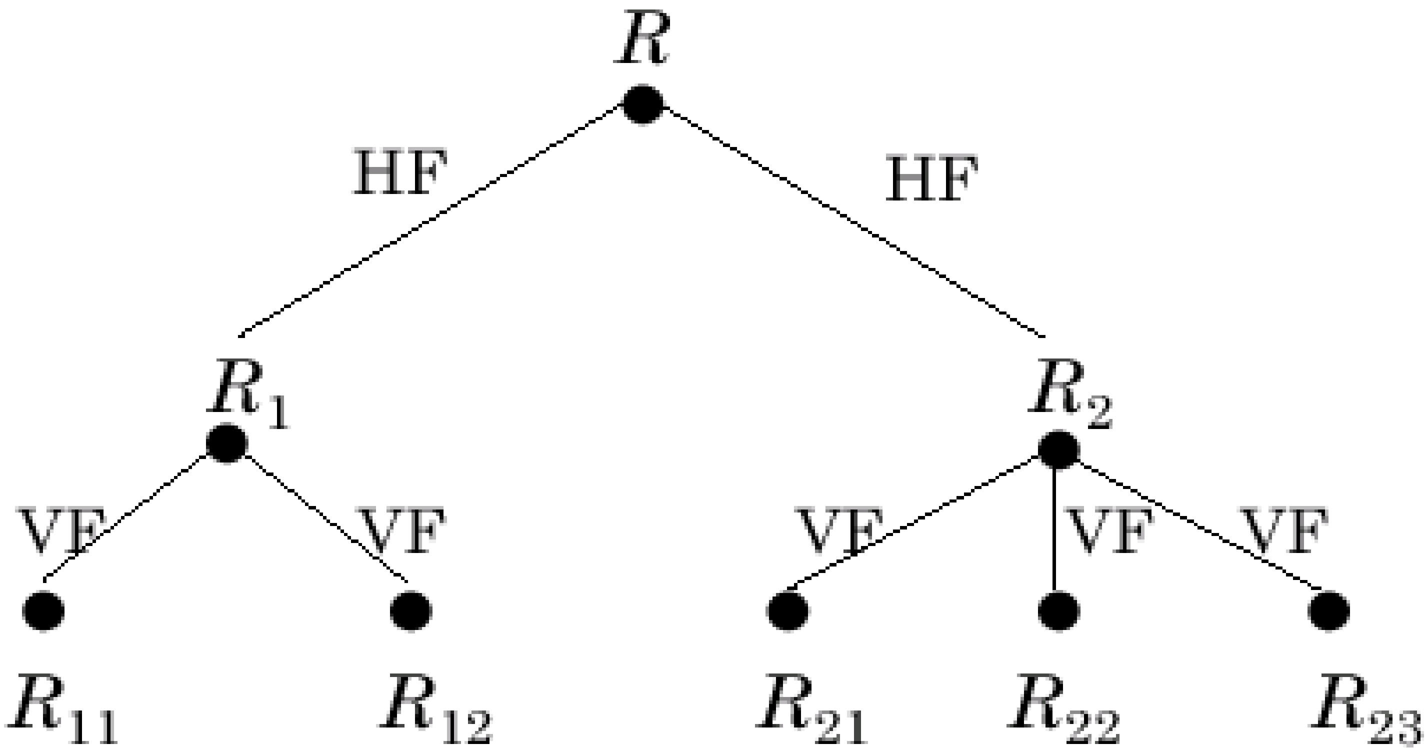
- Reconstruction
  - Reconstruction can be achieved by the join operation (key)

$$R = \infty_K R_i, R_i \in F_R$$

- Disjointness
  - TID's are not considered to be overlapping since they are maintained by the system
  - Duplicated keys are not considered to be overlapping

3.3.3 Hybrid Fragmentation

Hybrid Fragmentation



Hybrid Fragmentation

- **Tree-structured partitioning**
- **The number of levels of nesting can be large, but it is certainly finite**
- **In most practical application do not exceed 2**
  - Normalized global relations already have small degrees
  - One cannot perform too many vertical fragmentations before the cost of joins becomes very high

Correctness of HF

- Completeness
  - If the intermediate and leaf fragments are complete
- Reconstruction
  - Starts at the leaves of the partitioning tree and moves upward by performing joins and unions
- Disjointness
  - If the intermediate and leaf fragments are disjoint

3.4 Allocation

Allocation

- Problem Statement

  Given $F = \{F1, F2, \ldots, Fn\}$ fragments

  $S = \{S1, S2, \ldots, Sm\}$ network sites

  $Q = \{q1, q2, \ldots, qq\}$ applications

  Find the "optimal" distribution of *F* to S.

- **Optimality**
  - Minimal cost
    - Communication + storage + processing (read & update)
    - Cost in terms of time (usually)
  - Performance
    - Response time and/or throughput
  - Constraints
    - Per site constraints (storage & processing)

FAP vs DAP

FAP: File Allocation Problem

DAP: Database Allocation Problem

- Fragments are not individual files
  - relationships should be taken into account
- Access to databases is more complicated
  - remote file access model not applicable; relationship between allocation and query processing should be properly modeled
- Cost of integrity enforcement should be considered
- Cost of concurrency control should be considered

Information Requirements

- Database information
  - selectivity of fragments: $sel_i(F_j)$
  - size of a fragment: $size(F_j) = card(F_j) * length(Fj)$
- Application information
  - access types and numbers
  - access localities
  - Response-time constraint
- Site information
  - unit cost of storing data at a site
  - unit cost of processing at a site
- Network information
  - Bandwidth, latency, communication overhead
  - Here, the cost of communication is defined in terms of frame
    - **communication cost/frame between two sites**
    - **frame size**

Application information

- Number of read accesses of a query to a fragment
- Number of update accesses of query to a fragment
- A matrix indicating which queries updates which fragments
- A similar matrix for retrievals
- originating site of each query

Allocation Model

General Form

min(Total Cost)

subject to

response time constraint

storage constraint

processing constraint

Decision Variable

x_{ij} **=1 if fragment** F_i **is stored at site** S_j

0 otherwise

Total Cost

all queries **query** processing cost +

all sites all fragments cost of **storing** a fragment at a site

- Storage Cost **(of fragment F_j at S_k)**
  - (unit storage cost at S_k) *(size of F_j) $*x_{jk}$
- Query Processing Cost **(for one query)**
  - processing component + transmission component

Query Processing Cost – processing component

access cost +
integrity enforcement cost +
concurrency control cost

- Access cost
  - For all sites, all fragments

    (number of update accesses+ number of read accesses) * x_{ij} * local processing cost at a site
- Integrity enforcement / concurrency control costs
  - Can be similarly calculated

Query Processing Cost – transmission component

cost of processing updates +
cost of processing retrievals

- Cost of updates
  - For all sites, all fragments

    update message cost + acknowledgment cost
- Cost of Retrievals
  - For all fragments

    min $_{\text{all sites}}$ (cost of retrieval command +cost of sending back the result)

Constraints

- Response Time Constraints (for a query)
  - execution time of query ≤ maximum allowable response time for that query
- Storage Constraint (for a site)
  - For all fragments

    storage requirement of a fragment at that site ≤ storage capacity at that site
- Processing constraint (for a site)
  - For all queries

    processing load of a query at that site ≤ processing capacity of that site

Solution Methods

- FAP is NP-complete
- DAP is also NP-complete

- Heuristics
- Reducing the complexity
  - Assume all candidate partitions known; select the “best” partitioning
  - Ignore replication at first

3.5 Conclusion

Conclusion

- About fragmentation
- About allocation
  - Allocation is typically treated independently of fragmentation, that actually contributes to the complexity of the allocation models.
- About the static environment assumption
  - In a dynamic environment, the process becomes one of design-redesign-materialization.

References

- S. Ceri, G. Pelagatti, Distributed Databases: Principles and Systems, New York: McGraw-Hill, 1984
- S. Ceri, M. Negri, G. Pelagatti, Horizontal Data Partitioning in Database Design, In Proc. ACM SIGMOD Int. Conf. on Management of Data, June 1982, 128-136
- J. A. Hoffer, A Clustering Approach to the Generation of Subfiles for the Design of a Computer Data Base, Ph.D. dissertation, Ithaca, N.Y.: Dept. of Operations Research, Cornell U., Jan 1975
- S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, Vertical Partitioning of Algorithms for Database Design, ACM Trans. Database Syst.(Dec 1984), 9(4):680-710
- K. P. Eswaran, Placement of Records in a File and File Allocation in a Computer Network, In: Information Processing'74, 1974, 304-307