5. Distributed Database Design

Chapter 3

Distributed Database Design

Review of Traditional Relational Database Design

- Conceptual Design (概念设计)
  - Understand users' data requirements, processing requirements, security and integrity requirements, etc.
  - Design a conceptual model (E-R model) through data abstraction
- Logical Design (逻辑设计)
  - Design conceptual and external schema of the DB (mainly relational tables and views)
  - 1NF (attribute), 2NF(key), 3NF (foreign key), 4NF, 5NF, 6NF, ……
- Physical Design (物理设计)
  - Design data storage structure and access methods (e.g., indexes)

Outline

- Introduction
- Fragmentation （片段划分）
  - Horizontal fragmentation （水平划分）
  - Derived horizontal fragmentation （导出式水平划分）
  - Vertical fragmentation （垂直划分）
- Allocation （片段分配）

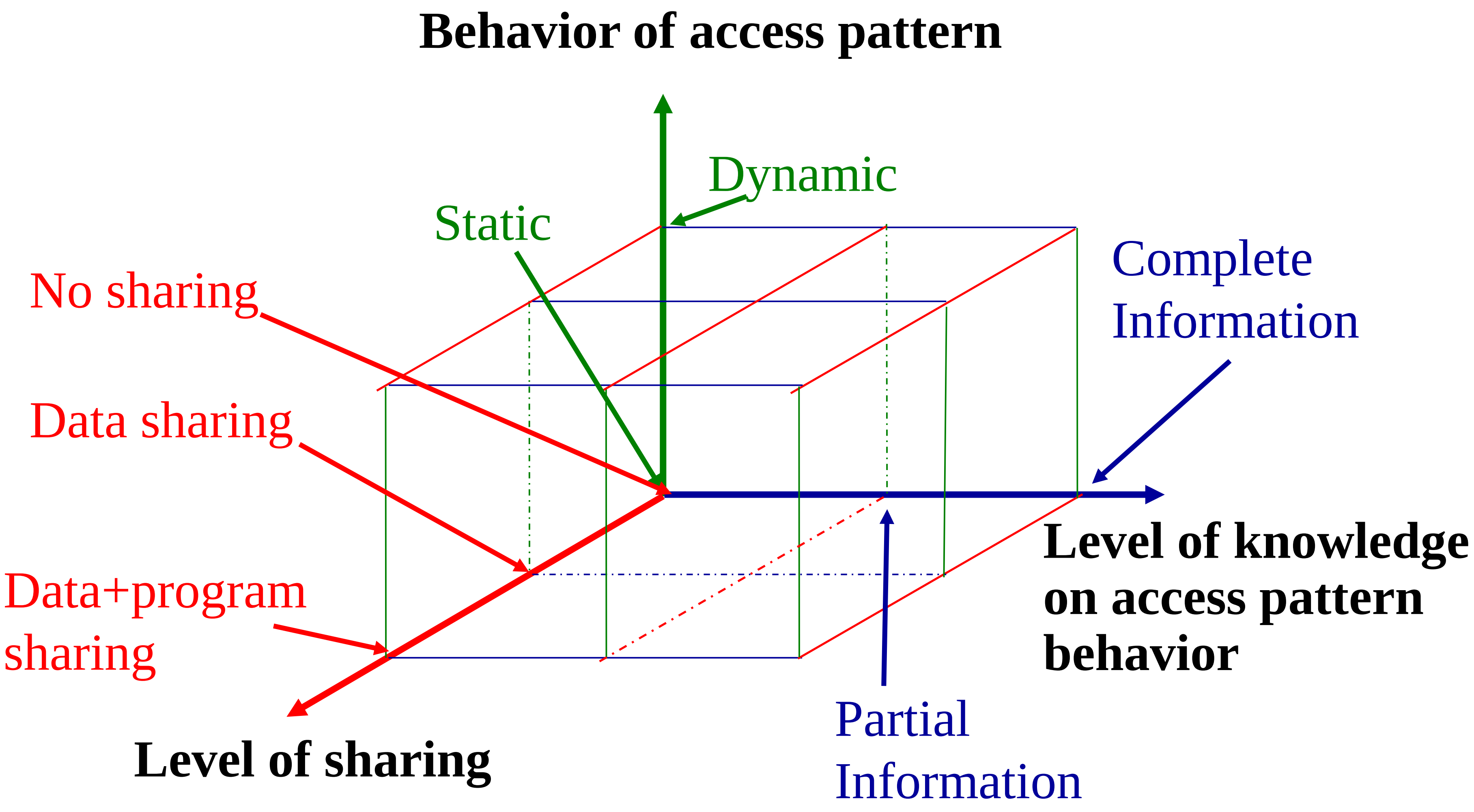
Outline

- ☞ Introduction
- ❖ Fragmentation（片段划分）
  - Horizontal fragmentation（水平划分）
  - Derived horizontal fragmentation（导出式水平划分）
  - Vertical fragmentation（垂直划分）
- ❖ Allocation（片段分配）

Distributed Database Design

- The design of a distributed computer system involves making decisions on the placement of *data* and *programs* across the sites of a computer network.
- In distributed DBMSs, such placement involves two things:
  - Placement of the DDBMS software
  - Placement of the applications that run on the database
  - Placement of data
- The course concentrates on distribution of data
  - The distribution of DDBMS and applications are given a priori.

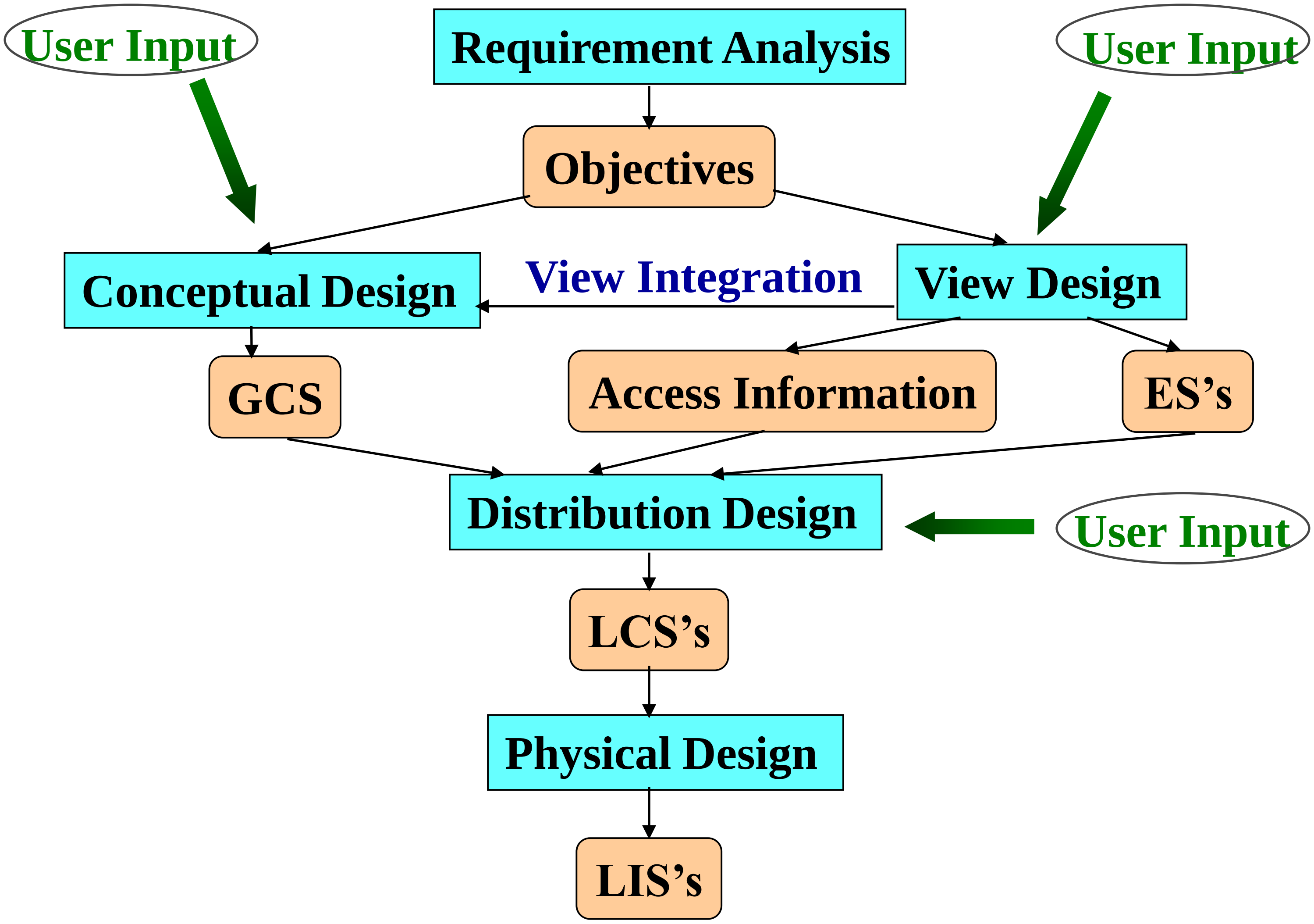
Framework of Distribution



Design Strategies

- Top-Down
  - Mostly in designing systems from scratch
  - Mostly in designing homogeneous systems
- Bottom-up
  - When the databases already exist at a number of sites
- Combining both

Top-Down Design Process



Distribution Design Issues

- Fragmentation
  - Why fragmentation at all?
  - How should we fragment?
  - How much should we fragment?
  - How to test correctness of decomposition?
- Allocation
- Necessary information required for fragmentation and allocation

Why Fragment?

- Can’t we just distribute relations?
  - Too big
  - Partial data is wanted
- What is a reasonable unit of distribution?
  - Fragments of relations (sub-relations)
  - Access locality

Fragmentation

- ❖ Unit of distribution
  = unit of data application accesses
- ☺ Reduce irrelevant data access
- ☺ Facilitate intra-query concurrency over different fragments
- ☺ Can be used with other performance enhancing methods (e.g., indexing and clustering)
- ☹ Applications have conflicting requirements, making disjoint fragmentation a very hard problem
- ☹ Multiple fragment access requires join or union
- ☹ Semantic data control (integrity enforcement) could be very costly

About fragmentation

- How should we fragment?
  - Vertical Fragments sub grouping of attributes
  - Horizontal Fragments sub grouping of tuples
  - Mixed/Hybrid Fragments combination of above two

- How much to fragment?
  - Too little: too much of irrelevant data access
  - Too much: too much processing cost
  - Need to find suitable level of fragmentation

Correctness Criteria

- Completeness no loss of data
  - 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 ∇ such that $R = \nabla_{1 \leq i \leq n} R_i$
- 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!=j).

Allocation Alternatives

- Full Replication
  - Each fragment resides at each site
- Partial Replication
  - Each fragment resides at some of the sites
- Not-replicated (Partitioned)
  - Each fragment resides at only one site

Q & A:

What are the advantages and disadvantages?

Allocation Alternatives

	Full-Replication	Partial -replication	Partitioning
Query Processing	Easy	Same Difficulty	
Directory Management	Easy or non-existent	Same Difficulty	
Concurrency Control	Moderate	Difficult	Easy
Reliability	High	High	Low
Reality	Possible application	Realistic	Possible application

Rule of Thumb for Allocation

If number-of-read-only-queries is more than number-of-update-queries,

then replication is advantageous, otherwise replication may cause problems.

Information Requirements

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

Outline

- Introduction
- Fragmentation （片段划分）
  - Horizontal fragmentation （水平划分）
  - Derived horizontal fragmentation （导出式水平划分）
  - Vertical fragmentation （垂直划分）
- Allocation （片段分配）

Fragmentation

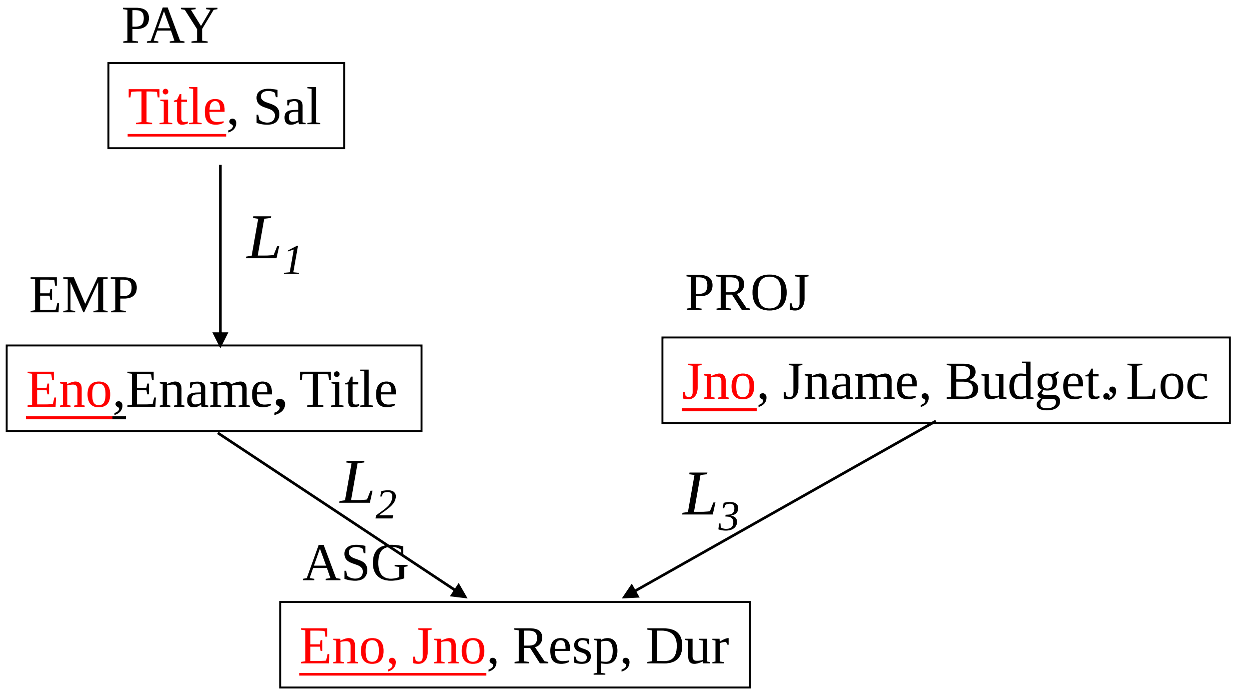
- Horizontal fragmentation (HF)
  - Primary horizontal fragmentation (PHF)
    - based on predicates accessing the relation
  - Derived horizontal fragmentation (DHF)
    - based on predicates being defined on another logically related relation

We shall first study algorithm for horizontal fragmentation, and then study issues related to derived horizontal fragmentation.

- Vertical fragmentation (VF)
- Hybrid fragmentation (HVF)

PHF Information Requirements

- Database Information
  - > links



PHF Information Requirements (*cont.*)

- Application Information 1
  - **Simple predicate**: Given $R(A_1, A_2, \ldots, A_n)$, with each A_i having domain of values dom(A_i), a simple predicate p_j is

$$p_j : \quad A_i \;\theta\; Value$$

where $\theta \in \{<, >, \leq, \geq, \neq\}$ and $Value \in$ dom(A_i).

Example

Jname="maintenance"
Budget $\leq$ 200000

PHF Information Requirements (cont.)

- Application Information 2
  - **minterm predicate:** Given R and a set of **simple predicates** $P_r = \{p_1, p_2, \ldots, p_m\}$ on R, the set of minterm predicates $M = \{m_1, m_2, \ldots, m_z\}$ is defined as

$$M = \{m_i \mid m_i = \wedge_{Pj \in Pr}\ p^*_j\} \quad (1 \le i \le z,\ 1 \le j \le z)$$

where $p^*_j = p_j$ or $\neg p_j$

Example

m_1: (Jname=“maintenance”) $\wedge$ (Budget $\le$ 200000)

m_2: $\neg$(Jname=“maintenance”) $\wedge$ (Budget $\le$ 200000)

m_3: (Jname=“maintenance”) $\wedge$ $\neg$(Budget $\le$ 200000)

m_4: $\neg$(Jname=“maintenance”) $\wedge$ $\neg$(Budget $\le$ 200000)

Primary Horizontal Fragmentation

- Each horizontal fragment R_i of relation R is defined by $R_i = \sigma_{F_i}(R)$, $1 \leq i \leq w$, where F_i is a selection formula, which is (preferably) a minterm predicate.
  - A horizontal fragment R_i of relation R consists of all the tuples of R which satisfy a minterm predicate m_i.
- Given a set of minterm predicates M, there are as many as horizontal fragments of relation R as there are minterm predicates.
- Set of horizontal fragments also referred to as minterm fragments

PHF - Algorithm

Input: Relation R and a set of simple predicates P_r

Output: The set of fragments of $R = \{R_1 , R_2 , ..., R_w \}$, which obey the fragmentation rules.

Preliminaries:

- P_r should be *complete*
- P_r should be *minimal*

Completeness of Simple Predicates

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

Completeness of Simple Predicates

- Example: Assume there are only 3 locations in the whole table.

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

Applications:

Q1: Find the projects at each location

Q2: Find projects with budget less than $200,000

incomplete

Predicates:

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

☺ Pr={ LOC="Montreal", LOC="New York", LOC="Paris",
BUDGET≤ 2000000, BUDGET >200000}

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 f_i and f_j differently.
- In other words, the simple predicate should be *relevant* in determining a fragmentation.
- If all the predicates of a set P_r are relevant, then P_r is minimal.

Minimality of Simple Predicates

Example

Applications:
Q1: Find the projects at each location
Q2: Find projects with budget less than $200,000

☺ **Pr={ LOC="Montreal", LOC="New York", LOC="Paris",**
BUDGET ≤ 2000000, BUDGET >200000}

minimal

☹ **Pr={ LOC="Montreal", LOC="New York", LOC="Paris",**
BUDGET ≤ 200000, BUDGET >200000,
PNAME="Instrumentation"}

COM_MIN Algorithm

Input: A relation *R* and a set of simple predicates P_r

Output: A *complete* and *minimal* set of simple predicates P_r' for P_r

Rule 1: A relation or fragment is partitioned into at least two parts which are accessed differently by at least one application.

COM_MIN Algorithm (*cont.*)

1. Initialization

- Find a $p_i \in P_r$ such that p_i partitions *R* according to *Rule 1*
- Set $P_r' = p_i$; $P_r \leftarrow P_r - p_i$; $F \leftarrow f_i$

2. Iteratively add predicates to P_r' until it is complete

- Find a $p_j \in P_r$ such that p_j partitions some f_k defined according to minterm predicate over P_r' according to *Rule 1*
- $P_r' \leftarrow p_j$; $P_r = P_r - p_j$; $F \leftarrow f_j$
- If $\exists p_k \in P_r'$ which is irrelevant, then

  $P_r' = P_r' - p_k$; $F \leftarrow F - f_k$

PHORIZONTAL Algorithm

Make use of COM_MIN to perform fragmentation

Input: A relation *R* and a set of simple predicates P_r

Output: A set of minterm predicates *M* according to which relation *R* is to be fragmented

Steps:

- $P_r' \leftarrow$ COM_MIN(R, P_r)
- Determine the set *M* of minterm predicates
- Determine the set *I* of implications among $p_i \in P_r'$
  Eliminate the contradictory minterms from *M*

Contradictory Minterms

- Given a minimal and complete set of simple predicates, containing *n* simple predicates
- Not all the minterm fragments derived are valid
  - A fragment can be self contradictory because of implications among simple predicates.

Example:

Dom(Sal): [10000, 200000]; Dom(Loc) = {HK,SF}

p_1 : sal < 50000; p_2 : Loc = HK; p_3 : Loc = SF

Note: $p_2 \Rightarrow (\neg\, p_3)$; $p_3 \Rightarrow (\neg\, p_2)$

the minterm $p_1 \wedge p_2 \wedge p_3$ is self contradictory.

PHORIZONTAL Algorithm (*cont.*)

Input: relation R and a set of simple predicates Pr

Output: a set of minterm fragments M

begin

$Pr' =$ COM-MIN(R, Pr);

M = set of minterm predicates from Pr'

I = set of implications among $p_i \in Pr'$

for each $m_i \in M$

if m_i is contradictory according to I then

$M = M - m_i$

end

PHF Example: PAY

PAY

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

Application

1) *Check the salary info and determine raise*

Employee records kept at two sites ==> application runs at two sites

Simple predicates: P_1: SAL ≤ 30000, P_2: SAL > 30000

$Pr = \{P_1, P_2\}$, which is complete and minimal

$Pr' = Pr$

Minterm predicates: m_1: (SAL ≤ 30000); m_2: (SAL > 30000)

PAY_1

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

PAY_2

TITLE	SAL
Elec. Eng.	40000
Syst. Anal.	34000

PHF Example: PROJ

PROJ

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

Applications

1) *Find the name and budget of projects given their locations*
   - Issued at three sites
2) *Access project information according to budget* (one site accesses ≤ 200000; other two access >200000)

p_1: LOC = "Montreal"
p_2 : LOC = "New York"
p_3 : LOC = "Paris"

p_4: BUDGET ≤ 200000
p_5 : BUDGET > 200000

m_1: LOC = "Montreal" ∧ BUDGET ≤ 200000
m_2 : LOC = "Montreal" ∧ BUDGET > 200000
m_3 : LOC = "New York"∧ BUDGET ≤ 200000
m_4 : LOC = "New York"∧ BUDGET > 200000
m_5 : LOC = "Paris" ∧ BUDGET ≤ 200000
m_6 : LOC = "Paris" ∧ BUDGET > 200000

PHF Example: PROJ - Result

PROJ

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

PNO	PNAME	BUDGET	LOC
P1	Instrumentation	150000	Mntreal

PNO	PNAME	BUDGET	LOC
P2	Database	135000	New York

PNO	PNAME	BUDGET	LOC
P3	CAD/CAM	250000	New York

PNO	PNAME	BUDGET	LOC
P4	Maintenance	310000	Paris

m_1: LOC = “Montreal” $\wedge$ BUDGET $\leq$ 200000

m_2 : LOC = “Montreal” $\wedge$ BUDGET > 200000

m_3 : LOC = “New York” $\wedge$ BUDGET $\leq$ 200000

m_4 : LOC = “New York” $\wedge$ BUDGET > 200000

m_5 : LOC = “Paris” $\wedge$ BUDGET $\leq$ 200000

m_6 : LOC = “Paris” $\wedge$ BUDGET > 200000

PHF - Correctness

- Completeness
  - Since *Pr* is complete and minimal, the selection predicates are complete
- Reconstruction
  - If relation *R* is fragmented into $F_R = \{R_1, R_2, \ldots, R_r\}$

$$R = \cup_{\forall R_i \in F_R} R_i$$

- Disjointness
  - Minterm predicates that form the basis of fragmentation should be mutually exclusive

More about PHF

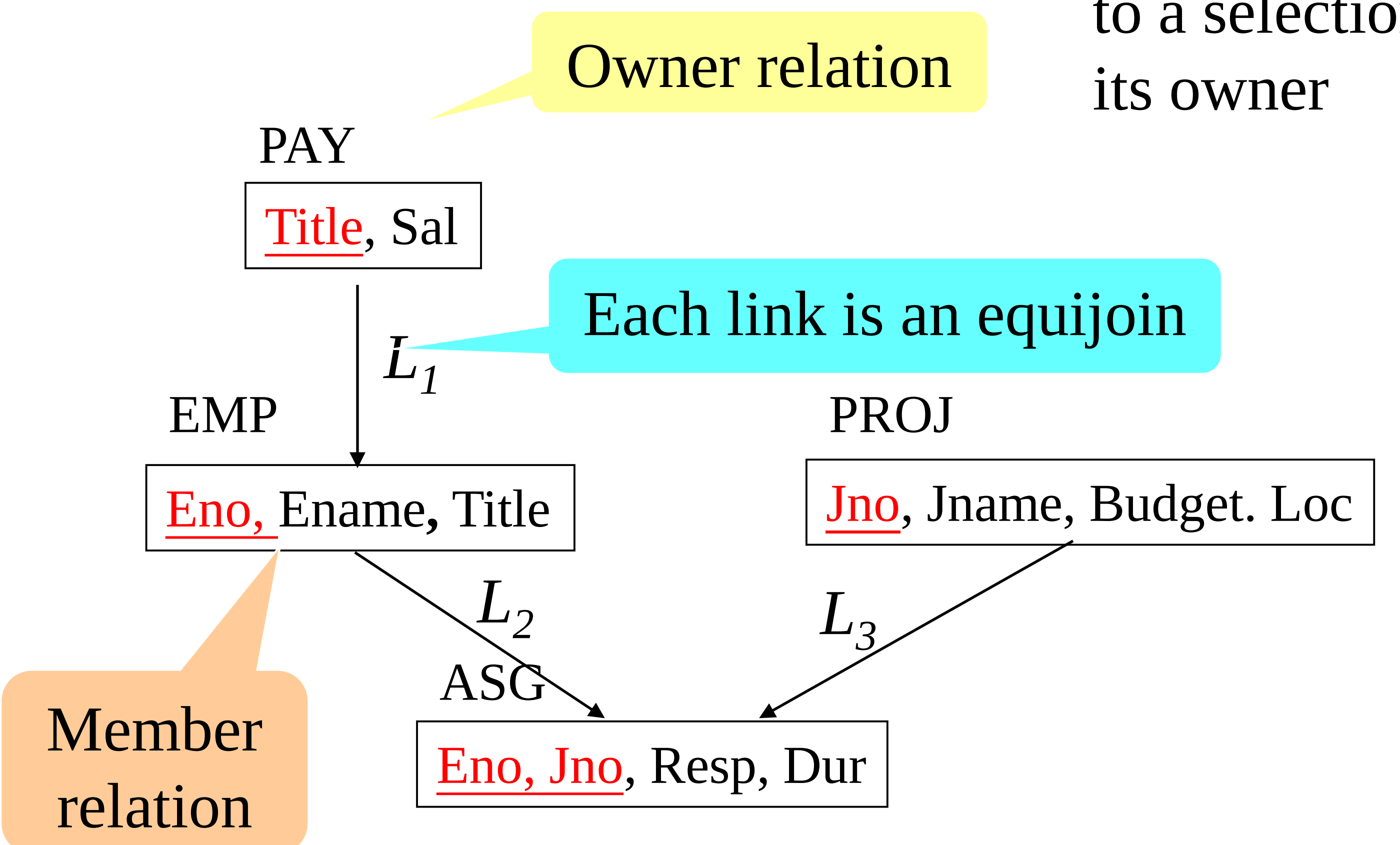
- Application Information 3
  - Minterm selectivity: ***sel*** **(*m_i*)**
    – number of tuples of the relation that would be accessed by a user query specified according to a given minterm predicate.
  - Access frequency: ***acc*** **(*q_i*)**
    – frequency with which user applications access data.
    If $Q = \{q_1, q_2, ..., q_n\}$ is the set of queries, *acc* (q_i) indicates access frequency of query q_i in a given period.

- Database Information
  - Cardinality of each relation: ***card (R)***

Follow ``80/20'' rule

DHF: Derived Horizontal Fragmentation

DHF: Defined on a member relation according to a selection operation on its owner



DHF – Example

PAY

Title, **Sal**

$\downarrow L_1$

EMP

Eno, **Ename, Title**

$PAY_1 = \sigma_{SAL \leq 30000}\ PAY$

TITLE	SAL
Mech. Eng.	27000
Programmer	24000

$PAY_2 = \sigma_{SAL > 30000}\ PAY$

TITLE	SAL
Elec. Eng.	40000
Syst. Anal.	34000

EMP

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

DHF $\rightarrow$

$EMP_1 = EMP \ltimes PAY_1$

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

$EMP_2 = EMP \ltimes PAY_2$

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

DHF: Derived Horizontal Fragmentation

- Let *S* be horizontally fragmented and let there be a link *L* with *owner*(*L*) = *S*, and *member*(*L*) = *R*, the derived horizontal fragments of *R* are defined as

$$R_i = R \ltimes S_i \text{ , } 1 \leq i \leq w$$

where S_i is the horizontal fragment of *S*, $\ltimes$ is the semijoin operator, and w is the maximum number of fragments.

- Inputs to derived horizontal fragmentation
  - partitions of owner relation
  - member relation
  - the semijoin condition
- The algorithm is straightforward.

DHF: Correctness

- Completeness
  - Primary horizontal fragmentation based on completeness of selection predicates. For derived horizontal fragmentation based on referential integrity
- Reconstruction
  - Same as primary horizontal fragmentation (via union)
- Disjointedness
  - Simple join graphs between the owner and the member fragments.

DHF: Issues

- Multiple owners for a member relation; how should we derive horizontally fragments of a member relation?
- There could be a chain of derived horizontal fragmentation.

Vertical Fragmentation (VF)

- Has been studied within the centralized context
- Vertical partitioning of a relation R produces fragments R_1, R_2, ..., R_m, each of which contains a subset of R's attributes as well as the primary key of R
- The objective of vertical fragmentation is to reduce irrelevant attribute access, and thus irrelevant data access
- "Optimal" vertical fragmentation is one that minimizes the irrelevant data access for user applications

VF Two Approaches

- Grouping: each individual attribute one fragment, at each step join some of the fragments until some criteria being satisfied
  - Attributes to fragments
- Splitting: start with global relation, and generate beneficial partitions based on access behavior of the applications
  - Relations to fragments
- Replicated key attributes
  - Advantage: easier to enforce functional dependencies (for integrity checking)

VF Information Requirements

- Application Information
  - Attribute affinities
    - A measure that indicates how closely related the attributes are
    - This is obtained from more primitive usage data
  - Attribute usage values
    - Given a set of queries $Q = \{q_1, q_2, \ldots, q_m\}$ that will run on relation $R(A_1, A_2, \ldots, A_n)$

$$use(q_i, A_j) = \begin{cases} 1 & \text{if attribute } A_j \text{ is referenced by query } q_i \\ 0 & \text{otherwise} \end{cases}$$

$use(q_i, .)$ can be defined accordingly.

VF Definition of use(q_i, A_j)

- Consider the following 4 queries for relation PROJ

  q_1: SELECT BUDGET FROM PROJ WHERE PNO = val;

  q_2: SELECT PNAME,BUDGET FROM PROJ;

  q_3: SELECT PNAME FROM PROJ WHERE LOC = val;

  q_4: SELECT SUM(BUDGET) FROM PROJ WHERE LOC=val;

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

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

VF - 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 $Q = \{q_1, q_2, ..., q_m\}$ is defined as:

$$Aff(A_i, A_j) = \sum_{\{k \mid use(q_k, A_i)=1 \wedge use(q_k, A_j)=1\}} \sum_{site\text{-}l} ref_{site\text{-}l}(q_k) * acc_{site\text{-}l}(q_k)$$

where $ref_{site\text{-}l}(q_k)$ is the number of accesses to attributes for each execution of application q_k at site *site-l*; $acc_{site\text{-}l}(q_k)$ is the access frequency of query q_k at site *site-l* .

VF - Affinity Measure aff(A_i , A_j)

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

Also assume the access frequency of query q_k at different sites is:

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

then $Aff(A_1, A_3) = 15*1 + 20*1 + 10*1 = 45$

and the attribute affinity matrix AA is:

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

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

A Matrix for Vertical Fragmentation

- This affinity matrix will be used to guide the fragmentation effort. The process involves first clustering together the attributes with high affinity for each other, and then splitting the relation accordingly.

VF - Correctness

A relation R, defined over attribute set *A* and key *K*, generates vertical partitioning

$$F_R = \{R_1, R_2, \ldots, R_r\}$$

- Completeness

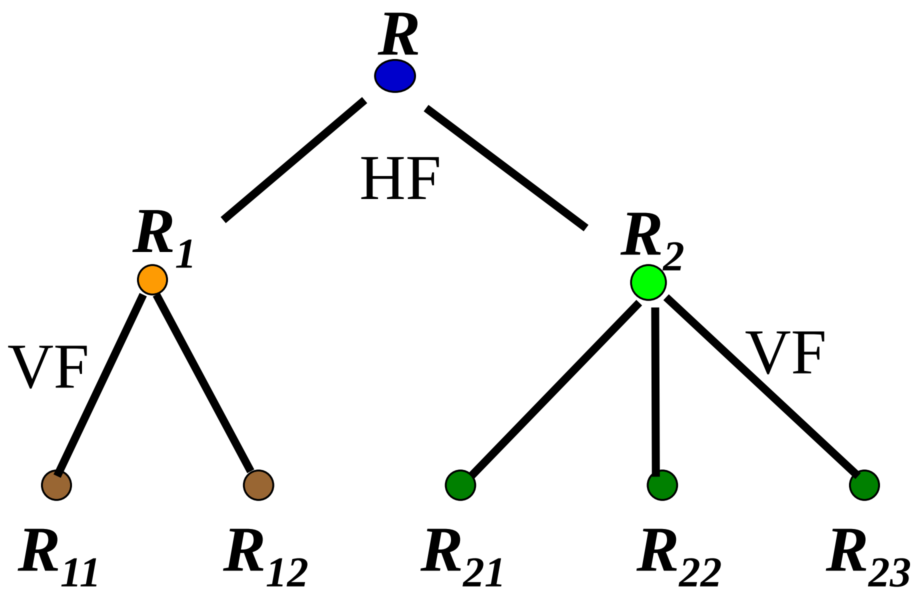
$$A = \cup A_{R_i}$$

- Reconstruction

$$R = \bowtie_k R_i \quad (\forall R_i \in F_R)$$

- Disjointness
  - Duplicate keys are not considered to be overlapping

Hybrid Fragmentation



Outline

- Introduction
- Fragmentation （片段划分）
  - Horizontal fragmentation （水平划分）
  - Derived horizontal fragmentation （导出式水平划分）
  - Vertical fragmentation （垂直划分）
- Allocation （片段分配）

Allocation

- File Allocation vs. Database Allocation
  - Fragments are not individual files
    - Relationships have to be maintained
  - Access to databases is more complicated
    - Remote file access model not applicable
    - Relationship between allocation and query processing
  - Cost of integrity enforcement should be considered
  - Cost of concurrency control should be considered

Allocation Problem

- Assume

  $F=\{F_1, F_2, \ldots, F_n\}$

  $S=\{S_1, S_2, \ldots, S_m\}$

  $Q=\{Q_1, Q_2, \ldots, Q_q\}$

Problem

Find the *optimal* distribution of F over S

Optimality with Two Aspects

- Minimal cost
  - Storing F_i at S_j
  - Querying F_i at S_j
  - Updating F_i at all S_j's with a copy of F_i
  - Communication
- Performance
  - Response time
  - Throughput
  - ……

Separate the two issues to reduce its complexity.

A Simple Formulation of the Cost Problem

$F=\{F_1,F_2, \ldots, F_n\}$
$S=\{S_1,S_2, \ldots, S_m\}$
$Q=\{Q_1,Q_2, \ldots, Q_q\}$

- For a single fragment F_i
  - $R = \{r_1, r_2, \ldots, r_m\}$

    r_j : read-only traffic generated at S_j for F_i
  - $U = \{u_1, u_2, \ldots, u_m\}$

    u_j : update traffic generated at S_j for F_i

A Simple Formulation of the Cost Problem (cont.)

- Assume the communication cost between any pair of sites S_i and S_j is fixed

$F=\{F_1, F_2, \ldots, F_n\}$
$S=\{S_1, S_2, \ldots, S_m\}$
$Q=\{Q_1, Q_2, \ldots, Q_q\}$

  - $C(T) = \{c_{1,1},\ c_{1,2},\ c_{1,3},\ \cdots,\ c_{1,m},\ \cdots,\ c_{m-1,m}\}$

    $c_{i,j}$: retrieval communication cost

  - $C'(U) = \{c'_{1,1},\ c'_{1,2},\ c'_{1,3},\ \cdots,\ c'_{1,m},\ \cdots,\ c'_{m-1,m}\}$

    $c'_{i,j}$: update communication cost

A Simple Formulation of the Cost Problem (cont.)

- $D = \{d_1, d_2, \ldots, d_m\}$

  cost for storing F_i at S_j

$F=\{F_1,F_2, \ldots, F_n\}$
$S=\{S_1,S_2, \ldots, S_m\}$
$Q=\{Q_1,Q_2, \ldots, Q_q\}$

No capacity constraints for sites and communication links

A Simple Formulation of the Cost Problem (cont.)

$F=\{F_1,F_2, \ldots, F_n\}$
$S=\{S_1,S_2, \ldots, S_m\}$
$Q=\{Q_1,Q_2, \ldots, Q_q\}$

- The allocation problem is a cost minimization problem for finding the set

  $$I \subseteq \{S_1, S_2, \ldots, S_m\}$$

  i.e., the sites I to store fragment F_i

A Simple Formulation of the Cost Problem (cont.)

$F=\{F_1,F_2, \ldots, F_n\}$
$S=\{S_1,S_2, \ldots, S_m\}$
$Q=\{Q_1,Q_2, \ldots, Q_q\}$

For *queries/updates from site* S_i

Site S_i Reads

$$r_i \cdot \min_{j|S_j \in I} c_{ij}$$

Site S_i Updates

$$\sum_{j|S_j \in I} u_i \cdot c_{ij}'$$

Site S_j Storage

$$\sum_{j|S_j \in I} d_j$$

Total Cost

$$\min\left[\sum_{i=1}^{m}\left(\sum_{j|S_j \in I} u_i c'_{ij} + r_i \cdot \min_{j|S_j \in I} c_{ij}\right) + \sum_{j|S_j \in I} d_j\right]$$

This formulation only considers one fragment F_i at site S_j.
It is NP-complete.

A Precise Formulation of the Cost Problem

- A precise formulation must consider:
  - All fragments together
  - How query is processed
  - The enforcement of integrity constraint
  - The cost of concurrency control and transaction control

Allocation Model in General

- Allocation Model

  min (total Cost)

  subject to

  – response time constraint

  – storage constraint

  – processing constraint

- Decision variable

$$x_{ij} = \begin{cases} 1 & \text{if fragment } F_i \text{ is stored at site } S_j \\ 0 & \text{otherwise} \end{cases}$$

Total Cost

$\sum_{\text{all sites}} \sum_{\text{all fragments}}$ cost of storing a fragment at a site +

$\sum_{\text{all queries}}$ query processing cost

where

- Storage Cost (on fragment F_i at site S_k):

  (unit storage cost at S_k) * (size of F_i) * x_{ik}

- Query Processing Cost (for one query)

  ①processing component + ②transmission component

Query Processing Cost

① Processing component

access cost + integrity enforcement cost + concurrency control cost

- access cost:

$$\sum_{\text{all sites}} \sum_{\text{all fragments}} (\text{number of update accesses} + \text{number of read accesses}) * x_{ik} * \text{local processing cost at site}$$

- integrity enforcement and concurrency control costs can be similarly calculated.

Query Processing Cost (*cont.*)

② Transmission component

cost of processing updates + cost of processing retrievals

- Cost of updates

$$\sum_{\text{all fragments}} \sum_{\text{all sites}} \text{update message cost} + \sum_{\text{all fragments}} \sum_{\text{all sites}} \text{acknowledgement cost}$$

- Retrieval costs

$$\sum_{\text{all fragments}} \min_{\text{all sites}} (\text{retrieval message cost} + \text{cost of sending back the result})$$

Constraints

- Response time for each query not longer than maximally allowed response time for that query.
- Storage constraint: The total size of all fragments allocated at a site must be less than the storage capacity at that site.
- Processing constraint: The total processing load because of all queries at a site must be less than the processing capacity at that site.

Solution Methods

- NP-complete
- Heuristics approaches
  - Exploring techniques developed in operational research (运筹学)
- Reduce problem complexity
  - ignore replication first, and then improve with a greedy algorithm

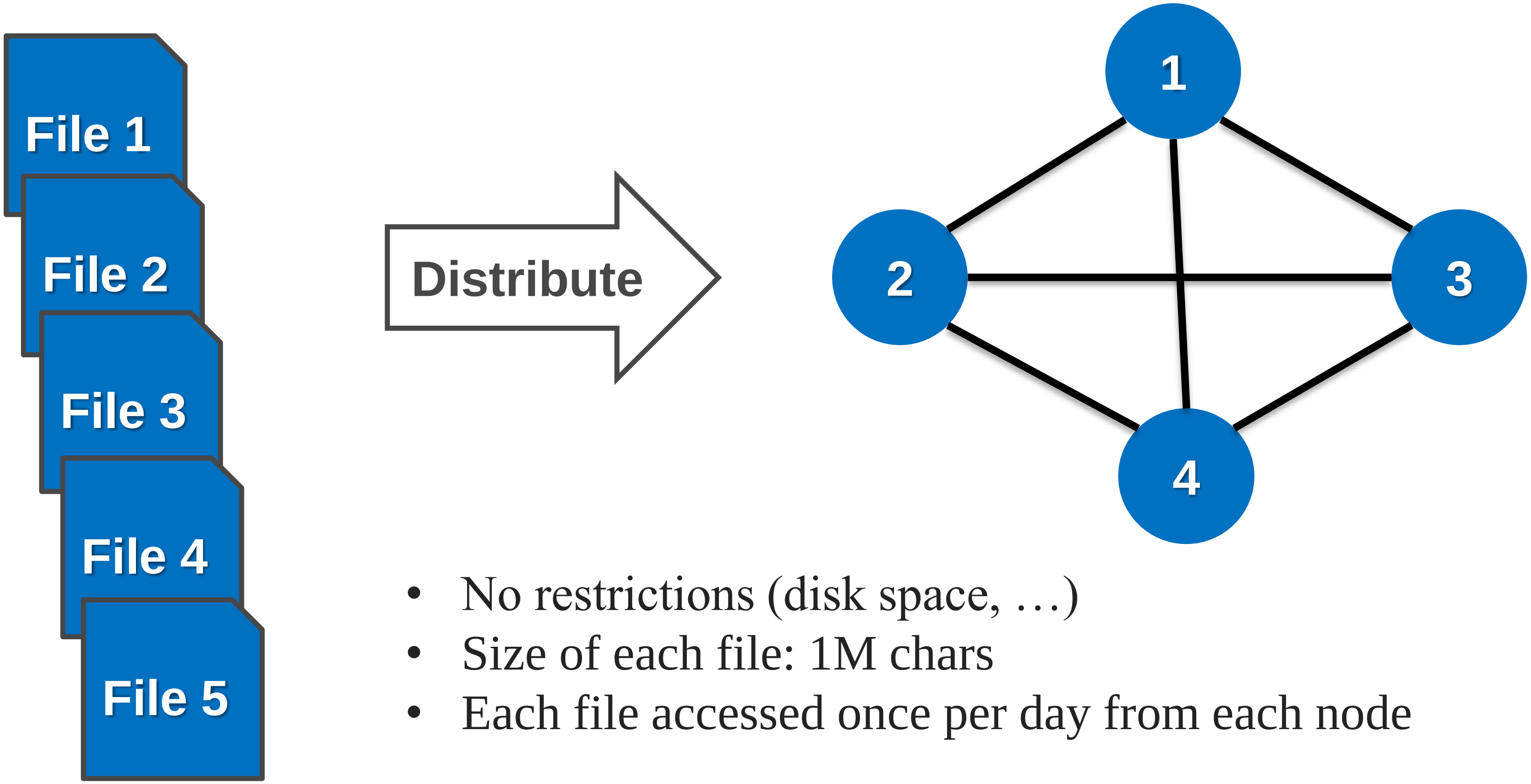
A Nested Genetic Method for Distributed Database Design

- Draw inspiration from nature
- Based on adaption in natural organisms
- Successfully applied to complex problems
- Result in a pool of *good* solutions

Components of Genetic Method

1. Representation of solutions (bit strings)
2. Pool of solutions (population)
3. Darwinian notion of “fitness”
4. Genetic operators
5. Survival of the fittest

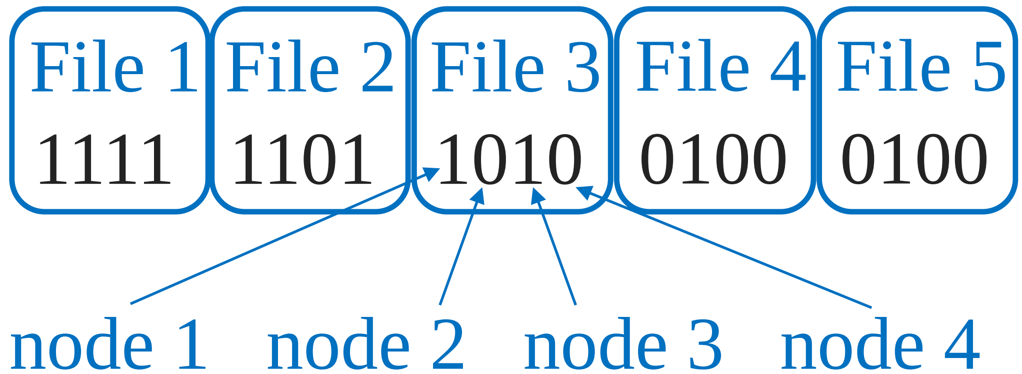
Example: distribute 5 files over 4 nodes



- No restrictions (disk space, …)
- Size of each file: 1M chars
- Each file accessed once per day from each node

Solution Representation

- 5×4 bits



Pool of Solutions

- Generate randomly initial pool of solutions
- Pool size *p* = input parameter
  - Large enough
  - Not too large

No	Solution
1	1111 1101 1010 0100 0100
2	0001 0100 0010 1000 1100
3	1011 1101 0110 1101 0101
4	0100 0010 1001 0100 0101
5	0010 0101 1000 0100 0101
6	1010 0101 1111 1111 0111

Notion of Fitness

- Solutions typically evaluated by performance
  (minimum communication volume per day)
  e.g., Fitness = 1- 9/60=0.85
  Selection Probability = 0.85/5.0=0.170

No	Solution	Performance	Fitness	Selection Probability
1	1111 1101 1010 0100 0100	9M characters	0.85	0.170
2	0001 0100 0010 1000 1100	14M characters	0.77	0.154
3	1011 1101 0110 1101 0101	7M characters	0.88	0.176
4	0100 0010 1001 0100 0101	12M characters	0.80	0.160
5	0010 0101 1000 0100 0101	13M characters	0.78	0.156
6	1010 0101 1111 1111 0111	5M characters	0.92	0.184
Total		60M characters	5.00	1.000

Genetic Operators

- 2 solutions = parent solutions
- Apply genetic operators
  - Crossover = Merge 2 solutions into 1
  - Mutation = Random bit switching
  - → New solutions

Parent 1:	1111 1101 1010 0100 0100	
Parent 2:	1010 0101 1111 1111 0111	
	crossover point	
Offspring 1:	1111 1101 1111 1111 0111	→ Performance = 2M Characters
Offspring 2:	1010 0101 1010 0100 0100	→ Performance = 12M Characters

Survival

- Choose *p* solutions from parents + children
  - Drop solution 2 (14M) and solution 5 (13M)

No	Solution	Performance	Fitness	Selection Probability
1	1111 1101 1010 0100 0100	9M characters	0.85	0.170
2	0001 0100 0010 1000 1100	14M characters	0.77	0.154
3	1011 1101 0110 1101 0101	7M characters	0.88	0.176
4	0100 0010 1001 0100 0101	12M characters	0.80	0.160
5	0010 0101 1000 0100 0101	13M characters	0.78	0.156
6	1010 0101 1111 1111 0111	5M characters	0.92	0.184
Total		60M→40M characters	5.00	1.000

Offspring 1: 1111 1101 1111 1111 0111 Performance = 2M Characters

Offspring 2: 1010 0101 1010 0100 0100 Performance = 12M Characters

Genetic Operators

- 1 solution can be mutated simply by choosing a bit at random and switching it from its current value to its complement (or another random value if genes are not binary)
  - 1111 11**0**1 1010 0100 0100 → Performance = 9M Characters

→ New solution:

1111 11**1**1 1010 0100 0100 → Performance = 8M Characters

Heuristics

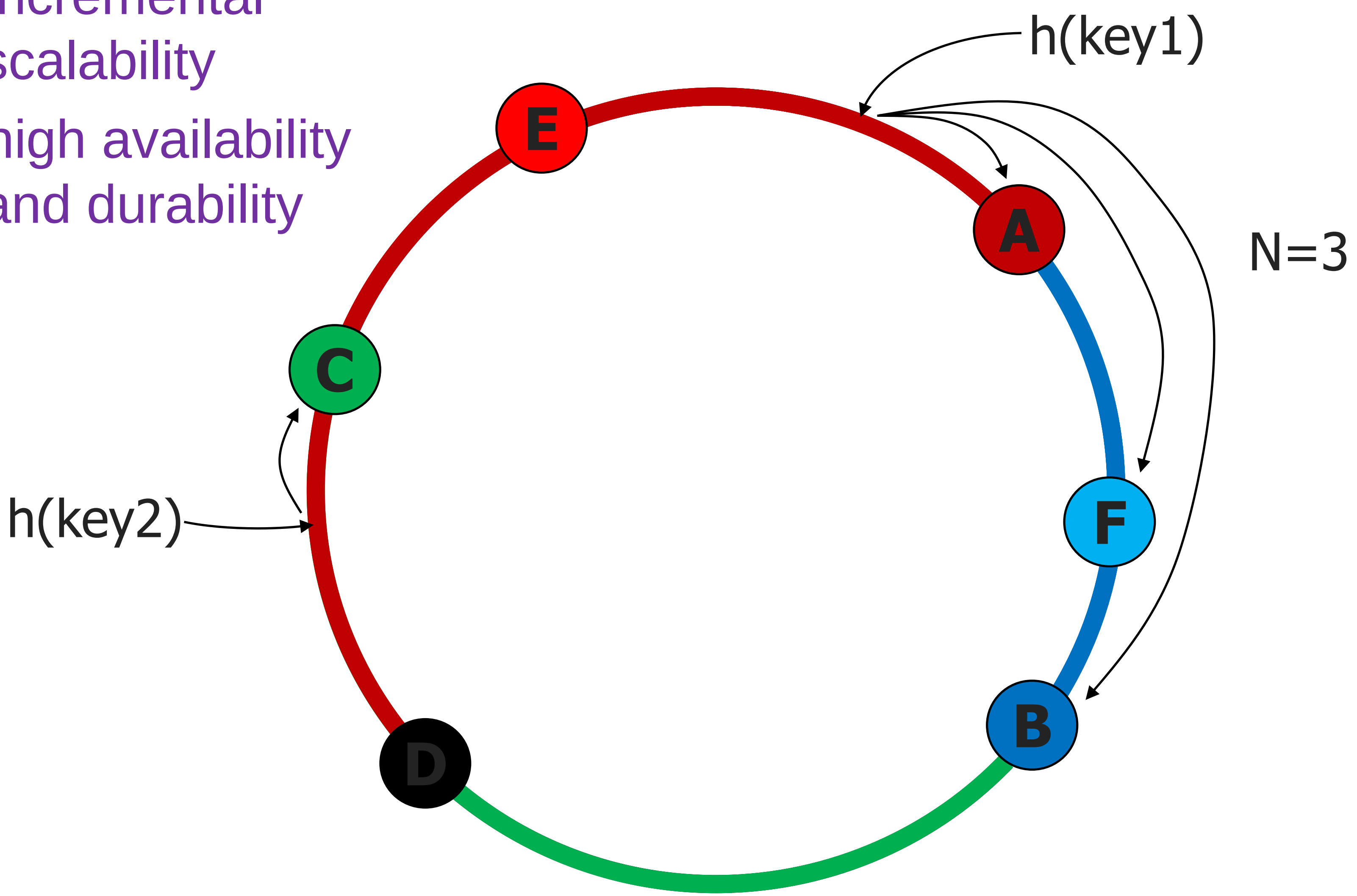
- Parents with good performance tend to contain some good partial solutions (schemas).
- Due to the stochastic selection, such parents are likely to produce more offspring than those with below average performance.
- Over successive iterations (generations), the number of good schemas represented in the pool tends to increase, and the number of bad schemas tends to decrease.
- The average performance of the pool tends to improve.

Challenges

- Effects of pool size and mutation rate on runtime and performance
- Effects of different crossover methods and stopping rules
- Compare performance with alternate solutions

Cassandra's Partitioning and Replication

- incremental scalability
- high availability and durability



Cassandra's Partitioning

- Nodes are logically structured in Ring Topology
- Each node is assigned a random value representing its position on the ring.
- Each data item identified by a key is assigned to a node by hashing its key to yield its position on the ring, and then walking the ring clockwise to find the first node with a position larger than the item's position. This node is deemed the coordinator for this key.
- Thus, each node is responsible for the region on the ring between it and its predecessor node on the ring.

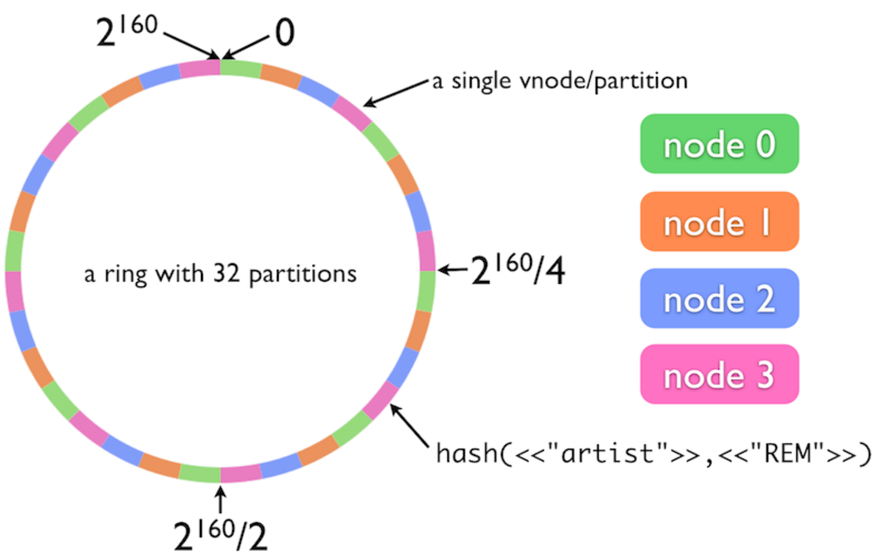
Consistent Hashing

- Advantages
  - Departure or arrival of a node only affects its immediate neighbors and other nodes remain unaffected

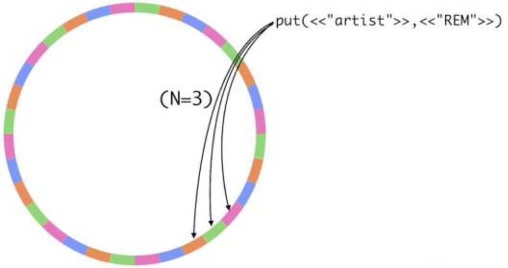
- Challenges
  - The random position assignment of each node on the ring leads to non-uniform data and load distribution
  - Oblivious to the heterogeneity in the performance of nodes

Two Solutions to Address the Limitations

1. Analyze load information on the ring and have lightly loaded nodes move on the ring to alleviate heavily loaded nodes
2. Assign nodes to multiple positions in the circle



Cassandra's Replication



- Each data item is replicated at N (replication factor) nodes.
- Different Replication Policies
  - Rack Unaware – replicate data at N-1 successive nodes after its coordinator
  - Rack Aware – uses a system called 'Zookeeper' to choose a leader node, which tells nodes the range they are replicas for
  - Datacenter Aware – similar to Rack Aware but leader is chosen at Data center level instead of Rack level.

Note: what is Zookeeper?

- Apache ZooKeeper is a service used by a cluster (group of nodes) to coordinate between themselves and maintain shared data with robust synchronization techniques.
- Common services provided include
  - **Naming service** − Identifying the nodes in a cluster by name.
  - **Configuration management** − Latest and up-to-date configuration information of the system for a joining node.
  - **Cluster management** − Joining / leaving of a node in a cluster and node status at real time.
  - **Leader election** − Electing a node as leader for coordination purpose.
  - **Locking and synchronization service** − Locking the data while modifying it.
  - **Highly reliable data registry** − Availability of data even when one or a few nodes are down.

Question & Answer