Distributed DB Design

Distributed DB Design

Chapter 5
Ozsu & Valduriez

- Multi-DBs (or bottom-up)
  - No design issues!
- Top-down approach
  - Given a DB (and workload), how to split and allocate to sites
  - Two design issues
    - Fragmentation
    - Allocation
  - Issues not independent but will cover separately

Example

Employee relation E (#,name,loc,sal,...)

<u>40% of queries:</u>

Qa: select *
from E
where loc=Sa
and...

<u>40% of queries:</u>

Qb: select *
from E
where loc=Sb
and ...

Motivation: Two sites: Sa, Sb

Qa → $\boxed{\text{Sa}}$ $\boxed{\text{Sb}}$ ← Qb

- It does not take a rocket scientist to figure out fragmentation...

E

#	NM	Loc	Sal
5	Joe	Sa	10
7	Sally	Sb	25
8	Tom	Sa	15
:			:

$\mathfrak{F}$

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
:			

At Sa

#	NM	Loc	Sal
7	Sally	Sb	25
:			

At Sb

$\mathfrak{F} = \{ F_1, F_2 \}$

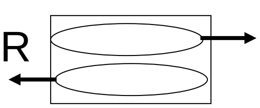
$F_1 = \sigma_{\, loc=Sa}\, E \qquad F_2 = \sigma_{\, loc=Sb}\, E$

$\Rightarrow$ called *primary horizontal* fragmentation

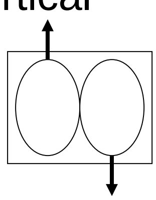
$\Rightarrow$ fragments are expressed in R.A

Fragmentation

- Horizontal

R 

- Vertical

R 

Primary

depends on local attributes

Derived

depends on foreign relation

Which are good fragmentations?

Example 1:

F = { F1, F2 }

F1 = $\sigma_{sal<10}$ E F2 = $\sigma_{sal>20}$ E

Problem: Some tuples lost!

Example 2:

F = { F3, F4 }

F3 = $\sigma_{sal<10}$ E F4 = $\sigma_{sal>5}$ E

Tuples with 5 < sal < 10 are duplicated...

$\Rightarrow$ Prefer to deal with replication explicitly

Example: $\mathfrak{F} = \{ F_5, F_6, F_7 \}$

$F_5 = \sigma_{sal \leq 5}$ E $F_6 = \sigma_{5 < sal < 10}$ E

$F_7 = \sigma_{sal \geq 10}$ E

☞ Then replicate F_6 if convenient (part of allocation problem)

Desired properties for horizontal fragmentation

$R \Rightarrow \mathfrak{F} = \{ F_1, F_2, \ldots \}$

(1) Completeness

$\forall t \in R,\ \exists F_i \in \mathfrak{F}$ such that $t \in F_i$

(2) Disjointness

$\forall t \in F_i,\ \neg\exists F_j$ such that

$t \in F_j,\ i \neq j,\ F_i, F_j \in \mathfrak{F}$

(3) Reconstruction – $\cup F_i$

E

#	NM	Loc	Sal
5	Joe	Sa	10
7	Sally	Sb	25
8	Tom	Sa	15
:			:

$\mathfrak{F}$

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
:			

At Sa

#	NM	Loc	Sal
7	Sally	Sb	25
:			

At Sb

How many fragments to generate? How to decide?

How do we get completeness and disjointness?

(1) Check it "manually"!

e.g., $F_1 = \sigma_{sal<10} E$; $F_2 = \sigma_{sal \geq 10} E$

(2) "Automatically" generate fragments with these properties

Desired simple predicates $\Rightarrow$ Fragments

Example of generation

- Say queries use predicates:
  $A<10$, $A>5$, $Loc = S_A$, $Loc = S_B$
- Assumption: Only 2 locations
- Next: - generate "minterm" predicates
  - eliminate useless ones

5 < A < 10

Minterm predicates (part I)

~~(1) $A<10 \wedge A>5 \wedge Loc=S_A \wedge Loc=S_B$~~

(2) $A<10 \wedge A>5 \wedge Loc=S_A \wedge$ ~~$\neg(Loc=S_B)$~~

(3) $A<10 \wedge A>5 \wedge$ ~~$\neg(Loc=S_A)$~~ $\wedge$ $Loc=S_B$

~~(4) $A<10 \wedge A>5 \wedge \neg(Loc=S_A) \wedge \neg(Loc=S_B)$~~

~~(5) $A<10 \wedge \neg(A>5) \wedge Loc=S_A \wedge Loc=S_B$~~

(6) $A<10 \wedge \neg(A>5) \wedge Loc=S_A \wedge$ ~~$\neg(Loc=S_B)$~~

(7) $A<10 \wedge \neg(A>5) \wedge$ ~~$\neg(Loc=S_A)$~~ $\wedge$ $Loc=S_B$

~~(8) $A<10 \wedge \neg(A>5) \wedge \neg(Loc=S_A) \wedge \neg(Loc=S_B)$~~

A ≤ 5

Minterm predicates (part II)

~~(9) $\neg(A<10) \wedge A>5 \wedge Loc=S_A \wedge Loc=S_B$~~

(10) $\neg(A<10) \wedge A>5 \wedge Loc=S_A \wedge$ ~~$\neg(Loc=S_B)$~~

(11) $\neg(A<10) \wedge A>5 \wedge$ ~~$\neg(Loc=S_A)$~~ $\wedge$ $Loc=S_B$

~~(12) $\neg(A<10) \wedge A>5 \wedge \neg(Loc=S_A) \wedge \neg(Loc=S_B)$~~

~~(13) $\neg(A<10) \wedge \neg(A>5) \wedge Loc=S_A \wedge Loc=S_B$~~

~~(14) $\neg(A<10) \wedge \neg(A>5) \wedge Loc=S_A \wedge \neg(Loc=S_B)$~~

~~(15) $\neg(A<10) \wedge \neg(A>5) \wedge \neg(Loc=S_A) \wedge Loc=S_B$~~

~~(16) $\neg(A<10) \wedge \neg(A>5) \wedge \neg(Loc=S_A) \wedge \neg(Loc=S_B)$~~

A ≥ 10

Final fragments:

F_2: $5 < A < 10 \wedge Loc=S_A$
F_3: $5 < A < 10 \wedge Loc=S_B$
F_6: $A \le 5 \wedge Loc=S_A$
F_7: $A \le 5 \wedge Loc=S_B$
F_{10}: $A \ge 10 \wedge Loc=S_A$
F_{11}: $A \ge 10 \wedge Loc=S_B$

Note: elimination of useless fragments depends on application semantics:

e.g.: if LOC could be $\neq S_A$, $\neq S_B$,
we need to add fragments

F_4: $5 < A < 10 \wedge Loc \neq S_A \wedge Loc \neq S_B$
F_8: $A \le 5 \wedge Loc \neq S_A \wedge Loc \neq S_B$
F_{12}: $A \ge 10 \wedge Loc \neq S_A \wedge Loc \neq S_B$

Summary

- Given simple predicates $P_r = \{ p_1, p_2, .. p_m \}$ minterm predicates are

$$M=\{m \mid m = \bigwedge_{p_k \in P_r} p_k^*, \ 1 \leq k \leq m \}$$

where p_k^* is p_k or is $\neg\, p_k$

- Fragments $\sigma_m R$ for all $m \in M$ are complete and disjoint

Which simple predicates should we use in Pr?

⇔ Desired property of Pr:

- minimality
- completeness

different from COMPLETENESS of fragmentation!

Return to example:

E(#, NM, LOC, SAL,...)

Common queries:

```
Qa: select *            Qb: select *
      from E                from E
      where LOC=Sa          where LOC=Sb
      and ...               and ...
```


Three choices:

(1) Pr = { } $\mathfrak{F}_1$ ={ E }

(2) Pr = {LOC=Sa, LOC=Sb}

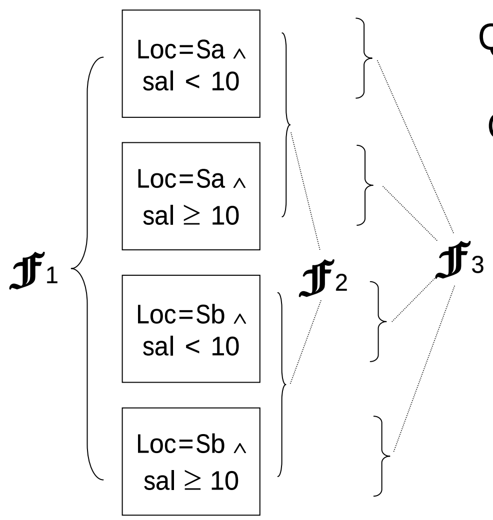
$\mathfrak{F}_2$={ $\sigma_{loc=Sa}$ E, $\sigma_{loc=Sb}$ E }

(3) Pr = {LOC=Sa, LOC=Sb, Sal<10}

$\mathfrak{F}_3$={ $\sigma_{loc=Sa \wedge sal<10}$ E, $\sigma_{loc=Sa \wedge sal\geq 10}$ E,

$\sigma_{loc=Sb \wedge sal<10}$E, $\sigma_{loc=Sb \wedge sal\geq 10}$ E }

In other words:



Q_a: Select ... loc = S_a ...

Q_b: Select ... loc = S_b ...

↓

$\mathfrak{F}_2$ is good...
(not $\mathfrak{F}_1$, $\mathfrak{F}_3$)

Informal definition

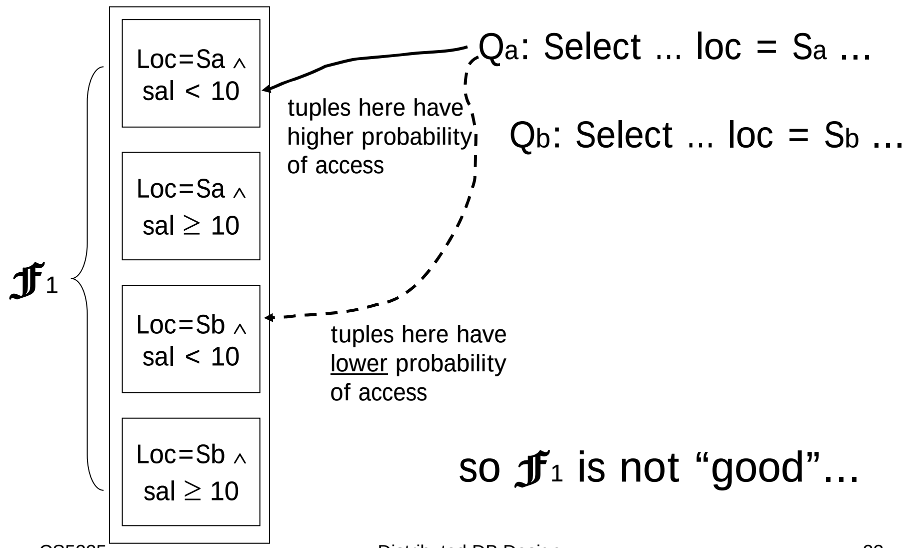
Set of predicates Pr is *complete*

if for every $F_i \in \mathfrak{F}[P_r]$,

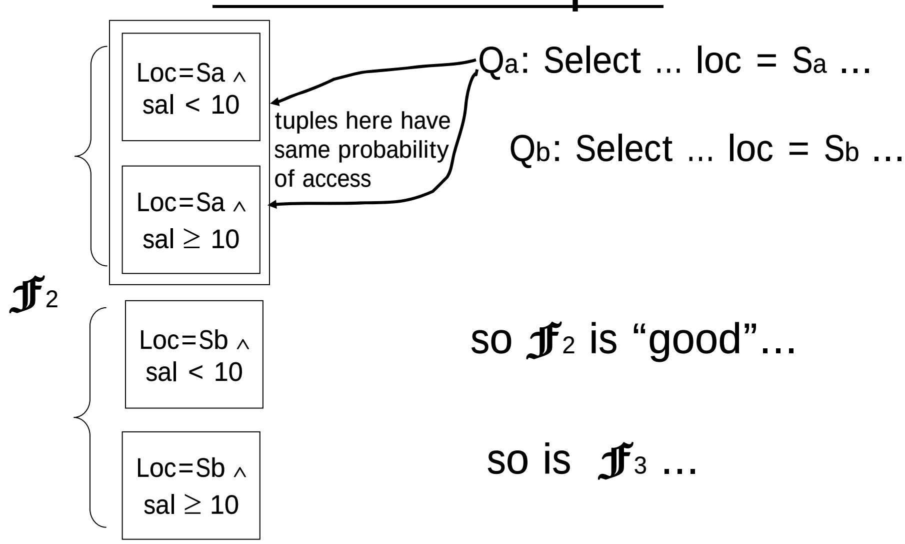
every $t \in F_i$ has equal probability of access by every (*major*) application.

Note: $\mathfrak{F}[P_r]$ is fragmentation defined by minterm predicates generated by P_r.

Back to example:



Back to example:



Is P_r complete a good thing?

Informal definition

Set of predicates P_r is *minimal* if no

$P_r' \subset P_r$ is complete

Back to example:

complete?

(1) Pr = { } ✗

(2) Pr = {LOC=Sa, LOC=Sb} ✓

(3) Pr = {LOC=Sa, LOC=Sb, Sal<10} ✓

Pr(2) is a subset of Pr(3), so Pr(3) is not minimal...

- How do we get complete and minimal Pr?

Answer: use predicates that are “relevant” in frequent queries

Example: Qa:

Select *
from E
where LOC=Sa and
SAL > input parameter

Derived horizontal fragmentation

Example:

E(#, NM, SAL, LOC)

Fragmentation of E: $\mathfrak{F}$={ E_1, E_2} by LOC

J(#, DES,...)

Common query for project:

Given employee name,

list projects (s)he works in

$$E \bowtie J = (E_1 \cup E_2) \bowtie J$$
$$= (E_1 \bowtie J) \cup (E_2 \bowtie J)$$

E_1

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
…			

(at S_a)

E_2

#	NM	Loc	Sal
7	Sally	Sb	25
12	Fred	Sb	15
…			

(at S_b)

How to fragment J? On which attribute? How to do it? (by looking at J alone?)

J

#	Description
5	project 1
7	project 2
5	project 3
12	project 4
…	

E_1

#	NM	Loc	Sal
5	Joe	Sa	10
8	Tom	Sa	15
…			

(at S_a)

E_2

#	NM	Loc	Sal
7	Sally	Sb	25
12	Fred	Sb	15
…			

(at S_b)

J_1

#	Des
5	project 1
5	project 3
…	

J_2

#	Des
7	project 2
12	project 4
…	

$J_1 = J \ltimes E_1$

$= \Pi_J (J \bowtie E_1)$

$J_2 = J \ltimes E_2$

Semijoin: $\ltimes$

Derived horizontal fragmentation

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

$\Downarrow$

S, $\mathfrak{D} = \{D_1, D_2, \ldots D_n\}$ where $D_i = S \ltimes F_i$

$\mathfrak{F}$ could be primary or derived

Convention: R is owner

S is member

Checking completeness and disjointness of derived fragmentation

Example: Say J is:

#	Des
…	
33	project 33
…	

☛ But no # = 33 in E_1 or in E_2!

This J tuple will not be in J_1 or J_2
Fragmentation not complete

To get completeness …

- Need to enforce referential integrity constraint:

join attr(#) of member relation

$\Downarrow$ ($\subseteq$)

join attr(#) of owner relation

Example:

E1

#	NM	Loc	Sal
5	Joe	Sa	10
…			

E2

#	NM	Loc	Sal
5	Fred	Sb	20
…			

J

#	Description
5	project 5
…	

Fragmentation is not disjoint!

J1

#	Description
5	project 5
…	

J2

#	Description
5	project 5
…	

To get disjointness …

- Join attribute(#) should be key of owner relation

Summary: horizontal fragmentation

- Type: primary, derived
- Properties: completeness, disjointness
- Predicates: minimal, complete

Vertical fragmentation (When?)

Example:

E

#	NM	Loc	Sal
5	Joe	Sa	10
7	Sally	Sb	25
8	Fred	Sa	15
…			

E_1

#	NM	Loc
5	Joe	Sa
7	Sally	Sb
8	Fred	Sa
…		

E_2

#	Sal
5	10
7	25
8	15
…	

$R[T] \Rightarrow \quad R_1[T_1] \quad T_i \subseteq T$

$\vdots$

$R_n[T_n]$

Just like normalization of relations

Properties: $R[T] \Rightarrow \quad R_i[T_i]$

(1) Completeness

$$\bigcup_{\text{all } i} T_i = T$$

(2) Disjointness

$T_i \cap T_j = \emptyset$ for all i,j $i \neq j$

$$E(\#,LOC,SAL) \rightarrow E_1(\#,LOC), \quad E(\#,LOC,SAL) \rightarrow E_2(SAL)$$

Not a desirable property!!
(could not reconstruct R!)

(3) Lossless join

$$\underset{\text{all } i}{\bowtie} R_i = R$$

☛ One way to achieve lossless join:
Repeat key in all fragments, i.e.,
Key $\subseteq T_i$ for all i

How do we decide what attributes are grouped with which?

Example:

E(#,NM,LOC,SAL) → ?

→ E_1(#,NM,LOC), E_2(#,SAL)

→ E_1(#,NM), E_2(#,LOC), E_3(#,SAL)

Attribute affinity matrix

	A_1	A_2	A_3	A_4	A_5
A_1	-	-	-	-	-
A_2	50	-	-	-	-
A_3	45	48	-	-	-
A_4	1	2	0	-	-
A_5	0	0	4	75	-

Aff(Ai, Aj) captures the access frequencies of applications for attributes Ai and Aj

R_1[K,A_1,A_2,A_3] R_2[K,A_4,A_5]

- Ref 1 (Ozsu & Valduriez, Chap 5) discusses
  - How to build affinity matrix (What about 3 or higher dimensions?)
  - How to identify attribute clusters
  - How to partition relation
- You are not responsible for
  - Clustering and partitioning algorithms
    (i.e., Skip pages 135-145)

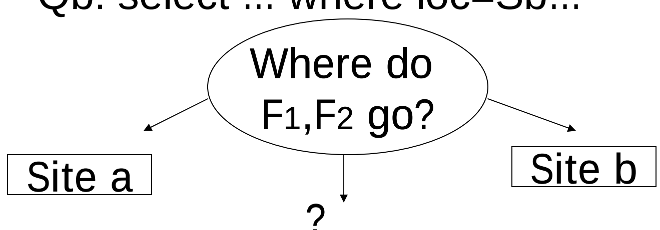
Allocation

Example: E(#,NM,LOC,SAL) $\Rightarrow$

$F_1 = \sigma_{loc=Sa}\, E$; $F_2 = \sigma_{loc=Sb}\, E$

Qa: select ... where loc=Sa...

Qb: select ... where loc=Sb...



Many Issues ...

- Where do queries originate
- What is communication cost?
  and size of answers, relations,...?
- What is storage capacity, cost at sites?
  and size of fragments?
- What is processing power at sites?
- What is query processing strategy?
  - How are joins done?
  - Where are answers collected?
- Do we replicate fragments?
  - Cost of updating copies?
  - Writes and concurrency control?

Optimization problem

- What is best placement of fragments and/or best number of copies to:
  - minimize query response time
  - maximize throughput
  - minimize "some cost"
  - ...
- Subject to constraints?
  - Available storage
  - Available bandwidth, power,...
  - Keep 90% of response time below X
  - ...

This is an incredibly hard problem!

Example: Single fragment F

Read cost: $\sum_{i=1}^{m} [t_i \times \underset{j}{\text{MIN}}\ C_{ij}]$

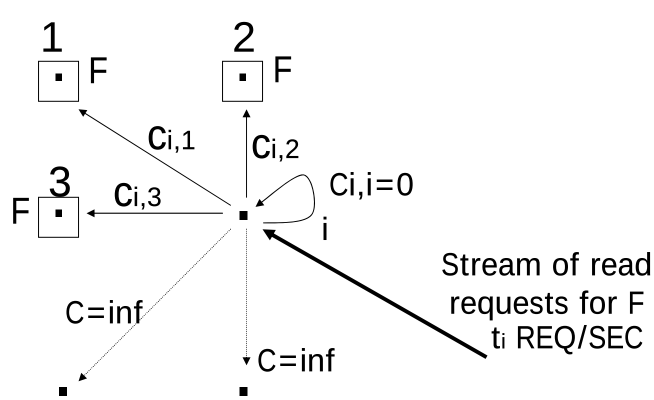
i: Originating site of request

t_i: Read traffic at S_i

C_{ij}: Retrieval cost

Accessing fragment F at Sj from Si

Scenario - Read cost



Write cost

$$\sum_{i=1}^{m} \sum_{j=1}^{m} X_j \, u_i \, C'_{ij}$$

i: Originating site of request

j: Site being updated

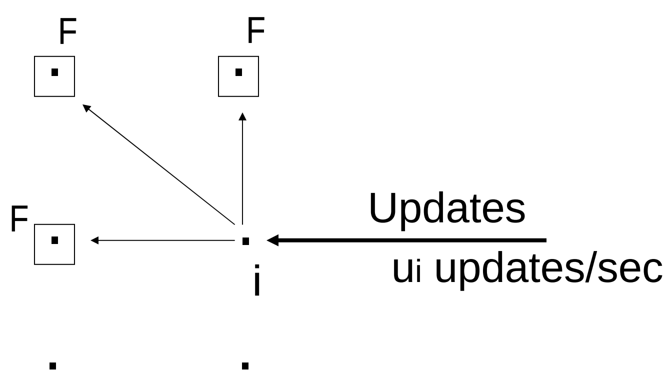
X_j: $\begin{cases} 0 \text{ if F not stored at } S_j \\ 1 \text{ if F stored at } S_j \end{cases}$

u_i: Write traffic at S_i

C'_{ij}: Write cost

Updating F at Sj from Si

Scenario - write cost



Storage cost:

$$\sum_{i=1}^{m} X_i \, d_i$$

X_i: $\begin{cases} 0 \text{ if F not stored at } S_i \\ 1 \text{ if F stored at } S_i \end{cases}$

d_i: storage cost at S_i

Target function

$$\min \left\{ \sum_{i=1}^{m} [t_i \times \underset{j}{\mathrm{MIN}}\, C_{ij} + \sum_{j=1}^{m} X_j \times u_i \times C'_{ij}] + \sum_{i=1}^{m} X_i \times d_i \right\}$$

Can add more complications:

Examples:

- Multiple fragments
- Fragment sizes
- Concurrency control cost

Example

Assumptions:

- The cost of storing fragments at sites is negligible.
- The cost of reading a non-local fragment is 1 unit and the cost of writing a non-local fragment is 2 units.
- The cost of reading or writing a local fragment is 0 unit.

- *app1* reads fragment F1 5 times and reads fragment F2 5 times.
- *app2* reads fragment F2 5 times and reads fragment F3 5 times.
- *app3* reads fragment F2 10 times and writes fragment F2 10 times.

- *app1* issues at site 1, *app2* issues at site 2, and *app3* issues at site 3.

  Which of the following three allocations of fragments minimizes total cost?
  - 1: F1 at site 1; F2 at site 2; F3 at site 3
  - 2: F1 and F2 at site 1; F2 and F3 at site 3.
  - 3: F1 at site 1, F3 at site 2, F2 at site 3.

Example (Cont)

Allocation-1: F1 at site 1; F2 at site 2; F3 at site 3

App1 5 non-local reads cost 5
App2 5 non-local read cost 5
App3 10 non-local reads cost 10, non-local writes cost 20.

Total cost is 40.

Allocation-2: F1 and F2 at site 1; F2 and F3 at site 3.
App1 costs 0
App2 10 non-local reads cost 10
App3 10 non-local writes cost 20

Total cost is 30.

Allocation-3: Total is 10.

“Best=fit” strategy for non-replicated allocation

- Place the fragment *i* at the site *j** where the total costs/savings to *i* is minimum/maximum
- What would be the allocation result (of our example) if we use the “best-fit” method?
- The number of total references for fragment F2 is 5+5+10+10x2 = 40.
  - When allocating F2 to sites 1, 2, and 3, the total cost will be 35, 35, and 10, respectively.
  - Correspondingly, the total saving will be 5, 5, and 30. Under “best-fit”, F2 is placed at site **3**.

Summary

- Description of fragmentation
- Good fragmentations
- Design of fragmentation
- Allocation