Distributed DB Design

Top-down approach:
- have DB...
- how to split and allocate the sites

Multi-DBs (or bottom-up): no design issues!

Two issues in DDB design:

- Fragmentation
- Allocation

Note: issues not independent, but will cover separately

Example

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

40% of queries:
Qa: select * from E
where loc=Sa and ...

40% of queries:
Qb: select * from E
where loc=Sb and ...

Motivation: Two sites: Sa, Sb

Qa → Sa
Qb ← Sb

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

- Horizontal
  - Primary: depends on local attributes
  - Derived: depends on foreign relation

- Vertical

Fragmentation also called Sharding

Three common horizontal partitioning techniques
- Round robin
- Hash partitioning
- Range partitioning
**Round robin**

- Evenly distributes data
- Good for scanning full relation
- Not good for point or range queries

**Hash partitioning**

- Good for point queries on key; also for joins
- Not good for range queries; point queries not on key
- If hash function good, even distribution

**Range partitioning**

- Good for some range queries on A
- Need to select good vector: else unbalance
  - Data skew
  - Execution skew

**Which are good fragmentations?**

**Example:**

\[
\mathcal{F} = \{ F_1, F_2 \}
\]

- \( F_1 = \sigma_{sal < 10} E \)
- \( F_2 = \sigma_{sal > 20} E \)

- Problem: Some tuples lost!
Which are good fragmentations?

Second example:

\[ \mathcal{F} = \{ F_3, F_4 \} \]

\[ F_3 = \sigma_{\text{sal}<10} \text{E} \quad F_4 = \sigma_{\text{sal}>5} \text{E} \]

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

⇒ Prefer to deal with replication explicitly

Example: \[ \mathcal{F} = \{ F_5, F_6, F_7 \} \]

\[ F_5 = \sigma_{\text{sal} \leq 5} \text{E} \quad F_6 = \sigma_{5<\text{sal}<10} \text{E} \quad F_7 = \sigma_{\text{sal} \geq 10} \text{E} \]

- Then replicate \( F_6 \) if convenient

(part of allocation problem)

Desired properties for horizontal fragmentation

\[ R \Rightarrow \mathcal{F} = \{ F_1, F_2, \ldots \} \]

(1) Completeness

\[ \forall t \in R, \exists F_i \in \mathcal{F} \text{ such that } t \in F_i \]

(2) Disjointness

\[ \forall t \in F_i, \not\exists F_j \text{ such that } t \in F_j, \ i \neq j, \ F_i, F_j \in \mathcal{F} \]

(3) Reconstruction - ignore

How do we get completeness and disjointness?

(1) Check it “manually”!

- e.g., \( F_1 = \sigma_{\text{sal}<10} \text{E} \); \( F_2 = \sigma_{\text{sal}>10} \text{E} \)

How do we get completeness and disjointness?

(2) “Automatically” generate fragments with these properties

Desired simple predicates ⇒ Fragments
Example of generation

- Say queries use predicates:
  \[ A < 10, \ A > 5, \ \text{Loc} = S_A, \ \text{Loc} = S_B \]
- Next: - generate "minterm" predicates
  - eliminate useless ones

Minterm predicates (part I)

1. \[ A < 10 \land A > 5 \land \text{Loc} = S_A \land \text{Loc} = S_B \]
2. \[ A < 10 \land A > 5 \land \text{Loc} = S_A \land \neg(\text{Loc} = S_B) \]
3. \[ A < 10 \land A > 5 \land \neg(\text{Loc} = S_A) \land \text{Loc} = S_B \]
4. \[ A < 10 \land A > 5 \land \neg(\text{Loc} = S_A) \land \neg(\text{Loc} = S_B) \]
5. \[ A < 10 \land \neg(A > 5) \land \text{Loc} = S_A \land \text{Loc} = S_B \]
6. \[ A < 10 \land \neg(A > 5) \land \text{Loc} = S_A \land \neg(\text{Loc} = S_B) \]
7. \[ A < 10 \land \neg(A > 5) \land \neg(\text{Loc} = S_A) \land \text{Loc} = S_B \]
8. \[ A < 10 \land \neg(A > 5) \land \neg(\text{Loc} = S_A) \land \neg(\text{Loc} = S_B) \]

Minterm predicates (part II)

9. \[ \neg(A < 10) \land A > 5 \land \text{Loc} = S_A \land \text{Loc} = S_B \]
10. \[ \neg(A < 10) \land A > 5 \land \text{Loc} = S_A \land \neg(\text{Loc} = S_B) \]
11. \[ \neg(A < 10) \land A > 5 \land \neg(\text{Loc} = S_A) \land \text{Loc} = S_B \]
12. \[ \neg(A < 10) \land A > 5 \land \neg(\text{Loc} = S_A) \land \neg(\text{Loc} = S_B) \]
13. \[ \neg(A < 10) \land \neg(A > 5) \land \text{Loc} = S_A \land \text{Loc} = S_B \]
14. \[ \neg(A < 10) \land \neg(A > 5) \land \text{Loc} = S_A \land \neg(\text{Loc} = S_B) \]
15. \[ \neg(A < 10) \land \neg(A > 5) \land \neg(\text{Loc} = S_A) \land \text{Loc} = S_B \]
16. \[ \neg(A < 10) \land \neg(A > 5) \land \neg(\text{Loc} = S_A) \land \neg(\text{Loc} = S_B) \]
Final fragments:
F2: $5 < A < 10 \land \text{Loc}=\text{SA}$
F3: $5 < A < 10 \land \text{Loc}=\text{SB}$
F6: $A \leq 5 \land \text{Loc}=\text{SA}$
F7: $A \leq 5 \land \text{Loc}=\text{SB}$
F10: $A \geq 10 \land \text{Loc}=\text{SA}$
F11: $A \geq 10 \land \text{Loc}=\text{SB}$

Note: elimination of useless fragments depends on application semantics:
e.g.: if LOC could be $\neq \text{SA}, \neq \text{SB}$,
we need to add fragments
F4: $5 < A < 10 \land \text{Loc} \neq \text{SA} \land \text{Loc} \neq \text{SB}$
F8: $A \leq 5 \land \text{Loc} \neq \text{SA} \land \text{Loc} \neq \text{SB}$
F12: $A \geq 10 \land \text{Loc} \neq \text{SA} \land \text{Loc} \neq \text{SB}$

Why does this work?
Predicates: $p_1 \land p_2 \land p_3 \land p_4$
$\vdash p_1 \land p_2 \land p_3 \land \neg p_4$
$
eg p_1 \land \neg p_2 \land \neg p_3 \land \neg p_4$

(1) Completeness: Take $t \in R$
p(t) must be T or F!
Say $p_1(t) = T, p_2(t) = T, p_3(t) = F, p_4(t) = F$
Then $t$ is in fragment with predicate
$p_1 \land p_2 \land \neg p_3 \land \neg p_4$

(2) Disjointness
Say $t \in \text{Fragment } p_1 \land p_2 \land \neg p_3 \land \neg p_4$
Then:
$p_1(t) = T, p_2(t) = T, p_3(t) = F, p_4(t) = F$
$\Rightarrow t$ cannot be in any other fragment!

Summary
• Given simple predicates $P_r = \{ p_1, p_2, \ldots, p_m \}$
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
Another Desired Fragmentation Property: 
Match Access Patterns

frequently accessed together

- data A
- data B
- data C

try to place in same fragment

Return to example:

E(#, NM, LOC, SAL,...)
Common queries:

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

Three choices:

(1) Pr = { } \(F_1 = \{ E \}\)
(2) Pr = {LOC=Sa, LOC=Sb} \(F_2 = \{ \sigma_{loc=Sa} E, \sigma_{loc=Sb} E \}\)
(3) Pr = {LOC=Sa, LOC=Sb, Sal<10} \(F_3 = \{ \sigma_{loc=Sa \land sal<10} E, \sigma_{loc=Sa \land sal\geq 10} E, \sigma_{loc=Sb \land sal<10} E, \sigma_{loc=Sb \land sal\geq 10} E \} \)

In other words:

Loc=Sa \land sal < 10
Loc=Sa \land sal \geq 10
Loc=Sb \land sal < 10
Loc=Sb \land sal \geq 10

Qa: Select ... loc = Sa ... 
Qb: Select ... loc = Sb ...

\(F_2\) is good... (not \(F_1, F_3\))

Derived horizontal fragmentation
Example:

E(#, NM, SAL, LOC)
\(\mathcal{F} = \{ E_1, E_2 \}\) by LOC
J(#, DES,...)
Common query for project:

[Given employee name, list projects (s)he works in]
Derived horizontal fragmentation

\[ R, F = \{ F_1, F_2, \ldots, F_n \} \]
\[ S, D = \{ D_1, D_2, \ldots, D_n \} \]

where \( D_i = S \bowtie F_i \)

**Convention:**
- \( R \) is owner
- \( S \) is member

\( F \) could be primary or derived

To get completeness:

Need to enforce referential integrity constraint:
- join attr(\#) of member relation
- joint attr(\#) of owner relation

• Checking completeness and disjointness of derived fragmentation

Example: Say \( J \) is:

\[
\begin{array}{|c|c|}
\hline
\# & Des \\hline
5 & work on 347 hw \hline
7 & go to moon \hline
5 & build table \hline
12 & rest \hline
\end{array}
\]

This \( J \) tuple will not be in \( J_1 \) nor \( J_2 \)

Fragmentation not complete
Join attribute(#) should be key of owner relation

Summary: horizontal fragmentation

- Type: primary, derived
- Properties: completeness, disjointness

Vertical fragmentation

Example:

<table>
<thead>
<tr>
<th></th>
<th>#</th>
<th>NM</th>
<th>Loc</th>
<th>Sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>5</td>
<td>Joe</td>
<td>Sa</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Sally</td>
<td>Sb</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Fred</td>
<td>Sa</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E(#,LOC,SAL)

R[T] \Rightarrow R_i[T_i] \hspace{1cm} Ti \subseteq T

R_n[T_n]

Just like normalization of relations

Properties: \( R[T] \Rightarrow R_i[T_i] \)

1. Completeness

\[ \bigcup_{i} Ti = T \]

2. Disjointness

\[ Ti \cap T_j = \emptyset \text{ for all } i, j \neq i \]

\[ E(#,LOC,SAL) \]

E1(#,LOC)

E2(SAL)
(2) Disjointness
\[ T_i \cap T_j = \emptyset \text{ for all } i, j \quad i \neq j \]
\[ E(\#,LOC,SAL) \]
\[ E_1(\#,LOC) \]
\[ E_2(SAL) \]
Not a desirable property!!
(could not reconstruct R!)

(3) Lossless join
\[ R_i = R \quad \forall i \]
One way to achieve lossless join:
Repeat key in all fragments, i.e.,
Key \subseteq T_i \text{ for all } i

\[ \implies \text{ How do we decide what attributes are grouped with which?} \]
\[ E(\#,NM,LOC) \]
\[ E_1(\#,NM) \]
\[ E_2(\#,LOC) \]
\[ E_3(\#,SAL) \]

Attribute affinity matrix
\[
\begin{array}{ccccc}
A_1 & A_2 & A_3 & A_4 & A_5 \\
A_1 & - & - & - & - & - \\
A_2 & 50 & - & - & - & - \\
A_3 & 45 & 48 & - & - & - \\
A_4 & 1 & 2 & 0 & - & - \\
A_5 & 0 & 0 & 4 & 75 & - \\
\end{array}
\]

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

• Textbook (Ozsu & Valduriez) discusses
  - How to build affinity matrix
  - 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: \( E(\#, NM, LOC, SAL) \implies F_1 = \sigma_{loc=Sa} E ; F_2 = \sigma_{loc=Sb} E \)

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

Where do \( F_1, F_2 \) go?

Site a

Site b

### 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?

### More Issues

- 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: \( \sum_{i=1}^{m} [t_i \times \text{MIN } C_{ij}] \)

- \( i \): Originating site of request
- \( t_i \): Read traffic at \( S_i \)
- \( C_{ij} \): Retrieval cost
  - Accessing fragment F at \( S_j \) from \( S_i \)

---

Scenario - Read cost

![Diagram of read cost scenario.]

Write cost

\[ \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} u_i C'_{ij} \]

- \( i \): Originating site of request
- \( j \): Site being updated
- \( X_{ij} \): 0 if F not stored at \( S_j \)
  - 1 if F stored at \( S_j \)
- \( u_i \): Write traffic at \( S_i \)
- \( C'_{ij} \): Write cost
  - Updating F at \( S_j \) from \( S_i \)

---

Scenario - write cost

![Diagram of write cost scenario.]

Storage cost:

\[ \sum_{i=1}^{m} X_i d_i \]

- \( X_i \): 0 if F not stored at \( S_i \)
  - 1 if F stored at \( S_i \)
- \( d_i \): storage cost at \( S_i \)

---

Target function:

\[
\min \left\{ \sum_{i=1}^{m} [t_i \times \text{MIN } C_{ij}] + \sum_{j=1}^{n} X_j \times u_i \times C'_{ij} \right\} + \sum_{i=1}^{m} X_i \times d_i
\]
Can add more complications:

Examples:
- Multiple fragments
- Fragment sizes
- Concurrency control cost

Case Study: PNUTS
• Where in the World is My Data?
  Sudarshan Kadambi, Jianjun Chen, Brian F. Cooper, David Lomax, Raghu Ramakrishnan, Adam Silberstein, Erwin Tam, Hector Garcia-Molina; VLDB 2011
• Distributed object/tuple store for Yahoo!

Case Study: PNUTS
• Issue: Where to locate data
• Issue: What and where to replicate

PNUTS Discussion
• Dynamic vs Static fragment placement
• Caching vs Replication

Policy Constraints
• MIN_COPIES: The minimum number of full replicas of the record that must exist.
• INCL_LIST: An inclusion list -- the locations where a full replica of the record must exist.
• EXCL_LIST: An exclusion list -- the locations where a full replica of the record cannot exist.

Example Rule
• Rule 1:
  IF TABLE_NAME = "Users"
  THEN
    SET 'MIN_COPIES' = 2
    CONSTRAINT_PRI = 0
Another Example Rule

• Rule 2:
  • IF TABLE_NAME = “Users” AND
    FIELD STR(‘home location’) = ‘France’
    THEN
      SET ‘MIN_COPIES’ = 3 AND
      SET ‘EXCL_LIST’ = ‘USWest, USEast’
      CONSTRAINT PRI = 1

Summary

• Description of fragmentation
• Good fragmentations
• Design of fragmentation
• Allocation