Query Processing

- Decomposition
- Localization
- Optimization

Decomposition

- Same as in centralized system
- Normalization
- Eliminating redundancy
- Algebraic rewriting

Normalization

- Convert from general language to a “standard” form (e.g., Relational Algebra)

Example

Select A, C
From R, S
Where (R.B = 1 and S.D = 2) or (R.C > 3 and S.D = 2)

Also: Detect invalid expressions

E.g.: Select * from R where R.A = 3

R does not have “A” attribute
Eliminate redundancy

E.g.: in conditions:
\[(S.A=1) \land (S.A>5) \Rightarrow \text{False} \]
\[(S.A<10) \land (S.A<5) \Rightarrow S.A<5 \]

E.g.: Common sub-expressions

Algebraic rewriting

E.g.: Push conditions down

\[\sigma_{\text{cond}} \quad \rightarrow \quad \sigma_{\text{cond1}} \quad \sigma_{\text{cond2}} \]

Localization steps

1. Start with query
2. Replace relations by fragments
3. Push \( \cup \): up (use CS245 rules)
   \[\pi, \sigma: \text{down} \]
4. Simplify – eliminate unnecessary operations

After decomposition:
- One or more algebraic query trees on relations
- Localization:
  - Replace relations by corresponding fragments

Notation for fragment

\[[R: \text{cond}]\]

fragment conditions its tuples satisfy
Example A
(1) $\sigma_{E=3}

\mid

R$

(2) $\sigma_{E=3}

\cup

[R_1: E < 10] \cup [R_2: E \geq 10]$

(3) $\sigma_{E=3} \cup \sigma_{E=3}

[R_1: E < 10] \cup [R_2: E \geq 10]$

(3) $\sigma_{E=3} \cup \sigma_{E=3}

[R_1: E < 10] \cup [R_2: E \geq 10]

\Rightarrow \emptyset$

(4) $\sigma_{E=3}

\mid

[R_1: E < 10]$

Rule 1

A $\sigma_{c_1[R: c_2]} \Rightarrow \sigma_{c_1[R: c_1 \land c_2]}$

B $[R: \text{False}] \Rightarrow \emptyset$
In example A:

\[ \sigma_{E=3}[R2: E \geq 10] \Rightarrow \sigma_{E=3}[R2: E=3 \land E \geq 10] \Rightarrow \sigma_{E=3}[R2: \text{False}] \Rightarrow \emptyset \]

Example B

(1) A=common attribute

RS

(2)

\([R1: A<5] \cup [R2: 5 \leq A \leq 10] \cup [R3: A>10]

\([S1: A<5] \cup [S2: A \geq 5] \]

(3)

\([R1: A<5] \cup [R2: 5 \leq A \leq 10] \cup [R3: A>10]

\([S1: A<5] \cup [S2: A \geq 5] \]

(4)

\([R1: A<5] \cup [R2: 5 \leq A \leq 10] \cup [R3: A>10]

\([S1: A<5] \cup [S2: A \geq 5] \]

\([R3: A>10] \cup [S2: A \geq 5] \]
Rule 2

\[ \text{[R: C1]} \bowtie [\text{S: C2} ] \Rightarrow \]
\[ [\text{R} \bowtie \text{S: C1 } \land \text{ C2 } \land \text{ R.A } = \text{ S.A} ] \]

In step 4 of Example B:

\[ \text{[R1: A<5]} \bowtie [\text{S2: A } \geq 5] \]
\[ \Rightarrow [\text{R1}\bowtie\text{S2: R1.A < 5 } \land \text{ S2.A } \geq 5 \land \text{ R1.A } = \text{ S2.A } ] \]
\[ \Rightarrow [\text{R1}\bowtie\text{S2: False}] \Rightarrow \emptyset \]

Localization with derived fragmentation

Example C

(2)

\[ \begin{align*}
\text{R1: } & A<10 \\
\text{R2: } & A \geq 10
\end{align*} \]
\[ \begin{align*}
\text{S1: } & K=R.K \\
\text{S2: } & K=R.K
\end{align*} \]

(3)

\[ \begin{align*}
\text{K} & \\
\text{R1} & [\text{S1}] \\
\text{R1} & [\text{S2}] \\
\text{R2} & [\text{S1}] \\
\text{R2} & [\text{S2}]
\end{align*} \]

In step 4 of Example C:

\[ \text{[R1:A<10]} \bowtie [\text{S2:K=R.K } \land \text{ R.A } \geq 10] \]
\[ \Rightarrow [\text{R1}\bowtie\text{S2: R1.A } < 10 \land \text{ S2.K } = \text{ R.K } \land \text{ R.A } \geq 10 \land \text{ R1.K } = \text{ S2.K} ] \]
\[ \Rightarrow [\text{R1}\bowtie\text{S2: False }] \quad (K \text{ is key of R, R1}) \]
\[ \Rightarrow \emptyset \]
(4)
\[ [R_1: A < 10] \land [S_1: K = R.K \land R.A < 10] \land [R_2: A \geq 10] \land [S_2: K = R.K \land R.A \geq 10] \]

(4) simplified more:
\[ \bigcup \bigwedge \bigcup \bigwedge \bigcup \bigwedge \]

• Localization with vertical fragmentation

Example D

(1) \[ \Pi_A \]
\[ R \]
\[ R_1(K, A, B) \]
\[ R_2(K, C, D) \]

(2) \[ \Pi_A \]
\[ \bigcup \bigwedge \]
\[ R_1 R_2 \]
\[ (K, A, B) \]
\[ (K, C, D) \]

(3) \[ \Pi_A \]
\[ \bigcup \bigwedge \]
\[ \Pi_{K,A} \]
\[ \Pi_{K,A} \]
\[ \bigcup \bigwedge \]
\[ R_1 R_2 \]
\[ (K, A, B) \]
\[ (K, C, D) \]

not really needed

(4) \[ \Pi_A \]
\[ \bigcup \bigwedge \]
\[ \bigcup \bigwedge \]
\[ R_1 \]
\[ (K, A, B) \]

Rule 3
• Given vertical fragmentation of R:
  \[ R_i = \Pi_{A_i}(R), ~ A_i \subseteq A \]
• Then for any \[ B \subseteq A \]:
  \[ \Pi_B(R) = \Pi_B(\bigcup \bigwedge R_i \mid B \cap A_i \neq \emptyset) \]
• Localization with hybrid fragmentation

Example E

\[ R_1 = \sigma_{k<5} [\Pi_{k,A} R] \]

\[ R_2 = \sigma_{k\geq 5} [\Pi_{k,A} R] \]

\[ R_3 = \Pi_{k,B} R \]

Query:

\[ \Pi_{k=3} R_1 \]

Reduced Query:

\[ \Pi_{k=3} R_1 \]

Summary - Query Processing

• Decomposition ✓
• Localization ✓
• Optimization
  – Overview
  – Tricks for joins + other operations
  – Strategies for optimization

Optimization Process:

Generate query plans → P1 → P2 → P3 → Pn
Estimate size of intermediate results → C1 → C2 → C3 → Cn
Estimate cost of plan ($, time, ...) → pick minimum

Differences with centralized optimization:

• New strategies for some operations (semi-join, range-partitioning, sort, ...)
• Many ways to assign and schedule processors
Parallel/distributed sort

Input:
(a) relation R on single site/disk
(b) R fragmented/partitioned by sort attribute
(c) R fragmented/partitioned by other attribute

Output
(a) sorted R on single site/disk
(b) fragments/partitions sorted

Basic sort
- R(K,...), sort on K
- Fragmented on K
  Vector: k₀, k₁, ... kn

Algorithm: each fragment sorted independently
- If necessary, ship results

⇒ Same idea on different architectures:

Shared nothing:

Shared memory:

Range partitioning sort
- R(K,...), sort on K
- R located at one or more site/disk, not fragmented on K
• Algorithm:
  (a) Range partition on K
  (b) Basic sort

  \[ \begin{array}{c}
  \text{Ra} \\
  \text{Rb}
  \end{array} \quad \begin{array}{c}
  \overset{R_1}{\text{Local sort}} \\
  \overset{R_2}{\text{Local sort}} \\
  \overset{R_3}{\text{Local sort}}
  \end{array} \quad \begin{array}{c}
  \overset{R'}{\text{Result}} \\
  \end{array} \]

• Selecting a good partition vector

  \[ \begin{array}{c}
  7 \quad \ldots \\
  10 \quad \ldots \\
  31 \quad \ldots \\
  12 \quad \ldots \\
  8 \\
  15 \\
  11 \\
  32 \\
  17 \\
  \end{array} \]

  \[ \begin{array}{c}
  \text{Ra} \\
  \text{Rb} \\
  \text{Rc}
  \end{array} \]

Example

• Each site sends to coordinator:
  - Min sort key
  - Max sort key
  - Number of tuples

• Coordinator computes vector and distributes to sites
  (also decides # of sites for local sorts)

Sample scenario:

Coordinator receives:
- \( \text{SA}: \text{Min}=5 \quad \text{Max}=10 \quad \# = 10 \text{ tuples} \)
- \( \text{SB}: \text{Min}=7 \quad \text{Max}=17 \quad \# = 10 \text{ tuples} \)

Expected tuples:

\[ \begin{array}{c}
  2 \\
  1 \\
  \text{ko?}
  \end{array} \]

[assuming we want to sort at 2 sites]
Expected tuples with key < \(k_o\) = \(\frac{\text{Total tuples}}{2}\)

\[2(k_o - 5) + (k_o - 7) = 10 \]

\[3k_o = 10 + 10 + 7 = 27\]

\(k_o = 9\)

Variations

- Send more info to coordinator
  - Partition vector for local site
    - Histogram

\[\begin{array}{cccc}
5 & 6 & 7 & 8 & 9 & 10 \\
\hline
3 & 3 & \text{# tuples} & 10 & \text{local vector} \\
\end{array}\]

More than one round

E.g.: (1) Sites send range and # tuples
  - (2) Coordinator returns “preliminary” vector \(V_0\)
  - (3) Sites tell coordinator how many tuples in each \(V_0\) range
  - (4) Coordinator computes final vector \(V_f\)

Can you come up with a distributed algorithm?

(no coordinator)

Parallel external sort-merge

- Same as range-partition sort, except sort first

Note: can use merging network if available (e.g., Teradata)
• Parallel/distributed Join

Input: Relations R, S
      May or may not be partitioned

Output: R \bowtie S
         Result at one or more sites

Notes:
• Same partition function f is used for both R and S (applied to join attribute)
• f can be range or hash partitioning
• Local join can be of any type (use any CS245 optimization)
• Various scheduling options e.g.,
  (a) partition R; partition S; join
  (b) partition R; build local hash table for R; partition S and join

More notes:
• We already know why part-join works:

Even more notes:
• Selecting good partition function f very important:
  - Number of fragments
  - Hash function
  - Partition vector

• Good partition vector
  - Goal: | R_1 | + | S_1 | the same
  - Can use coordinator to select
Asymmetric fragment + replicate join

Notes:
- Can use any partition function f for R (even round robin)
- Can do any join — not just equi-join
e.g.: $R \bowtie_S S$

General fragment and replicate join

Notes:
- Asymmetric F+R join is special case of general F+R
- Asymmetric F+R may be good if S small
- Works for non-equi-joins

S is partitioned in similar fashion

Semi-join
- Goal: reduce communication traffic
- $R \bowtie_A S \Rightarrow (R \bowtie_A S) \bowtie_A S$ or
- $R \bowtie_A (S \bowtie_A R)$ or
- $(R \bowtie_A S) \bowtie_A (S \bowtie_A R)$
Example: $R \bowtie S$

$T = 4|A| + 2|A+C| + \text{result}$

Computing transmitted data in example:

- with semi-join $R \bowtie (S \bowtie R)$:
  $T = 4|A| + 2|A+C| + \text{result}$

- with join $R \bowtie S$:
  $T = 4|A+B| + \text{result}$

In general:

- Say $R$ is smaller relation
- $(R \bowtie S) \bowtie S$ better than $R \bowtie S$ if
  $\text{size } (R \bowtie S) + \text{size } (R \bowtie S) < \text{size } (R)$

better if say $|B|$ is large
• Similar comparisons for other semi-joins
• Remember: only taking into account transmission cost

• Trick:
  Encode \( \Pi_A S \) (or \( \Pi_A R \)) as a bit vector
  key in \( S \):
  \[
  0 0 1 1 0 1 0 0 0 0 1 0 1 0 0
  
  \]
  <---one bit/possible key----->

---

Three way joins with semi-joins

Goal: \( R \bowtie S \bowtie T \)

Option 1: \( R' \bowtie S' \bowtie T \)
  \( R' = R \leftarrow S; \ S' = S \leftarrow T \)

Option 2: \( R'' \bowtie S' \bowtie T \)
  \( R'' = R \leftarrow S'; \ S' = S \leftarrow T \)

---

Many options!

Number of semi-join options is exponential in # of relations in join
Privacy Preserving Join

- Site 1 has \( R(A, B) \)
- Site 2 has \( S(A, C) \)
- Want to compute \( R \bowtie S \)
- Site 1 should NOT discover any \( S \) info not in the join
- Site 2 should NOT discover any \( R \) info not in the join

Semi-Join Does Not Work

- If Site 1 sends \( I_A R \) to Site 2, site 2 learns all keys of \( R \)!

Fix: Send hashed keys

- Site 1 hashes each value of \( A \) before sending
- Site 2 hashes (same function) its own \( A \) values to see what tuples match

What is problem?

- Dictionary attack!
  Site 2 takes all keys, \( a_1, a_2, a_3 \ldots \) and checks if \( h(a_1), h(a_2), h(a_3) \) matches what Site 1 sent...

Adversary Model

- Honest but Curious
  - dictionary attack is possible (cheating is internal and can’t be caught)
  - sending incorrect keys not possible (cheater could be caught)
One Solution (Agrawal et al)

- Use commutative encryption function
  - $E_i(x) = x$ encryption using site $i$’s private key
  - $E_1(E_2(x)) = E_2(E_1(x))$
  - Shorthand for example:
    - $E_1(x)$ is $x$
    - $E_2(x)$ is $x$
    - $E_1(E_2(x))$ is $x$

Why does this solution work?

Other Privacy Preserving Operations?

- Inequality join $R > \times S$
- Similarity Join $R \times \times S$ with $\text{sim}(R.A,S.A) < e$

Other parallel operations

- Duplicate elimination
  - Sort first (in parallel)
  - Then eliminate duplicates in result
  - Partition tuples (range or hash) and eliminate locally

- Aggregates
  - Partition by grouping attributes; compute aggregate locally

Example:

```
R_a  |  #  | dept | sal
1    | 1   | toy  | 10
2    | 2   | toy  | 20
3    | 3   | sales| 15
```

```
R_b  |  #  | dept | sal
4    | 4   | sales| 5
5    | 5   | toy  | 20
6    | 6   | mgmt | 15
7    | 7   | sales| 10
8    | 8   | mgmt | 30
```

• sum (sal) group by dept
Example:

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>mgmt</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

- sum (sal) group by dept

Example:

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>sales</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>mgmt</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

- sum (sal) group by dept

Example:

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

less data!

Example:

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

less data!

Example:

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ra</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>2</td>
<td>toy</td>
<td>toy</td>
</tr>
<tr>
<td>3</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rb</th>
<th># dept</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>5</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>6</td>
<td>sales</td>
<td>sales</td>
</tr>
<tr>
<td>7</td>
<td>mgmt</td>
<td>mgmt</td>
</tr>
</tbody>
</table>

less data!

Preview: Map Reduce

- data A1
- data A2
- data A3
- data B1
- data B2
- data C1
- data C2
**Enhancements for aggregates**
- Perform aggregate during partition to reduce data transmitted
- Does not work for all aggregate functions...
  Which ones?

**Selection**
- Range or hash partition
- Straightforward
  But what about indexes?

**Indexing**
- Can think of partition vector as root of distributed index:

![Diagram of partition vector as root of distributed index]

**Index on non-partition attribute**

![Diagram of index on non-partition attribute]

Notes:
- If index is not too big, it may be better to keep whole and make copies...
- If updates are frequent, can partition update work...
  (Question: how do we handle split of B-Tree pages?)

**Extensible or linear hashing**

![Diagram of extensible or linear hashing]

R1
f → R2
  ↓
R3
R4 ← add
• How do we adapt schemes?
• Where do we store directory, set of participants...?
• Which one is better for a distributed environment?
• Can we design a hashing scheme with no global knowledge (P2P)?

Summary: Query processing
• Decomposition and Localization ✓
• Optimization
  - Overview ✓
  - Tricks for joins, sort... ✓
  - Tricks for inter-operations parallelism
  - Strategies for optimization

Inter-operation parallelism
• Pipelined
• Independent

Pipelined parallelism

Site 1
\( \sigma \)

Site 2
\( S \)

Join

Probe

Tuples matching \( \sigma \)

result

Site 1

R

Tuples

Site 1

S

Independent parallelism

Site 1

R

S

T

V

Site 2

(1) \( \text{temp1} \leftarrow R \bowtie S \); \( \text{temp2} \leftarrow T \bowtie V \)
(2) \( \text{result} \leftarrow \text{temp1} \bowtie \text{temp2} \)

• Pipelining cannot be used in all cases e.g.: Hash Join

Stream of R tuples

Stream of S tuples
Summary

As we consider query plans for optimization, we must consider various tricks:
- for individual operations
- for scheduling operations