Distributed Databases

CS347
Lecture 14
May 30, 2001
Topics for the Day

• Query processing in distributed databases
  – Localization
  – Distributed query operators
  – Cost-based optimization
Query Processing Steps

• Decomposition
  – Given SQL query, generate one or more algebraic query trees

• Localization
  – Rewrite query trees, replacing relations by fragments

• Optimization
  – Given cost model + one or more localized query trees
  – Produce minimum cost query execution plan
Decomposition

- Same as in a centralized DBMS
- Normalization (usually into relational algebra)

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

\[ \sigma (R.B = 1 \lor R.C > 3) \land (S.D = 2) \]

 Conjunctive normal form
Decomposition

• Redundancy elimination
  \[(S.A = 1) \land (S.A > 5) \implies \text{False} \]
  \[(S.A < 10) \land (S.A < 5) \implies S.A < 5 \]

• Algebraic Rewriting
  – Example: pushing conditions down

\[\sigma_{\text{cond}}\]
\[\rightarrow\]
\[\sigma_{\text{cond}1} \quad \sigma_{\text{cond}2} \]

\[S \quad T\]
\[S \quad T\]
Localization Steps

1. Start with query tree
2. Replace relations by fragments
3. Push $\cup$ up & $\pi,\sigma$ down (CS245 rules)
4. Simplify – eliminating unnecessary operations

Note: To denote fragments in query trees

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

Relation that fragment belongs to  Condition its tuples satisfy
Example 1

\[ \sigma_{E=3} \quad \bigcup \quad \sigma_{E=3} \quad \bigcup \quad \sigma_{E=3} \]

\[ [R: E < 10] \quad [R: E \geq 10] \]

\[ \sigma_{E=3} \]

\[ [R: E < 10] \quad [R: E \geq 10] \]

\[ \sigma_{E=3} \quad \bigcup \quad \sigma_{E=3} \]

\[ [R: E < 10] \quad [R: E \geq 10] \]

\[ [R: E < 10] \quad [R: E \geq 10] \]
Example 2

\[
\begin{align*}
\text{R} & \quad \text{S} \\
\text{A} & \quad \text{A}
\end{align*}
\]

\[
\begin{align*}
\cup & \quad \cup \\
\text{[R: A<5]} & \quad \text{[R: 5 \leq A \leq 10]} & \quad \text{[R: A>10]} & \quad \text{[S: A<5]} & \quad \text{[S: A \geq 5]}
\end{align*}
\]

\[
\begin{align*}
\text{R}_1 & \quad \text{R}_2 & \quad \text{R}_3 & \quad \text{S}_1 & \quad \text{S}_2
\end{align*}
\]
Rules for Horiz. Fragmentation

- $\sigma_{C_1}[R: C_2] \implies [R: C_1 \land C_2]$
- $[R: \text{False}] \implies \emptyset$
- $[R: C_1] \bowtie_A [S: C_2] \implies [R \bowtie_A S: C_1 \land C_2 \land R.A = S.A]$

- In Example 1:
  $\sigma_{E=3}[R_2: E \geq 10] \implies [R_2: E=3 \land E \geq 10]$
  $\implies [R_2: \text{False}] \implies \emptyset$

- In Example 2:
  $[R: A < 5] \bowtie_A [S: A \geq 5]$
  $\implies [R \bowtie_A S: R.A < 5 \land S.A \geq 5 \land R.A = S.A]$
  $\implies [R \bowtie_A S: \text{False}] \implies \emptyset$
Example 3 – Derived Fragmentation

S’s fragmentation is derived from that of R.

\[ \text{[R: } A < 10\text{]} \quad \text{[R: } A \geq 10\text{]} \quad \text{[S: } K = R.K \land R.A < 10\text{]} \quad \text{[S: } K = R.K \land R.A \geq 10\text{]} \]

\[ R_1 \quad R_2 \quad S_1 \quad S_2 \]
[R: A<10] [S: K=R.K ∧ R.A<10]  
[R: A ≥ 10] [S: K=R.K ∧ R.A ≥ 10]
Example 4 – Vertical Fragmentation

\[ \Pi_A \]
\[ R \]

\[ \Pi_A \]
\[ R_1(K,A,B) \]

\[ \Pi_A \]
\[ R_2(K,C,D) \]

\[ \Pi_{K,A} \]
\[ R_1(K,A,B) \]

\[ \Pi_{K,A} \]
\[ R_2(K,C,D) \]
Rule for Vertical Fragmentation

• Given vertical fragmentation of R(A):
  \[ R_i = \Pi_{A_i}(R), \ A_i \subseteq A \]

• For any \( B \subseteq A \):
  \[ \Pi_B(R) = \Pi_B \left[ \bigotimes_{i} R_i \bigg| B \cap A_i \neq \emptyset \right] \]
Parallel/Distributed Query Operations

• Sort
  – Basic sort
  – Range-partitioning sort
  – Parallel external sort-merge

• Join
  – Partitioned join
  – Asymmetric fragment and replicate join
  – General fragment and replicate join
  – Semi-join programs

• Aggregation and duplicate removal
Parallel/distributed sort

• **Input:** relation R on
  – single site/disk
  – fragmented/partitioned by sort attribute
  – fragmented/partitioned by some other attribute

• **Output:** sorted relation R
  – single site/disk
  – individual sorted fragments/partitions
Basic sort

- Given $R(A,\ldots)$ range partitioned on attribute $A$, sort $R$ on $A$

- Each fragment is sorted independently
- Results shipped elsewhere if necessary
Range partitioning sort

- Given R(A,....) located at one or more sites, not fragmented on A, sort R on A

- **Algorithm**: range partition on A and then do basic sort

![Diagram showing the range partitioning sort process with R_a, R_b, R_1, R_2, R_3, a_0, a_1, R_{1s}, R_{2s}, R_{3s}, and the result set.]

result
Selecting a partitioning vector

• Possible centralized approach using a “coordinator”
  – Each site sends statistics about its fragment to coordinator
  – Coordinator decides # of sites to use for local sort
  – Coordinator computes and distributes partitioning vector

• For example,
  – Statistics could be (min sort key, max sort key, # of tuples)
  – Coordinator tries to choose vector that equally partitions relation
Example

• Coordinator receives:
  – From site 1: Min 5, Max 9, 10 tuples
  – From site 2: Min 7, Max 16, 10 tuples

• Assume sort keys distributed uniformly within [min,max] in each fragment

• Partition R into two fragments

What is $k_0$?
Variations

• Different kinds of statistics
  – Local partitioning vector
  – Histogram

• Multiple rounds between coordinator and sites
  – Sites send statistics
  – Coordinator computes and distributes initial vector \( V \)
  – Sites tell coordinator the number of tuples that fall in each range of \( V \)
  – Coordinator computes final partitioning vector \( V_f \)
Parallel external sort-merge

- Local sort
- Compute partition vector
- Merge sorted streams at final sites
Parallel/distributed join

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

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

Join attribute A

Local join

Result

Note: Works only for equi-joins
Partitioned Join

- Same partition function (f) for both relations
- f can be range or hash partitioning
- Any type of local join (nested-loop, hash, merge, etc.) can be used
- Several possible scheduling options. Example:
  - partition R; partition S; join
  - partition R; build local hash table for R; partition S and join
- Good partition function important
  - Distribute join load evenly among sites
Asymmetric fragment + replicate join

Join attribute A

\[ R_a \]
\[ R_b \]

Local join

\[ R_1 \]
\[ R_2 \]
\[ R_3 \]

\[ S \]

\[ S_a \]
\[ S_b \]

Partition function \( f \)

Result

- Any partition function \( f \) can be used (even round-robin)
- Can be used for any kind of join, not just equi-joins
General fragment + replicate join

\[ \begin{align*}
\mathbf{R}_a & \rightarrow \mathbf{R}_1 \\
\mathbf{R}_b & \rightarrow \mathbf{R}_2 \\
\mathbf{R}_b & \rightarrow \mathbf{R}_n \\
\mathbf{S}_a & \rightarrow \mathbf{S}_1 \\
\mathbf{S}_b & \rightarrow \mathbf{S}_2 \\
\mathbf{S}_b & \rightarrow \mathbf{S}_m
\end{align*} \]

Partition

Replicate m copies

Replicate n copies
All \( n \times m \) pairings of R,S fragments

Result

- Asymmetric F+R join is a special case of general F+R.
- Asymmetric F+R is useful when S is small.
Semi-join programs

- Used to reduce communication traffic during join processing
- \( R \bowtie S = (R \bowtie S) \bowtie S \)
  \[= R \bowtie (S \bowtie R)\]
  \[= (R \bowtie S) \bowtie (S \bowtie R)\]
Compute $\Pi_A(S) = [2, 10, 25, 30]$

$S \bowtie (R \bowtie S)$

$R \bowtie S = \begin{bmatrix} 10 & y \\ 25 & w \end{bmatrix}$

- Using semi-join, communication cost = $4A + 2(A + C) + \text{result}$
- Directly joining $R$ and $S$, communication cost = $4(A + B) + \text{result}$
Comparing communication costs

- Say $R$ is the smaller of the two relations $R$ and $S$
- $(R \bowtie S) \bowtie S$ is cheaper than $R \bowtie S$ if
  \[ \text{size (} \Pi_A S \text{)} + \text{size (} R \bowtie S \text{)} < \text{size (} R \text{)} \]
- Similar comparisons for other types of semi-joins
- Common implementation trick:
  - Encode $\Pi_A S$ (or $\Pi_A R$) as a bit vector
  - 1 bit per domain of attribute $A$

\[
\begin{array}{ccccccccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}
\]
n-way joins

• To compute $R \bowtie S \bowtie T$
  - **Semi-join program 1:** $R' \bowtie S' \bowtie T$
    where $R' = R \bowtie S$ & $S' = S \bowtie T$
  - **Semi-join program 2:** $R'' \bowtie S' \bowtie T$
    where $R'' = R \bowtie S'$ & $S' = S \bowtie T$
  - Several other options

• In general, number of options is exponential in the number of relations
Other operations

• Duplicate elimination
  – Sort first (in parallel), then eliminate duplicates in the result
  – Partition tuples (range or hash) and eliminate duplicates locally

• Aggregates
  – Partition by grouping attributes; compute aggregates locally at each site
Example

\[ \text{sum(sal) group by dept} \]
Example

Aggregate during partitioning to reduce communication cost

Does this work for all kinds of aggregates?
Query Optimization

• Generate query execution plans (QEPs)
• Estimate cost of each QEP ($, time, …)
• Choose minimum cost QEP

• What’s different for distributed DB?
  – New strategies for some operations (semi-join, range-partitioning sort, …)
  – Many ways to assign and schedule processors
  – Some factors besides number of IO’s in the cost model
Cost estimation

- In centralized systems - estimate sizes of intermediate relations
- For distributed systems
  - Transmission cost/time may dominate
  - Account for parallelism
  - Data distribution and result re-assembly cost/time
Optimization in distributed DBs

• Two levels of optimization
• Global optimization
  – Given localized query and cost function
  – Output optimized (min. cost) QEP that includes relational and communication operations on fragments
• Local optimization
  – At each site involved in query execution
  – Portion of the QEP at a given site optimized using techniques from centralized DB systems
Search strategies

1. Exhaustive (with pruning)
2. Hill climbing (greedy)
3. Query separation
Exhaustive with Pruning

- A fixed set of techniques for each relational operator
- Search space = “all” possible QEPs with this set of techniques
- Prune search space using heuristics
- Choose minimum cost QEP from rest of search space
Example

$|R| > |S| > |T|$

1. Prune because cross-product not necessary
2. Prune because larger relation first

(R $\bowtie$ S) $\bowtie$ T

(R $\bowtie$ S) $\bowtie$ T

(T $\bowtie$ S) $\bowtie$ R

Ship T to S
Semi-join

Ship S to R
Semi-join
Hill Climbing

- Begin with initial feasible QEP
- At each step, generate a set S of new QEPs by applying ‘transformations’ to current QEP
- Evaluate cost of each QEP in S
- Stop if no improvement is possible
- Otherwise, replace current QEP by the minimum cost QEP from S and iterate
Example

- **Goal**: minimize communication cost

- **Initial plan**: send all relations to one site
  - To site 1: cost = 20 + 30 + 40 = 90
  - To site 2: cost = 10 + 30 + 40 = 80
  - To site 3: cost = 10 + 20 + 40 = 70
  - To site 4: cost = 10 + 20 + 30 = 60

- **Transformation**: send a relation to its neighbor
Local search

- Initial feasible plan
  
P0: \( R (1 \rightarrow 4); \; S (2 \rightarrow 4); \; T (3 \rightarrow 4) \)
  
  Compute join at site 4

- Assume following sizes:
  
  \( R \bowtie S \Rightarrow 20 \)

  \( S \bowtie T \Rightarrow 5 \)

  \( T \bowtie V \Rightarrow 1 \)
No change

Worse

cost = 30

cost = 40
Next iteration

- P1: \( S (2 \to 3); \ R (1 \to 4); \ \alpha (3 \to 4) \)
  where \( \alpha = S \bowtie T \)
  Compute answer at site 4
- Now apply same transformation to R and \( \alpha \)
Resources