Query Optimization
Techniques for
Partitioned Tables

Herodotos Herodotou
Nedyalko Borisov
Shivnath Babu

Duke University
Table Partitioning

- Split parent table into smaller child tables (partitions)

- Partitioning methods: hash, range, list

- Benefits
  - Improve query performance
  - Faster data loading, archival, backup
  - Efficient statistics maintenance
  - Better cardinality estimation
  - Fine-grained control for tuning
  - . . .

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sales_Jan_11**

**Sales_Feb_11**

**Sales_Mar_11**
Recent Trends

- **Growing usage** due to increased data sizes

- **Growing user control** due to new SQL extensions
  - Partitioning conditions for derived tables

- DBA must satisfy multiple **objectives and constraints** regarding partitioning

- **Implications**
  - DBA may have limited control over partitioning scheme
  - Diverse mix of partitioning schemes
Partitioning Schemes

- Multidimensional (Hierarchical) partitioning
- Tables partitioned on same key but different ranges
- Range partitioning with non-equi ranges
Partition-aware Query Optimization

SELECT * 
FROM Sales S, Payments P 
WHERE S.id = P.id 
    AND S.d > 'Feb-15' AND P.id < 25
SELECT *  
FROM Sales S, Payments P  
WHERE S.id = P.id  
    AND S.d > Feb-15  
    AND P.id < 25

P₁: “join of unions”

HJ

Union

Union

TS(S₁₂)
TS(S₁₃)
TS(S₂₂)
TS(S₂₃)

TS(P₁)
TS(P₂)
TS(P₃)

P₂: “union of (partition-wise) joins”

Union

Union

Union

IS(P₃)

TS(P₁)
TS(S₁₂)
TS(S₂₂)
TS(S₂₃)

TS(P₂)
TS(S₁₃)
TS(S₂₂)
TS(S₂₃)
Partition-aware Query Optimization

- More efficient partition-wise joins
- More appropriate join orders
- More appropriate join operators

Legend
- S = Sales
- P = Payments
- L = Loans
Problem & Challenges

- **Problem Definition**
  - Given a partitioning scheme and a query find the optimal query execution plan

- **Challenges**
  - Dealing with plan space explosion
  - Incorporating into state-of-the-art optimizers
  - Partitions as physical or logical properties?
  - Supporting a wide range of partitioning conditions
Overview

1. Matching
   - $S_{12}$ matches $P_1$
   - $S_{13}$ matches $P_2$
   - $S_{22}$ matches $P_3$

2. Clustering
   - $S_{12}$ clusters $P_1$
   - $S_{13}$ clusters $P_2$
   - $S_{22}$ clusters $P_3$
   - $S_{23}$ clusters $P_3$

3. Path Selection
   - $\text{Union} \hspace{10pt} \text{Union} \hspace{10pt} \text{Union}$
   - $TS(S_{22})$ and $TS(P_1)$
   - $TS(S_{23})$ and $TS(P_2)$
   - $\text{HJ}$
   - $\text{Union} \hspace{10pt} TS(P_3)$
   - $TS(S_{22})$ and $TS(S_{23})$
Matching Phase

• Goal
  • Identify partition-wise join pairs that can generate output records

• New data structure: Partition Index Tree (PIT)
  • Core idea: associate each partition with intervals
  • Functionalities: index intervals, efficient lookups
  • Implementation: Augmented red-black tree
Matching Algorithm

**Inputs:** Table S, Table P, Join condition J

**Step 1:** Convert partitioning conditions to intervals

**Step 2:** Build PIT with intervals of S

**Step 3:** Probe PIT with intervals of P

**Output:** Partition-wise join pairs \((S_i, P_j)\)

<table>
<thead>
<tr>
<th>Notation:</th>
<th>Partition</th>
<th>Interval</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_{12}</td>
<td>[0, 20)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>S_{13}</td>
<td>[0, 20)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>S_{22}</td>
<td>[20, 40)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>S_{23}</td>
<td>[20, 40)</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>

**Probe:**
- P_{2} [10, 20)

**Output:**
- Partition Join Pairs
  - \((S_{12}, P_{1})\)
  - \((S_{12}, P_{2})\)
  - \((S_{13}, P_{1})\)
  - \((S_{13}, P_{2})\)
  - \((S_{22}, P_{3})\)
  - \((S_{23}, P_{3})\)
Matching Algorithm

• Benefits
  • $O(n \log(n))$, $n =$ number of partitions of $S$
  • $\Theta(n)$ memory needs
  • Build & reuse PITs multiple times

• Additional support
  • Numeric, dates, and string ranges or lists
  • Complex partitioning conditions (AND, OR)
  • Complex join conditions (AND, OR)
  • Non-equi joins
  • Details in the paper
Clustering Phase

**Input:** Partition join pairs (output from Matching Phase)

**Step 1:** Build bipartite join partition graph

**Step 2:** Find connected components using Breadth-First-Search

**Output:** Clustered join pairs

---

**Input:** Partition join pairs

- \((S_{12}, P_1)\)
- \((S_{12}, P_2)\)
- \((S_{13}, P_1)\)
- \((S_{13}, P_2)\)
- \((S_{22}, P_3)\)
- \((S_{23}, P_3)\)

**Intermediate:** Join Partition Graph

**Output:** Clustered join pairs

- \((\{S_{12}, S_{13}\}, \{P_1, P_2\})\)
- \((\{S_{22}, S_{23}\}, \{P_3\})\)
Path Creation and Selection

- **Goal**
  - Create and cost partition-wise join paths for child tables
Bottom-up Query Optimization

Find & retain best 3-way join paths per interesting order

Find & retain best 2-way join paths per interesting order

Find & retain best access paths
Bottom-up Query Optimization

Logical Relation (Join)

Best (physical) join path

Best (physical) join path with interesting order
Extended Enumeration

- Original enumeration
  - Create and cost join of unions
- Extended enumeration
  - Create and cost union of joins
  - Retain best path per interesting order
- Not enough!
  - Not considering entire plan space (e.g., if $P_j$ is best, no 3-way partition-wise joins)
Treating Partitions as Physical Properties

• Interesting partitions in joins
  • Can make later joins less expensive

• Approach
  • Retain best path for each interesting order
  • Retain best path for each interesting partition

• Limitation
  • Not considering entire plan space (e.g., cannot create union of joins with different join orders)
Treating Partitions as Logical Properties

Logical child joins
- Treated like logical parent joins
- Retain best join paths per interesting order

Logical child joins
- Treated like logical parent joins
- Retain best join paths per interesting order
Treating Partitions as Logical Properties

• Property 1
  • Interesting orders independent across child joins
• Property 2
  • Child joins can have different join orders/operators
• Property 3
  • Entire extended plan space is enumerated
• Optimality guarantee
  • Our bottom-up optimizer will find the optimal plan in the extended plan space
Experimental Evaluation

- Prototype using PostgreSQL 8.3.7
- TPC-H benchmark (scale 30)

Evaluation Methodology

- DBA has full/limited control over partitioning scheme
- State-of-the-art Vs. Our partition-aware optimizer

Optimizer evaluation metrics

- Execution time
- Optimization time
- Memory utilization
Evaluation: Execution Time

![Chart showing execution time for different queries and conditions. The chart compares State-of-the-art and Partition aware methods. Queries 2, 3, 4, 5, 7, 8, 9, 10, 12, and 14 are measured.]
Evaluation: Optimization Time

- **State-of-the-art**
- **Partition aware**

**Optimization Time (ms)**

**Query**

1. Query 2
2. Query 3
3. Query 4
4. Query 5
5. Query 7
6. Query 8
7. Query 9
8. Query 10
9. Query 12
10. Query 14
Evaluation: Memory Utilization

![Memory Usage Chart](chart.png)

- **State-of-the-art**
- **Partition aware**

Memory Usage (MB) vs. Query

- **Query**: 2, 3, 4, 5, 7, 8, 9, 10, 12, 14
- **State-of-the-art**
- **Partition aware**

06/14/2011
Duke University
Summary

- **Extended plan space** to include plans with multiway partition-wise joins
- Developed **new partition-aware optimization techniques**
- Easy incorporation into bottom-up query optimizers

Thank You
Extensions to Parallel Databases

- **Data placement strategy**
  - Hash partitioning to nodes
  - Range/list partitioning within each node

- **Our extensions**
  - Create partition-wise joins \( S_i \bowtie P_i \) for each node \( N_i \)
  - Produce child joins for \( S_i \bowtie P_i \)

- **Data placement strategy**
  - Replicate dimension tables
  - Partition fact tables

- **Our extensions**
  - Further partition all tables
  - Produce child joins on each node
Evaluation: Vary Partition Size

![Bar graph showing optimization time vs partition size (MB).]

- **State-of-the-art**
- **Partition aware**

Optimization Time (ms) vs Partition Size (MB):

- 64 MB
- 96 MB
- 128 MB
- 192 MB
- 256 MB
Evaluation: Vary Data Size

- State-of-the-art
- Partition aware

Data Size (GB): 10, 20, 30, 40

Query:
- 3
- 4
- 7
- 5
- 8
- 9
Cardinality Estimation

![Bar chart showing estimated number of records for different queries. The chart compares State-of-the-art, Partition aware, and Actual results.](chart.png)

- **Query Q02**
  - State-of-the-art: ~1.0E+08
  - Partition aware: ~1.0E+08
  - Actual: ~1.0E+08

- **Query Q03**
  - State-of-the-art: ~1.0E+09
  - Partition aware: ~1.0E+09
  - Actual: ~1.0E+09

- **Query Q04**
  - State-of-the-art: ~1.0E+08
  - Partition aware: ~1.0E+08
  - Actual: ~1.0E+08

- **Query Q05**
  - State-of-the-art: ~1.0E+08
  - Partition aware: ~1.0E+08
  - Actual: ~1.0E+08

- **Query Q07**
  - State-of-the-art: ~1.0E+09
  - Partition aware: ~1.0E+09
  - Actual: ~1.0E+09

- **Query Q08**
  - State-of-the-art: ~1.0E+10
  - Partition aware: ~1.0E+10
  - Actual: ~1.0E+10

- **Query Q09**
  - State-of-the-art: ~1.0E+10
  - Partition aware: ~1.0E+10
  - Actual: ~1.0E+10

- **Query Q10**
  - State-of-the-art: ~1.0E+10
  - Partition aware: ~1.0E+10
  - Actual: ~1.0E+10

- **Query Q12**
  - State-of-the-art: ~1.0E+10
  - Partition aware: ~1.0E+10
  - Actual: ~1.0E+10

- **Query Q14**
  - State-of-the-art: ~1.0E+10
  - Partition aware: ~1.0E+10
  - Actual: ~1.0E+10