Skipping-oriented Partitioning for Columnar Layouts

Sun, et al.

Presented by Ryan Wallace
Problem

Data Blocks

Data Skipping
Problem
Importance

More skipping → Less:

- I/O
- CPU work (decompression)
- Locking conflict
Existing Solutions

Range Partitioning

Problem: Too coarse-grained.
Existing Solutions

Skipping Oriented Partitioning

Filters are:
1. Common
2. Stable
**Existing Solutions**

**Skipping Oriented Partitioning:** **Workload Analysis**

- Feature: a single predicate or a conjunction of predicates.
- Importance weights

```sql
ALTER TABLE table change
CREATE DATABASE database
CREATE TABLE (col def.....PRIMARY
    KEY(col))
DELETE FROM tablename WHERE clause
DROP database/table
INSERT INTO table (col,col,...)
    VALUES (col,col,...)
LOAD DATA INFILE _filename_ INTO
    TABLE table
SELECT col,col,... FROM table
    WHERE clause
SELECT statement UNION SELECT
    statement
SHOW DATABASES|TABLES
SHOW COLUMNS FROM table
UPDATE table SET
    col=value,...WHERE clause
```
Existing Solutions

Skipping Oriented Partitioning: **Partitioning**

Union vector: bitwise OR of all feature vectors in block
Existing Solutions

Skipping Oriented Partitioning: **Querying**

Two steps:
1. Check which features *subsume* the query
2. Skip blocks based on their *union vectors*

Example:

```
SELECT A, D FROM T WHERE A='m' and D= 2
```
### Difficulty

<table>
<thead>
<tr>
<th>Filter</th>
<th>grade='A'</th>
<th>year&gt;2011 and course='DB'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best partitioning</td>
<td>t₁, t₂</td>
<td>t₃, t₄</td>
</tr>
</tbody>
</table>

“Feature conflict”
Resolution: **Generalized Skipping-Oriented Partitioning**

No more feature conflict!
Difficulty

But, a **new difficulty**!

At query time, we must **undo** all the row rearrangements.
Intuition for Solution

Key challenges:
1. Column Grouping
2. Local Feature Selection
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20
Solution: Column Grouping

\[ c_1 > 10 \text{ and } c_2 = 5 \text{ and } c_6 < 20 \]

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, \mathcal{G}) = \sum_{G_i \in \mathcal{G}^q} |G_i \cap \mathcal{C}^q| \cdot r_i^q + \text{overhead}(q, \mathcal{G})
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

\[
\text{COST}(q, G) = \sum_{G_i \in \mathcal{G}^q} |G_i \cap C^q| \cdot r_i^q + \text{overhead}(q, G)
\]
Solution: Column Grouping

c1 > 10 and
c2 = 5 and
c6 < 20

Approach:
Exact

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r^q_i + \text{overhead}(q, G)
\]
Solution: Column Grouping

\[ c_1 > 10 \text{ and } c_2 = 5 \text{ and } c_6 < 20 \]

Approach:
Selectivity Estimate
Solution: Column Grouping

\[ c_1 > 10 \text{ and } c_2 = 5 \text{ and } c_6 < 20 \]

**Approach:**
Block Estimate

\[
\text{COST}(q, G) = \sum_{G_i \in G^q} |G_i \cap C^q| \cdot r_{i}^q + \text{overhead}(q, G)
\]
Solution: Local Feature Selection

Global features

\[ c_1 > 10 \]
\[ c_2 = 5 \]
\[ c_5 = 1 \]
\[ c_6 < 20 \]

Approach:
One-to-one
Solution: Local Feature Selection

Problem:
SELECT c1, c6
WHERE c1 > 10

Approach:
One-to-one
Solution: Local Feature Selection

Global features

c1 > 10

c2 = 5

c5 = 1

c6 < 20

Approach:
Feature weighting
Solution: Local Feature Selection

Approach:
Feature weighting

Question:
How many local features in each column group?
Analysis and Experiments

System prototype:
- In Apache Spark
- GSOP applied to individual Parquet files

Benchmarks:
1. Big Data Benchmark
2. TPC-H
3. Sloan Digital Sky Survey
Analysis and Experiments

Big Data Benchmark - *micro-benchmarking*

Parameters:
- Number of columns and columns templates
- Filter skewness
- Selectivity
Analysis and Experiments

TPC-H - query performance
- Large table (70 x 600 million); selective filter predicates

GSOP-single suffers from tuple ID reads and tuple reconstructions
Analysis and Experiments

TPC-H - *column grouping*
- Varying the column grouping subroutine

Hyrise and HillClimb: column co-access patterns

GSOP: column co-access patterns + *feature conflict* + *skipping horizontal blocks*
Analysis and Experiments

TPC-H - *objective function evaluation*

- Used in column grouping algorithm

Block estimation approaches the quality of full computation
Analysis and Experiments

SSDS
- Real-world workload; relatively few columns selected

- GSOP-hy: 8 column groups
- GSOP-hc: 20 column groups
- GSOP: 2 column groups
Proof of Claims

- GSOP theoretically guaranteed to perform better than SOP and GSOP-single

- Empirically confirmed in a variety of workloads
Gaps

1. Uniform data I/O cost

2. Optimal size of GSOP atomic “chunks”
Next Steps

1. Compression awareness
2. More advanced workload analysis
3. Additional layout options (e.g., replication)
4. Adaptivity