Workload-Aware Aggregate Maintenance in Columnar In-Memory Databases

Stephan Müller, Lars Butzmann, Stefan Klauck, Hasso Plattner
Motivation: Mixed Workloads

- **OLTP**
  - Transactions

- **OLAP**
  - Analytics

---

**Data Warehouse**

**ETL**

**Relational Database**
Motivation: Mixed Workloads

OLTP + OLAP

IMDB

Transactions

Analytics
Materialized Views

600 Inserts, 400 Selects, 1M Base Table, 10 Products

Workload Execution Time in s

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

- w/o materialized aggregate
- w/ materialized aggregate

- Maintenance
- Select
- Insert
Materialized Views

CREATE MATERIALIZED VIEW Aggregate AS
SELECT Product, SUM(Amount) as Amount
FROM Fact
GROUP BY Product

<table>
<thead>
<tr>
<th>Fact</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>Amount</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Aggregate</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>Amount</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
Materialized View Tradeoff

Overhead of maintenance

vs.

Timeliness of maintenance
In-Memory Database Technology
SanssouciDB

Column store  Compression  Main-Delta-Architecture
Related Work

- Materialized view maintenance
  - Gupta & Mumick (1995)
  - Müller et al. (2013)

- View maintenance in data warehouses
  - Zhuge et al. (1995)
  - Agrawal et al. (1997)
  - Jain & Gosain (2012)

- Summary-Delta Tables Concept
  - Mumick & Quass (1997)
Maintenance Categories

- Timing
  - Eager
  - Lazy
  - Merge

- Type of update algorithm
  - Full
  - Incremental
Maintenance Timing

<table>
<thead>
<tr>
<th>Insert</th>
<th>Insert</th>
<th>Select</th>
<th>Insert</th>
<th>Select</th>
<th>Merge</th>
<th>Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy</td>
<td>Lazy</td>
<td>Eager</td>
<td>Eager</td>
<td>Eager</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Update Algorithm

Full update

Complete table

Incremental update

Changes only
Merge Update Strategy
Persisted aggregate

Aggregated on-the-fly
Performance
Influencing Factors

- **Select-insert ratio**
  - 10 Select : 1 Insert
  - 1 Select: 1000 Insert

- **Delta size**

- **Order of queries**
  - Insert...
  - Insert...
  - Select...

- **Compaction ratio**
  - Pareto
  - Uniform
  - ...
1M Records Base Table
1000 Statements,
Inserts excluded

Query + Maintenance Time in ms

No Materialization
Eager Incremental
Lazy Incremental
Merge Update
Break-Even Point
Lazy Incremental vs. Merge Update

![Graph showing the query and maintenance time in ms for Lazy Incremental and Merge Update against the ratio of #Inserts / (#Queries + #Inserts).]
Maintenance Strategy Switching
Requirements

Materialized Aggregate Engine

<table>
<thead>
<tr>
<th>Maintenance Strategies</th>
<th>Cost Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switching Engine</td>
<td>Cost Estimator</td>
</tr>
</tbody>
</table>

SanssouciDB
Cost Estimator

- **Cost model** for all strategies
- **Configurator** measures basic operations

<table>
<thead>
<tr>
<th>Input</th>
<th>Workload Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Execution time of strategies</td>
</tr>
</tbody>
</table>
## Cost Model

<table>
<thead>
<tr>
<th>Merge Strategy</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{\text{Merge}} = R_{\text{Select}} \cdot (T_{\text{Select}} + T_{\text{Delta}} + T_{\text{Union}}) )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lazy Strategy</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{\text{Lazy}} = R_{\text{Select}} \cdot T_{\text{Select}} + R_{\text{Insert}} \cdot \text{opt}(R_{\text{Insert}}) \cdot (T_{\text{Dict}} + T_{\text{Maintenance}}) )</td>
<td></td>
</tr>
</tbody>
</table>
Model vs. Benchmark

<table>
<thead>
<tr>
<th>Query + Maintenance Time in ms</th>
<th>#Inserts / (#Inserts + #Selects)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lazy Incremental</td>
<td>Merge Update</td>
</tr>
<tr>
<td>Lazy Incremental Cost Model</td>
<td>Merge Update Cost Model</td>
</tr>
</tbody>
</table>
Workload Monitoring

- Query logging

Insert...  
Select...  
Select...  
Insert...

$n$ queries forming a window

Select-insert ratio  
$= \frac{\#\text{Inserts}}{\#\text{Inserts} + \#\text{Selects}}$

Example workload

Best performing strategy

Merge Update  
Break-Even Point  
Lazy Incremental
Evaluation
1M Records Base Table
20k statements (devided into 200 windows)
Future Work

- Update/delete support
  - Currently only insert-only
- Cost aware strategy switch
- Object-aware joins
Summary

- Cost models for view maintenance strategies in columnar IMDB
- Aggregate engine adapting maintenance strategies based on current workload
- Performance gain by on-the-fly switching between maintenance strategies
Thank you

stephan.mueller@hpi.uni-potsdam.de
http://epic.hpi.uni-potsdam.de/Home/AggCache