Balanced Query Processing Based on Lightweight Compression of Intermediates

Patrick Damme, 1st TAB presentation

Supervisor: Wolfgang Lehner
Co-supervisor: Thorsten Strufe
RoSI Post-Doc: Dirk Habich
Importance of Analytical Queries

Analytical workloads

- Growing data volumes in numerous areas
- Online Analytical Processing (OLAP)
  Workloads gain importance
Importance of Analytical Queries

Analytical workloads

- Growing data volumes in numerous areas
- Online Analytical Processing (OLAP) Workloads gain importance

- Star Schema Benchmark Query 3 (SQL)

```sql
SELECT c_nation, s_nation, d_year, SUM(lo_revenue) AS revenue
FROM customer, lineorder, supplier, date
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey
AND lo_orderdate = d_datekey
AND c_region = 'ASIA' AND s_region = 'ASIA'
AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year ASC, revenue DESC;
```
Importance of Analytical Queries

Analytical workloads

▪ Growing data volumes in numerous areas
▪ Online Analytical Processing (OLAP) Workloads gain importance

▪ Star Schema Benchmark Query 3 (prose)

We want the total revenue generated by all orders posted by a customer from Asia and delivered by a supplier from Asia between 1992 and 1997 for all distinct combinations of the customer’s nation, the supplier’s nation, and the year sorted chronologically and in decreasing order of revenue
Importance of Analytical Queries

Analytical workloads
▪ Growing data volumes in numerous areas
▪ Online Analytical Processing (OLAP) Workloads gain importance

▪ Star Schema Benchmark Query 3 (prose)

We want the total revenue
generated by all orders
posted by a customer from Asia and
delivered by a supplier from Asia
between 1992 and 1997
for all distinct combinations of
the customer’s nation,
the supplier’s nation, and
the year
sorted
chronologically and
in decreasing order of revenue

Column-Stores
▪ DB architecture of choice for OLAP workloads
▪ Store data elements of one column in subsequent memory locations
▪ Nowadays fully in-memory for efficiency
OLAP Workloads in In-Memory Column-Stores
OLAP Workloads in In-Memory Column-Stores
OLAP Workloads in In-Memory Column-Stores

Main Memory

Data

Bottleneck

- increasingly fast CPUs
- comparably low main memory bandwidth
OLAP Workloads in In-Memory Column-Stores

- Reduced data sizes
  → lower transfer times
  → better cache utilization
- Direct processing of compressed data
  - Computational overhead

Bottleneck
increasingly fast CPUs
comparably low main memory bandwidth
OLAP Workloads in In-Memory Column-Stores

**Bottleneck**
- Increasingly fast CPUs
- Comparably low main memory bandwidth

**Main Memory**
- Base data
- Intermediates

**Lightweight data compression**
- Reduced data sizes
  - Lower transfer times
  - Better cache utilization
- Direct processing of compressed data

- Computational overhead
OLAP Workloads in In-Memory Column-Stores

**Bottleneck**
- Increasingly fast CPUs
- Comparably low main memory bandwidth

**Main Memory**
- Base data
- Intermediates

**Equal costs**
- For accessing base data and intermediates

**Lightweight data compression**
- Reduced data sizes → lower transfer times
  → better cache utilization
- Direct processing of compressed data

**Treat intermediates efficiently too!**
- Computational overhead

**CPU**
OLAP Workloads in In-Memory Column-Stores

**Bottleneck**
- Increasingly fast CPUs
- Comparably low main memory bandwidth

**Equal costs**
- For accessing base data and intermediates

**Main Memory**
- Base data
- Intermediates

**Lightweight data compression**
- Reduced data sizes → lower transfer times → better cache utilization
- Direct processing of compressed data
- Computational overhead

**Avoid intermediates**
- Special code generation
- Composite operators
  - High effort
  - Not always possible

**Treat intermediates efficiently too!**
OLAP Workloads in In-Memory Column-Stores

- Reduced data sizes
  → lower transfer times
  → better cache utilization
- Direct processing of compressed data
- Computational overhead

Equal costs for accessing base data and intermediates

Bottleneck:
- Increasingly fast CPUs
- Comparably low main memory bandwidth

Main Memory:
- Base data
- AND intermediates

Lightweight data compression

Treat intermediates efficiently too!
Our Vision: Balanced Query Processing based on Compressed Intermediate Results
Our Vision

Lightweight compression of all intermediate result
Our Vision: Balanced Query Processing based on Compressed Intermediate Results

- Compression-aware physical operators
- Lightweight compression of all intermediate result
Our Vision

- Benefit vs. overhead, optimize overall query performance
- Compression-aware physical operators
- Lightweight compression of all intermediate result

Our Vision: Balanced Query Processing based on Compressed Intermediate Results
Our Vision

Benefit vs. overhead, optimize overall query performance
Compression-aware physical operators
Lightweight compression of all intermediate result

Our Vision: **Balanced** Query Processing **based on** Compressed Intermediate Results

Requires a Context-sensitive DBMS
Our Vision

Benefit vs. overhead, optimize overall query performance
Compression-aware physical operators
Lightweight compression of all intermediate result

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics
Surrounding operators
System utilization, user preferences
Our Vision

Benefit vs. overhead, optimize overall query performance
Compression-aware physical operators
Lightweight compression of all intermediate result

Requires a Context-sensitive DBMS

Data characteristics
Surrounding operators
System utilization, user preferences

Structural Aspect
Investigation of lightweight compression
Our Vision

Data characteristics
- Investigation of lightweight compression

Structural Aspect

Surrounding operators
- Integration of compression into query execution

Operational Aspect

System utilization, user preferences

Benefit vs. overhead, optimize overall query performance

Compression-aware physical operators

Lightweight compression of all intermediate result

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS
Our Vision

**Data characteristics**
- Investigation of lightweight compression

**Structural Aspect**

**Operational Aspect**
- Integration of compression into query execution

**Optimization Aspect**
- Compression-aware query optimization

Our Vision: Balanced **Query Processing** based on **Compressed Intermediate Results**

Requires a Context-sensitive DBMS

- Benefit vs. overhead, optimize overall query performance
- Compression-aware physical operators
- Lightweight compression of all intermediate result

Data characteristics

Surrounding operators

System utilization, user preferences
Our Vision

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics
- Investigation of lightweight compression

Structural Aspect

Surrounding operators
- Integration of compression into query execution

Operational Aspect

System utilization, user preferences
- Compression-aware query optimization

Optimization Aspect

Completed work
Focus on Integers

- Due to their outstanding importance in column-stores
- Many columns contain integers by nature
- Other fixed-width types: representable as integers
- Variable-width types: representation by integer codes required for efficient processing
Lightweight Data Compression Techniques

<table>
<thead>
<tr>
<th>RLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Length Encoding</td>
</tr>
<tr>
<td>Replace run by value &amp; length</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1200</th>
<th>1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
</tbody>
</table>
# Lightweight Data Compression Techniques

## RLE

**Run Length Encoding**
- Replace run by value & length

<table>
<thead>
<tr>
<th>Data</th>
<th>RLE</th>
<th>DELTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1200</td>
<td>1000</td>
</tr>
<tr>
<td>1200</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
<td>50</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td>200</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>5</td>
</tr>
</tbody>
</table>

## DELTA

**Differential Coding**
- Replace data elem. by difference to predecessor

<table>
<thead>
<tr>
<th>Data</th>
<th>DELTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1100</td>
<td>100</td>
</tr>
<tr>
<td>1150</td>
<td>50</td>
</tr>
<tr>
<td>1350</td>
<td>200</td>
</tr>
<tr>
<td>1355</td>
<td>5</td>
</tr>
</tbody>
</table>
## Lightweight Data Compression Techniques

<table>
<thead>
<tr>
<th>RLE</th>
<th>DELTA</th>
<th>FOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Length Encoding</td>
<td>Differential Coding</td>
<td>Frame-of-Reference</td>
</tr>
<tr>
<td>Replace run by value &amp; length</td>
<td>Replace data elem. by difference to predecessor</td>
<td>Replace data elem. by difference to reference value</td>
</tr>
<tr>
<td>1200</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>1200</td>
<td>4</td>
<td>200</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
<td>100</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>200</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50</td>
</tr>
</tbody>
</table>
## Lightweight Data Compression Techniques

<table>
<thead>
<tr>
<th>RLE</th>
<th>DELTA</th>
<th>FOR</th>
<th>DICT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Run Length Encoding</strong>&lt;br&gt;Replace run by value &amp; length</td>
<td><strong>Differential Coding</strong>&lt;br&gt;Replace data elem. by difference to predecessor</td>
<td><strong>Frame-of-Reference</strong>&lt;br&gt;Replace data elem. by difference to reference value</td>
<td><strong>Dictionary Coding</strong>&lt;br&gt;Replace data elem. by 0-based key in dictionary</td>
</tr>
<tr>
<td>1200</td>
<td>1200</td>
<td>1200</td>
<td>200</td>
</tr>
<tr>
<td>1200</td>
<td>4</td>
<td>1100</td>
<td>100</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
<td>1150</td>
<td>50</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td>1350</td>
<td>200</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
<td>1355</td>
<td>5</td>
</tr>
<tr>
<td>300</td>
<td>200</td>
<td>1200</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **RLE**: Run Length Encoding
- **DELTA**: Differential Coding
- **FOR**: Frame-of-Reference
- **DICT**: Dictionary Coding
Lightweight Data Compression Techniques

<table>
<thead>
<tr>
<th>RLE</th>
<th>DELTA</th>
<th>FOR</th>
<th>DICT</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Length Encoding</td>
<td>Differential Coding</td>
<td>Frame-of-Reference</td>
<td>Dictionary Coding</td>
<td>Null Suppression</td>
</tr>
<tr>
<td>Replace run by value &amp; length</td>
<td>Replace data elem. by difference to predecessor</td>
<td>Replace data elem. by difference to reference value</td>
<td>Replace data elem. by 0-based key in dictionary</td>
<td>Eliminate leading zeroes in binary representation</td>
</tr>
<tr>
<td>1200</td>
<td>1000</td>
<td>1200</td>
<td>1000</td>
<td>00...001011</td>
</tr>
<tr>
<td>1200</td>
<td>1100</td>
<td>1100</td>
<td>1000</td>
<td>1011</td>
</tr>
<tr>
<td>1200</td>
<td>1150</td>
<td>1000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>1350</td>
<td>1000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>1355</td>
<td>1050</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>1050</td>
<td>2</td>
<td>1011</td>
</tr>
</tbody>
</table>
## Lightweight Data Compression Techniques

### RLE (Run Length Encoding)
- Replace run by value & length
- Examples: 1200 1200, 1200 4, 1200 300, 1200 2, 300 300

### DELTA (Differential Coding)
- Replace data elem. by difference to predecessor
- Examples: 1000 1000, 1100 100, 1150 50, 1350 200, 1355 5

### FOR (Frame-of-Reference)
- Replace data elem. by difference to reference value
- Examples: 1200 200, 1100 100, 1000 0, 1050 50

### DICT (Dictionary Coding)
- Replace data elem. by 0-based key in dictionary
- Examples: 1000 0, 1200 1, 1000 0, 1050 2, 1050 2

### NS (Null Suppression)
- Eliminate leading zeroes in binary representation
- Example: 00...001011

---

**Tailored to data characteristics**
- determining their
  - compression rate
  - performance
# Lightweight Data Compression Techniques

## RLE

<table>
<thead>
<tr>
<th>Run Length Encoding</th>
<th>Replace run by value &amp; length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>1200</td>
<td>4</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
</tr>
<tr>
<td>300</td>
<td>2</td>
</tr>
</tbody>
</table>

## DELTA

<table>
<thead>
<tr>
<th>Differential Coding</th>
<th>Replace data elements by difference to predecessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>1100</td>
<td>100</td>
</tr>
<tr>
<td>1150</td>
<td>50</td>
</tr>
<tr>
<td>1350</td>
<td>200</td>
</tr>
<tr>
<td>1355</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simple-16</th>
<th>4-Wise NS</th>
<th>4-Gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1050</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

## NS

<table>
<thead>
<tr>
<th>Null Suppression</th>
<th>Eliminate leading zeroes in binary representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00...001011</td>
<td></td>
</tr>
</tbody>
</table>

## Tailored to data characteristics
- determining their compression rate
- performance

## High number of algorithms
- many for each technique
- even more by combinations

## Algorithms
- Varint-G8IU
- PFOR
- Masked-VByte
- SIMD-BP128
- SIMD-FastPFOR
- Simple-8b
- PFOR2008
- SIMD-GroupSimple
- Simple-9
- VByte
- 4-Wise NS
- 4-Gamma
- NewPFOR
- ...
## Lightweight Data Compression Techniques

<table>
<thead>
<tr>
<th>RLE</th>
<th>DELTA</th>
<th>FOR</th>
<th>DICT</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Length Encoding</td>
<td>Differential Coding</td>
<td>Frame-of-Reference</td>
<td>Dictionary Coding</td>
<td>Null Suppression</td>
</tr>
<tr>
<td>Replace run by value &amp; length</td>
<td>Replace data elem. by difference to predecessor</td>
<td>Replace data elem. by difference to reference value</td>
<td>Replace data elem. by 0-based key in dictionary</td>
<td>Eliminate leading zeroes in binary representation</td>
</tr>
<tr>
<td>1200</td>
<td>1200</td>
<td>1000</td>
<td>1200</td>
<td>0</td>
</tr>
<tr>
<td>1200</td>
<td>4</td>
<td>1100</td>
<td>1200</td>
<td>1</td>
</tr>
<tr>
<td>1200</td>
<td>300</td>
<td>1150</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>1200</td>
<td>2</td>
<td>1350</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td>1355</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Tailored to data characteristics

- Determining their compression rate
- Performance

### High number of algorithms

- Many for each technique
- Even more by combinations

### Efficiency is crucial for us

- (De)compression during query execution
- Use of SIMD extensions
A Sample of Our Experimental Survey
A Sample of Our Experimental Survey

**Compression rate [bits/int]**

Dataset 1: Algorithm A - 64, Algorithm B - 48, Algorithm C - 32
Dataset 2: Algorithm A - 32, Algorithm B - 16, Algorithm C - 8
Dataset 3: Algorithm A - 16, Algorithm B - 8, Algorithm C - 4

**Compression time [sec]**

Dataset 1: Algorithm A - 0.8, Algorithm B - 0.6, Algorithm C - 0.4
Dataset 2: Algorithm A - 0.4, Algorithm B - 0.2, Algorithm C - 0.1
Dataset 3: Algorithm A - 0.2, Algorithm B - 0.1, Algorithm C - 0.05

**Decompression time [sec]**

Dataset 1: Algorithm A - 0.2, Algorithm B - 0.1, Algorithm C - 0.05
Dataset 2: Algorithm A - 0.1, Algorithm B - 0.05, Algorithm C - 0.02
Dataset 3: Algorithm A - 0.05, Algorithm B - 0.02, Algorithm C - 0.01
There is no single-best compression algorithm

The choice is non-trivial and depends on:
A Sample of Our Experimental Survey

There is no single-best compression algorithm.
The choice is non-trivial and depends on:

Data characteristics
A Sample of Our Experimental Survey

There is no single-best compression algorithm.

The choice is non-trivial and depends on:

- Data characteristics
- Objective
A Sample of Our Experimental Survey

Compression rate [bits/int]

Dataset 1 Dataset 2 Dataset 3
Algorithm A Algorithm B Algorithm C

Compression time [sec]

Dataset 1 Dataset 2 Dataset 3

Decompression time [sec]

Dataset 1 Dataset 2 Dataset 3
Algorithm A Algorithm B Algorithm C

Missing the right choice can lead to
- performance decrease of factors
- memory consumption increase of factors
Changing Data Characteristics
Changing Data Characteristics

Base data

INSERT
UPDATE
DELETE

t_1

data distribution

value

t_2

time
Changing Data Characteristics

Base data

INSERT
UPDATE
DELETE

t_1  t_2

data distribution

frequency
value

frequency
value
Changing Data Characteristics

We might need to adapt the format.
Changing Data Characteristics

We might need to adapt the format and this should be done efficiently.
Changing Data Characteristics

- base data
- value
- frequency
- data distribution

base data
Changing Data Characteristics

![Diagram showing data distribution and base data values]

- Frequency
- Value
- Data distribution
- Base data
- B: 2
- A: 1
- C: 3
Changing Data Characteristics

Diagram showing the process of changing data characteristics:

1. **Base Data**
   - Frequency distribution
   - Value distribution

2. **Selection**
   - Intermediate

The diagram illustrates the flow from base data to intermediate through the selection process.
Changing Data Characteristics

- Base data
  - Value
  - Frequency
  - Data distribution

- Intermediate

- Selection

- Base data
  - Value
  - Frequency
  - Data distribution

- 2
- 1
- 3
- A
- B
- C
Changing Data Characteristics

Base data

Intermediate

Selection

Data distribution

Frequency

Value
Changing Data Characteristics

- base data
- intermediate
- selection

Data distribution:
- frequency vs. value
Changing Data Characteristics

- **Base Data**
- **Selection**
- **Intermediate**
- **Aggregation**

![Data Distribution](image1)

![Data Distribution](image2)
Changing Data Characteristics

- **Base data**
- **Selection**
- **Intermediate**
- **Aggregation**

**Data distribution**

**Frequency** vs. **Value**

**Total query execution time**

- * keeping format A
- * changing to format C

![Graph](chart.png)
Changing Data Characteristics

We might need to adapt the format.
Changing Data Characteristics

We might need to adapt the format. And this must be done very efficiently.
Data Transformation Techniques

Data Transformation

Adapting the representation of the data from a source format to a destination format
Data Transformation Techniques

Data Transformation

Adapting the representation of the data from a source format to a destination format

Indirect transformation

Complete decompression and recompression

Relies only on existing (de)compression algorithms

Inefficient materialization of the uncompressed data
Data Transformation Techniques

Data Transformation
Adapting the representation of the data from a source format to a destination format

Indirect transformation
Complete decompression and recompression

- uncompressed
- compressed (src. format)
- compression (src. format)
- decompression (dest. format)
- compressed (dest. format)

- Relies only on existing (de)compression algorithms
- Inefficient materialization of the uncompressed data

Direct transformation
Transformation in one step

- compressed (src. format)
- direct transformation (src. to dest. format)
- compressed (dest. format)

- Novel class of algorithms related to (de)compression
- Can be much more efficient depending on the data

Our contribution
Example: From RLE to 4-Wise NS
Example: From RLE to 4-Wise NS

Indirect transformation

main memory

RLE

12

main memory

4-Wise NS
Example: From RLE to 4-Wise NS

Indirect transformation

main memory

RLE

main memory

uncompr.

main memory

4-Wise NS

12
Example: From RLE to 4-Wise NS

Indirect transformation:
- Main memory
  - RLE
  - Uncompressed
  - 4-Wise NS

Direct transformation:
- Main memory
  - CPU vector registers
  - RLE
  - Uncompressed | 4-Wise NS
  - 4-Wise NS
Example: From RLE to 4-Wise NS

Indirect transformation:
- RLE
  - main memory
  - uncompr.
  - 12
- 4-Wise NS

Direct transformation:
- CPU vector registers
- RLE
- uncompr.
- 4-Wise NS
- 4-Wise NS

We save:
- Storing/loading uncompressed data to/from main memory
- Repeated block recompression
Our Vision

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics

Structural Aspect

Investigation of lightweight compression

Surrounding operators

Operational Aspect

Integration of compression into query execution

System utilization, user preferences

Optimization Aspect

Compression-aware query optimization

Completed work
Our Vision

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics
- Investigation of lightweight compression
  - Completed work

Structural Aspect

Surrounding operators
- Integration of compression into query execution
  - Ongoing work

Operational Aspect

System utilization, user preferences
- Optimization Aspect
  - Compression-aware query optimization
  - Ongoing work
Uncompressed Processing

Processing Model
- Column-at-a-Time
- Operators consume and produce columns
Uncompressed Processing

**Processing Model**
- Column-at-a-Time
- Operators consume and produce columns

**Inputs and outputs**
- Each might be represented in its individual compressed format
Uncompressed Processing

**Processing Model**
- Column-at-a-Time
- Operators consume and produce columns

**Inputs and outputs**
- Each might be represented in its individual compressed format

**We assume**
- We want to support
  - $n$ compressed formats
  - 1 operator
Uncompressed Processing

**Processing Model**
- Column-at-a-Time
- Operators consume and produce columns

**Inputs and outputs**
- Each might be represented in its individual compressed format

**We assume**
- We want to support
  - $n$ compressed formats
  - 1 operator

Different degrees of integration possible
Wrapper Employing (De)Compression

Idea

- Wrapper around original operator
  - Decompresses inputs
  - Recompresses outputs
- Efficiency: work on blocks fitting into the L1-cache
- Format changes through wrapper configuration

Requires

- 1 operator variant
- $n$ compressions
- $n$ decompressions

Advantage

- Simplicity: relies only on existing
  - operator implementations
  - (de)compression algorithms

Disadvantage

- Wastes potential of working directly on compressed data
Wrapper Employing Direct Transformation

Idea
- Direct processing of compressed data elements
- Common format for all inputs and outputs
- Arbitrary format combinations through a wrapper employing our direct transformations

Requires
- \( n \) operator variants
- \( n^2 - n \) transformations reusable for other operators

Advantage
- Compr. data within operators
  - Better cache utilization
  - Higher bit-level-parallelism when using SIMD instructions

Disadvantage
- Wrapper still causes overhead
Specialized Operator

Idea
- Operator tailored to a specific format combination
- No wrapper

Requires
- $n^{i+o}$ operator variants

Advantage
- Maximum efficiency

Disadvantage
- Highest integration effort due to high number of operator variants
Next Steps: …
Next Steps: Hashing and Compression

Hashing

- Integral part of many DB-operators
- Grouping
- Aggregation
- Join
- Partitioning
Next Steps: Hashing and Compression

Hashing
Integral part of many DB-operators

Join
Grouping
Aggregation
Partitioning

Hash Tables
Next Steps: Hashing and Compression

Hashing
Integral part of many DB-operators
Grouping
Aggregation

Hash Tables

Compressed Hash Tables
- To fit hash table closer to CPU in memory hierarchy
- Important due to random access pattern

Join
Partitioning
Next Steps: Hashing and Compression

Concise Hash Table by IBM
- Eliminates empty buckets in huge virtual linear probing hash table
- No compression of the entries

Integral part of many DB-operators
- Join
- Grouping
- Hashing

Aggregation
- Partitioning

Compressed Hash Tables
- To fit hash table closer to CPU in memory hierarchy
- Important due to random access pattern

Existing works
- Integral part of many DB-operators
- Grouping
- Aggregation
- Join
- Partitioning

Hash Tables

Dresden Database Systems Group
Next Steps: Hashing and Compression

Hashing

Integral part of many DB-operators

Join

Partitioning

Grouping

Aggregation

Hash Tables

Compressed Hash Tables

Existing works

- Concise Hash Table by IBM
- Eliminates empty buckets in huge virtual linear probing hash table
- No compression of the entries

Our ideas

- Compressed entries
- Splitting hash table into several ones: one per key bit width
- Storing compressed hash table entirely in processor (vector) registers, possible?

- To fit hash table closer to CPU in memory hierarchy
- Important due to random access pattern
Our Vision

Our Vision: **Balanced Query Processing based on Compressed Intermediate Results**

Requires a Context-sensitive DBMS

**Structural Aspect**
- Data characteristics
- Investigation of lightweight compression

**Operational Aspect**
- Surrounding operators
- Integration of compression into query execution

**Optimization Aspect**
- System utilization, user preferences
- Compression-aware query optimization

Completed work

Ongoing work
Our Vision

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics
- Investigation of lightweight compression
  - Completed work

Structural Aspect

Surrounding operators
- Integration of compression into query execution
  - Ongoing work

Operational Aspect

System utilization, user preferences
- Compression-aware query optimization
  - Completed/Future work

Optimization Aspect
Compression-Aware Query Optimization
Compression-Aware Query Optimization

Existing query optimizer

SQL query

“optimal” plan
Compression-Aware Query Optimization

Existing query optimizer

- Algebraic restructuring
- Join ordering
- Logical ops to physical ops
- ...

“optimal” plan

SQL query
Compression-Aware Query Optimization

Deep integration is beyond our scope.

Existing query optimizer

- Algebraic restructuring
- Join ordering
- Logical ops to physical ops
- ...

“optimal” plan

highly complex

SQL query
Compression-Aware Query Optimization

- Algebraic restructuring
- Join ordering
- Logical ops to physical ops
- ...

Existing query optimizer

Deep integration is beyond our scope.

Our contribution

Compression-aware optimization strategies

“optimal” plan w/o compression

SQL query

improved plan w/ compression
Compression-Aware Query Optimization

Deep integration is beyond our scope.

Existing query optimizer

- Algebraic restructuring
- Join ordering
- Logical ops to physical ops
- ...

“optimal” plan w/o compression

Our contribution

Compression-aware optimization strategies

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data

improved plan w/ compression
Decision in Context

User side
- Run time requirements (interactive vs. batch)
- Importance of the query (e.g. who submitted it)

System side
- Current system load
- Optimize run time with limited memory consumption
- Optimize memory consumption with limited run time

Our contribution

Compression-aware optimization strategies

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data

improved plan w/ compression
Challenges to Be Addressed

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data
Challenges to Be Addressed

Estimation of the intermediates' data characteristics

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data
Challenges to Be Addressed

Compression-aware optimization strategies

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data
Challenges to Be Addressed

Cost model for compression algorithms and operators

Our contribution

Compression-aware optimization strategies

- Select suitable compressed format for each intermediate
- Replace physical operators by our operators for compressed data

improved plan w/ compression
Selecting a Suitable Compression Algorithm
Selecting a Suitable Compression Algorithm

Data characteristics

Selection strategy

Set of available algorithms

Best algorithm
Selecting a Suitable Compression Algorithm

Cost model

<table>
<thead>
<tr>
<th>Performance</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression</td>
<td>Compression rate</td>
</tr>
<tr>
<td>Decompression</td>
<td></td>
</tr>
<tr>
<td>Aggregation</td>
<td></td>
</tr>
</tbody>
</table>

Data characteristics

Set of available algorithms

Minimal-cost algorithm
Grey-Box Approach
Grey-Box Approach

Modelling

Everything known from the algorithm’s specification
(and our experimental survey)

block size
available bit widths
... decisiive data characteristics
Grey-Box Approach

Modelling

Everything known from the algorithm’s specification (and our experimental survey)

- block size
- bit widths
- decisive data characteristics

Measuring

Everything related to the algorithm’s execution on the hardware

- clock freq.
- cache size
- memory bandwidth
- branch mispred.

One-time calibration phase
## Selecting a Suitable Compression Algorithm

### Data characteristics

<table>
<thead>
<tr>
<th>Cost model</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compression rate</td>
</tr>
<tr>
<td></td>
<td>Compression</td>
</tr>
<tr>
<td></td>
<td>Decompression</td>
</tr>
<tr>
<td></td>
<td>Aggregation</td>
</tr>
</tbody>
</table>

### Algorithms’ profiles (calibration)

- Minimal-cost algorithm

### Set of available algorithms

- Dresden Database Systems Group
Our Vision

**Our Vision: Balanced Query Processing based on Compressed Intermediate Results**

Requires a Context-sensitive DBMS

**Structural Aspect**
- Data characteristics
  - Investigation of lightweight compression
  - Completed work

**Operational Aspect**
- Surrounding operators
  - Integration of compression into query execution
  - Ongoing work

**Optimization Aspect**
- System utilization, user preferences
  - Compression-aware query optimization
  - Completed/Future work
Our Vision

Our Vision: Balanced Query Processing based on Compressed Intermediate Results

Requires a Context-sensitive DBMS

Data characteristics
- Investigation of lightweight compression
  - Completed work

Surrounding operators
- Integration of compression into query execution
  - Ongoing work

System utilization, user preferences
- Compression-aware query optimization
  - Completed/Future work

Structural Aspect
- Operational Aspect
- Optimization Aspect
Our Vision: **Balanced Query Processing** based on **Compressed Intermediate Results**

**Contributions so far**

- **Experimental survey** of lightweight compr. algos  
  EDBT'17  
  (ACM TODS)

- **Direct transformations** for compressed data  
  ADBIS'15

- **Cost model** for compression algorithms  
  (ACM TODS)

- **Benchmark framework** for lightweight compr. algos  
  TPCTC'15
Summary and Contributions

Our Vision: **Balanced Query Processing** based on **Compressed Intermediate Results**

**Contributions so far**

- **Experimental survey** of lightweight compr. algos
  - EDBT’17
    - (ACM TODS)

- **Direct transformations** for compressed data
  - ADBIS’15

- **Cost model** for compression algorithms
  - (ACM TODS)

- **Benchmark framework** for lightweight compr. algos
  - TPCTC’15

**Contributions still in the works**

- **Operators for compressed data** with different degrees of integration

- **Cost model** for compression-aware operators

- **Compression-aware strategies** for the query optimizer