HYRISE – In-Memory Storage Engine

Martin Grund\textsuperscript{1}, Jens Krueger\textsuperscript{1}, Philippe Cudre-Mauroux\textsuperscript{3}, Samuel Madden\textsuperscript{2}, Alexander Zeier\textsuperscript{1}, Hasso Plattner\textsuperscript{1}

\textsuperscript{1}Hasso-Plattner-Institute, Germany
\textsuperscript{2}MIT CSAIL, USA
\textsuperscript{3}University of Fribourg, Switzerland
Enterprise applications have evolved: not just OLAP vs. OLTP

- Demand for real-time analytics on transactional data
- High throughput analytics \(\Rightarrow\) completely in memory
  - Massive RAMs (>1TB/node) enable this for many apps

Example:

- **Available-To-Promise Check** – Perform real-time ATP check directly on transactional data during order entry, without materialized aggregates of available stocks.
- **Dunning** – Search for open invoices interactively instead of scheduled batch runs.
- **Operational Analytics** – Instant customer sales analytics with always up-to-date data.
Our System: HYRISE

- High throughput on structured enterprise data
  - Completely in main memory

- Efficiently executed both OLTP and OLAP requests
  - Key idea: Vertically partition tables

- New algorithms to find the best partitioning for all tables
  - Based on a workload profile
  - Using a cache-miss based cost model
  - Scalable to huge number of tables, wide relations
    - E.g., Many SAP apps have 10K+ tables w/ 100+ columns
Memory Hierarchy - Recap

- Memory hierarchy does not stop with main memory.
- Motivation for disk-based column stores, remains valid for main memory; **Avoid loading data that is not accessed.**

![Diagram of memory hierarchy with CPU, Registers, CPU Caches, Main Memory, Flash, and Hard Disk]

- Accessing memory with different strides introduces different latencies.

![Graph showing CPU Cycles per Value vs. Stride in Bytes with a legend: Sequential accesses 10x-100x faster]
ENTERPRISE BACKGROUND
Enterprise Application Characteristics

- Identify: Why are enterprise applications so complex?

- Detailed customer data analysis from SAP installations of 12 companies (~32 billion event records analyzed)

- Enterprise applications have
  - Extremely wide schemas – up to 300 attributes on heavily used tables
  - Thousands of tables – every ERP installation ~ 70k
  - Changing workload
Example Enterprise Workload

- Range selects occur often
- Real world is more complicated than single tuple access
- With new applications the “read”-gap will even increase
Summary

- Traditional View
  - OLTP Systems for transactional scenarios
  - OLAP Systems for analytical scenarios
- Our View: Single System
  - Main Memory
  - Vertically partitioned
  - Single copy of data (no redundancy)
    - To reduce maintenance and overhead of multiple copies

Key challenge: How to perform vertical partitioning to optimize performance on a given hybrid workload
HYBRID IN-MEMORY STORAGE ENGINE DESIGN
HYRISE Architecture

- Our focus is on three key aspects
  - In-Memory Data Storage
    - Predicting access costs
  - Layout Decisions
  - Optimizing Query Execution

- Layout Engine integrates cost model and workload data
HYRISE Partitioning Problem

- Each table split into a set of non-overlapping *containers* (partitions)
  - Each container consists of one or more attributes

\[
\begin{align*}
C_1 (a_1) & \quad \quad \quad C_2 (a_2 \ldots a_6) & \quad \quad \quad C_2 (a_7 \ldots a_8) \\
\end{align*}
\]

- Uses workload as input to find best partitioning

- The performance of each workload operator on a given layout is calculated based on cache misses

- Container overhead cost defines the cost of loading data that is not accessed by a query operator
Cost Model – Projections

- Goal is to predict cost of basic accesses to a container
  - Based on access to multiple attributes over all rows (projection) and access to all attributes of a container to a selection of rows (selectivity)
- Cache misses are precisely calculated, using the offset and width of the columns projected from the container
  - Not enough to calculate #accessed bytes → understand how the accessed data is laid out
Experimental validation shows the match of the model and reality
HYRISE cost model provides means to calculate cache misses for:
- Full projections / partial projections
- Selections – capturing both independent and overlapping selections

More complex operators can be composed out of the basic elements

Experiments show that cache misses are a good predictor for performance of in-memory database systems.
Cache misses are a good predictor for performance.
LAYOUT SELECTION
For narrow tables, finding the optimal layout is easy and can be done through exhaustive enumeration.

Enterprise applications have super-wide schemas:
- Up to 300 attributes in our study
- Millions of possible layouts
First Approach

- Exponential, but multiple pruning steps that reduce the number of possible layouts in practice

1. Candidate Generation
   - Determine all primary partitions (the largest partitions that will not incur any container overhead cost)

2. Candidate Merging
   - Inspect all permutations of primary partitions to generate partitions that minimize the overall cost

3. Layout Generation
   - Generate all valid layouts by exhaustively exploring all possible combinations of partitions from the second phase
Candidate Generation

- Determining all primary partitions
  - Primary Partition: Largest partition that does not incur container overhead cost
- Each operation on a table implicitly splits the attributes into two subsets
  - The order of the operations can be ignored
- Recursively splitting each set of attributes of the workload into subsets for each operation
Candidate Generation

Table

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>EMAIL</th>
<th>COMPANY</th>
<th>PHONE</th>
<th>ORG</th>
</tr>
</thead>
</table>

Query 1 - Select ID,NAME from Table where ORG = 9

OP 1

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
</table>

OP 2

| ORG |

Query 2 - Select ID,COMPANY from Table where ORG = 9

OP 3

| ID | COMPANY |

OP 4

| ORG |
## Candidate Generation

<table>
<thead>
<tr>
<th>OP 1</th>
<th>ID</th>
<th>NAME</th>
<th>ID</th>
<th>NAME</th>
<th>EMAIL</th>
<th>COMPANY</th>
<th>PHONE</th>
<th>ORG</th>
</tr>
</thead>
</table>

HYRISE | Martin Grund | VLDB 2011
Candidate Merging

- Generate possible permutations of primary partitions
- Identify partitions that reduce the overall cost for the workload
  - Based on the assumption that the access cost for two partitions with the same attribute set can be independently computed
  - Calculation based on the cost model
Candidate Merging

**Primary Partitions**

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>COMPANY</td>
</tr>
<tr>
<td>ID</td>
<td>ORG</td>
</tr>
</tbody>
</table>

**Merged Permutation**

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>COMPANY</td>
</tr>
</tbody>
</table>

| ID | ORG |

- **Subset 1**
  - 12,000
  - ✔ 11,764

- **Subset 2**
  - 12,000
  - ✔ 11,764

- **Subset 3**
  - 12,000
  - ❌ 36,764

Will be inserted into the global candidate list

Only an excerpt, 5 attributes
Generate 31 permutations.
Candidate Merging

Result of Phase 2

<table>
<thead>
<tr>
<th>ID</th>
<th>EMAIL</th>
<th>PHONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>ID</td>
<td>NAME</td>
</tr>
<tr>
<td>COMPANY</td>
<td>ID</td>
<td>COMPANY</td>
</tr>
<tr>
<td>ORG</td>
<td>NAME</td>
<td>COMPANY</td>
</tr>
</tbody>
</table>
Layout Generation

- Generate all possible valid layouts from the result of phase 2
- Exhaustively explore all combinations
- A valid layout contains all attributes exactly once
## Layout Generation

The table illustrates different layouts for organizing data fields such as `ORG`, `EMAIL`, `PHONE`, `NAME`, `COMPANY`, and `ID`. The table shows the cost in 1000 for each layout configuration.

<table>
<thead>
<tr>
<th>Layout</th>
<th>Cost in 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORG EMAIL PHONE</td>
<td>27.7</td>
</tr>
<tr>
<td>ORG COMPANY EMAIL PHONE</td>
<td>28.2</td>
</tr>
<tr>
<td>NAME ORG EMAIL PHONE</td>
<td>28.2</td>
</tr>
<tr>
<td>NAME ORG COMPANY ID EMAIL PHONE</td>
<td>28.5</td>
</tr>
</tbody>
</table>

The layout configurations with the lowest cost are highlighted.

![Cost in 1000](image)
With huge numbers of attributes the scalability of the original algorithm degrades

Proposal: approximation that clusters frequently used attributes, by generating optimal sub-layouts for each cluster of primary partitions
EVALUATION
Sample Workload

- Mixed workload that is loosely based on the SAP Sales and Distribution scenario
  - Total benchmark size of 28 GB data
- 13 Queries
  - 9 OLTP Queries with typical CRUD operations
  - 3 OLAP-like Queries with high selectivity
  - 1 Planning like query with incrementally decreasing selectivity
- Layout Example – Input table are sales order headers
  - 3 containers: VBELN (id) is used by many different queries; (KUNNR, AEDAT) are evaluated as predicates together; third partition is accessed by “SELECT *” operators

<table>
<thead>
<tr>
<th>VBELN</th>
<th>...</th>
<th>KUNNR</th>
<th>...</th>
<th>...</th>
<th>AEDAT</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBELN</td>
<td></td>
<td>KUNNR</td>
<td></td>
<td>AEDAT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
HYRISE Workload Evaluation

- HYRISE uses 4x less cycles than the all row layout, and is about 1.6 times faster than the all column layout.
- Depending on the query weight, HYRISE’s advantage can vary.
HYRISE Workload Evaluation

- Strong tension between the layouts, since most of the times the hybrid layout can only be as good as one of them.
- The mixed workload increases the benefit of a hybrid layout.
- Hybrid layout is usually better than the comparable layout.

**Normalized CPU Cycles**

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>C</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q0</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>C</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In summary, HYRISE uses 8x less cycles than the all-f row layout.
HYRISE Layout Tension

- **Q6 (Insert)** – HYRISE has to update multiple containers, row must be better.
- **Q5 (Select)** – HYRISE clearly outperforms both approaches.

![Bar chart](image)
CONCLUSIONS
Conclusions

- Presented HYRISE
  - Main memory *hybrid database* for mixed (OLTP + OLAP) workloads
  - Novel algorithms to find optimal workload aware vertical partitioning
    - Using a highly accurate cache-miss based model
  - On SAP-based benchmark, 4x better than all rows and 60% better than all columns
- Come see HYRISE live at our Demo booth
THANK YOU