

The background of the slide is a blurred photograph of a modern, multi-story building with a glass facade and a large tree in front of it. The building is reflected in a body of water in the foreground.

Research and Implementation of Database Concepts

Kickoff

Martin Boissier, Thomas Bodner, Stefan Halfpap, Jan Koßmann, Dr. Michael Perscheid,
Dr. Daniel Ritter, Marcel Weisgut
Enterprise Platform and Integration Concepts

EPIC Teaching Activities

	Winter Term	Summer Term
Bachelor	Scalable Software Engineering (Lecture, 5 th term, revised SWT II)	Foundation of Business Software (Lecture, 4 th term)
	Bachelor's Project (5 th and 6 th term) Online Marketplace Simulation: A Testbed for Self-Learning Agents	
Master	Trends and Concepts in the Software Industry II (Seminar with Prof. Plattner and target-actual comparison of enterprise software at customers)	Trends and Concepts in the Software Industry I (Lecture with Prof. Plattner and industry partners)
	Trends and Concepts in the Software Industry III (Optional Project Seminar)	
	Data-driven Decision Support (Lecture and Project)	Causal Inference (Lecture and Project)
	Research and Implementation of Database Concepts (Research Seminar)	Build Your Own Database (Lecture and Project)
	Master Projects	
	Data-driven Decision Support	Autonomous Data Management

What to expect?

- Better understand how database systems work
- Learn how to familiarize yourself with a larger code base
- Work in small teams on a larger project

Same as in the
Develop your own Database
(DYOD) seminar

- Gain experience in systems development
- Improve your C++(20) skills

Less of a focus than in DYOD

- Research experience
- Related work, Conduct experiments, visualize results, communicate findings

New in this seminar

How does this relate to Develop your own Database?

- We found that thesis students often have little experience in communicating their results
- This seminar is supposed to be a „thesis light“, including literature research, implementation, designing and executing experiments, and presenting the results in speech and writing
- It is both suitable for those students who have taken DYOD and for those who have not
- BUT: No weekly meetings with the entire group, thus no DBMS/C++ introduction
 - Previous experience, e.g, from Trends and Concepts or the DBS lectures is helpful
 - DYOD slides and sprint documents are [available](#) if you want to read up on details
- More research-oriented, i.e., the projects are proposals, not full specifications

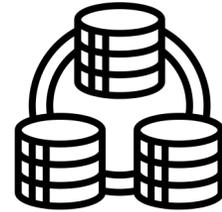
Who are we?

Cloud Data Management



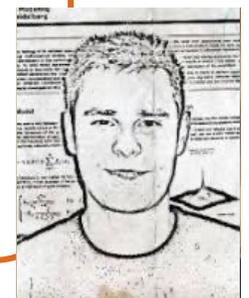
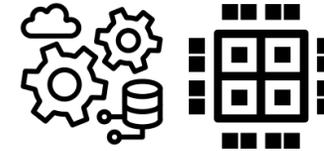
Thomas Bodner

Query-Driven
Data Allocation



Stefan Halfpap

In-Memory, Cloud DB &
Next Gen. Hardware



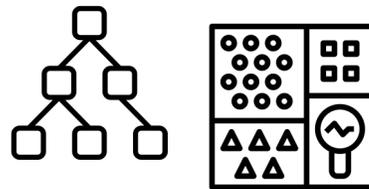
Daniel Ritter

Compression



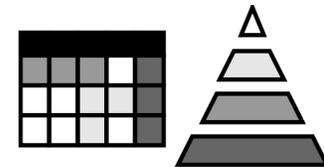
Martin Boissier

Unsupervised DB
Optimization



Jan Kossmann

Data Management on
Modern Storage Technol.



Marcel Weisgut

Hyrise

- An In-Memory Storage Engine for Hybrid Transactional and Analytical Processing
- HYRISE is a research database for the systematic evaluation of new concepts for hybrid transactional and analytical data processing on modern hardware
- Developed with and by HPI students
- Open Source (<https://git.io/hyrise>)
- System paper published at EDBT'19



- Modern, documented C++20 code base, 93% test coverage
- SQL interface, PostgreSQL network protocol
- Easy to extend via plug-in interface
- Supported benchmarks: TPC-(C|H|DS), JCC-H, Join-Order
- Runs on Intel, AMD, IBM Mainframe, ARM, Apple M1, Raspberry PI

Hyrise in three* pictures

```

multi-predicate joins are expensive, we do not want to create semi join reductions.
look at each predicate of the join independently. We can do this as a JoinNode's pre
Disjunctive predicates are currently not supported and if they were, they would b
join_predicates entry.
r (const auto& join_predicate : join_node->join_predicates()) {
const auto predicate_expression = std::dynamic_pointer_cast<BinaryPredicateExpres
DebugAssert(predicate_expression, "Expected BinaryPredicateExpression");
if (predicate_expression->predicate_condition != PredicateCondition::Equals) {
continue;
}

```

```

// Since semi join reductions might be beneficial for both sides of the join, v
// which can deal with both sides.
const auto reduce_if_beneficial = [&](const auto side_of_join) {

```

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Hyper										
Umbra	498	47	969							
MonetDB	1,258	119	676	725	821					
DuckDB	7,488	31	629	629	536	207	804	467	1,782	8
Hyrise Paper	4,874	2,083	816	1,185	1,505	196	665	435	1,938	57
Hyrise WIP	13,559	57	2,080	2,283	2,533	372	1,965	1,136	1,938	70
	6,369	50	1,719	1,796	3,579	723	4,832	2,217	1,252	
			976	643	2,854	50	1,404	2,217	32,681	3,37
						36	564	1,078	9,740	4,496
								422	6,712	2,282

Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms

Marcel Jankrift²
Potsdam, Germany

Rainer Schlosser¹



Hyrise Re-engineered: An Extensible Database System for Research in Relational In-Memory Data Management

Markus Dreseler, Jan Kossmann, Martin Boissier, Stefan Klauk,
Matthias Uflacker, Hasso Plattner
Hasso Plattner Institute
Potsdam, Germany
firstname.lastname@hpi.de

ABSTRACT

Research in data management profits when the performance evaluation is based not only on individual components in isolation, but uses an actual DBMS end-to-end. Facilitating the integration and benchmarking of new concepts within a DBMS requires a simple setup process, well-documented code, and the possibility to execute both standard and custom benchmarks without tedious preparation. Fulfilling these requirements also makes it easy to reproduce the results later on.

The relational open-source database Hyrise (VLDB, 2010) was presented to make the case for hybrid row- and column-format data storage. Since then, it has evolved from being a single-purpose research DBMS towards becoming a platform for various projects, including research in the areas of indexing, data partitioning, and non-volatile memory. With a growing diversity of topics, we have found that the original code base grew to a point where new experimentation became unnecessarily difficult. Over the last two years, we have re-written Hyrise from scratch and built an extensible multi-purpose research DBMS that can serve as an easy-to-extend platform for a variety of experiments and prototyping in database research.

In this paper, we discuss how our learnings from the previous version of Hyrise have influenced our re-write. We describe the new architecture of Hyrise and highlight the main components. Afterwards, we show how our extensible plugin architecture facilitates research on diverse DBMS-related aspects without compromising the architectural tidiness of the code. In a first performance evaluation, we show that the execution time of most TPC-H queries is competitive to that of other research databases.

1 INTRODUCTION

Hyrise was first presented in 2010 [19] to introduce the concept of hybrid row- and column-based data layouts for in-memory databases. Since then, several other research efforts have used Hyrise as a basis for orthogonal research topics. This includes work on data tiering [7], secondary indexes [16], multi-version concurrency control [42], different replication schemes [43], and non-volatile memories for instant database recovery [44].

Over the years, the uncontrolled growth of code and functionality has become an impediment for future experiments. We have identified four major factors leading to this situation:

- The lack of SQL support required query plans to be written by hand and made executing standard benchmarks tedious.
- Accumulated technical debt made it difficult to understand the code base and to integrate new features.

For these reasons, we have completely re-written Hyrise and incorporated the lessons learned. We redesigned the architecture to provide a stable and easy to use basis for holistic evaluations of new data management concepts. Hyrise now allows researchers to embed new concepts in a proper DBMS and evaluate performance end to end, instead of implementing and benchmarking them in isolation. At the same time, we allow most components to be selectively enabled or disabled. This way, researchers can exclude unrelated components and perform isolated measurements. For example, when developing a new join implementation, they can bypass the network layer or disable concurrency control.

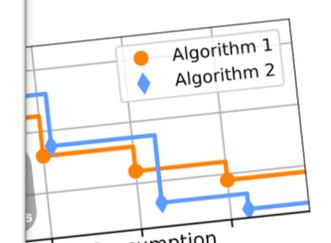
In this paper, we describe the new architecture of Hyrise and how our prior learnings have led to a maintainable and comprehensible database for researching concepts in relational in-memory data management (Section 2). Furthermore, we present a plugin concept that allows testing different optimizations without having to modify the core DBMS (Section 3). We compare Hyrise to other database engines, show which approaches are similar, and highlight key differences (Section 4). Finally, we evaluate the new version and show that its performance is competitive (Section 5).

1.1 Motivation and Lessons Learned

The redesign of Hyrise reflects our past experiences in developing, maintaining, and using a DBMS for research purposes. We motivate three important design decisions.

Decoupling of Operators and Storage Layouts. The previous version of Hyrise was designed with a high level of flexibility in the storage layout model: each table could consist of an arbitrary number of containers, which could either hold data (in uncompressed or compressed, mutable or immutable forms) or other containers with varying horizontal and vertical spans. In consequence, each operator had to be implemented in a way where it could deal with all possible combinations of storage containers. This made the process of adding new operators cumbersome and led to a system where some operators made undocumented assumptions about the data layout (e.g., that all partitions

distributed equally



Storage Consumption
Dimensions need to be considered
index selection algorithms.

indexes [45]. Third, it is challenging to estimate the impact of an index on the workload of indexes and actually running queries, as they are inherently inaccurate [15]. Performance enhancements by indexes are often not reflected in the complexity of the problem of research work in this area. Early in the 1970s [29, 41]. Since then, many approaches, based on different approaches, have tried to optimize index configurations. The difference in calculation time, solution quality, and the method for cost estimation. In our knowledge, there is no comparison of algorithms in different dimensions, e.g., container budgets, the algorithms' runtimes, and loads (see Figure 1). For example, Schmitt presents a specific benchmark for online algorithms to evaluate two index selection algorithms and evaluate their performance across multiple dimensions [44]. In contrast, in this paper, we present a specific benchmark for online algorithms' performance across multiple dimensions, which presents new selection algorithms. For these reasons, we developed a publicly available evaluation platform.

Comparison

	Hyrise	Skyrise
Target workload	HTAP on hot to warm data	Interactive OLAP on cold data
Dataset size sweetspot	Gigabytes to a few Terabytes	Gigabytes to (hundreds of) Terabytes
Architecture	Scale-up within large bare metal machines	Independent scale-out of decoupled FaaS-based compute and cloud object storage
Pricing model	Pay upfront for machine and provisioning, pay as you go for maintenance and energy	Cloud object storage is \$23/TB/month, pay as you go per query, as an example TPC-H Q1 @ SF1000 is currently \$0.16

Research Topics

1. In-Memory Pipelined Query Execution
2. Analyzing Traces of Serverless Query Execution
3. Incorporating Distributed Plans into Query Optimization
4. Learned Indexes on Dynamic Data
5. Efficient and Accurate Histograms
6. Database Node Placement in the Cloud
7. Partial Indexes
8. Dynamic Data Placement Algorithms

In-Memory Pipelined Query Execution

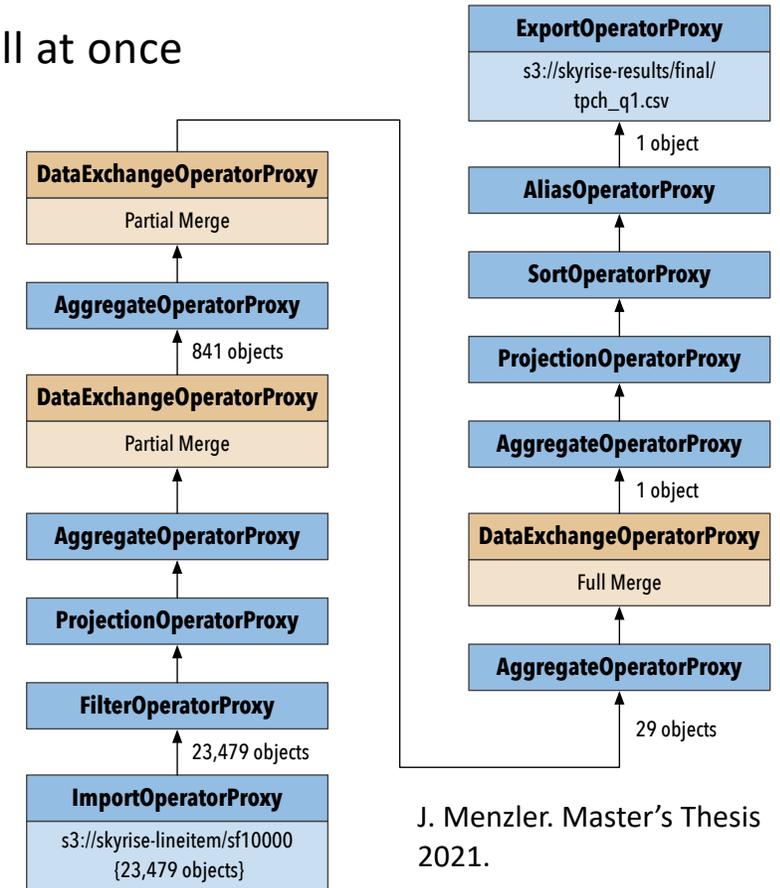
- Initially, Skyrise adopted Hyrise's materialized execution model

- Each operator consumes its input all at once and then produces its output all at once
- Easy to reason about and only option for cross-worker processing
- Intermediate query execution results may exhibit large footprint
- No opportunity for parallelism along query pipelines of operators

- We study a hybrid materialized/pipelined execution model

- Workers have little memory capacity and run single query pipeline each
- Intermediates are materialized across and pipelined within workers
- Extend operator set (import, filter, projection, ..) to work on „chunks“
- Analyze worker main memory usage and query pipeline runtimes

- Prior experience with cloud services beneficial



Analyzing Traces of Serverless Query Execution

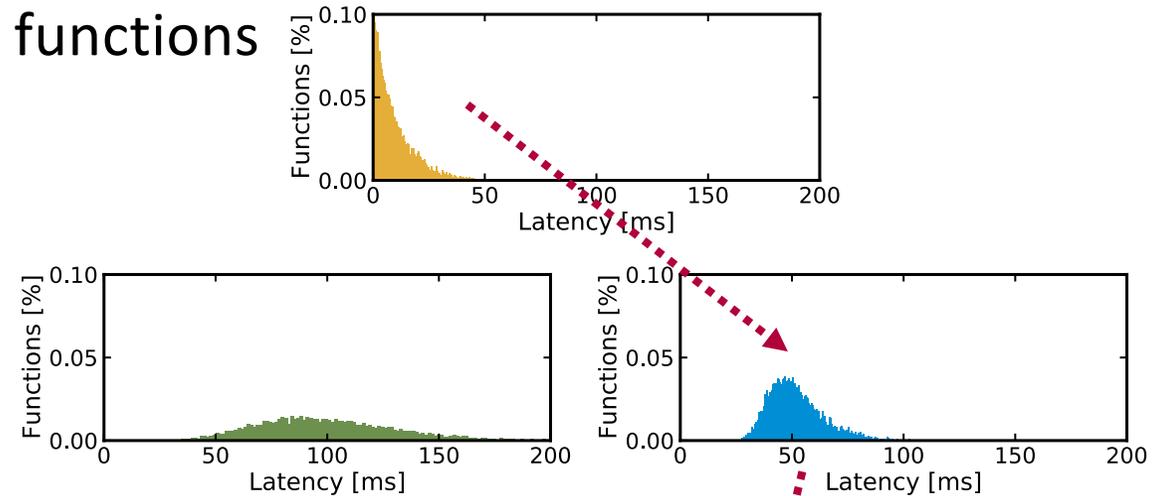
- Skyrise executes queries in parallel across cloud functions

- Cloud functions run pipelines of query operators
- Concurrency to several thousand function invocations
- Skyrise inherits properties of FaaS platform, i.e., elastic scalability, reliability, performance and security isolation across/within queries
- Skyrise also inherits the observability issue, rendering debugging and profiling cumbersome

- Skyrise collects a multitude of runtime data

- Operator and operator step transitions, timings, throughput, costs, ..
- Aggregate data and make it consumable for debugging or profiling

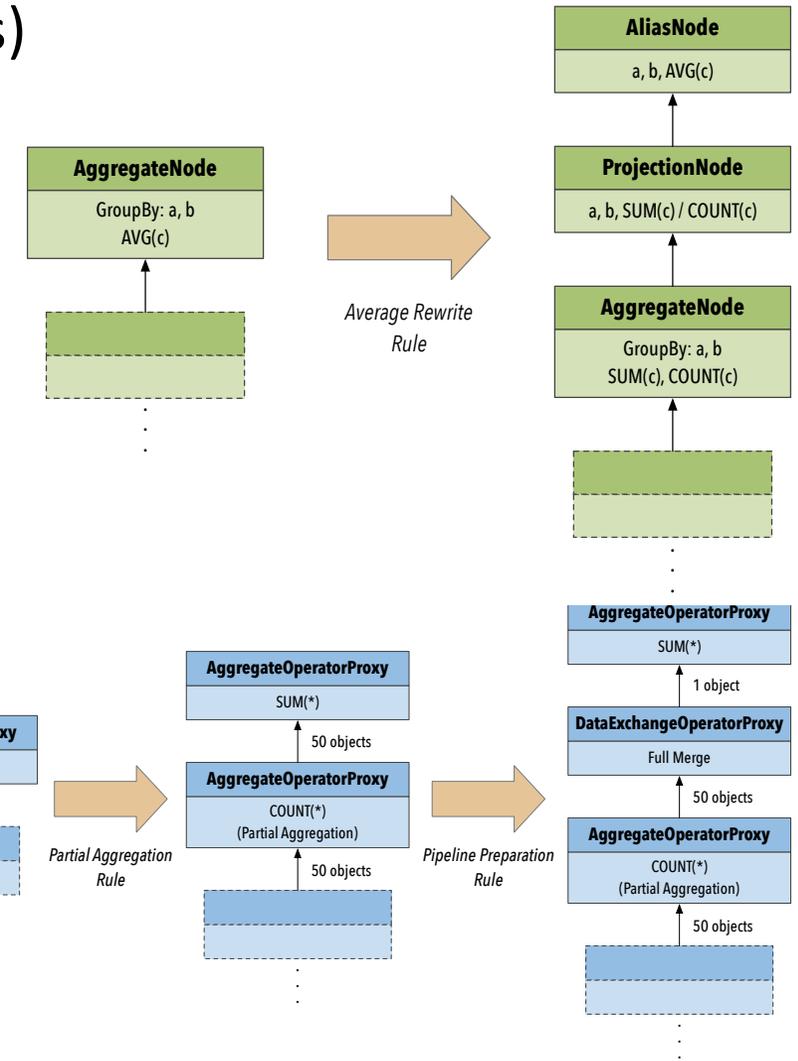
- Prior experience with cloud (function) services beneficial



```
[skyrise> Run q=6 sf=10000 w=2500
[ DONE ] List database table partitions in S3 SF10000
[ DONE ] Building PQP
[ DONE ] 100% of workers done
Query result:
+-----+
|          revenue          |
+-----+
| 1233162480045.8884 |
+-----+
Query runtime: 36880 ms
Query cost:    1.1771 $
              Lambda: 1.0966 $      32831529ms x 2048 MB
              S3:    0.0805 $      2551xPut and 169427xGe
```

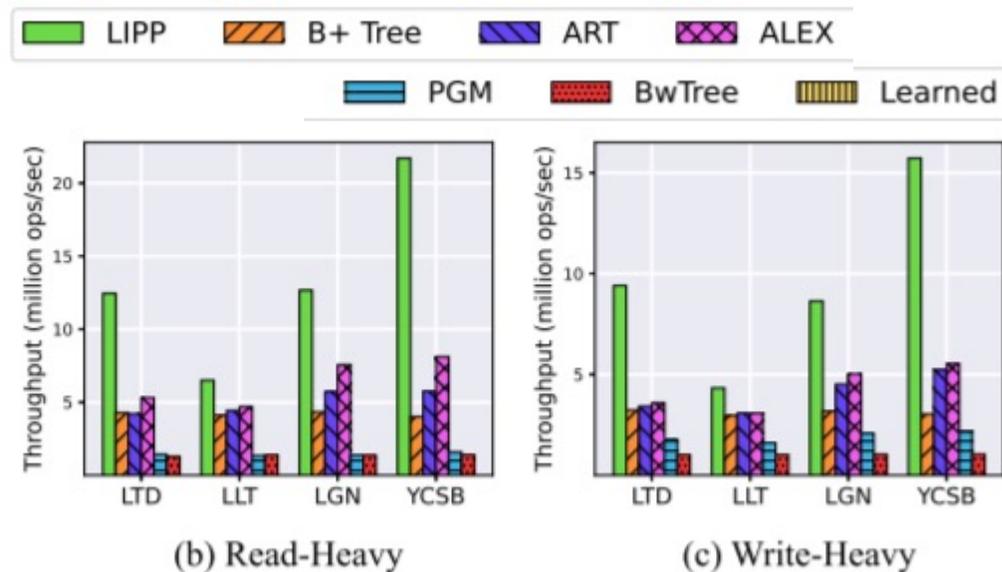
Incorporating Distributed Plans into Query Optimization

- FaaS Platforms offer massive parallelism (>10,000s of workers)
- To exploit underlying parallelism, Skyrise optimizer must be aware of data partitioning, distribution, and shuffling
- Extend rule sets for both logical and physical query plans, for now based on heuristics
- Systematically evaluate individual effects and interplay
- Prior experience with cloud services beneficial



Learned Indexes on Dynamic Data

- Learned indexes (LIs) with better **performance** than common tree indexes
- LIs supporting **dynamic** data: **PGM**, **ALEX**, **LIPP**
- Datasets (from ALEX paper + new String data)
- Assessment criteria: **index** lookup times, throughput, size; **construction** time + memory



- Tasks:
 - Understand + run dynamic, open source LIs
 - Reproduce results on Integers (compare btree)
 - Extend for further **data types** (e.g. Strings → INT)
 - Select / generate data type-specific datasets
 - Benchmark on String datasets
 - (Stretch: integrate LIs into Hyrise + benchmark)
- Learning potentials:
 - ML-techniques in databases
 - Indexing data
 - Benchmarking
 - (Hyrise index integration)

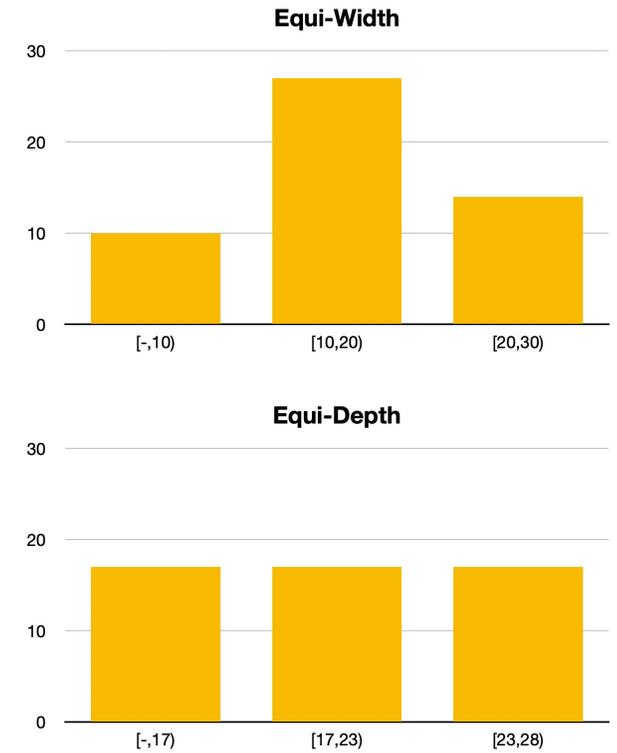
Efficient and Accurate Histograms

Motivation

- Histograms are database statistics that allow the query optimizer to find efficient and fast query plans
- Improving the accuracy of histograms can have a large positive impact as inefficient query plans are often recognized and avoided
- However, creating and maintaining histograms can be expensive

Current Situation in Hyrise

- Hyrise builds histograms for the entire column
 - Building histograms for a +1TB data set can take hours, even with 240 cores
- The currently used histograms can be inaccurate when data is heavily skewed (often the case in the real world)



Efficient and Accurate Histograms

Goal

- Enable Hyrise to efficiently create histograms for large data sets
- Improve cardinality estimations by using skew-aware histogram types

Implementation

- Implementation of text book histograms (e.g., equi-width) and max-diff histogram [Hist96]
- Creation of histograms using stable sampling
- Efficient implementation for data sets > 1TB on large server (240 cores and 8 sockets)

Evaluation

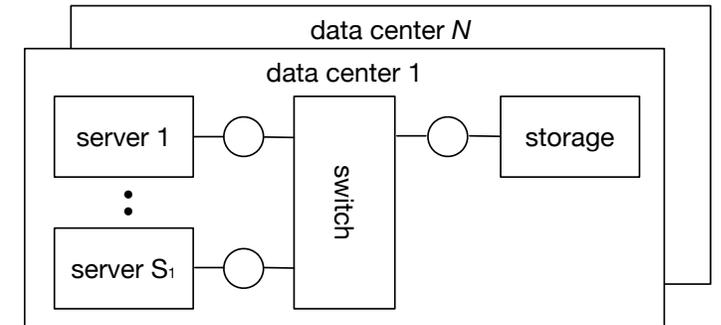
- Evaluation on synthetic (TPC-H) and real-world (IMDB movie data) data sets
 - What is the accuracy of the evaluated histograms?
 - How efficient is their creation?

Expected Results

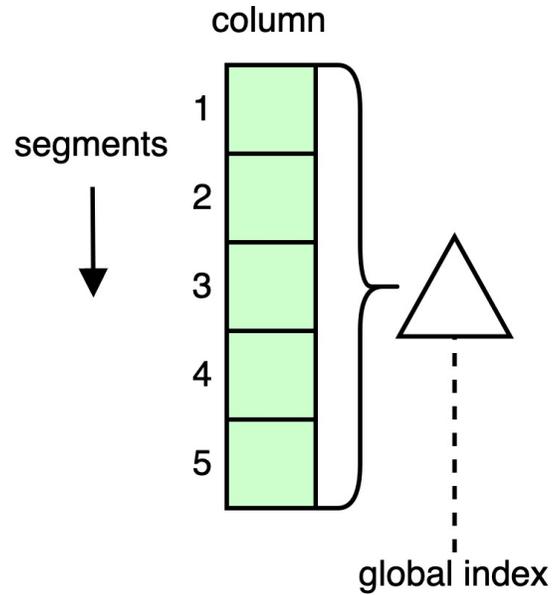
- Thorough implementation and evaluation of different histograms types
- For the histogram type that performs best: efficient and scalable implementation that can ideally find its way to the Hyrise main branch

Database Node Placement in the Cloud

- **Motivation: Database systems** are increasingly deployed **in the cloud**
- **Problem:** Optimize the **assignment of** (database system) **VMs to physical resources** under **constraints**
 - Problem size: hundreds of servers and thousands of VMs
 - Exemplary VM settings:
#cores & speed, RAM, storage affinity & anti-affinity rules
 - Exemplary server settings:
#cores & speed, RAM, connectivity
- **Task: Implement and evaluate allocation algorithms** (greedy vs. linear programming based)
- **Learning goals** (specific to this topic):
 - Approaches for solving **optimization problems**, in particular, **linear programming**
 - Characteristics of the architecture of **cloud data centers**



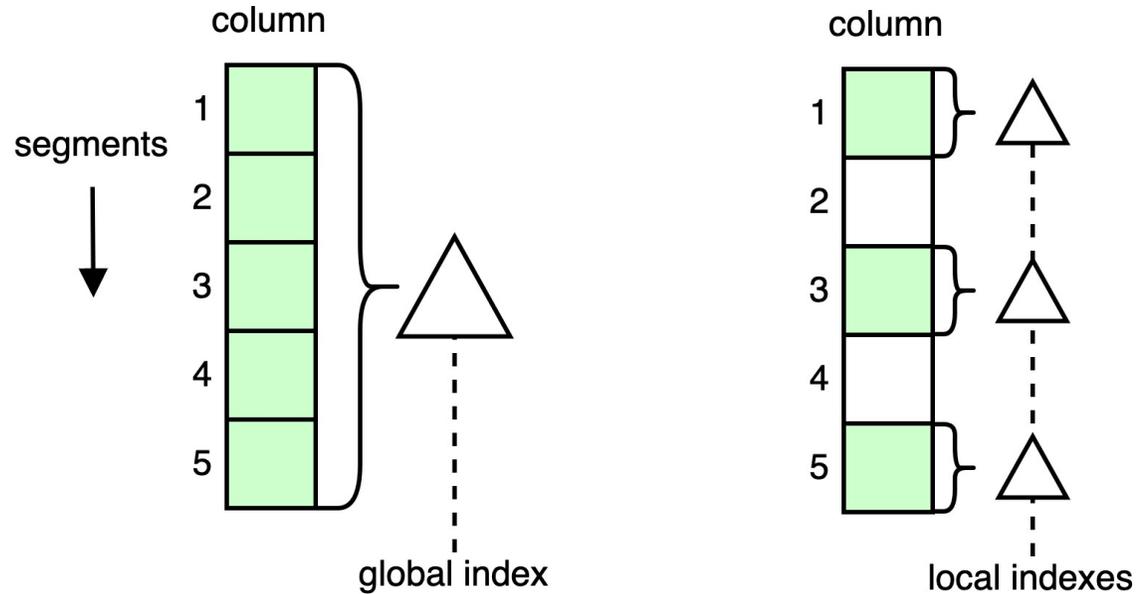
Partial Indexes



Memory Consumption Issue

Indexing all tuples of a table results in a high memory footprint.

Partial Indexes



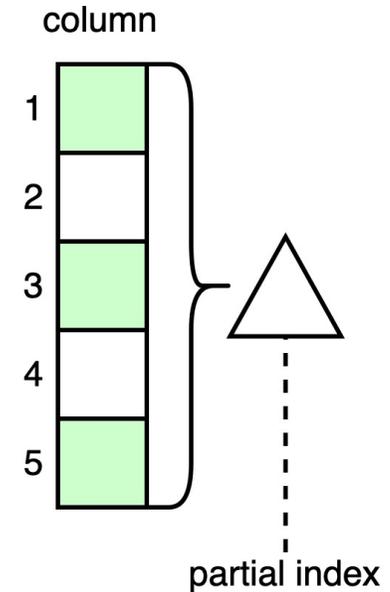
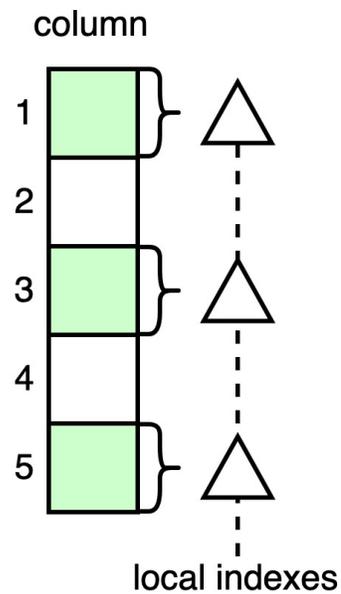
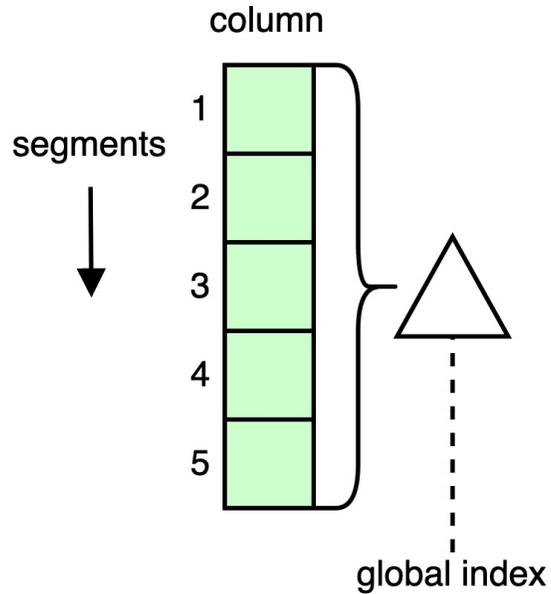
Memory Consumption Issue

Indexing all tuples of a table results in a high memory footprint.

Scalability Issue

The number of lookup operations grows linearly with the number of existing partitions.

Partial Indexes



Memory Consumption Issue

Indexing all tuples of a table results in a high memory footprint.

Scalability Issue

The number of lookup operations grows linearly with the number of existing partitions.

Partial Indexing

- Store index entries of multiple partitions in one global data structure.
- Only a subset of the partitions is indexed.

Partial Indexes

Implementation

- (Partial hash index) – majority implemented in DYOD 21
- Partial B-Tree index
- Index scan operator (currently not compatible with PI)
- Index join operator: fallback join for non-indexed partitions
- Micro benchmarks
- Optimizer rules to use index scans/joins

Evaluation

- Latencies of index lookup operations
- Latencies of index maintenance operations
- Index memory consumption
- Using various benchmarks (micro, TPC-H, JCC-H)
- Performance effects of implemented/modified optimizer rules

Expected Results

- Index implementations (hash and B-Tree)
- Partial index compatible index scan implementation
- Partial index compatible index join implementation
- Optimizer rules to use index scans/joins
- Experimental performance evaluation of partial indexes in comparison to global indexes (used in scans and joins)
- Experimental performance evaluation of Hyrise using the new optimizer rules (using TPC-H and JCC-H)

Dynamic Data Placement Algorithms

Motivation

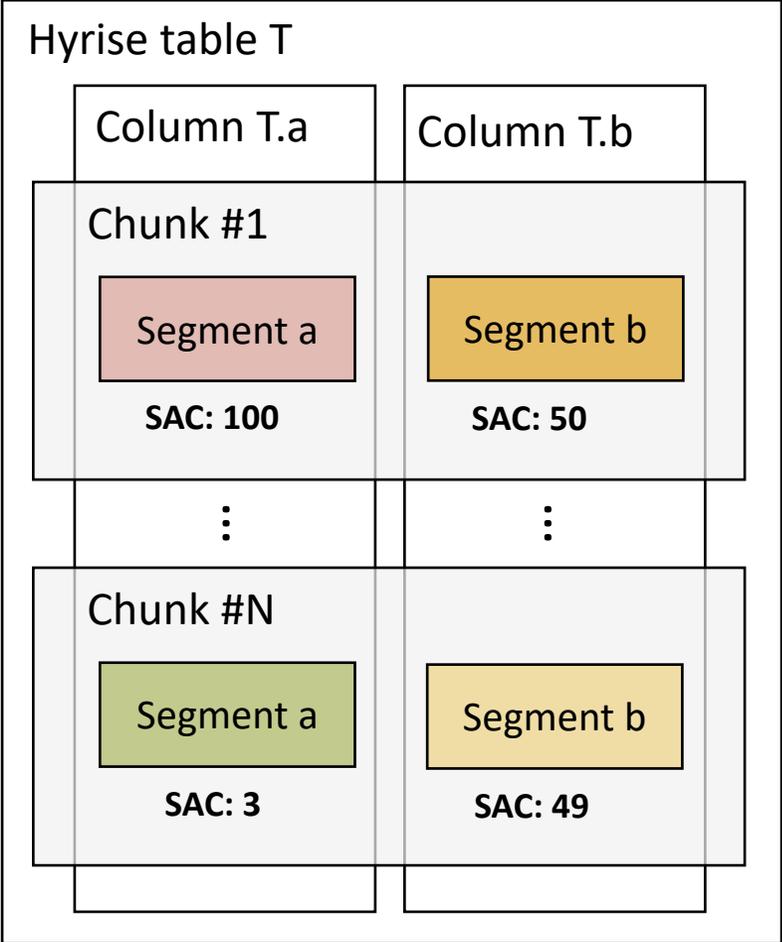
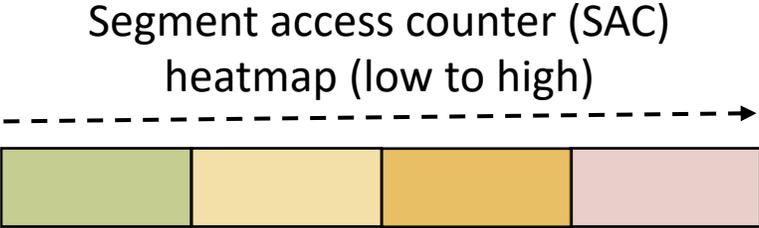
- Storing data in DRAM allows significantly lower access latencies compared to other data tiers, such as SSDs or HDDs
- DRAM in main-memory databases is limited:
“[...] the amount of data to be processed keeps growing while DRAM capacity does not” [1]
- To tackle this issue, data can be placed on different data tiers, such as SSDs.

Guiding Questions

- Which data (structures) should be placed on the slower data tier?
- Given a DRAM budget and a workload, what is an optimal data placement?

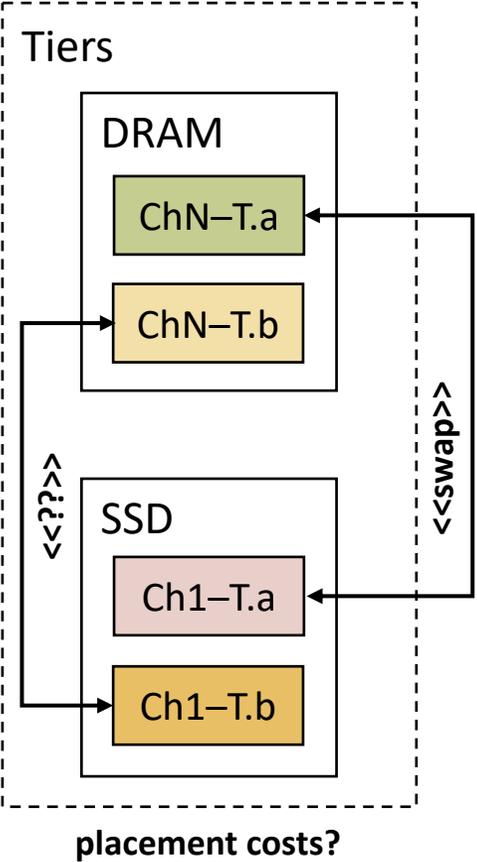
Dynamic Data Placement Algorithms

Hyrise automatic tiering



tiering plugin

<tier>-CAP: X GB,
<tier>-COST: Y USD



Dynamic Data Placement Algorithms

Implementation

- Algorithms that determine the optimum data placement for a given workload, having DRAM or SSD as the data tiers (as a Hyrise plugin).
- Micro benchmarks for manual data placement experiments.

Evaluation

- Manual data placements with different shares of segments stored on DRAM/SSD
- TPC-H performance with (a) all segments are stored in main memory, (b) all segments stored on SSD, and (c) segments stored on both DRAM and SSD, according to the developed algorithms
- Metrics: query latencies, memory consumption
- Compare the developed algorithms with a provided reference algorithm

Expected Results

- Different data placement algorithm implementations
- Experimental performance evaluation with segments manually placed on DRAM/SSD
- Experimental performance eval. of the data placement algorithms compared to a reference algorithm

Timeline



Administration

- Specialization areas:
 - ITSE: BPET, OSIS, ITSE-Analyse, ITSE-Maintenance
 - DATA: SCAL
- Official deadline to register was 22 October
- Grading
 - 50% project result and presentation
 - 40% scientific report (4-8 pages ACM format, depending on group size)
 - 10% personal engagement

Bringing groups and topics together

- You are welcome to hang out in this Zoom call after the introduction to figure out groups
- If you have found a topic (and a group), please mail Jan.Kossmann@hpi.de and Daniel.Ritter@guest.hpi.de
 - Include three (or more) topic preferences
 - The assignment algorithm is strategy-proof ;)
- If you have any questions or are still looking for a group partner, please mail us, too