Research and Implementation of Database Concepts

Introduction for Winter Term 2022/23

Thomas Bodner, David Justen, Daniel Lindner, Martin Boissier, Stefan Halfpap, Dr. Michael Perscheid
Enterprise Platform and Integration Concepts Group

EPIC Teaching Activities

	Winter Term	Summer Term
Bachelor	**Scalable Software Engineering** (Lecture, 5th semester, revised SWT II)	**Foundations of Business Software** (Lecture, 4th semester)
	Bachelor's Project (5th and 6th semester)	
Master	**Trends and Concepts in the Software Industry II** (Seminar with Prof. Plattner and 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)	**Develop Your Own Database** (Lecture and Project)
	Master Project	

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) lecture and project

- Gain experience in systems development
- Improve your C++ skills

Less of a focus than in DYOD

- Research experience
- Study related work, conduct experiments, visualize results, communicate findings

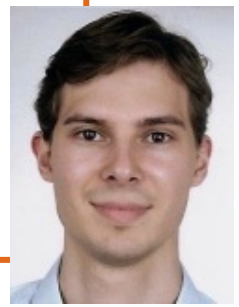
New in this research seminar

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

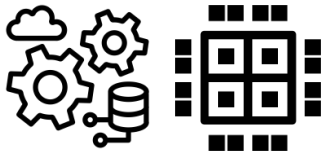
Who are we?

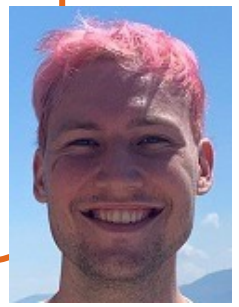
Elastic
Query Processing



Thomas Bodner

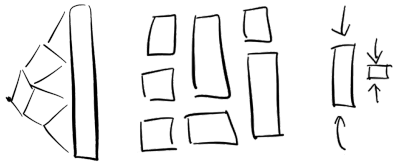
Robust
Query Processing

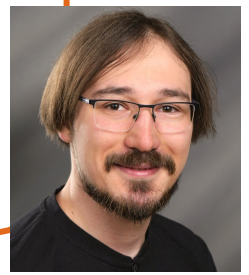




David Justen

Cost Models
for Database Tuning



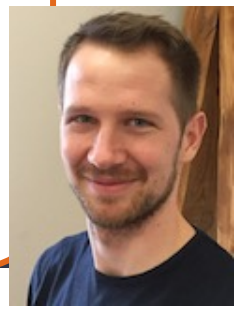


Daniel Lindner

Footprint Reduction

Martin Boissier

Workload Distribution



Stefan Halfpap

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

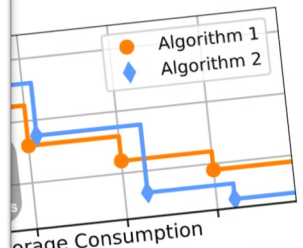
```
s multi-predicate joins are expensive, we do not want to create semi join reductions
look at each predicate of the join independently. We can do this as a JoinNode's pre
Disjunctive predicates are currently not supported and if they were, they would b
join_predicates entry.
r (const auto& join_predicate : 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,
// 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	Q1
Hyper	**498**	47	969	725	821	207	804	467	1,782	8
Umbra	1,258	119	**676**	**629**	**536**	196	665	435	1,938	**57**
MonetDB	7,488	**31**	816	1,185	1,505	372	1,965	1,136	**1,252**	70
DuckDB	4,874	2,083	2,080	2,283	2,533	723	4,832	2,217	32,681	3,37
Hyrise Paper	13,559	57	1,719	1,796	3,579	50	1,404	1,078	9,740	4,49
Hyrise WIP	6,369	50	976	643	2,854	**36**	**564**	**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 Jankrift[2] Rainer Schlosser[1]

tributed equally



orage Consumption

mensions need to be considered
index selection algorithms.

xes [45]. Third, it is challenging to
impact of an index on the workload
indexes and actually running queries,
s are inherently inaccurate [15].
performance enhancements by indexes
iation with the complexity of the prob-
of research work in this area. Early
the 1970s [29, 41]. Since then, many
s, based on different approaches, have
nd optimized index configurations. The
iffer in calculation time, solution quality,
a, and the method for cost estimation.
r knowledge, there is no comparison of
orithms in different dimensions, e.g., con-
orage budgets, the algorithms' runtimes,
oads (see Figure 1). For example, Schnait-
present a specific benchmark for online
d evaluate two index selection algorithms
kloads [44]. In contrast, in this paper, we
gorithms for three workloads and focus on
gorithms' performance across multiple di-
existing work, which presents new selection
ally contains comparisons, these are often
g the evaluated dimensions and choice of
thms.
we compare and evaluate eight existing al-
e index selection problem: [4, 9, 10, 14, 26,
interpret their results in different scenarios.
gation should (i) be reproducible, (ii) offer the
d further algorithms, database workloads, and
ms in the future, as well as (iii) automate the
acilitate practical use for database researcher
rators. For these reasons, we developed a
d publicly available evaluation platform.

open proceedings

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

Markus Dreseler, Jan Kossmann, Martin Boissier, Stefan Klauck,
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

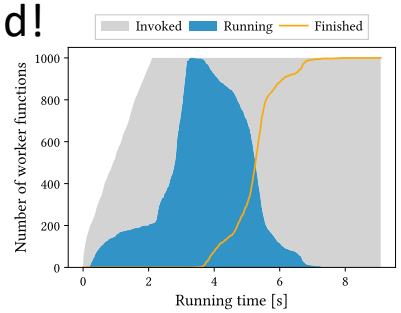
The redesign of Hyrise reflects our past experiences in developing, maintaining, and using a DBMS for research purposes. We motivate three important design decisions.

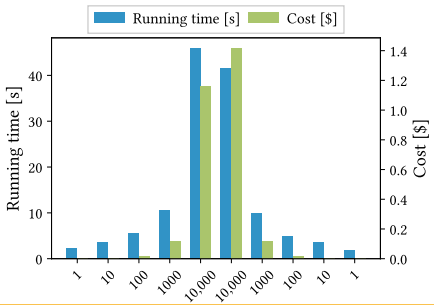
Decoupling of Operators and Storage Layouts. The previous version of Hyrise was designed with a high level of flexibility in the storage layout model: each table could consist of an arbitrary number of containers, which could either hold data (in uncompressed or compressed, mutable or immutable forms) 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

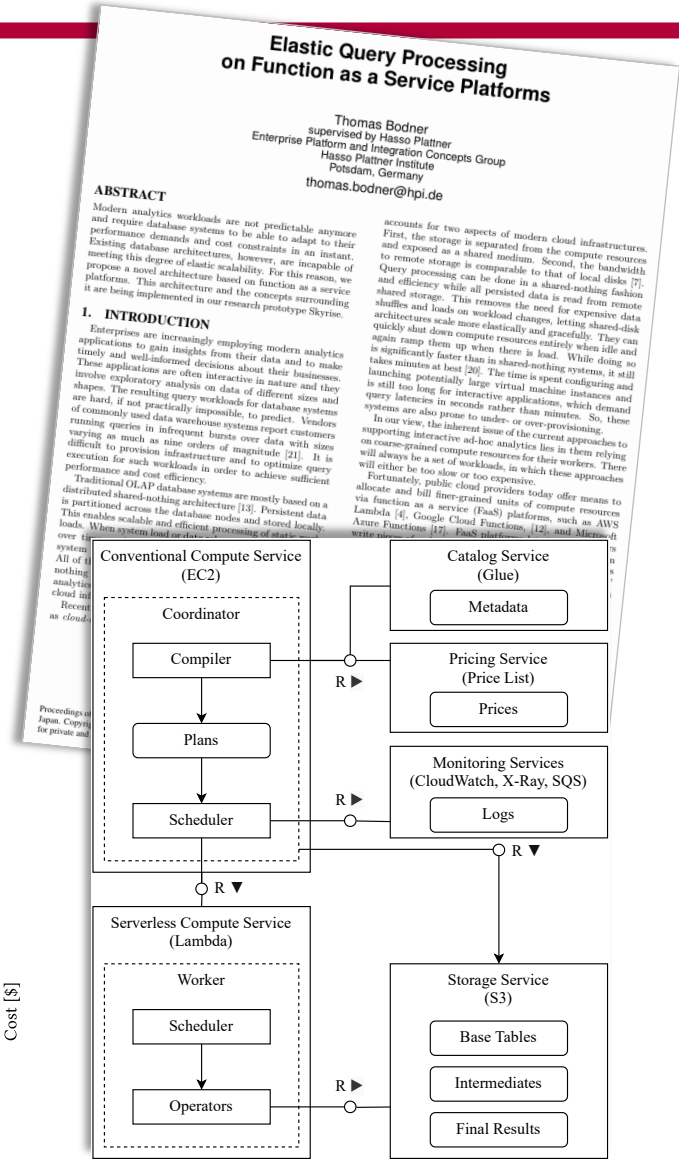
HPI Hasso Plattner Institut

Skyrise

- A serverless query processor for interactive in-situ analytics on cold data
  - **Serverless:** Built on function as a service platforms and object storage
  - **(SQL) query processor**: Relational query execution and optimization
  - **Interactive:** Aims at query latencies in seconds
  - **In-situ:** Processes data without upfront load/align/sort/compress/index/..(ing)
  - **Cold data:** Infrequently accessed TB/PB-scale historical, IoT and Web data
- Initiated in fall 2019 to explore modern cloud infrastructure for databases
  - Exploits scalability, elasticity and reliability of the cloud, deals with its challenges
  - Modern C++ (17), documented, tested (> 90% coverage) codebase
  - Just starting out, plenty of research ahead!
  - Vision paper published at VLDB ’20
  - 3x Master’s theses, 2x seminar papers







Comparison

	Hyrise	Skyrise
Target workload	HTAP on hot to warm data	Interactive OLAP on cold data
Dataset size sweetspot	Gigabytes	Gigabytes to (tens 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. Dynamic Data and Data Dependency-based Optimization
2. Efficient and Accurate Histograms
3. Interactive Database Index Selection and Evaluation
4. Serverless Data Shuffle with Long-standing Resources
5. Understanding Serverless Query Execution
6. Extending Serverless Query Execution with Custom Code

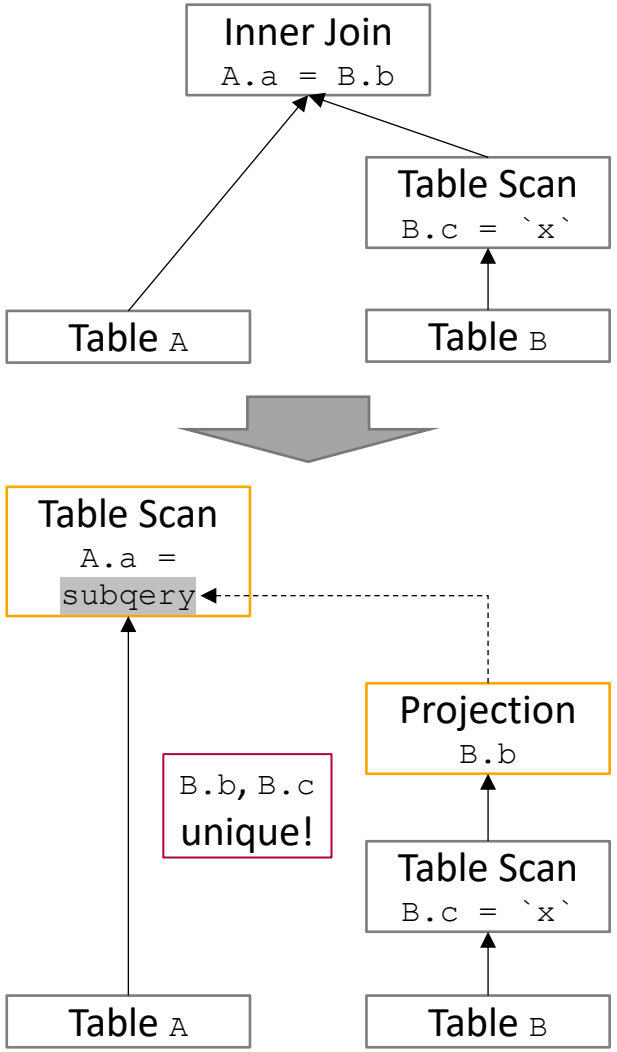
Dynamic Data and Data Dependency-based Optimization

Motivation

- Interrelations in data, especially so-called *data dependencies*, can be used to optimize queries
- Expensive operations, e.g., joins and aggregates, can be simplified and performance increases
- However, data dependencies can be invalidated by updated or inserted data, leading to incorrect query results

Current Situation in Hyrise

- Hyrise supports three data dependency-based query rewrite techniques and efficient automated discovery of *UCCs* [1]
- Currently, there is no mechanism for re-validating data dependencies when data changes



[1] Kossmann et al. 2022. *Workload-driven, Lazy Discovery of Data Dependencies for Query Optimization.* CIDR.

Dynamic Data and Data Dependency-based Optimization

Goal

- Enable automated re-validation of data dependencies when data changes
- Ensure correct query results regardless at any time

Implementation

- Mechanism for recognizing invalidated UCCs
- Implementation of an incremental validation approach exploiting Hyrise's append-only data layout (cf. [2])
- Driver to decide on when to perform incremental/full data dependency re-validation

Evaluation

- Evaluation on synthetic (TPC-H) and real-world (IMDB movie data) data sets
  - Is incremental validation beneficial faster than the current validation strategies?
  - What is the overhead of re-validation given different update frequencies?

Expected Results

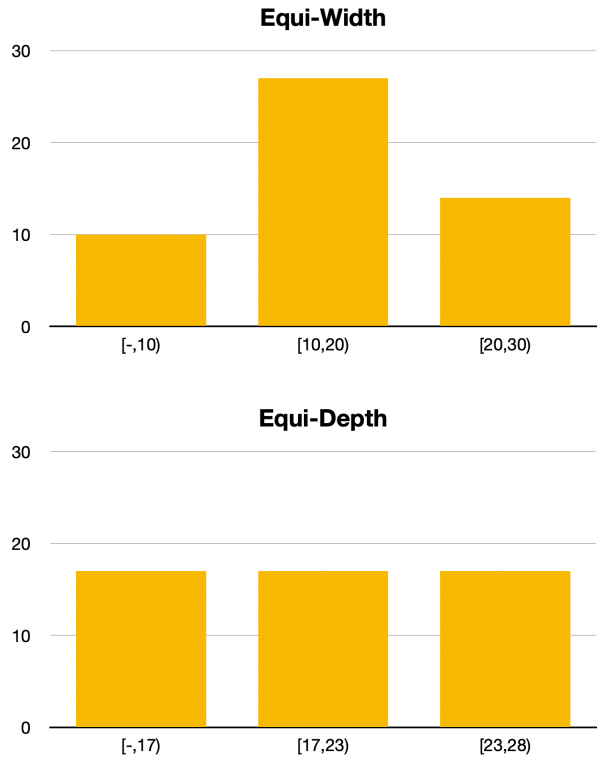
- Thorough implementation and evaluation of an incremental UCC validation algorithm
- Means to recognize that a UCCs is not valid anymore (e.g., using transaction IDs) that can ideally be merged into Hyrise's master branch

[2] Abedjan et al. 2014. *Detecting Unique Column Combinations on Dynamic Data.* ICDE, pp. 1036–1047.

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

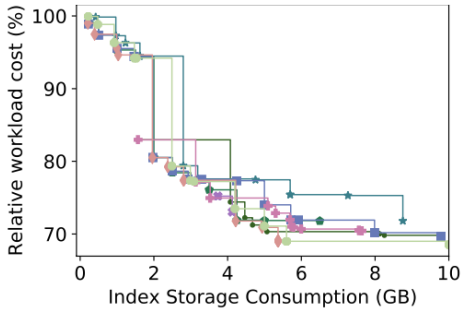
- Thorough implementation and evaluation of different histograms types
- For the histogram type that performs best: efficient and scalable implementation that can ideally find its way to the Hyrise main branch

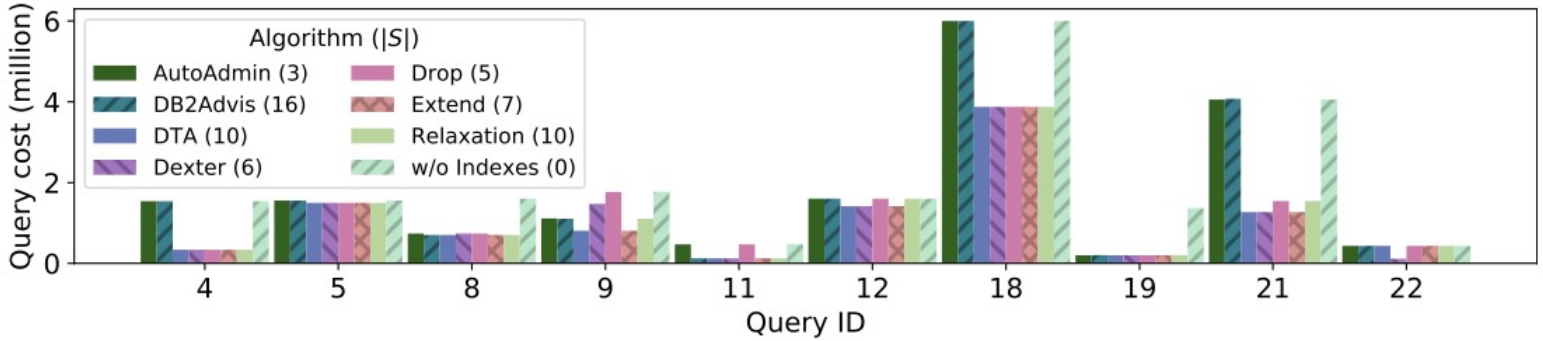
[Hist96] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, Eugene J. Shekita:
Improved Histograms for Selectivity Estimation of Range Predicates. SIGMOD 1996: 294-305

Interactive Database Index Selection and Evaluation

Motivation

- Index selection is an essential database optimization problem
- Various approaches find different solutions [1], which are difficult to compare, because of index sizes, index applicability to different queries, index interaction
- Current evaluations focus on overall (i.e., aggregated) results and effects of changes to an existing selection are unclear





Goal

- Build on existing index evaluation platform [2] (9 algorithms; PostgreSQL+HypoPG; TPC-H, TPC-DS, JOB)
- Compare and evaluate index selection results of different algorithms on a per index and query level
- Enable interactive exploration of index selections

[1] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, Rainer Schlosser: Magic mirror in my hand, which is the best in the land? An Experimental Evaluation of Index Selection Algorithms. Proc. VLDB Endow. 13(11): 2382-2395 (2020)

[2] https://github.com/hyrise/index_selection_evaluation

Serverless Data Shuffle with Long-Standing Resources

- Problem
  - Serverless query processors (SQP) shuffle data on auto-scaling storage (e.g., Amazon S3) to enable distributed processing
  - These storage services are pay-per-use and elastic but have high request costs, high latency, and strict request rate limits
  - Using these services makes data shuffles slow and expensive

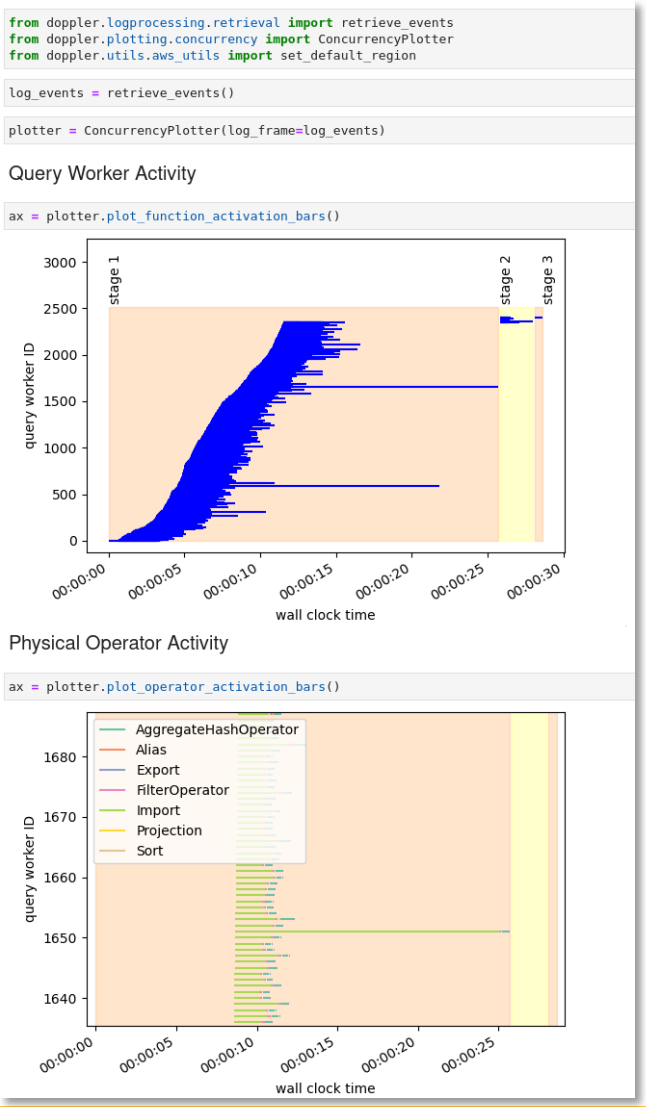
- Approach
  - Integrate our in-house SQP Skyrise with Jiffy, an elastic key-value store for self-hosting
  - Examine the potential for a system with serverless compute but self-hosted (serverful) ephemeral storage

- Leading Questions
  - How does data shuffling on self-hosted, in-memory KV stores compare to serverless, auto-scaling storage price- and performance-wise?
  - For which workloads is it viable to start Jiffy clusters on-demand?
  - What are the inherent trade-offs on deciding whether to shuffle data on serverless or self-hosted storage?

Justen, Cost-efficiency and Performance Robustness in Serverless Data Exchange, SIGMOD 2022 SRC.
Khandelwal. Jiffy: Elastic Far-Memory for Stateful Serverless Analytics. EuroSys 2022.

Understanding Serverless Query Execution

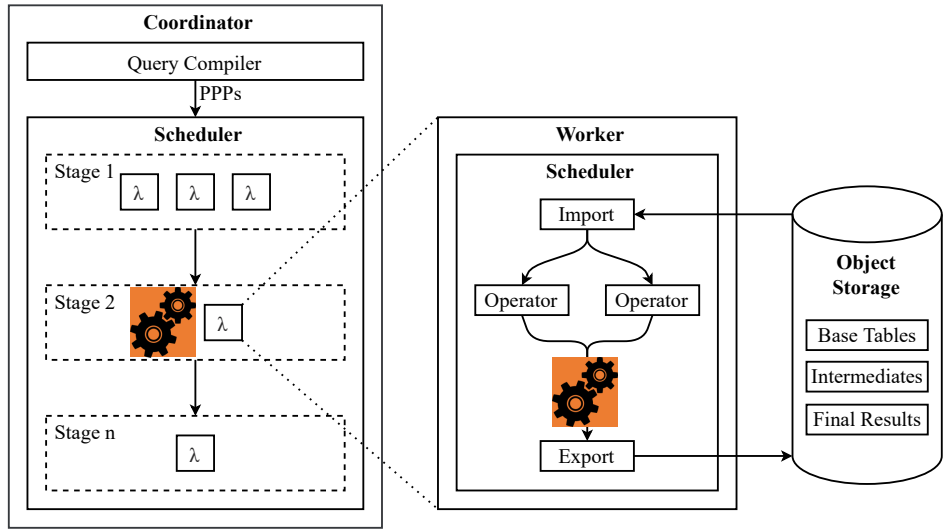
- Skyrise executes queries distributed across cloud functions
  - Cloud functions run pipelines of query operators
  - Concurrency to several thousand function invocations
  - Skyrise inherits properties of FaaS platform, i.e., elastic scalability, reliability, performance and security isolation across/within queries
  - Skyrise also inherits the observability issue, difficult debugging and profiling
- Skyrise monitoring subsystem traces and analyzes queries
  - Straggling query workers, fatal failures, function concurrency issues
  - Many involved scenarios are not covered and need attention
  - Storage leaks across the storage hierarchy, service deadlocks, ..



F. Engel. Straggler Mitigation in Distributed Query Execution on Cloud Functions. Master's Thesis 2021.
Bodner et al., Doppler: Understanding Serverless Query Execution, SIGMOD 2022 BiDEDE Workshop.

Extending Serverless Query Execution with Custom Code

- Database systems offer their users ways to run application logic close to the data
- This often happens in the form of so-called user-defined functions or aggregates (UDFs/UDAs)
- You will explore how to integrate UDFs into Skyrise:
  - What should be the language, API/ABI?
  - What should be the wrapping primitive (query, pipeline, operator, ..)?
  - How should deployment work?



Bodner, Elastic Query Processing on Function as a Service Platforms, VLDB 2020 PhD Workshop.
Dean et al., MapReduce: Simplified data processing on large clusters. OSDI 2004.

Timeline


17 Oct 2022
Introduction
23 Oct 2022 by end of day
Submit (your group and) topic preferences
24 Oct 2022
Announcement of topic assignments
Between 24 Oct and 7 Nov 2022
First meeting with your supervisor(s) based on individual arrangement
Weekly meetings with advisors
6 Mar 2022
Final presentations – 20 min + 10 min Q&A
17 Mar 2022
Submission of written report (4 to 8 pages)

Administration

- Specialization areas:
  - IT-Systems Engineering MA: BPET; OSIS
  - Data Engineering MA: SCAL (PO 2018); DASY (PO 2022)
  - Digital Health MA: SCAD
  - Software Systems Engineering MA: SSE-D; SSYS; DSYS
- Official deadline to register is 31 October 22
- 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 email Thomas.Bodner@hpi.de
  - Include names of group members
  - Include three prioritized topic preferences
- If you have any questions or are still looking for a group partner, please mail us, too

Background reading: https://en.wikipedia.org/wiki/Strategyproofness
https://en.wikipedia.org/wiki/Gale%E2%80%93Shapley_algorithm