Adaptive Optimizations for Databases

Christoph Anneser
Technical University of Munich

21.05.2024
Introduction

https://ourworldindata.org/
Introduction

https://ourworldindata.org/
Introduction

https://ourworldindata.org/

Introduction

Introduction

Adaptive Hybrid Indexes

AutoSteer

Programming Model

Conclusions
Introduction

Price USD / GB (log)

Year


Price USD / GB (log)

Memory Flash SSD HDD

https://ourworldindata.org/


Optimize Data Management Systems for Resource Efficiency Sustainably
Optimize Data Management Systems for Resource Efficiency Sustainably
Static and Adaptive Optimizations

Static Optimizations:
- At development time
- Independent of input data
- Theoretical runtime

Adaptive Optimizations:
- At execution time
- Data distribution & patterns
- Hardware avail. & utilization
Static and Adaptive Optimizations

- **Static Optimizations**
  - At development time
  - Independent of input data
  - Theoretical runtime

- **Adaptive Optimizations**
  - At execution time
  - Data distribution & patterns
  - Hardware avail. & utilization
Static and Adaptive Optimizations

Static Optimizations

⇒ At development time
⇒ Independent of input data
⇒ Theoretical runtime

Adaptive Optimizations

⇒ At execution time
⇒ Data distribution & patterns
⇒ Hardware avail. & utilization
Static and Adaptive Optimizations

Static Optimizations

⇒ At development time
⇒ Independent of input data
⇒ Theoretical runtime

Adaptive Optimizations

⇒ At execution time
⇒ Data distribution & patterns
⇒ Hardware avail. & utilization
While index structures are crucial components in high-performance query-processing systems, they occupy a large fraction of the available memory. Recently, proposed compact indexes reduce the space overhead and thus speed up queries by altering the database to keep large indexes in memory. These compact indexes, however, are driven by performance optimizations in index structures because they adopt encodings that trade memory for performance efficiency. Applying different encodings within a single index might allow optimizing both dimensions at the same time—but if so, it is not clear how much of the index can be optimized at build time.

In many cases, more than half of the available memory of a DBMS can be attributed to index structures [54]. To take advantage of multiple encodings in a single index structure, we present a novel framework leveraging the benefits of workload-adaptive hybrid indexes which move encoding decisions to run time instead. By sampling incoming queries adaptively, it tracks accesses to index parts and keeps fine-grained statistics which are used for space- and performance-optimized encoding negotiations. We evaluated our framework using in-memory and on-chip, and examine the adaptivity process and space-performance tradeoffs for real-world and synthetic workloads. For multi-dimensional workloads, our framework is competitive and allows optimizing both dimensions at the same time while extending more than 90% of the original performance.

ACM Reference Format:

https://doi.org/10.1145/3514221.3526121

CCS CONCEPTS
•Hardware •Operating systems •Data structures and algorithms •Databases — Query processing — Query execution

KEYWORDS
Space-efficient index, Adaptive Index, Hybrid Index

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

1 INTRODUCTION

Back in 2006, Jim Gray stated that memory is the new disk and disk is the new tape [2]. This also applies to modern database systems that store the entire data in random-access memory (RAM) to allow real-time analytics for trading companies and financial services. For example, they need to process large datasets efficiently to react to new developments and updates within a few milliseconds. While the DRAM prices have been stable during the last six years, the data collected by sensors, smartphones, and media platforms, IoT-devices, and digital market-places increases at a high rate resulting in data overflows [54], and storing all data in memory becomes infeasible in many cases. However, they do not need to store entire data for in-memory database systems [1]. These instances are equipped with in-memory capacities of up to 24 TB, but the hourly cost of such an instance is more than $120.

To address high-performance query processing for real-time analytics, in-memory DBs are ideal because they offer high scalability, high concurrency, and low latency. However, in-memory DBs run out of memory very quickly and are not designed for bursty workloads. For example, OLTP DBMSs, the change rate for millions of database rows can be significant in many cases. Therefore, more than half of the available memory in a DRAM can be attributed to index structures [9].
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance
    and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

- Real-world workloads are **skewed**
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the **workload**
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the **workload**

Optimize at run-time
Adaptive Hybrid Indexes – Problem

- **Index structures** are essential for fast query processing
  - Typically optimized for performance and not for memory efficiency
  - Make up to 50% of the memory footprint of a DBMS
  - Compression almost always incurs some overhead!

- Real-world workloads are **skewed**
  - Information is available at run-time and depends on the **workload**

Optimize at run-time
Adaptive Hybrid Indexes – Solution

Index Structure

Adaptive Hybrid Index
Adaptive Hybrid Indexes – Solution

Adaptive Hybrid Index

Index Structure

1 Lightweight Workload Tracking
Adaptive Hybrid Indexes – Solution

Index Structure

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification

anneser@in.tum.de Adaptive Hybrid Indexes 2
Adaptive Hybrid Indexes – Solution

**Adaptive Hybrid Index**

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations

---

Index Structure

--

Introduction  Adaptive Hybrid Indexes  AutoSteer  Programming Model  Conclusions  6
Adaptive Hybrid Indexes – Solution

Index Structure

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations
Adaptive Hybrid Indexes – Solution

Lightweight Workload Tracking
Classification
Adaptive Optimizations
Adaptive Hybrid Index
Perf.-Optimized
Compressed

Adaptive Hybrid Index

1. Lightweight Workload Tracking
2. Classification
3. Adaptive Optimizations

Index Structure

- Perf.-Optimized
- Compressed
Adaptive Hybrid Trie

Experiment Setup:
- Dataset: 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- Workload: 50% Reads, 50% Scans, key selection follows a Zipf distribution  
- Setup: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM  
- Compiler: GCC 9.3.0 with flags `-O3` and `march=native`

Figure: Query latency and index size of ART and FST
Adaptive Hybrid Trie is a level-wise combination of ART and FST!

**Figure:** Query latency and index size of ART and FST

**Experiment Setup:**
- **Dataset:** 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- **Workload:** 50% Reads, 50% Scans, key selection follows a Zipf distribution  
- **Setup:** 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM  
- **Compiler:** GCC 9.3.0 with flags `-O3` and `march=native`
Adaptive Hybrid Trie

1. Expand hot nodes
2. Compact cold nodes

Introduction
Adaptive Hybrid Indexes
AutoSteer
Programming Model
Conclusions
Adaptive Hybrid Trie

Queries

ART

FST

1. Expand hot nodes
2. Compact cold nodes
Adaptive Hybrid Trie

Queries ➔ ART

ART

FST

Adaptation Manager

track
optimize

1. Expand hot nodes
2. Compact cold nodes
Adaptive Hybrid Trie

Queries

ART

FST

1. Expand hot nodes

track

optimize

Adaptation Manager

Introduction Adaptive Hybrid Indexes AutoSteer Programming Model Conclusions
Adaptive Hybrid Trie

Queries

ART

1. Expand **hot nodes**
2. Compact **cold nodes**

FST

track

optimize

Adaptation Manager
Evaluation: Hybrid Trie – Space & Performance

Experiment Setup:
- **Dataset**: 33M unique email addresses (host-reversed order, e.g. `com.foo@<username>`)  
- **Workload**: 50% Reads, 50% Scans, key selection follows a Zipf distribution  
- **Setup**: 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM  
- ** Compiler**: GCC 9.3.0 with flags `-O3` and `march=native`
**Evaluation: Hybrid Trie – Space & Performance**

**Conclusions:**
For point lookups, Hybrid Trie
⇒ reduces index size by 63% comp. to ART
⇒ improves performance by 2.7x comp. to FST

**Experiment Setup:**
- **Dataset:** 33M unique email addresses (host-reversed order, e.g. com.foo@<username>)
- **Workload:** 50% Reads, 50% Scans, key selection follows a Zipf distribution
- **Setup:** 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- **Compiler:** GCC 9.3.0 with flags `-O3` and `march=native`
**Evaluation: Hybrid Trie – Space & Performance**

**Conclusions:**
For point lookups, Hybrid Trie
⇒ reduces index size by 63% comp. to ART
⇒ improves performance by 2.7x comp. to FST

The Pre-Trained Hybrid Trie does not include tracking-related overhead

**Experiment Setup:**
- **Dataset:** 33M unique email addresses (host-reversed order, e.g. com.foo@<username>)
- **Workload:** 50% Reads, 50% Scans, key selection follows a Zipf distribution
- **Setup:** 16-core AMD Ryzen 9 3950X CPU @ 3.5GHz, 64GB DDR4 RAM
- **Compiler:** GCC 9.3.0 with flags -O3 and march=native
ABSTRACT

While index structures are crucial components in high-performance query processing systems, they carry a big fraction of the available memory. Recently, proposed compact indexes reduce the space overhead and thus speed up queries by altering the database to keep larger working sets in memory. These compact indexes, however, are slower than performance-optimized in-memory indexes because they adapt encodings that trade performance for memory efficiency. Applying different encodings within a single index might enable optimizing both dimensions at the same time. However, it is unclear how to build, dynamically monitor and adaptively balance this trade-off at build-time.

To solve this problem, we propose Algorithmic Indexing (AI) that automatically optimizes the index structure. We separate the optimization into two phases: (1) warm-up phase, in which we explore the search space by applying a number of encoding configurations to the database and record both the performance and space overhead, and (2) runtime phase, in which we keep sampling to track index node accesses and dynamically update and balance the encoding configuration. We evaluate AI on real-world and synthetic workloads. For skewed workloads, our framework can reduce the space by up to 82% while retaining more than 90% of the original performance.

KEYWORDS

space-efficient index, adaptive indexing, hybrid indexes

1 INTRODUCTION

While the DRAM prices have been stable during the last six to seven years, the data collected by sensors, smartphones, social media platforms, IoT-devices, and digital market-places that store the entire data in random access memory (RAM) to allow real-time analyses for trading companies and financial services has been growing at a high rate (see figures 1a and 1b). When the amount of data becomes too large to store in RAM, databases need to be designed in such a way that the full data can be kept in memory. For example, AWS offers RAM instances that are equipped with an in-memory capacity of up to 11 TB, but the hourly cost of each instance is more than $120.

To reduce the high-performance-query-processing for real-time analysis, modern database systems use hybrid indexes [1], where certain parts are stored in RAM and other parts are stored in solid-state disks (SSDs) or disk drives. Hybrid indexes are memory-efficient and are widely used by DBMSs. Because there might be multiple indexes per table, especially in OLTP DBMSs, the storage overhead for indexes can be significant. In many cases, more than half of the available memory of a DBMS can be attributed to index structures [54].

To take advantage of multiple encodings in index structures, we present a new framework (AI) that manages encoding decisions in a cost-optimized manner. By sampling incoming queries adaptively, it tracks accesses to index nodes and keeps the encoding mixture that is used for space and performance-optimized encoding navigation. We evaluated our framework using micro-benchmarks and validate the adaptation process and space-performance trade-off for real-world and synthetic workloads. In detailed experiments, our framework is able to reduce the space by up to 82% while retaining more than 90% of the original performance.
P2: AutoSteer

Database Management System

SQL
Parser
Tables
Indexes
Buffer Manager
Query Optimizer
Execution Engine
Result

Software
Hardware

Programming Model & RTS

AutoSteer: Learned Query Optimization for Any SQL Database
Christoph Anneser 1
Technical University of Munich
anner@in.tum.de


PVLDB Reference Format:

AutoSteer: Learned Query Optimization for Any SQL Database

Christoph Anneser 1
Technical University of Munich
anner@in.tum.de

This paper presents AutoSteer, a learning-based solution that automatically drives query optimizations on any SQL database that exposes (and optimizes) hints. AutoSteer builds on the Bandit framework and extends it with software capabilities (e.g., automated hint-set discovery) to manage integration of hint and facilitate usability in both enterprise and open-source SQL systems. We successfully applied AutoSteer on PostgreSQL, Vertica, SparkSQL, MySQL, and FlinkDB. It proposes a new database engine architecture with diverse query optimizers. We then conducted a detailed experimental evaluation with public benchmarks (e.g., TPC-DS, TPC-H, TPC-EM, and TPC-CE) and compared AutoSteer against state-of-the-art systems. Our main conclusion is that AutoSteer can find the right hint-set on a per-query basis and can improve the performance of a whole engine (e.g., up to 40% improvements for PostgreSQL) but also match the performance of state-of-the-art systems with reduced human supervision and decreased adaptivity, as it replaces hand-picked hint-sets with those that are automatically discovered. We also provide an open-source implementation of AutoSteer together with a visual tool for interactive user experiences.

PVLDB Artifact Availability:
This paper is accompanied by an open-source implementation of AutoSteer together with a visual tool for interactive user experiences. The source code, data, and/or other artifacts have been made available at https://github.com/IntelLabs/Auto-Steer.

PVLDB Artistic License.

1 Introduction

AutoSteer is a framework for steering query optimizers of SQL databases autonomously. For each query, we search for effective rewrite rules and store them in the query plans efficiently. The results can be used to train predictive models or to improve existing query optimization policies. We tackle two such challenges in this paper:

1. Learning query optimization policies
2. Learning rewrite rules

We first discuss related work in Section 2, which sets the stage for our main contribution. Section 3 presents AutoSteer and its design. Section 4 evaluates AutoSteer on several benchmarks, while Section 5 discusses future work. Section 6 concludes the paper.

1. Introduction

Our research community has been making rapid strides in applying modern machine learning (ML) techniques to tackle longstanding problems in databases [5, 27, 28]. Learned query optimization lies at the foundation of such efforts [27]. Various techniques from query execution and database-driven query optimization have been proposed in prior work [27, 28, 33]. We only discuss how our work extends current research.

2. Related Work

Modern machine learning (ML) and deep learning have been successfully applied to various fields of database management systems (DBMS) [27, 28]. This section reviews related work in the area of learned query optimizer (LQO).

2.1 Learned Query Optimizers

ML and deep learning have been successfully applied to various fields of database management systems (DBMS). This section reviews related work in the area of learned query optimizer (LQO).

2.2 Learning Rewrite Rules

In addition to optimizing query execution, modern DBMS also optimize query rewrite rules (RRs) based on user-defined preferences. As a result, LQOs must learn to choose the right rewrite rules for each query.

2.3 Learning Query Optimization Policies

Finally, modern DBMS also learn to choose the optimal query optimization policies for each query. This section reviews related work in the area of learned query optimization policies.

3 AutoSteer

Figure 1: AutoSteer is a framework for steering query optimizers of SQL databases autonomously. For each query, we search for effective rewrite rules and store them in the query plan. Then, we use a greedy algorithm to explore alternative query plans efficiently. The results can be used to train predictive models or to improve existing query optimization policies.

3 AutoSteer

AutoSteer is a framework for steering query optimizers of SQL databases autonomously. For each query, we search for effective rewrite rules and store them in the query plan. Then, we use a greedy algorithm to explore alternative query plans efficiently. The results can be used to train predictive models or to improve existing query optimization policies.

3.1 System Design

The AutoSteer framework consists of the following components:

1. Query Optimizer
2. Feature Extractor
3. Policy Learner
4. Query Plan Predictor

The query optimizer is responsible for generating query plans for each incoming query. The feature extractor is responsible for extracting features from the query plan. The policy learner is responsible for learning rewrite rules that are most effective for each query. The query plan predictor is responsible for predicting the query plan that will be selected by the policy learner.

3.2 Experimental Evaluation

We evaluate AutoSteer on several benchmarks, including TPC-DS, TPC-H, TPC-EM, and TPC-CE. The results show that AutoSteer can significantly improve query optimizer performance compared to state-of-the-art systems. In particular, AutoSteer outperforms the TPC-DS benchmark by up to 40% and the TPC-H benchmark by up to 30%.

3.3 Conclusion

In this section, we conclude the paper by summarizing our contributions and discussing future work.

AutoSteer: Learned Query Optimization for Any SQL Database

AutoSteer: Learned Query Optimization for Any SQL Database


@VLDB’23
Many Database Management Systems expose tunable optimizer knobs.
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization
Background – Steering Query Optimizers

Many Database Management Systems expose tunable optimizer knobs.
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization

SQL Query → Database

Hints
```sql
SET enable_indexscan=false;
```

Background – Steering Query Optimizers

Many Database Management Systems expose **tunable optimizer knobs**.
- Usually belong to **rewrite rules** of the rule-based optimizer
- Can be used to **steer** query optimization
Many Database Management Systems expose tunable optimizer knobs.
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization

Example Hint:
```
SET enable_indexscan=false;
```
Many Database Management Systems expose tunable optimizer knobs.
- Usually belong to rewrite rules of the rule-based optimizer
- Can be used to steer query optimization

```
SET enable_indexscan=false;
```
Lot of recent work on steered query optimizers:
- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.
Background – Steering Query Optimizers Automatically

Lot of recent work on steered query optimizers:
- SIGMOD'21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD'22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.

SQL Query → Query Optimizer
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.
Lot of recent work on steered query optimizers:
- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.
Lot of recent work on steered query optimizers:

- SIGMOD'21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD'22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.

SQL Query → Hint-Set 1 → Database

<table>
<thead>
<tr>
<th>Hint-Set 1</th>
<th>predict/estimate</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hint-Set 2</th>
<th>predict/estimate</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hint-Set 3</th>
<th>predict/estimate</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lot of recent work on steered query optimizers:
- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.
Lot of recent work on steered query optimizers:

- SIGMOD’21: “Steering Query Optimizers: A Practical Take on Big Data Workloads”, Negi et al.
- SIGMOD’22: “Deploying a Steered Query Optimizer in Production at Microsoft”, Zhang et al.

![Diagram of query optimization process]

SQL Query → Query Optimizer

- **Hint-Set 1**: predict/estimate 65 (Wrong)
- **Hint-Set 2**: predict/estimate 12 (Correct)
- **Hint-Set 3**: predict/estimate 42 (Wrong)
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
- Requires good knowledge of the query optimizer
- Tight integration into the DBMS
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
- Requires good knowledge of the query optimizer
- Tight integration into the DBMS
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
- Requires good knowledge of the query optimizer
- Tight integration into the DBMS

AutoSteer is a generic framework to steer query optimizers outside the DBMS!
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
- Requires good knowledge of the query optimizer
- Tight integration into the DBMS

AutoSteer is a generic framework to steer query optimizers outside the DBMS!
Limitations of Previous Approaches

- Databases usually expose up to hundreds of knobs
- Requires good knowledge of the query optimizer
- Tight integration into the DBMS

AutoSteer is a generic framework to steer query optimizers outside the DBMS!
AutoSteer – Overview

AutoSteer

Training Mode
AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

Introduction  Adaptive Hybrid Indexes  AutoSteer  Programming Model  Conclusions
AutoSteer – Overview

AutoSteer generates training data by exploring and executing alternative plans. Inference Mode: AutoSteer steers queries at runtime and uses the TCNN to infer execution times.
Introduction

Adaptive Hybrid Indexes

AutoSteer

Programming Model

Conclusions
AutoSteer – Overview

A Query Span contains all knobs that affect the query’s optimization!
A **Query Span** contains all knobs that affect the query's optimization!
The greedy search aims at finding the top hint-sets for a query.
AutoSteer – Overview

Introduction
Adaptive Hybrid Indexes
AutoSteer
Programming Model
Conclusions
AutoSteer – Overview

**Training Mode**

AutoSteer *generates training data* by exploring and *executing* alternative plans.

---

Introduction  Adaptive Hybrid Indexes  AutoSteer  Programming Model  Conclusions
AutoSteer – Overview

Training Mode
AutoSteer generates training data by exploring and executing alternative plans.

Inference Mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.

AutoSteer generates training data by exploring and executing alternative plans. Inference Mode
AutoSteer steers queries at runtime and uses the TCNN to infer execution times.
Dashboard Application at Meta – PrestoDB

- Focus on **tail latencies**
- >3000 Queries, scanning PBs of data, hundreds of worker nodes
- Workload runs multiple times per day

AutoSteer significantly reduces tail latencies of production workloads at Meta
1 INTRODUCTION

Our research community has been making rapid strides in applying modern machine learning (ML) techniques to tackle longstanding problems in databases [6, 24, 48]. Learned query optimization lies at the forefront of this progress [1]. Various techniques from query-driven and data-driven fine-tuning have been proposed in [27, 28, 29, 31, 32, 37, 39, 43], including query rewrite rules (RRs) should be considered in query optimization, but current approaches require manual feature tuning, and ad hoc search. We address this challenge by proposing a novel approach to steer a query optimizer by learning from a rich dataset of query execution traces. Our approach allows for easy adaptation to different query workloads and database systems, and it outperforms existing techniques in terms of both accuracy and efficiency.

2 AUTOSTEER

AutoSteer is a framework for steering query optimizers of SQL databases automatically. For each query, we search for effective rewrite rules and store them in the query optimizer. This can be useful in a variety of scenarios, such as cardirality estimation [22, 30, 31, 32, 37, 39, 43, 48], join order enumeration [29, 32, 39], and query rewriting [50]. Our approach leverages a large dataset of query execution traces to learn effective rewrite rules that can be used to improve query optimization.

3 Evaluation

We evaluate AutoSteer on a variety of workloads, including synthetic and real-world datasets. The results show that AutoSteer outperforms existing techniques in terms of both accuracy and efficiency. For example, on the TPC-H benchmark, AutoSteer achieves a 25% reduction in execution time compared to the baseline query optimizer. The performance gains are consistent across different workloads and database systems, demonstrating the general applicability of our approach.

4 Conclusion

In conclusion, AutoSteer is a powerful tool for steering query optimization in SQL databases. It is simple to use, and its performance is superior to existing techniques. We hope that this work will lead to further developments in the field of automated query optimization.

References

With full resource disaggregation on the horizon, it is unclear what the most suitable programming model is that enables dataflow developers to fully harvest the potential that recent hardware developments offer. Our vision is to move away from the abstraction level to allow developers to primarily reason about their dataflow and the requirements that need to be met by the underlying system in a declarative fashion. Underneath, the system works with typed memory regions and uses the notion of ownership that allows for more flexible memory management across the different compute devices and the tasks mapped onto these. This requires a holistic approach that crosses multiple layers of the system stack, operating existing systems research questions.

Figure 1: Moving from a compute-centric to a memory-centric architecture.

©HotOS’23
Motivation

How to develop & optimize software for heterogeneous, disaggregated hardware?
Motivation

How to develop & optimize software for heterogeneous, disaggregated hardware?
Motivation

Introduction

Adaptive Hybrid Indexes

AutoSteer

Programming Model

Conclusions
Motivation

How to develop & optimize software for heterogeneous, disaggregated hardware?
Motivation

How to develop & optimize software for heterogeneous, disaggregated hardware?

<table>
<thead>
<tr>
<th>Name</th>
<th>Bw.</th>
<th>Lat.</th>
<th>Gran.</th>
<th>Attached</th>
<th>Sync</th>
<th>Persist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>++</td>
<td>++</td>
<td>1 B</td>
<td>CPU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>HBM</td>
<td>++</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DRAM</td>
<td>+</td>
<td>+</td>
<td>64 B</td>
<td>CPU</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>PMem</td>
<td>o</td>
<td>o</td>
<td>256 B</td>
<td>CPU</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CXL-DRAM</td>
<td>o</td>
<td>o</td>
<td>64 B</td>
<td>PCIe</td>
<td>✓/✗</td>
<td>✓/✗</td>
</tr>
<tr>
<td>Disagg. Mem.</td>
<td>o</td>
<td>−</td>
<td>?</td>
<td>NIC</td>
<td>x</td>
<td>✓/✗</td>
</tr>
<tr>
<td>SSD</td>
<td>−</td>
<td>−</td>
<td>4 KiB</td>
<td>PCIe</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>HDD</td>
<td>−−</td>
<td>−−</td>
<td>4 KiB</td>
<td>SATA</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>
Motivation

How to develop & optimize software for heterogeneous, disaggregated hardware?
CXL Enables a Memory-Centric View

- CPU₁
- CPU₂
- TPU
- Memory Pool
  - DRAM₁
  - DRAM₂
  - PMEM
  - CXL DRAM
- GPU₁
- GPU₂
- FPGA

Latency: low
Bandwidth: high
Latency: medium
Persistent: ✓
CXL Enables a Memory-Centric View
CXL Enables a Memory-Centric View

Diagram:
- **CPU** 1
- **CPU** 2
- **TPU**
- **DRAM** 1
- **DRAM** 2
- **PMEM**
- **CXL DRAM**
- **GPU** 1
- **GPU** 2
- **FPGA**

**Abstraction Layer**

**Memory Pool**

- Latency: low
- Bandwidth: high
- Latency: medium
- Persistent: ✓
CXL Enables a Memory-Centric View

Abstraction Layer

Memory Pool

- DRAM₁
- DRAM₂
- PMEM
- CXL DRAM

CPU₁
CPU₂
TPU

GPU₁
GPU₂
FPGA

Latency: low
Bandwidth: high
Latency: medium
Persistent: ✓
CXL Enables a Memory-Centric View

Latency: low  Bandwidth: high
Latency: medium  Persistent: ✓
CXL Enables a Memory-Centric View
Mapping Memory Regions to Devices

⇒ Task Placement

<table>
<thead>
<tr>
<th>Memory Region Properties:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MR 1</td>
<td>low lat., sync</td>
</tr>
<tr>
<td>MR 2</td>
<td>low lat., persistent, async</td>
</tr>
<tr>
<td>MR 3</td>
<td>low lat., high bandwidth, sync</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Handovers:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MR 1</td>
<td>T0 Output, T1 Input</td>
</tr>
<tr>
<td>MR 2</td>
<td>T1 Output, T2 Input</td>
</tr>
</tbody>
</table>

⇒ Device Utilization

<table>
<thead>
<tr>
<th>DRAM</th>
<th>PMEM</th>
<th>GDDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR 1</td>
<td>MR 2</td>
<td>MR 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR 1</td>
<td>MR 2</td>
</tr>
</tbody>
</table>

⇒ Memory Abstraction Layer

RTS needs a comprehensive cost model and late binding
Mapping Memory Regions to Devices

⇒ Task Placement

Task 0 --- Task 1 --- Task 2

CPU    GPU

Memory Region Properties:

- MR\textsubscript{1}: low latency, synchronous
- MR\textsubscript{2}: low latency, persistent, asynchronous
- MR\textsubscript{3}: low latency, high bandwidth, synchronous

Handovers:

- MR\textsubscript{1}: \( T_0 \) output, \( T_1 \) input
- MR\textsubscript{2}: \( T_1 \) output, \( T_2 \) input

Device Utilization:

DRAM, PMEM, GDDR

RTS needs a comprehensive cost model and late binding.
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:

- $\text{MR}_1$: low lat., sync
- $\text{MR}_2$: low lat., persistent, async
- $\text{MR}_3$: low lat., high bandwidth, sync

![Diagram showing task placement and memory region properties](image_url)
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:
  - MR$_1$: low lat., sync
  - MR$_2$: low lat., persistent, async
  - MR$_3$: low lat., high bandwidth, sync

Handovers:
  - MR$_1$: T$_0$ Output, T$_1$ Input
  - MR$_2$: T$_1$ Output, T$_2$ Input

⇒ Device Utilization

![Diagram showing memory regions and devices]

- DRAM
- PMEM
- GDDR
- CPU
- GPU
- MR$_1$
- MR$_2$
- MR$_3$

RTS needs a comprehensive cost model and late binding

Introduction  Adaptive Hybrid Indexes  AutoSteer  Programming Model  Conclusions
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:
- \( \text{MR}_1 \): low lat., sync
- \( \text{MR}_2 \): low lat., persistent, async
- \( \text{MR}_3 \): low lat., high bandwidth, sync
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:

- **MR\_1**: low lat., sync
- **MR\_2**: low lat., persistent, async
- **MR\_3**: low lat., high bandwidth, sync

![Diagram of Memory Abstraction Layer with memory regions and devices]

Introduction  Adaptive Hybrid Indexes  AutoSteer  Programming Model  Conclusions
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:
   - $\text{MR}_1$: low lat., sync
   - $\text{MR}_2$: low lat., persistent, async
   - $\text{MR}_3$: low lat., high bandwidth, sync

⇒ Handovers:
   - $\text{MR}_1$: $T_0$ Output, $T_1$ Input
   - $\text{MR}_2$: $T_1$ Output, $T_2$ Input
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:
  MR₁: low lat., sync
  MR₂: low lat., persistent, async
  MR₃: low lat., high bandwith, sync

⇒ Handovers:
  MR₁: T₀ Output, T₁ Input
  MR₂: T₁ Output, T₂ Input

⇒ Device Utilization
Mapping Memory Regions to Devices

⇒ Task Placement

⇒ Memory Region Properties:
  - \( MR_1 \): low lat., sync
  - \( MR_2 \): low lat., persistent, async
  - \( MR_3 \): low lat., high bandwidth, sync

⇒ Handovers:
  - \( MR_1 \): \( T_0 \) Output, \( T_1 \) Input
  - \( MR_2 \): \( T_1 \) Output, \( T_2 \) Input

⇒ Device Utilization

\[ \text{RTS needs a comprehensive cost model and late binding} \]
Typed Memory Regions

- **Communication**
  - Purpose: Syncing tasks, message passing, ...
  - Properties: coherent, sync

- **Exchange**
  - Properties: coherent, async

- **Thread-local State**
  - Properties: non-coherent, sync, fast
Communication

- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync
Communication
- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data
- Properties: coherent, async
Communication
– Purpose: Syncing tasks, message passing, ...
– Properties: coherent, sync

Exchanging Data
– Properties: coherent, async

Thread-local State
– Properties: non-coherent, sync, fast
**Communication**
- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

**Exchanging Data**
- Properties: coherent, async

**Thread-local State**
- Properties: non-coherent, sync, fast
Typed Memory Regions

Communication
- Purpose: Syncing tasks, message passing, ...
- Properties: coherent, sync

Exchanging Data
- Properties: coherent, async

Thread-local State
- Properties: non-coherent, sync, fast
Typed Memory Regions

Communication
– Purpose: Syncing tasks, message passing, ...
– Properties: coherent, sync

Exchanging Data
– Properties: coherent, async

Thread-local State
– Properties: non-coherent, sync, fast

Global State
Global Scratch
Private Scratch
Conclusions

Adaptive Hybrid Indexes reduce storage overheads & retain high-performance
Adaptive Hybrid Indexes reduce storage overheads & retain high-performance

AutoSteer is a framework to steer rule-based query optimizers
Conclusions

Adaptive Hybrid Indexes reduce storage overheads & retain high-performance

AutoSteer is a framework to steer rule-based query optimizers

Memory-centric programming model for disaggregated memory
Questions?
Thank you for your attention!