Special Topics:
Self-Driving Database Management Systems
Bao: Making Learned Query Optimization Practical
Best Paper of SIGMOD’21

@Lichen Jin // 15-799 // Spring 2022
LAST CLASS

Automatic SQL Rewriting

→ Works at changing query at the SQL Language Level

→ MCTS to efficiently enumerate and select based on rewriting rules

→ Deep Learning model to estimate tree node costs
TODAY’S AGENDA

→ Overview: Learned Query Optimization
→ Prior Work: Neo
→ Bao Introduction
→Enumerating Query Hints
→ Contextual Multi-arm Bandit
→ Predictive Model: TCNN Cost Model
→ Integration and Improvements
→ Experiments
→ Parting Thought
Overview: Learned Query Optimization

Can execution data help with it?

Source: Ryan Marcus
Overview: Learned Query Optimization

Cost Estimation Model

Source: Ryan Marcus
Prior Work: Neo

→ Neural Optimizer
→ Complete replacement of default query optimizer
→ List alternative plans based on expert rules
→ Deep Reinforcement Learning Guided Search

Source: Ryan Marcus
Prior Work: Neo

Promising Results But

Sample Inefficiency
→ Typically takes more than 1 day for pre-train

Brittleness to Workload and Schema Change
→ The encoding of cardinality estimate needs retrain

Tail Catastrophe
→ Deep RL making wrong estimations due to sample inefficiency

Other state-of-the-art: Similar Issues

Source: Ryan Marcus
Bao: Bandit optimizer

By steering a traditional query optimizer, Bao:

→ Outperforms PG after 1 hour of training
→ Reduces 99% latency
→ Adapts to changes in workload, schema, and data.

Source: Ryan Marcus
Bao: Introduction

Three key points:

Offload Alternative Plan Enumeration
→ Query Hint Sets on top of the optimizer

Model Plan Selection as CMAB
→ Exploration and Exploitation
→ Reduce chances of wrong tail estimation

Deep Estimation Model
→ Tree Convolutional Neural Networks (TCNN)
→ Change-adaptive encoding approach

Figure 2: Bao system model
Query Hints

Slow query.
Run EXPLAIN.
- Loop join plan,
- Low selectivity

Try disabling loop join
- Huge improvement

Source: Ryan Marcus
Query Hints

Slow query. Run EXPLAIN.
➢ Loop join plan,
➢ Low selectivity

Try disabling loop join
➢ Huge improvement
Apply this hint globally
➢ ... other regressions,
 Undo that, need local hints. Now What?
➢ Opt 1: Apply the hint to every instance of the query
➢ Opt 2: Set as default, find regressions, add hints to those queries
➢ Opt 3: Give up

Source: Ryan Marcus
Enumerating Query Hints

Query Hint Set: A combination of configuration knobs.
→ Example: enable nested loop join
Query plans are enumerated using different query hint sets.
Bao tries to automatically determine the optimal hint to use.

Source: Ryan Marcus
Contextual Multi-arm Bandit

Suppose you are in a casino
But don’t know which bandit wins money

Time = 1

$+$

$-$

$++$

$--$
Contextual Multi-arm Bandit

Suppose you are in a casino
But don’t know which bandit wins money

Time = 2
Suppose you are in a casino
You want a mental model to estimate the best
Contextual Multi-arm Bandit

You really try the ‘best’ for ground truth
Used for updating the model

Time = X

<(Time, Arm), $!>
Contextual Multi-arm Bandit

Suppose you are in a casino
The model gradually estimates better on winning

Time = X

No addiction to gambling!

<(Time, Arm), $!>
Mapping Back to Query Hints

Consider different hints as arms in a CMAB Predictive Model (Query Latency Predict)
Experience: <Plan, Latency>

Source: Ryan Marcus
Mapping Back to Query Hints

Exploration-Exploitation Tradeoff
→ Losing Exploration: Local optimal, Wrong estimations
→ Losing Exploitation: Model cannot converge

Source: Ryan Marcus
Mapping Back to Query Hints

Exploration-Exploitation Tradeoff
→ Solution: Thompson Sampling

Traditional Query Optimizer

Predicted Performance

- Loop plan: 20
- Hash plan: 25
- Merge plan: 18

Execution

Merge plan: 38

Model Training

Source: Ryan Marcus
CMAB in Bao: Thompson Sampling

An old, well-studied algorithm for balancing exploration and exploitation.

Usual ML training (exploitation)  Weight = E[Weight | data]

Pick a random model (exploration)  Weight = sample P(Weight)

Sample model weights (Thompson Sampling)  Weight = sample P(Weight | data)

Source: Ryan Marcus
CMAB in Bao: Thompson Sampling

An old, well-studied algorithm for balancing exploration and exploitation.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Weight Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usual ML training (exploitation)</td>
<td>$\text{Weight} = \mathbb{E}[\text{Weight} \mid \text{data}]$</td>
</tr>
<tr>
<td>Pick a random model (exploration)</td>
<td>$\text{Weight} = \text{sample } P(\text{Weight})$</td>
</tr>
<tr>
<td>Sample model weights (Thompson Sampling)</td>
<td>$\text{Weight} = \text{sample } P(\text{Weight} \mid \text{data})$</td>
</tr>
</tbody>
</table>

Bao Implementation:
Retrain on a random sampled data set sized $|E|$, randomly drawn from $E$ with replacement

Other complex approaches: Bayesian Networks

Source: Ryan Marcus
**Predictive Model: TCNN Cost Model**

Tree Convolutional Network: Efficient catching correlation as high-level patterns between operators in the tree structure.
→ Long chain of merge joins without sort.
→ Bushy tree of hash operators.

**Figure 5: Bao prediction model architecture**
**Predictive Model: Plan Encoding**

**Binarize:** All internal nodes have two children; *supply with null node*

Multiple Children: *Convert to Binary*
**Predictive Model: Plan Encoding**

**Binarize:** All internal nodes have two children; *supply with null node*

**Multiple Children:** *Convert to Binary*

**Vectorize:** One-hot encoding of operator type, card. Estimator, cost
Integration and Improvements

→ Generate query plans based on hint sets
→ Select the “best” query hint set predicted by TCNN, execute the corresponding plan
→ Add ground truth to Experience data

→ Retrain with Thompson Sampling per n executions
→ Only keep most recent k as Experience data
**PostgreSQL Integration**

Per-query Activation
→ SET enable_bao TO [on/off]

Active vs. advisor mode
→ Active: use bao to select query plan
→ Advisor: default; implicitly train the model

Triggered Exploration
→ Mark query as critical to avoid regression
→ Weighted retrain on critical misprediction
PostgreSQL Integration

Per-query Activation
→ SET enable_bao TO [on/off]

Active vs. advisor mode
→ Active: use bao to select query plan
→ Advisor: default; implicitly train the model

Triggered Exploration
→ Mark query as critical to avoid regression
→ Weighted retrain on critical misprediction

Source Code:
https://github.com/learnedsystems/BaoForPostgreSQL/tree/master/pg_extension
**Experiment: Settings**

**OLAP Workload:**
- Tail Dominate Pattern
- Dynamic Changes
- IMDb (Default)

**Machine Types:**
- GCP N1-2; N1-4 (Default); N1-8; N1-16

**Baseline Databases:**
- PostgreSQL (Default); Comsys [Oracle]
- Bao integrated in corresponding DBMS

**Concurrency:** 1 (Default), 2, 4

**Bao configuration:** n=100, k=2000

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Queries</th>
<th>WL</th>
<th>Data</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDb</td>
<td>7.2 GB</td>
<td>5000</td>
<td>Dynamic</td>
<td>Static</td>
<td>Static</td>
</tr>
<tr>
<td>Stack</td>
<td>100 GB</td>
<td>5000</td>
<td>Dynamic</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>Corp</td>
<td>1 TB</td>
<td>2000</td>
<td>Dynamic</td>
<td>Static</td>
<td>Dynamic</td>
</tr>
</tbody>
</table>

*The schema change did not introduce new data, but did normalize a large fact table.*

**Model:**
- 3 Layer convolution, 2 Linear, Relu, Batch Norm; Adam Batch=16; epoch=100 with early stop

**Time-split training:**
- Model only updated with data collected from previous time split
Experiment: Results

Significant Lower Cost and Runtime

(a) Across our three evaluation datasets, Bao on the PostgreSQL engine vs. PostgreSQL optimizer on the PostgreSQL engine.

(b) Across our three evaluation datasets, Bao on the ComSys engine vs. ComSys optimizer on the ComSys engine.

(a) Across four different VM types, Bao on the PostgreSQL engine vs. PostgreSQL optimizer on the PostgreSQL engine.

(b) Across four different VM types, Bao on the ComSys engine vs. ComSys optimizer on the ComSys engine.
Tail Latency Analysis

IMDB: Significant Lower Tail Latency
Analysis over Runtime

IMDB-PostgreSQL: Model converges in a few hours

Figure 10: Number of IMDb queries processed over time for Bao and the PostgreSQL optimizer on the PostgreSQL engine. The IMDb workload contains 5000 unique queries which vary over time.
Per-query Analysis

IMDB-PostgreSQL: Only 3 of 105 has minor regression
Mostly with significant improvement
Varying # of Arms (Hint Sets)

IMDB-PostgreSQL: 48 arms ordered by observed benefits
Only a few arms is sufficient for optimization
Varying Concurrency and Memory

IMDB-PostgreSQL: Good performance when I/O bound
High CPU usage contention when CPU bound
Adaptivity Analysis

Bao better adapts to changes compared to state-of-the-art

Figure 14: Comparison of number of queries finished over time for Bao, Neo [51], DQ [40], and PostgreSQL for a stable query workload (left) and a dynamic query workload (right).
Modeling Evaluation

→ TCNN is necessary for Query Latency Predict
→ Bao avoids regression when model not converged
→ GPU time linear to the experience window size

(a) Random forest (RF) and linear models (Linear) used as Bao’s model. “Best hint set” is the single best hint set. IMDb, N1-16 VM, on PostgreSQL.

(b) Median Q-Error (0 is a perfect prediction) of Bao’s predictive model vs. the number of queries processed. IMDb workload on N1-16 VM using PostgreSQL engine.

(c) Simulated and observed time to train Bao’s performance prediction model (GPU) based on the sliding window size (number of queries used during each training iteration).
Model Regret (Loss) Analysis

\[ R_q = \left( P(B(q)(q)) - \min_i P(HSet_i(q)) \right)^2 \]

(a) CPU time regret

(b) Physical I/O regret
Bao on Distributed Databases

After SIGMOD submission: Applying Bao to distributed systems

Bao works on commercial and distributed systems as well

(3 node clusters, each ran on their preferred cloud provider with the cheapest nodes available)

Source: Ryan Marcus
PARTING THOUGHTS

→ Support searching on larger enumeration space (different query hints at different points) with approaches like MCTS

→ Inject learnt query optimizer changes to lower level like JIT bytecode with minimum recompiling

→ Why Bao eliminates tail latency better? Just the model is better sampled and hence more accurate?

→ Not suitable for OLTP optimizing? (Simple queries, no tail patterns)
NEXT CLASS

→ Query Optimization II

→ P2 Discussion