Bao: Making Learned Query Optimization Practical

Authors: Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, Tim Kraska

Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)

Presented By: Anirudh Bharadwaj Krishna, Divij Mishra, Jonathan Li Xu, Khushi Talesra, Richik Vivek Sen

Introduction

- Query optimization is a critical task for database management systems
- It is essential for guaranteeing effective database performance and responsiveness
- Cardinality estimation and cost modeling, two fundamental components of query optimization, remain challenging to solve

Problems with existing ML based Optimizers

- **Long Training Time:** Impractically long training times and large amounts of data to surpass traditional methods
- **Inability to adjust to data workload and changes:** Difficulty in adapting to changes in data or workload, often requiring costly retraining
- **Tail catastrophe:** Fail catastrophically on rare queries, especially with sparse training data
- **Black-box decision:** Opaque, black-box decisions, making it harder for DBAs to understand or influence query optimization
- **Integration cost:** High integration costs, lacking full support for standard SQL and vendor-specific features

Bao (Bandit Optimizer): Enhancing Query Optimization

- Bao is the first learned optimizer which overcomes these problems
- It works alongside an existing query optimizer (e.g., PostgreSQL's optimizer) to improve its performance, rather than replacing or eliminating the traditional optimizer
- PostgreSQL may choose a less efficient join strategy (e.g., loop join) in some cases
- Bao learns a mapping between queries and optimal execution strategies via **query hint sets** and thus guides traditional optimizers
- Bao adapts to data and schema changes, reduces tail latency, and improves query performance using **Thompson sampling**, without the need for extensive retraining

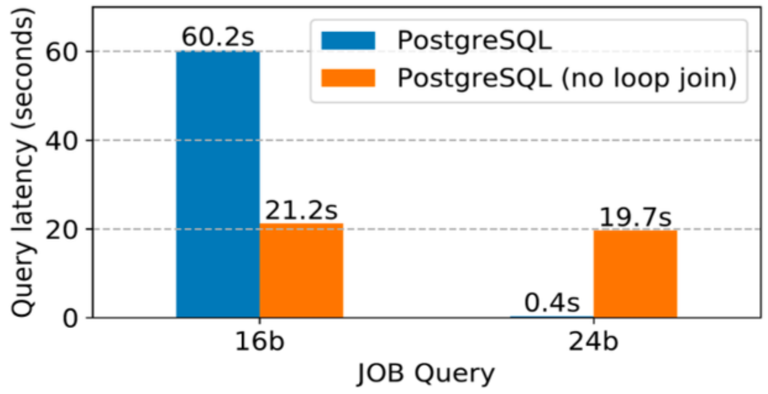


Figure1: Disabling loop join can improve or harm query performance in PostgreSQL

Key features of Bao Optimizer

- Short Training Time
- Robustness to Schema, Data, and Workload Changes
- Better Tail Latency
- Interpretability and Easier Debugging
- Low Integration Cost
- Extensibility

Downsides of Bao

- Longer optimization time
- Limited flexibility
- Best for longer queries

System Model

- **Generate Query Plans:**
  Bao uses the existing query optimizer to create multiple query plans based on different sets of hints (like disabling loop joins or forcing index usage).

- **Estimate run time:**
  Each query plan is turned into a vector tree. These vector tree are then fed into Bao's model, a tree convolution neural network which predicts the quality (e.g., execution time) for each plan.

- **Selecting Query Plan:**
  - Bao selects query plans using **Thompson Sampling**, a technique that balances exploration and exploitation.
  - It explores new query plans to find better ones while exploiting known good plans.
  - This approach allows Bao to learn and improves its query optimization over time.

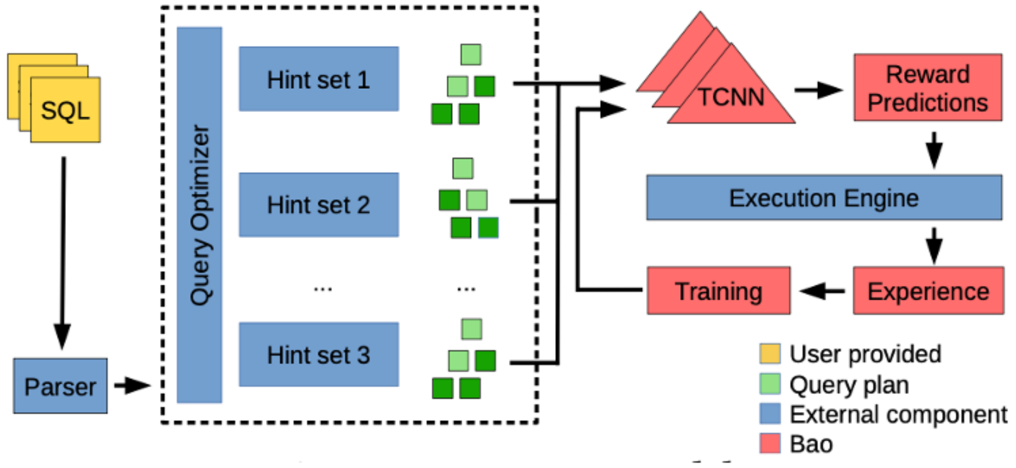


Figure 2: Bao System Model

System Model

- **Feedback Loop:**
  - After executing a plan, Bao learns from the performance (e.g., execution time) and adds this experience to its model
  - Bao periodically retrains its predictive model using collected experience
  - This helps in improving the accuracy of future plan selections

- **Assumptions and Limitations:**
  - Bao assumes that all hints result in semantically equivalent query plans
  - Bao applies hints to the entire query
  - Fine-grained actions, like adjusting individual joins are possible but not explored due to the high overhead involved
  - To ensure faster learning, a smaller action space is chosen, as its size directly impacts the convergence time of reinforcement learning algorithms

Selecting Query Hints - Query Hints

- Query hints are special (optionally user-defined) directives that the DB optimizer must account for while creating the query plan
- For e.g.
  - SET ENABLE_NESTLOOP = False; -> Discourages the planner from using nested-loop joins
  - SET ENABLE_SEQSCAN = False; -> The planner uses index scans instead of sequential full table scans
- Multiple query hints enabled -> query set
- The user must specify all possible hint sets that Bao should optimize over
- Bao's goal: for each query, select the appropriate query hint set

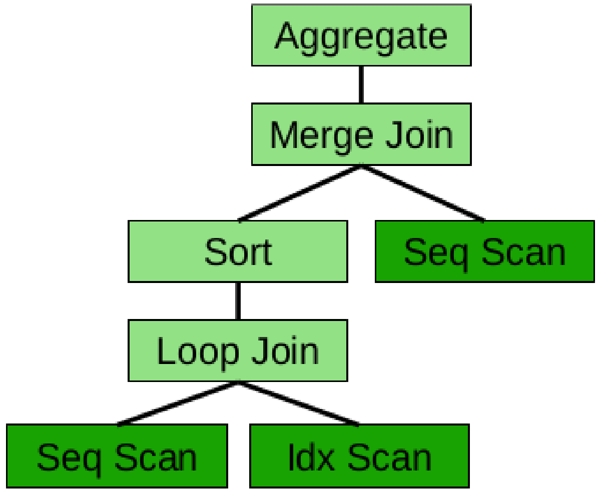
Selecting Query Hints – CMABs

gambling bad

- ***CMAB – Contextual Multi-Armed Bandits***
  - **Multi-Armed Bandits:**
    - Bao sees the same query hint sets at every query – over multiple trials, learns the best query hint sets (MAB problem)
  - **Contextual:**
    - There's no single best query hint set, depends on the query (contextual MAB)
- ***Thompson Sampling***
  - Generally, want to choose our best guess for the model (exploitation), but sometimes, intentionally choose bad models to discover better models later (exploration)
  - More formally, sample parameters according to our guess for the underlying probability distribution
  - Bao - Thompson sampling is implemented using bootstrapping for the predictive model (next slide)

Selecting Query Hints – Predictive Model

- For each query hint set, the base optimizer generates a query plan -> looks like a tree!
- Bao uses a Tree Convolutional Neural Network (TCNN) to capture relationships within the tree and predict the query plan's performance.



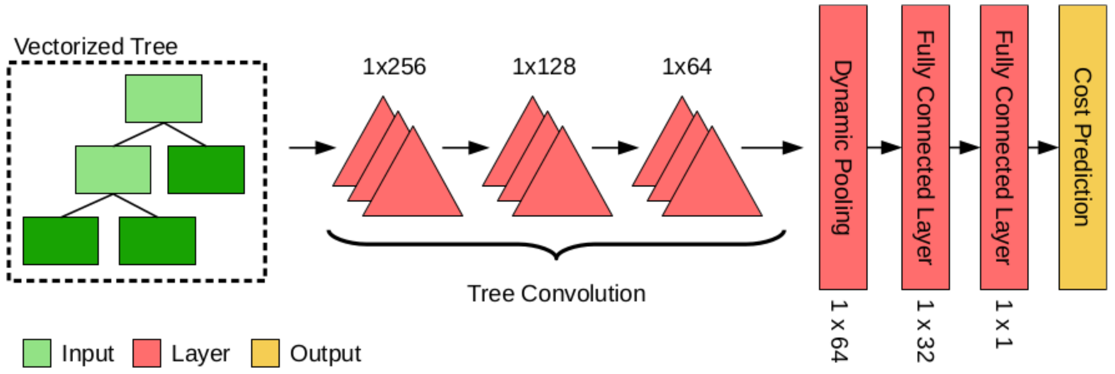
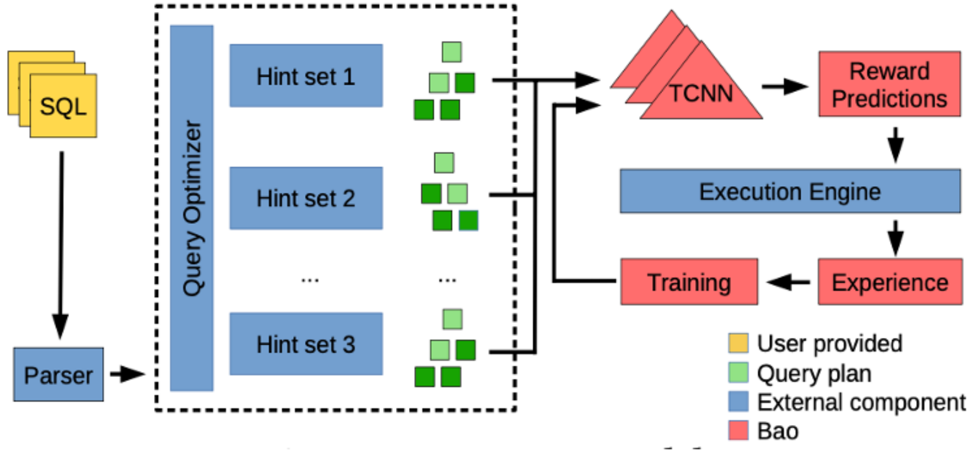


Figure 5: Bao prediction model architecture

Selecting Query Hints – Training Loop

- Training loop:
  - New query comes in
  - For each query hint set (~ 50 sets in the paper), the base optimizer generates a query tree plan
  - Bao estimates each plan's performance using TCNN – selects the plan with lowest cost and executes it
  - Bao adds the selected plan and actual cost to its dataset
- Thompson sampling: Every *n* (~ 100) queries, retrain TCNN on a bootstrapped sample of the previous *k* (~ 2000) queries.



PostgreSQL Integration

Why PostgreSQL?

- Bao may be used with any database system
- PostgreSQL provides a *hooks* system (callback mechanism to call custom functions)
  - Code does not need to be recompiled

Usability Features

- Per-query activation
- Active vs. Advisor mode
- Triggered exploration

PostgreSQL Integration

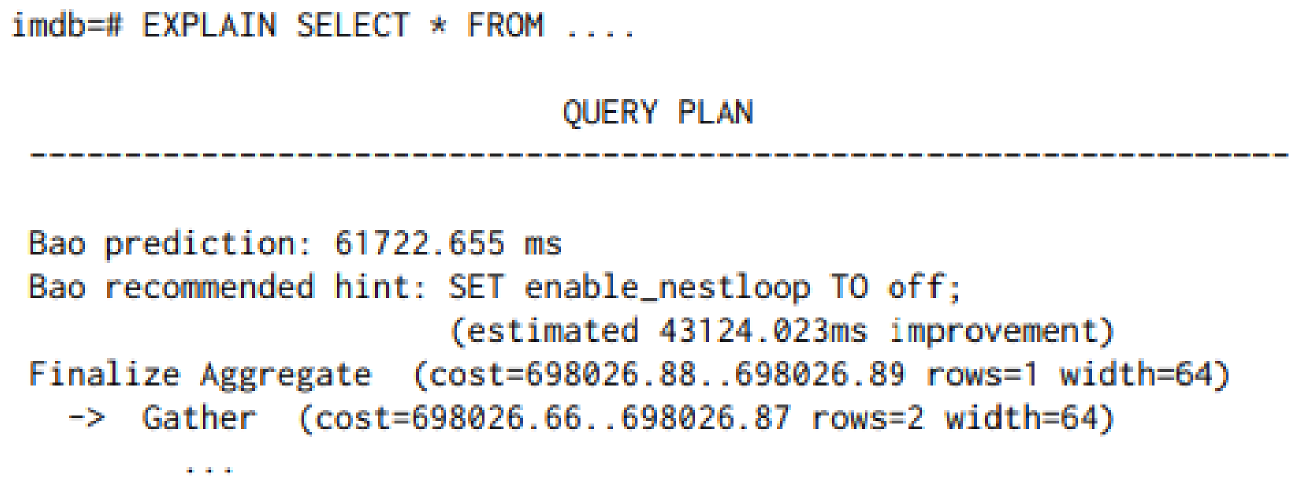
Per-query Activation

- Bao sits upon the existing PostgreSQL optimizer
  - Easy to activate or deactivate Bao on per-query basis
  - Activated: Bao will use Thompson sampling to select query hints
  - Deactivated: PostgreSQL is used
- Why Disable Bao? (`SET enable_bao TO [on/off]`)
  - Short queries may execute in less time than it takes for Bao's optimizer to run
  - DBA may have already set hints or a query plan
  - Bao can still learn from query executions when disabled

PostgreSQL Integration

Active vs. Advisor mode

- Active Mode
  - Automatically applies optimizations that are selected when Bao is *active*
  - Learns based off query performance
- Advisor Mode
  - Does not apply the optimizations, but will suggest them
  - The `EXPLAIN` query adds expected performance, hint set recommended, and predicted improvement
  - Learns based off query performance



```
imdb=# EXPLAIN SELECT * FROM ....

                                   QUERY PLAN
 -------------------------------------------------------------------

 Bao prediction: 61722.655 ms
 Bao recommended hint: SET enable_nestloop TO off;
                       (estimated 43124.023ms improvement)
 Finalize Aggregate  (cost=698026.88..698026.89 rows=1 width=64)
   ->  Gather  (cost=698026.66..698026.87 rows=2 width=64)
         ...
```

Figure 6: Example output from Bao's advisor mode

PostgreSQL Integration

Triggered exploration

- Problem: *Query regression* occurs when a selected optimization decreases the performance of a query vs what was chosen before
  - Bao actively explores new query plans, meaning it can be more erratic
- Solution: give DBAs more control when running queries
  - DBAs may mark queries as *performance critical*
  - A query marked as *critical* triggers Bao to periodically execute the query with each hint set and save the performance to experience set
  - Upon retraining, Bao ensures the new model will select the fastest hint set for each *critical* experience
  - The model is retrained every time it incorrectly predicts a *critical* experience

Related Work

1. Early Efforts in Query Optimization (Leo)

What Leo Did:

Adjusted histogram estimators incrementally based on repeated query executions.

Focused only on improving cardinality estimates (predicting query result sizes) rather than full query execution plans.

How Bao Differs:

Bao goes beyond improving cardinality estimation—it actively adjusts the execution strategies for queries (e.g., disabling certain operators like nested loops or hash joins).

Unlike Leo, Bao doesn't need repeated executions of the same query to improve. It learns on-the-fly from diverse queries.

Related Work

2. Machine Learning for Cardinality Estimation

(Deep Learning , Query-Driven , Monte Carlo Integration , and CRN)

What They Did:

Focused on accurately predicting cardinalities to improve query plans. These methods rely heavily on supervised learning, unsupervised models, or statistical techniques like Monte Carlo sampling.

While improving cardinality accuracy, they did not directly focus on improving query performance (e.g., latency or cost).

How Bao Differs:

Bao combines cardinality estimation with real-world execution feedback (e.g., query performance and execution times). It doesn’t just predict cardinalities—it evaluates entire query execution strategies.

Bao uses reinforcement learning, which adjusts query plans dynamically, rather than relying on pre-trained models.

Related Work

Reinforcement Learning for Query Optimization (Neo) and Broader Applications (Elastic Clusters, Job Scheduling, Physical Design)

What Prior Work Did

Reinforcement Learning for Query Optimization:

- Showed RL could outperform traditional optimizers like PostgreSQL with enough training.
- **Neo**: Optimized query latency using deep RL but required **24 hours of training** and struggled with schema or workload changes.
- **Adaptive Query Processing**: Adjusted query plans during execution but worked only with specific adaptive databases.

Broader RL Applications:

- Used RL for managing **server resources (elastic clusters)**, **task scheduling**, or **physical database design**.
- Focused on **system-wide problems** like resource management, not query optimization.

Related Work

5. Machine Learning for Other Database Applications

(Index Structures , Data Matching, Query Representation)

What They Did:

- Applied ML to auxiliary database tasks like indexing, finding matching records, or representing queries in numerical formats.
- These methods helped optimize database components indirectly but didn't address query plan optimization.

How Bao Differs:

- Bao directly optimizes query execution plans rather than peripheral tasks.
- It works on the critical optimization layer that determines query latency, throughput, and cost.

Related Work

6. Datasets and Evaluation

(IMDb, Stack, Corp)

How Prior Work Evaluated Their Systems:

- Many related works used static workloads or pre-determined datasets that didn't reflect real-world variability in workloads, schema, or data.

How Bao Differs:

- Bao was tested on **dynamic workloads, data, and schema changes** (e.g., Stack and Corp datasets) to show its adaptability.
- Bao integrates with real-world systems like PostgreSQL, whereas prior works often built stand-alone prototypes or controlled environments.

Bao's Distinctive Advantages Over Previous Works

Integrated Approach:
- Bao enhances traditional query optimizers instead of replacing them, making it easier to adopt in existing systems like PostgreSQL.

Fast Learning:
- Bao’s reinforcement learning-based approach requires much less training time and adapts dynamically, unlike prior work that often-needed significant training or retraining.

Focus on Real-World Performance:
- Bao directly improves latency, tail performance, and overall cost for diverse workloads, addressing practical challenges ignored by some prior works.

Adaptability:
- Bao can handle changing workloads, data, and schema, unlike static models or systems like Neo that struggle in dynamic environments.

Experiments

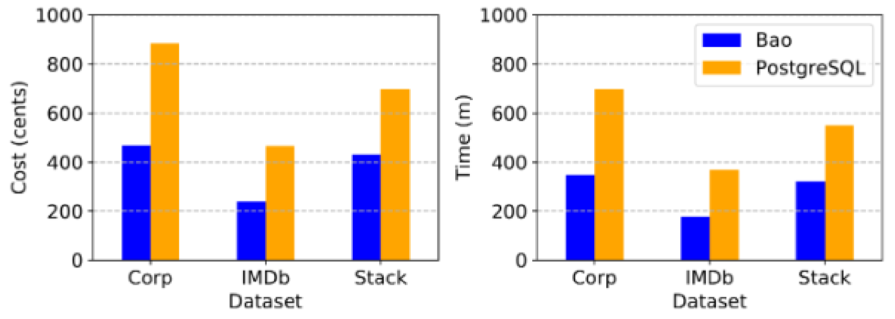
Three carefully selected datasets representing real scenarios:

- IMDb (7.2 GB):
  - 5000 queries
  - Dynamic workload, static data/schema
  - Augmented Join Order Benchmark
- Stack (100 GB):
  - 5000 queries
  - Dynamic workload & data
  - Real StackExchange data over 10 years
- Corp (1 TB):
  - 2000 queries
  - Dynamic workload & schema
  - Real corporate dashboard workload

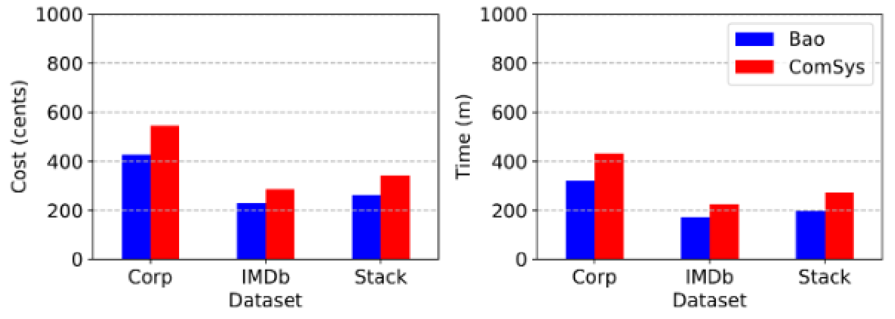
Testing Environments:

- Google Cloud: N1-4 VM + Tesla T4 GPU
- VM: 4 CPU cores, 15GB RAM
- All costs measured include GPU training time

Is Bao Practical? - Cost & Performance



(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.

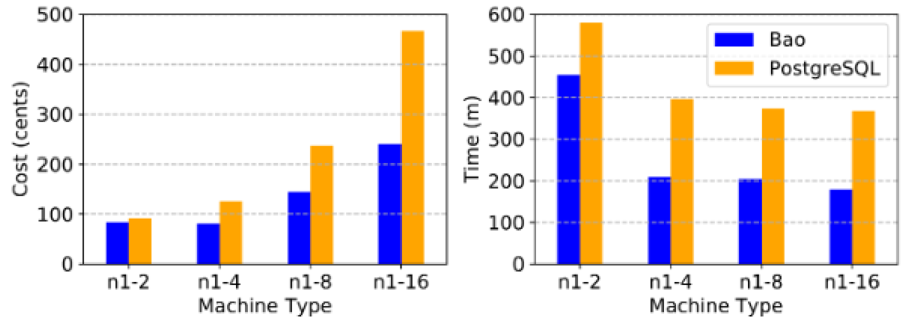
PostgreSQL Comparison:

- ~50% reduction in both cost and latency
- Consistent across all datasets
- Includes all training overhead
- Successfully handles Workload changes (IMDb), Data changes (Stack), Schema changes (Corp)

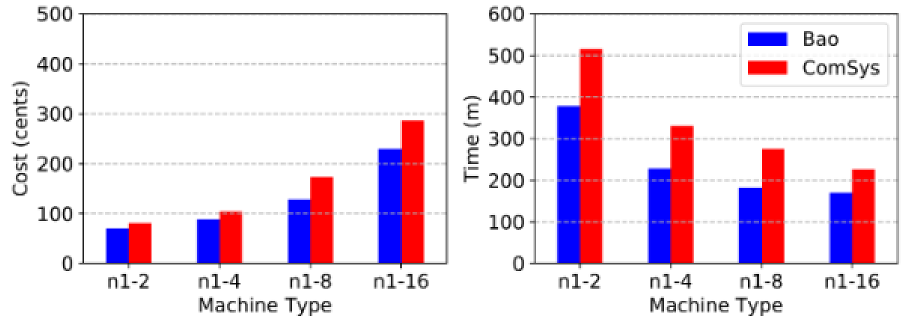
Commercial System Comparison:

- ~20% improvement in performance
- Notable achievement given maturity of commercial system
- Cost savings don't include commercial licensing fees

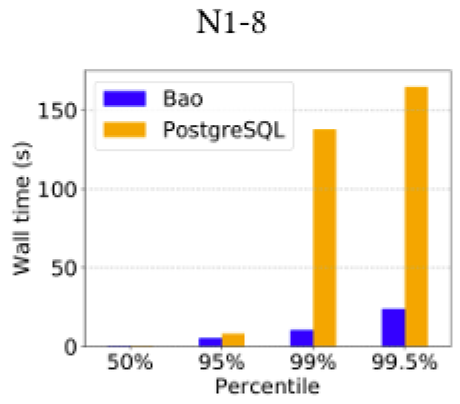
Is Bao Practical? - Hardware Impact & Tail Latency



(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.



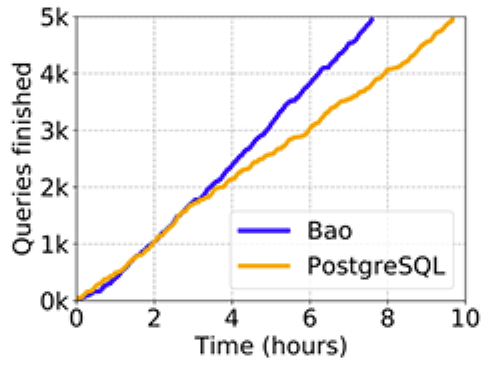
Hardware Scaling:

- Benefits increase with larger VMs for PostgreSQL
- More consistent across VM types for commercial system
- N1-16 shows best improvements

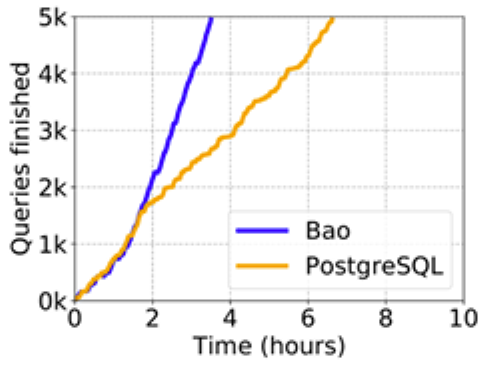
Tail Latency Improvements:

- - 99th percentile latency on N1-8:
  - PostgreSQL: 130 seconds
  - With Bao: 20 seconds
- Most gains from handling problematic queries
- Consistent improvements across VM types
- Especially effective for workloads following 80/20 rule

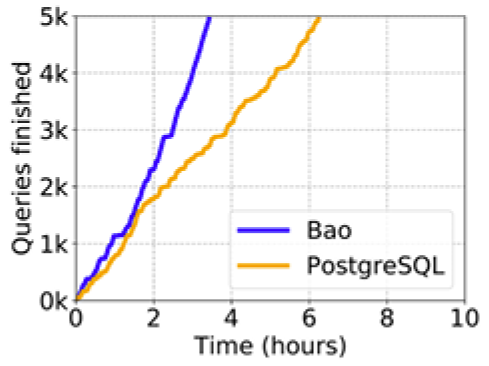
Is Bao Practical? - Training Time & Convergence



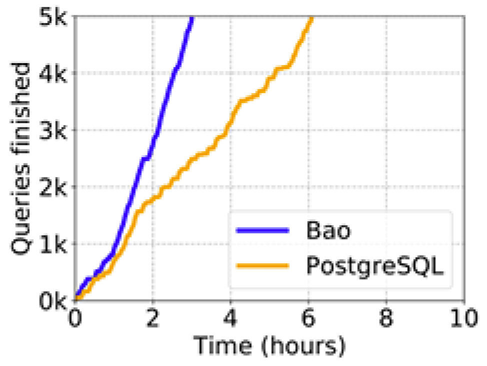
(a) VM type N1-2



(b) VM type N1-4



(c) VM type N1-8



(d) VM type N1-16

Quick Convergence:

- Matches PostgreSQL within 2 hours
- Exceeds baseline after ~3 hours
- Maintains performance through workload changes
- Much faster than previous approaches:
  - Neo: 24 hours
  - DQ: Even longer

Training Overhead:

- GPU can be attached/detached as needed
- ~3 minutes training time for 5000 queries
- Can be scheduled during low-load periods

What Hints Make the Biggest Difference?

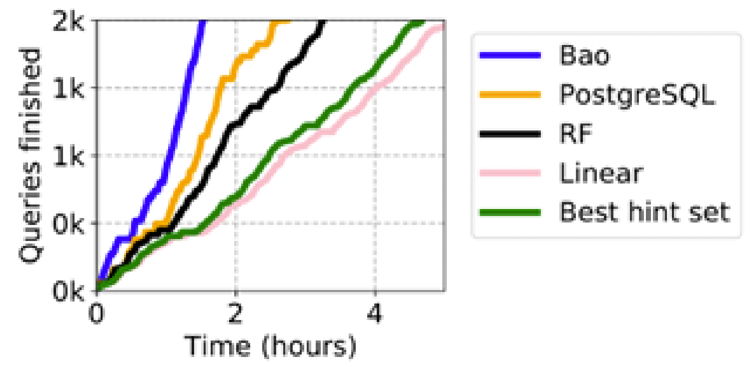
Most Impactful Hint Sets:

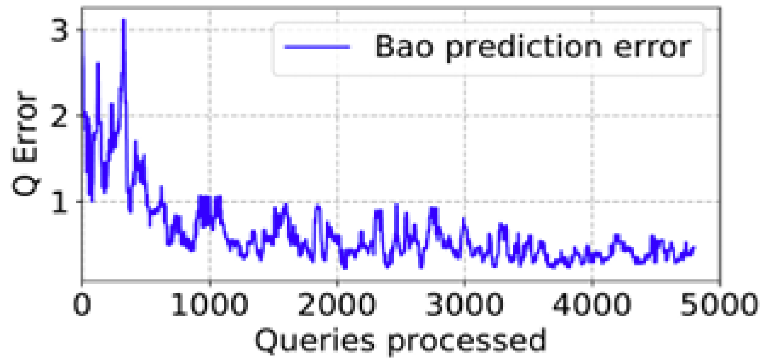
- Disable nested loop join: 35% improvement
  - Helps when cardinality is underestimated
- Disable index scan & merge join: 22%
- Disable nested loop join, merge join & index scan: 16%
- Disable hash join: 10%
  - Helps when cardinality is overestimated
- Disable merge join: 10%

Important Note:

- No single hint set consistently outperforms PostgreSQL
- Combination and context-awareness is key

Bao's ML Model Performance





Model Comparison:

- Neural network significantly outperforms:
  - Random Forest
  - Linear Regression
- Justifies architectural complexity

Learning Progress:

- Initial Q-error around 3
- Steadily improves with experience
- Maintains good decisions despite initial inaccuracy

Major Experimental Conclusions

Practicality Demonstrated Through:

- Quick training (hours vs days)
- Robust to workload/data/schema changes
- Significant cost & performance improvements
- Better tail latency handling

Hint Selection Intelligence:

- Context-aware hint application
- No single "best" hint set
- Most gains from few key combinations

Architecture Validation:

- Neural network complexity justified
- Minimal regression risks

Conclusion

Bao- a reinforcement learning-based bandit optimizer that enhances traditional query optimizers by steering them toward better execution strategies.

Fast Learning and Practical Performance:

Bao achieves performance comparable to advanced open-source and commercial optimizers with **only one hour of training**.

Improved Latency:

Bao effectively reduces both **median latencies** (how long most queries take) and **tail latencies** (how long the slowest queries take).

Handles Dynamic Environments:

Bao performs well even under **dynamic workloads**, changing data, and evolving database schemas.

Future Work

Testing in Cloud Systems:

- The team plans to evaluate Bao in **cloud environments**, particularly in **multi-tenant systems** where resources like disk, RAM, and CPU are shared and often scarce.
- Focus will be on improving overall **resource utilization**.

Integrating with Traditional Optimizers:

- Investigate if Bao's **predictive model** can be used as a **cost model** for traditional query optimizers.
- This could combine the strengths of machine learning with established optimization techniques for better query performance.

2 Study Questions

1. What is query regression? How do the authors mitigate query regression from occurring with Bao, given that it explores to train?
2. How are query plans generated in Bao? How does Bao estimate their runtimes?