Self-Driving Components

Weichen Li  Saatvik Shah
Progress

✅ 75% Goal: Complete
✅ 100% Goal: Complete
😊 125% Goal: Ongoing
75% Goal
Tensorflow Integration

- Model Development Pipeline
- Working Example:
  - LSTM for Workload Prediction
- Tests:
  - Basic Tests
  - Multiple-Sine Wave Workload Forecasting
Workload Prediction

Sinewave Workload Prediction

Predicted:  TRUE
Recommended Indexes

Task: Given a SQLStatement (a query), return all the indexable columns in a vector.

Returned vector: a vector of triplets
Each element in the vector has a format as (database_oid, table_oid, column_oid)
100% Goal
Setting up a RL framework for Dynamic Knob Changes

Recreation of NUS Dynamic Index Tuning “COREIL”

Cost-Model Oblivious Database Tuning with Reinforcement Learning

Debabrota Bard (1), Qin Lin (1), Weikong Chen (1), Hung Tan Vu (1), Zhong Yuan (2), Pierre Senellart (1,2), and Stéphane Bresson (2)

1 School of Computing, National University of Singapore, Singapore, Singapore debabrota.bard@nus.edu
2 Institut Mines-Telecom, Télécom Paristech, CNES-LTCI, Paris, France

Abstract. In this paper, we propose a learning approach to adaptive performance tuning of database applications. The objective is to validate the opportunity to devise a tuning strategy that does not need prior knowledge of a cost model. Instead, the cost model is learned through reinforcement learning. We instantiate our approach to the use case of index tuning. We model the execution of queries and updates as a Markov decision process whose states are database configurations, actions are configuration changes, and rewards are functions of the cost of configuration change and query and update evaluation. During the reinforcement learning process, we face two important challenges: not only the inimicability of a cost model, but also the size of the state space. To address the latter, we devise strategies to prune the state space, both in the general case and for the use case of index tuning. We empirically and comparatively evaluate our approach on a standard OLTP dataset. We show that our approach is competitive with state-of-the-art adaptive index tuner, which is dependent on a cost model.
Birds Eye View

Get Batch
(10 queries)

Metrics Collection

Ask for Metrics

- (Query, Metric) Pair

Update

Tunable Knobs

Execute

Workload
Testing Correctness – I

- **Compressed Index Configuration Modules**
  - Equality of Compression vs. Decompression on varying sample Index Configurations
  - Tested Add and Drop candidates against what might be expected

- **ML Models**
  - Synthetic Workloads
    - LSTM: Sine Wave workload
    - RLSE: Linear Workload
Testing Correctness – II

“RL” Implementation

◆ We setup synthetic Cyclic Workloads*
  ● Simple: Cyclic SELECTs with varying WHERE clauses on a Single Table
  ● Complex: Cyclic SELECTs/INSERTs/UPDATEs clauses on Multiple Tables
◆ Checks for drop in Cost over time
◆ Checks for drop in Cost over an untuned DBMS

*Modifications on top of the Autoadmin teams Testing Utility
Quality of Code Assessment

Code Quality (1: need more work to improve  2: good  3: production-quality)

- (3) `compressed_index_config.h/.cpp`: internal representation of index configurations
- (3) `compressed_index_config_util.h/.cpp`: utility functions to manipulate index configs
- (2) `compressed_idx_config_test.cpp`: test file for index configs
- (3) `lspi_tuner.h/.cpp`: Least-Squares Policy Iteration class
- (3) `lstdq.h/.cpp`: LSTDQ Estimator class
- (2) `rlse.h/.cpp`: Recursive Least Squares Estimator
- (2) `lspi_test.cpp`: test file for LSPI tuner
- 75% Work: All in Production
Benchmarks and Results
Tuning Modes Analyzed

1. No Tuning
2. What-If Exhaustive/Brute Force
   a. With Index Dropping
   b. Without Index Dropping
3. RL(LSPI) Tuning
   a. Non-Exhaustive: Only Single Column Indexes
   b. Exhaustive: With Multicolumn Indexes

Why Index Dropping is Tricky?
Simple Cyclic Workloads

- Simple Mix of SELECTs with a variable number of WHERE clauses on Single Table: ~10 queries
- Number of Cycles Set to 2
- Since we have SELECTs only, the RL algorithm will not perform Drops
Simple Cyclic Workloads: Cost Analysis of RL vs. Best Case

WhatIf Cost of LSPI (Exhaustive) v/s WhatIf (Exhaustive No-Dropping) v/s No Tuning

Log (WhatIf Index Cost)

LSPI(Exhaustive) Tuning Cost
No Tuning Cost
WhatIf(Exhaustive No-Dropping) Tuning Cost

Query #
Simple Cyclic Workloads: Cost Analysis of MultiCol vs. SingleCol Search
Simple Cyclic Workloads: Time Analysis

The graph compares the Tuning Time of LSPI (Exhaustive) against that of LSPI (Non-Exhaustive) and WhatIf (Exhaustive No-Dropping) across a series of queries. The x-axis represents the query number, while the y-axis shows the time in seconds. The graph illustrates how the tuning times vary for each workload type as the queries progress.
Simple Cyclic Workloads: LSPI Trend of Adding Indexes

LSPI(Exhaustive) Adds v/s LSPI(Exhaustive) Drops

- LSP(Exhaustive) Adds
- LSP(Exhaustive) Drops

Count

Query #
Complex Cyclic Workloads

➔ Mix of SELECTs/UPDATE/DELETE/JOIN over Multiple Tables: ~20 queries
➔ Number of Cycles Set to 2
➔ Two Cases
  ◆ Dropping Disabled
  ◆ Dropping Enabled
Complex Cyclic Workloads (Dropping Disabled): Cost Analysis of RL vs. Best Case
Complex Cyclic Workloads (Dropping Disabled): LSPI Trend of Adding Indexes

![Chart showing LSPI Exhaustive Adds vs LSPI Exhaustive Drops](chart.png)
Complex Cyclic Workloads (Dropping Enabled): Cost Analysis of RL vs. Best Case
Complex Cyclic Workloads (Dropping Enabled): LSPI Trend of Adding/Dropping Indexes
# Cumulative Costs

<table>
<thead>
<tr>
<th>Workload</th>
<th>No Tuning</th>
<th>LSPI (Exhaustive)</th>
<th>WhatIf (Exhaustive)</th>
<th>LSPI (Non-Exhaustive)</th>
<th>WhatIf (Exhaustive No-Dropping)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Cyclic</td>
<td>180.0000000</td>
<td>20.921036</td>
<td>50.772381</td>
<td>21.044231</td>
<td>20.905663</td>
</tr>
<tr>
<td>Complex Cyclic (No Dropping)</td>
<td>440.0000000</td>
<td>191.685303</td>
<td>261.120544</td>
<td>191.824112</td>
<td>191.680405</td>
</tr>
<tr>
<td>Complex Cyclic (With Dropping)</td>
<td>440.0000000</td>
<td>211.481110</td>
<td>261.116150</td>
<td>-</td>
<td>191.680405</td>
</tr>
</tbody>
</table>
Future Work

➔ TPCC!!
➔ Incorporate AutoAdmin Indexes as Add Candidates with Prefix Closures
➔ Incorporate Better Metrics:
  ◆ Number of Tuples Read
  ◆ Latency(??)
➔ More Error Analysis
  ◆ Effects across different sizes of Multicolumn Indexes
  ◆ Methods to Improve Quality of Drops
  ◆ Longer Training Sessions
  ◆ More Real World Datasets
➔ Application of Better RL algorithms
  ◆ We have Tensorflow + Eigen within Peloton at our disposal
➔ More Knobs!!