Cost-Model Oblivious
Database Tuning with Reinforcement Learning

Debabrota Basu\textsuperscript{1}, Qian Lin\textsuperscript{1}, Weidong Chen\textsuperscript{1}, Zihong Yuan\textsuperscript{1}, Hoang Tam Vo\textsuperscript{3}, Pierre Senellart\textsuperscript{1,2}, Stéphane Bressan\textsuperscript{1}

\textsuperscript{1}School of Computing, National University of Singapore, Singapore
\textsuperscript{2}Institut Mines–Téléc\textsuperscript{2}com; Téléc\textsuperscript{2}com ParisTech; CNRS LTCI, France
\textsuperscript{3}SAP Research and Innovation, Singapore
Motivation

Is Query Optimization a “solved” problem? If not, are we attacking the “right” problems? How should we identify the “right” problems to solve?
Motivation

- Current query optimizers depend on pre-determined cost models

- But cost models can be highly erroneous

the cardinality model. In my experience, the cost model may introduce errors of at most 30% for a given cardinality, but the cardinality model can quite easily introduce errors of many orders of magnitude! I’ll give a real-world example in a moment. With such errors, the wonder isn’t “Why did the optimizer pick a bad plan?” Rather, the wonder is “Why would the optimizer ever pick a decent plan?”
We propose and validate a tuning strategy to do without such a pre-defined model.

The process of database tuning is modelled as a Markov decision process (MDP).

A reinforcement learning based algorithm is developed to learn the cost function.

COREIL replaces the need of pre-defined knowledge of cost in index tuning.
Problem

Database Schema:

- **Warehouse**
- **Customer 1**
- **Customer 2**
- **Customer 3**

Queries:
1) New order
2) Delivery
3) Stock

Tables:
1) History
2) Stock
3) New orders
4) Stocks

Set of all Database Configurations: \( S = \{s\} \)

Schedule of queries and updates: \( Q \)
Transition

Query $q_t$

Configuration update $\delta(s_{t-1}, s_t)$

DB configuration $s_{t-1}$

Per-stage cost $C(s_{t-1}, s_t, q_t) = \delta(s_{t-1}, s_t) + \text{cost}(s_t, q_t)$

Updated DB configuration $s_t$

Query execution $\text{cost}(s_t, q_t)$
Mapping to MDP

Per-stage cost: $C(s_{t-1}, s_t, q_t) = \delta(s_{t-1}, s_t) + \text{cost}(s_t, q_t)$

Penalty function

States

Action

Query execution

Configuration update

September 4, 2015
Debabrota Basu
Cost-Model Oblivious Database Tuning
MDP Formulation

- **State**: Database configurations $s \in S$

- **Action**: Configuration changes $s_{t-1} \rightarrow S_t$ along with query $q_t$ execution

- **Penalty function**: Per-stage cost of the action $C(s_{t-1}, s_t, \hat{q}_t)$

- **Transition function**: Transition from one state to another on an action are deterministic

- **Policy**: A sequence of configuration changes depending on the incoming queries
Problem Statement

- For a policy $\pi$ and discount factor $0 < \gamma < 1$ the cumulative penalty function or the **cost-to-go function** can be defined as,

$$V^\pi(s) \triangleq \mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t, \hat{q}_t) \right]$$

satisfying

$$\begin{cases} 
  s_0 = s \\
  s_t = \pi(s_{t-1}, \hat{q}_t), \\
  t \geq 1 
\end{cases}$$

- **Goal**: Find out an optimal policy $\pi^*$ that minimizes the cumulative penalty or the cost-to-go function
Features of The Model

- The schedule is sequential
- The issue of concurrency control is orthogonal
- Query $q_t$ is a random variable generated from an unknown stochastic process
- It is always cheaper to do a direct configuration change
- There is no free configuration change
Policy Iteration

A dynamic programming approach to solve MDP.

- Begin with an initial policy $\pi_0$ and initial configuration $s_0$

- Find an estimate $V^{\pi_0}(s_0)$ of the cost-to-go function

- Incrementally improve the policy using the current estimate of the cost-to-go function. Mathematically,

$$V^{\pi_t}(s) = \min_{s' \in S} \left( \delta(s, s') + \mathbb{E} \left[ \text{cost}(s', q) \right] + \gamma V^{\pi_{t-1}}(s') \right)$$

- Carry on the improvement till there is no (or $\epsilon$) change in policy
Problems with Policy Iteration

- **Problem 1**: The **curse of dimensionality** makes direct computation of $\overline{V}$ hard.

- **Problem 2**: There may be **no proper model** available beforehand for the **cost function** $cost(s, q)$.

- **Problem 3**: The **probability distribution of queries** being **unknown**, it impossible to compute the expected cost of query execution.
Solution: Reducing the Search Space

**Theorem**

Let $s$ be any configuration and $\hat{q}$ be any observed query. Let $\pi^*$ be an optimal policy. If $\pi^*(s, \hat{q}) = s'$, then $\text{cost}(s, \hat{q}) - \text{cost}(s', \hat{q}) \geq 0$. Furthermore, if $\delta(s, s') > 0$, i.e., if the configurations certainly change after query, then $\text{cost}(s, \hat{q}) - \text{cost}(s', \hat{q}) > 0$.

Thus, the **reduced subspace** of interest

$$S_{s, \hat{q}} = \{ s' \in S \mid \text{cost}(s, \hat{q}) > \text{cost}(s', \hat{q}) \}$$
Solution: Learning the Cost Model

- Changing the configuration from $s$ to $s'$ can be considered as executing a special query $q(s, s')$

- Then the cost model can be approximated as

$$\delta(s, s') = \text{cost}(s, q(s, s')) \approx \zeta^T \eta(s, q(s, s'))$$

- This approximation can be improved recursively using Recursive Least Square Estimation (RLSE) algorithm

- Similar linear projection $\phi(s)$ can be used to approximate the cost-to-go function $V^\pi_t(s)$
What is COREIL?

COREIL is an index tuner, that

- instantiates our reinforcement learning framework
- tunes the configurations differing in their secondary indexes
- handles the configuration changes corresponding to the creation and deletion of indexes
- inherently learns the cost model and solve a MDP for optimal index tuning
COREIL: Reducing the State Space

- $I$ be the set of all possible indexes

- Each configuration $s \in S$ is an element of the power set $2^{|I|}$

- $r(\hat{q})$ be the set of recommended indexes for a query $\hat{q}$

- $d(\hat{q})$ be the set of indexes being modified (update, insertion or deletion) by $\hat{q}$

- The reduced search space is

$$S_{s,\hat{q}} = \{ s' \in S \mid (s - d(\hat{q})) \subseteq s' \subseteq (s \cup r(\hat{q})) \}$$

- For $B^+$ trees, prefix closure $\langle r(\hat{q}) \rangle$ replaces $r(\hat{q})$ for better approximation
We can define

\[ \phi_{s'}(s) \triangleq \begin{cases} 1, & \text{if } s' \subseteq s \\ -1, & \text{otherwise.} \end{cases} \forall s, s' \in S \]

**Theorem**

*There exists a unique* \( \theta = (\theta_{s'})_{s' \in S} \) *which approximates the value function as*

\[ V(s) = \sum_{s' \in S} \theta_{s'} \phi_{s'}(s) = \theta^T \phi(s) \]
COREIL: Feature Mapping Per-stage Cost

- \(\beta(s, \hat{q})\) captures the **difference between the index set** recommended by the database system and that of the current configuration.

- \(\alpha(s, \hat{q})\) take values either 1 or 0 whether a **query modifies any index** in the current configuration.

- We define the feature mapping

\[ \eta = (\beta^T, \alpha^T)^T \]

**to approximate the functions** \(\delta\) and **cost**
The dataset and workload conform to the TPC-C specification.

They are generated by the OLTP-Bench tool.

Each of the 5 transactions are associated with 3 ~ 5 SQL statements (query/update).

Response time of processing corresponding SQL statement is measured using IBM DB2.

The scale factor (SF) used here is 2.
Efficiency

![Graph showing efficiency comparison between COREIL and WFIT]

- **COREIL**
- **WFIT**

<table>
<thead>
<tr>
<th>Query #</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>500</td>
<td>5,000</td>
</tr>
<tr>
<td>1,000</td>
<td>4,000</td>
</tr>
<tr>
<td>1,500</td>
<td>3,000</td>
</tr>
<tr>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>2,500</td>
<td>1,000</td>
</tr>
<tr>
<td>3,000</td>
<td>0</td>
</tr>
</tbody>
</table>
Box-plot Analysis
Overhead Cost Analysis

![Graph comparing COREIL and WFIT over time for query #]
Effectiveness

![Effectiveness Graph](image)

- **COREIL**
- **WFIT**

- **Query #**
- **Time (ms)**

- **Y-axis:** \(10^0, 10^1, 10^2, 10^3\)
- **X-axis:** \(0, 500, 1,000, 1,500, 2,000, 2,500, 3,000\)
Conclusion

- Database tuning can be modelled as a Markov decision process.
- Our reinforcement learning algorithm solves the problem of cost-model oblivious database tuning.
- COREIL instantiates the approach for index tuning problem.
- It shows competitive performance with respect to the state-of-the-art WFIT algorithm.
Future Work

- Study the trade-off of effectiveness and efficiency of COREIL
- Validate this algorithm on different datasets like TPC-H and benchmark for online index tuning
- Check sensitivity of COREIL on set-up and parameters
- Extend our approach to other aspects of database configuration, including partitioning and replication
Questions?
Thank you
Algorithm: Least Square Policy Iteration with RLSE

1: Initialize the configuration $s_0$.
2: Initialize $\theta^0 = \theta = 0$ and $B^0 = \epsilon I$.
3: Initialize $\zeta^0 = 0$ and $\overline{B}^0 = \epsilon I$.
4: for $t=1,2,3,\ldots$ do
5: Let $\hat{q}_t$ be the just received query.
6: $s_t \leftarrow \arg\min_{s \in S_{s_{t-1}, \hat{q}_t}} (\zeta^{t-1})^T \eta(s_{t-1}, q(s_{t-1}, s)) + (\zeta^{t-1})^T \eta(s, \hat{q}_t) + \gamma \theta^T \phi(s)$
7: Change the configuration to $s_t$.
8: Execute query $\hat{q}_t$.
9: $\hat{C}^t \leftarrow \delta(s_{t-1}, s_t) + \text{cost}(s_t, \hat{q}_t)$.
10: $\hat{\epsilon}^t \leftarrow (\zeta^{t-1})^T \eta(s_{t-1}, \hat{q}_t) - \text{cost}(s_{t-1}, \hat{q}_t)$
11: $B^t \leftarrow B^{t-1} - \frac{B^{t-1} \phi(s_{t-1})(\phi(s_{t-1})-\gamma \phi(s_t))^{T} B^{t-1}}{1+(\phi(s_{t-1})-\gamma \phi(s_t))^{T} B^{t-1} \phi(s_{t-1})}$.
12: $\theta^t \leftarrow \theta^{t-1} + \frac{\hat{C}^t - (\delta(s_{t-1}) - \gamma \phi(s_t))^{T} \theta^{t-1} B^{t-1} \phi(s_{t-1})}{1+(\phi(s_{t-1})-\gamma \phi(s_t))^{T} B^{t-1} \phi(s_{t-1})}$.
13: $(\overline{B}^t, \zeta^t) \leftarrow \text{RLSE}(\hat{\epsilon}^t, \overline{B}^{t-1}, \zeta^{t-1}, \eta^t)$
14: if $(\theta^t)$ converges then
15: $\theta \leftarrow \theta^t$.
16: end if
17: end for
Cost of Configuration Change Analysis
**Theorem**

If for any policy $\pi$, there exist a vector vector $\theta$ such that $V^\pi(s) = \theta^T \phi(s)$ for any configuration $s$, then the proposed algorithm will converge to an optimal policy.