Cost-Model Oblivious Database Tuning with Reinforcement Learning

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

$^1$School of Computing, National University of Singapore, Singapore
$^2$Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI, France
$^3$SAP Research and Innovation, Singapore
Is Query Optimization a Solved Problem?

- 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?”
Proposed Solution

- 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:** \( W \)
- **Customer:** \( C_1, C_2, C_3 \)

**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:**

<table>
<thead>
<tr>
<th>( t )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>Customer 1, New order</td>
</tr>
<tr>
<td>202</td>
<td>Stock</td>
</tr>
<tr>
<td>203</td>
<td>Customer 2, Delivery</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
</tr>
</tbody>
</table>

**Indexes:**

- **Partitions**
- **Parameters**
- **Replicas**
Transition

\[ C(s_{t-1}, s_t, q_t) = \delta(s_{t-1}, s_t) + \text{cost}(s_t, q_t) \]
Mapping to MDP

Mapping to MDP

Per-stage cost

Policy: Sequence of configuration changes that minimizes the cumulative penalty
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

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

- **Goal**: Find out an optimal policy $\pi^*$ that minimizes the cumulative penalty or the cost-to-go function
Policy Iteration

A dynamic programming approach to solve MDP

- Begin with an initial policy $\pi_0$ and initial configuration $s_0$
- Find an estimate $\overline{V}^{\pi_0}(s_0)$ of the cost-to-go function
- Incrementally improve the policy using the Bellman equation based on current estimate of the cost-to-go function

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

- 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.
A reduced subspace is searched for a query $\hat{q}$ at state $s$ that includes states $s'$, such that $\text{cost}(s, \hat{q}) > \text{cost}(s', \hat{q})$

The cost model can be approximated using linear projection given by

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

where, changing the configuration from $s$ to $s'$ is considered as executing a special query $q(s, s')$

Similar linear projection $\phi(s)$ can be used to approximate the cost-to-go function $V^{\pi_t}(s)$

These approximations are then improved recursively by minimizing the least square error
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: Realization of Adaptive Tuning

- For a given query $\hat{q}$, it searches in a reduced space that includes the set of recommended indexes for that query but excludes the set of indexes being modified.

- To approximate $V^{\pi_t}(s)$, we define the feature mapping $\phi(s)$ that indicates whether an index is modified by a configuration or not.

- To approximate $\delta$ and cost, we define the feature mapping $\eta = (\beta^T, \alpha^T)^T$.

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

- $\alpha(s, \hat{q})$ indicates whether a query modifies any index in the current configuration.
The dataset and workload conform to the TPC-C specification.

They are generated by the OLTP-Bench tool.

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

The scale factor (SF) used here is 2.

We are comparing with WFIT algorithm that depends on what-if optimizer for the cost model.
Efficiency

**Figure**: Efficiency = total time per query
Overhead Cost Analysis

Figure: Overhead = Time of the optimization itself
Effectiveness

Figure: Effectiveness = Query execution time in the DBMS alone
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

- Validate the proposed 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
Thank you