# 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

# Motivation

ACM SIGMOD Blog

Guy Lohman

APRIL 10, 2014

## IS QUERY OPTIMIZATION A "SOLVED" PROBLEM?

Databases

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

# Motivation

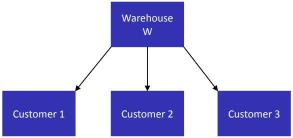
- Current query optimizers depend on pre-determined cost models
- But cost models can be highly erroneous

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?"

# 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




| Queries |
|---|
| 1) New order |
| 2) Delivery |
| 3) Stock |

| Tables |
|---|
| 1) History |
| 2) Stock |
| 3) New orders |
| 4) Stocks |

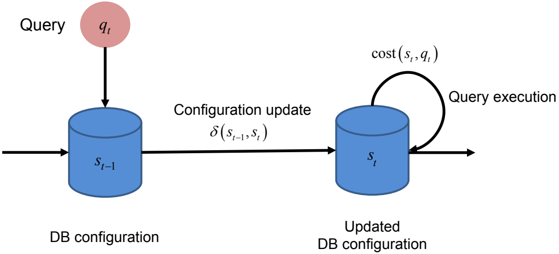
Database Schema: **R**


Set of all Database Configurations: **S = {s}**

| | |
|---|---|
| ... | ..... |
| t = 201 | Customer 1, New order |
| t = 202 | Stock |
| t = 203 | Customer 2, Delivery |
| ... | ..... |

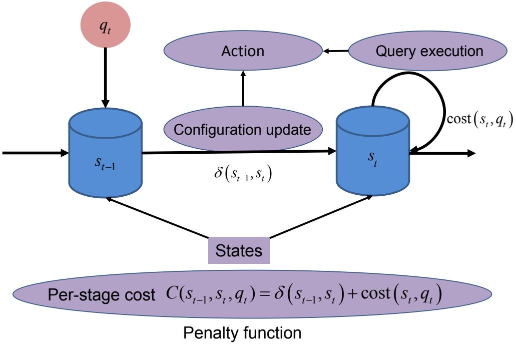
Schedule of queries and updates: **Q**

# Transition




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

# Mapping to MDP



# 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,

$$V^{\pi}(s) \triangleq \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} C(s_{t-1}, s_t, \hat{q}_t)\right] \text{ satisfying } \begin{cases} s_0 = s \\ s_t = \pi(s_{t-1}, \hat{q}_t), \\ t \geq 1 \end{cases}$$

- **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 $\overline{V}^{\pi_0}(s_0)$ of the cost-to-go function
- Incrementally improve the policy using the current estimate of the cost-to-go function. Mathematically,

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

- 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

# 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 $cost(s, \hat{q}) - cost(s', \hat{q}) \geq 0$. Furthermore, if $\delta(s, s') > 0$, i.e., if the configurations certainly change after query, then $cost(s, \hat{q}) - cost(s', \hat{q}) > 0$.*

Thus, the **reduced subspace** of interest

$$S_{s,\hat{q}} = \{s' \in S \mid cost(s, \hat{q}) > cost(s', \hat{q})\}$$

# 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

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

- This approximation can be improved recursively using Recursive Least Square Estimation (RLSE) algorithm
- Similar linear projection $\boldsymbol{\phi}(s)$ can be used to approximate the cost-to-go function $\overline{V}^{\pi_t}(s)$

# 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

# COREIL: Feature Mapping Cost-to-go Function

- 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* $\boldsymbol{\theta} = (\theta_{s'})_{s' \in S}$ *which approximates the value function as*

$$V(s) = \sum_{s' \in S} \theta_{s'} \phi_{s'}(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$$

# COREIL: Feature Mapping Per-stage Cost

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

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

- We define the feature mapping

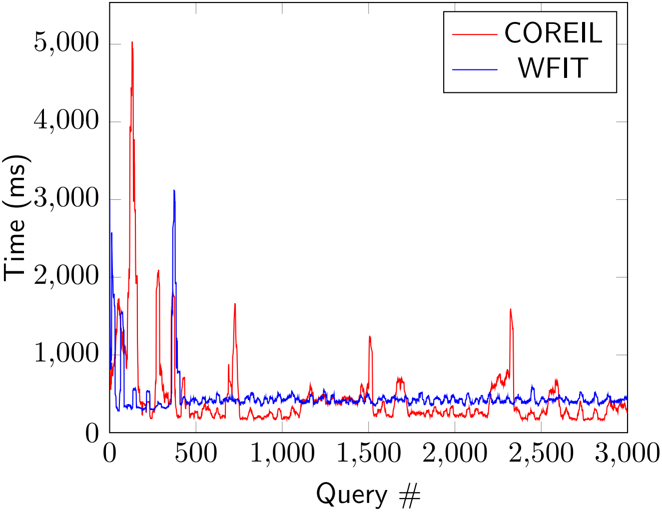
$$\boldsymbol{\eta} = (\boldsymbol{\beta}^T, \boldsymbol{\alpha}^T)^T$$

to approximate the functions $\delta$ and $cost$

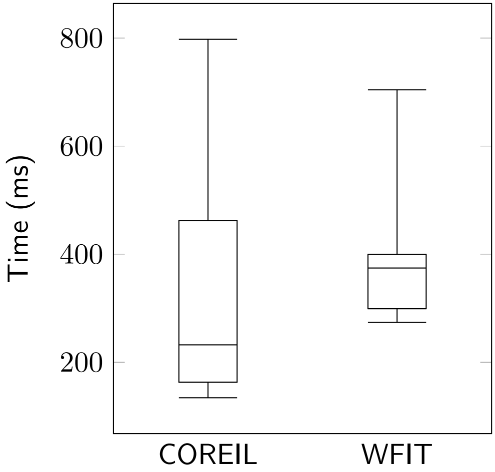
# Dataset and Workload

- 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 $3 \sim 5$ SQL statements (query/update)
- Response time of processing corresponding SQL statement is measured using IBM DB2
- The scale factor (SF) used here is 2

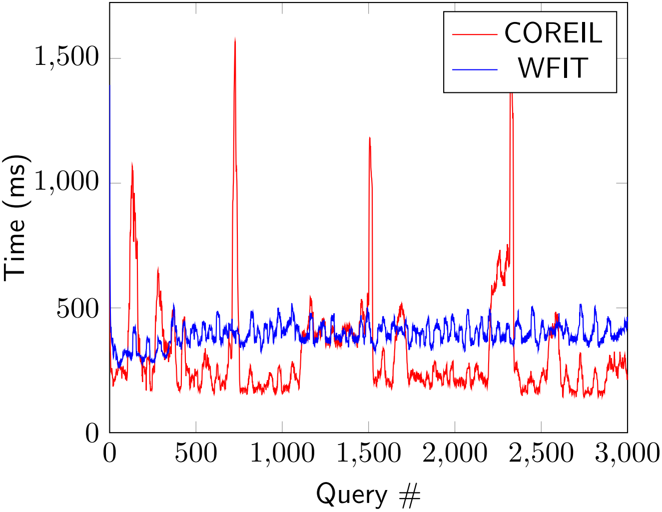
# Efficiency


COREIL
WFIT
Time (ms)
0
1,000
2,000
3,000
4,000
5,000
0
500
1,000
1,500
2,000
2,500
3,000
Query #

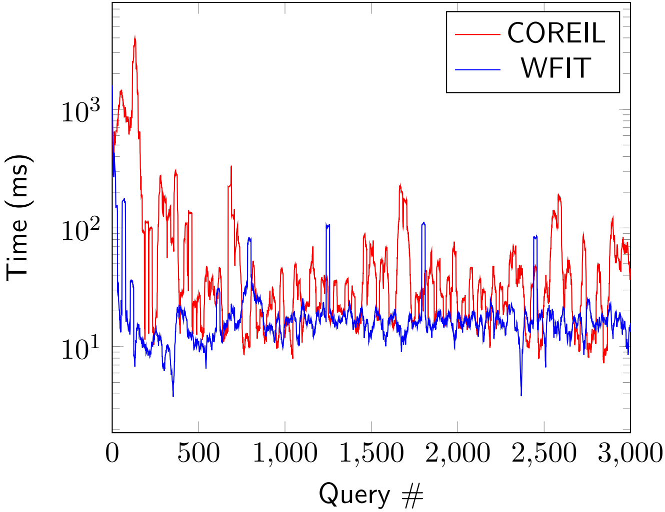
# Box-plot Analysis



# Overhead Cost Analysis



# Effectiveness



# 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

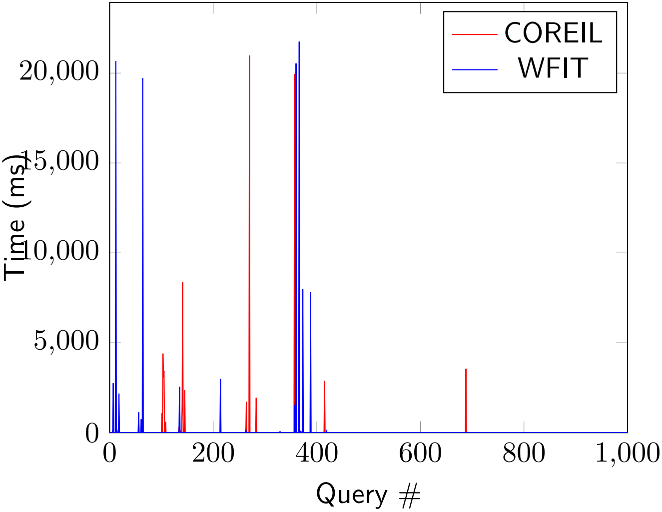
# Questions?

# Thank you

# Algorithm: Least Square Policy Iteration with RLSE

Initialize the configuration $s_0$.
Initialize $\boldsymbol{\theta}^0 = \boldsymbol{\theta} = \mathbf{0}$ and $B^0 = \epsilon I$.
Initialize $\boldsymbol{\zeta}^0 = \mathbf{0}$ and $\overline{B}^0 = \epsilon I$.
**for** t=1,2,3,... **do**
    Let $\hat{q}_t$ be the just received query.
    $s_t \leftarrow \underset{s \in S_{s_{t-1},\hat{q}_t}}{\arg\min} \; (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, q(s_{t-1}, s)) + (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s, \hat{q}_t) + \gamma \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$
    Change the configuration to $s_t$.
    Execute query $\hat{q}_t$.
    $\hat{C}^t \leftarrow \delta(s_{t-1}, s_t) + cost(s_t, \hat{q}_t)$.
    $\hat{\epsilon}^t \leftarrow (\boldsymbol{\zeta}^{t-1})^T \boldsymbol{\eta}(s_{t-1}, \hat{q}_t) - cost(s_{t-1}, \hat{q}_t)$
    $B^t \leftarrow B^{t-1} - \frac{B^{t-1}\boldsymbol{\phi}(s_{t-1})(\boldsymbol{\phi}(s_{t-1})-\gamma\boldsymbol{\phi}(s_t))^T B^{t-1}}{1+(\boldsymbol{\phi}(s_{t-1})-\gamma\boldsymbol{\phi}(s_t))^T B^{t-1}\boldsymbol{\phi}(s_{t-1})}$.
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \frac{(\hat{C}^t-(\boldsymbol{\phi}(s_{t-1})-\gamma\boldsymbol{\phi}(s_t))^T\boldsymbol{\theta}^{t-1})B^{t-1}\boldsymbol{\phi}(s_{t-1})}{1+(\boldsymbol{\phi}(s_{t-1})-\gamma\boldsymbol{\phi}(s_t))^T B^{t-1}\boldsymbol{\phi}(s_{t-1})}$.
    $(\overline{B}^t, \boldsymbol{\zeta}^t) \leftarrow RLSE(\hat{\epsilon}^t, \overline{B}^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\eta}^t)$
    **if** $(\boldsymbol{\theta}^t)$ converges **then**
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^t$.
    **end if**
**end for**

# Cost of Configuration Change Analysis



# Convergence

**Theorem**

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