Budget-aware Index Tuning with Reinforcement Learning

• Wentao Wu (Microsoft Research)
• Chi Wang (Microsoft Research)
• Tarique Siddiqui (Microsoft Research)
• Junxiong Wang (Cornell University)
• Vivek Narasayya (Microsoft Research)
• Surajit Chaudhuri (Microsoft Research)
• Philip A. Bernstein (Microsoft Research)
Cost-based index Tuning

Index Tuner
- Workload Parsing/Analysis
- Candidate Index Generation
- Configuration Enumeration

Best \( C \subseteq \{l_i\} \)
with respect to \( W, \Gamma \)

Database Server
- Query Optimizer (Extended)

What-If Calls
\( (q_i, C) \)

Cost \( (q_i, C) \)
What-if Calls are Expensive

- A what-if call is as expensive as a regular query optimizer call
- What-if calls dominate index tuning time
  - TPC-DS, 99 queries, 20 recommended indexes
Existing Work on Reducing What-if Calls

Reduce the search space of configuration enumeration.

- The configuration enumeration problem is NP-hard.
- There are exponentially many possible configurations and thus what-if calls.
- A classic solution is a greedy search approach that reduces the search space to polynomial size, which remains huge for large/complex workloads.

Other technologies

- Restrict the what-if calls to configurations with certain properties, e.g., atomic configurations.
- Effective reuse of cached what-if calls, which requires further extension/support from the query optimizer.
End user of index tuning needs to constrain the tuning time instead of letting it run forever.
- Microsoft’s Database Tuning Advisor (DTA) allows user to specify the maximum tuning time.

Under constrained tuning time, for large/complex workloads
- The number of what-if calls will go beyond the tuning time allowed, despite the previous techniques on reducing the number of what-if calls.

In this work, we study index tuning from a (new) constrained perspective, where
- The number of what-if calls (e.g., based on the tuning time budget) is given as a constraint.
- We focus on configuration enumeration under constrained number of what-if calls.
### Budget-constrained Configuration Search

**Budget allocation matrix**
- Row – configuration
- Column – query
- Cell – “X” if a what-if call is used

**For cells where what-if calls are not used, we use “derived cost.”**
- \( d(q, C) = \min_{S \subseteq C} \text{cost}(q, C) \)

**Problem formulation**
- Input: \( W, B \) (and other constraints \( \Gamma' \))
- Output: Best configuration \( C^* \)
- Budget constraint: The number of cells marked “X” = \( B \)

#### Example Configuration Table

<table>
<thead>
<tr>
<th>( C/q )</th>
<th>( q_1 )</th>
<th>( q_2 )</th>
<th>( q_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {I_1} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I_2} )</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( {I_3} )</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( {I_1, I_2} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I_1, I_3} )</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>( {I_2, I_3} )</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( {I_1, I_2, I_3} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Budget-aware Variants of Greedy Search

- **Greedy search**
  - (Base) Find the best singleton.
  - (Induction) Find the best configuration of size \( k + 1 \) by extending the best configuration of size \( k \).

- **Budget allocation in greedy search**
  - First come first serve (FCFS)
  - Two-phase
  - Atomic configuration

![Diagram of Greedy Search](image1)

- **(a) Greedy search**

<table>
<thead>
<tr>
<th>C/q</th>
<th>q₁</th>
<th>q₂</th>
<th>q₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {I₁} )</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( {I₂} )</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( {I₃} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₁, I₂} )</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>( {I₁, I₃} )</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>( {I₂, I₃} )</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( {I₁, I₂, I₃} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **(b) FCFS**

- **(c) Two-phase**

<table>
<thead>
<tr>
<th>C/q</th>
<th>q₁</th>
<th>q₂</th>
<th>q₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {I₁} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₂} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₃} )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₁, I₂} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₁, I₃} )</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {I₂, I₃} )</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>( {I₁, I₂, I₃} )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **(d) Atomic configuration**
Budget-aware Configuration Search using Reinforcement Learning (RL)

An exploration/exploitation trade-off

- **Exploration**: New configurations that have not yet been visited.
- **Exploitation**: Expand known promising configurations to include more indexes.

Reinforcement learning

- A principled way of dealing with exploration/exploitation trade-off.
Configuration Search as Markov Decision Process (MDP)

- **State** $s$: Configuration
- **Action** $a$: Index (to be included)
- **Transition probability** $p$: Deterministic
- **Reward** $r$: Percentage improvement of the workload $W$ over the state/configuration $C$

$$
\eta(W, C) = \left(1 - \frac{\text{cost}(W, C)}{\text{cost}(W, \emptyset)}\right) \times 100\%
$$
Monte Carlo Tree Search
Action Selection Policy

• **UCT**
  • Pick the action $a$ that maximizes the UCB (upper-confidence bound) score:
    \[
    \text{argmax}_a \left[ \hat{Q}(s, a) + \lambda \cdot \sqrt{\frac{\ln N(s)}{n(s, a)}} \right]
    \]
  • $\hat{Q}(s, a)$ is the estimated action-value function.
  • $N(s)$ is the number of times that $s$ is visited.
  • $n(s, a)$ is the number of times that the action $a$ is taken.

• **$\epsilon$-greedy**
  • Pick the action $a$ with respect to the probability:
    \[
    \text{Pr}(a|s) = \frac{\hat{Q}(s, a)}{\sum_{b \in \mathcal{A}(s)} \hat{Q}(s, b)}
    \]
Action Selection Policy (Cont.)

• Address *sparsity* in the estimated action-value function $\hat{Q}(s, a)$.
  • Choose a “prior distribution” for $\hat{Q}(s, a)$.
  • Refine the “prior distribution” after observing rewards.

• For each action/index $a$, estimate its percentage improvement.
  • Independent of the state $s$.
  • Needs to be done in a budget-aware manner.
  • For each budget what-if call, first select a query, and then select one of its index $a$ (see the paper for details).
• General rollout policy in MCTS
  • Expand the visited configuration $s$ by randomly inserting $l$ indexes.

• If UCT is used as the action selection policy
  • Insert $l$ indexes uniformly randomly.

• If $\epsilon$-greedy is used as the action selection policy
  • Insert $l$ indexes based on their “prior distribution.”
Extraction of the Best Configuration

Best configuration explored (BCE)
- Return the best configuration found during MCTS.
- This includes both the configurations explored by MCTS and the configurations generated by rollout.

Best greedy (BG)
- Use a greedy strategy to traverse the search tree.
- There are various options for the greedy strategy.
- Our current implementation
  - Run the greedy search algorithm again and return the configuration with the minimum derived cost.
Experiment Settings

• Datasets and workloads

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th># Queries</th>
<th># Tables</th>
<th>Avg. # Joins</th>
<th>Avg. # Filters</th>
<th>Avg. # Scans</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB</td>
<td>9.2GB</td>
<td>33</td>
<td>21</td>
<td>7.9</td>
<td>2.5</td>
<td>8.9</td>
</tr>
<tr>
<td>TPC-H</td>
<td>9.2GB</td>
<td>33</td>
<td>21</td>
<td>7.9</td>
<td>2.5</td>
<td>8.9</td>
</tr>
<tr>
<td>TPC-DS</td>
<td>9.2GB</td>
<td>33</td>
<td>21</td>
<td>7.9</td>
<td>2.5</td>
<td>8.9</td>
</tr>
<tr>
<td>Real-D</td>
<td>587GB</td>
<td>32</td>
<td>7,912</td>
<td>15.6</td>
<td>0.2</td>
<td>17</td>
</tr>
<tr>
<td>Real-M</td>
<td>26GB</td>
<td>317</td>
<td>474</td>
<td>20.2</td>
<td>1.5</td>
<td>21.7</td>
</tr>
</tbody>
</table>

• Baselines
  • Budget-aware variants of greedy search
  • Existing RL approaches to index tuning
Budget-aware Variants of Greedy Search

- **Vanilla greedy**
  - Standard greedy + FCFS (first come first serve)

- **Two-phase greedy**
  - Two-phase search + FCFS

- **Auto-admin greedy**
  - Two-phase greedy + atomic configuration
Comparison with Budget-aware Greedy (Benchmark Workloads)

Results on TPC-H

Results on TPC-DS
Comparison with Budget-aware Greedy (Real Workloads)

Results on Real-D

Results on Real-M
Existing RL Approaches to Index Tuning

**DBA bandits (ICDE 2021)**
- Model index selection as a “contextual bandit” problem.
- Customized to make it budget-aware.

**No DBA (arXiv 2018)**
- Solve the index selection problem using deep RL (e.g., deep Q-learning).
- Customized to make it budget-aware.
Comparison with Existing RL (Benchmark Workloads)

Results on TPC-H

Results on TPC-DS
Comparison with Existing RL (Real Workloads)

Results on Real-D

Results on Real-M
Summary of Contributions

• We proposed a problem formulation of budget-aware configuration search.

• We proposed a MCTS-based framework for budget-aware configuration search.

• We demonstrated that our MCTS-based framework outperforms both budget-aware variants of greedy search and existing RL techniques for index tuning, on both industrial benchmarks and real workloads.