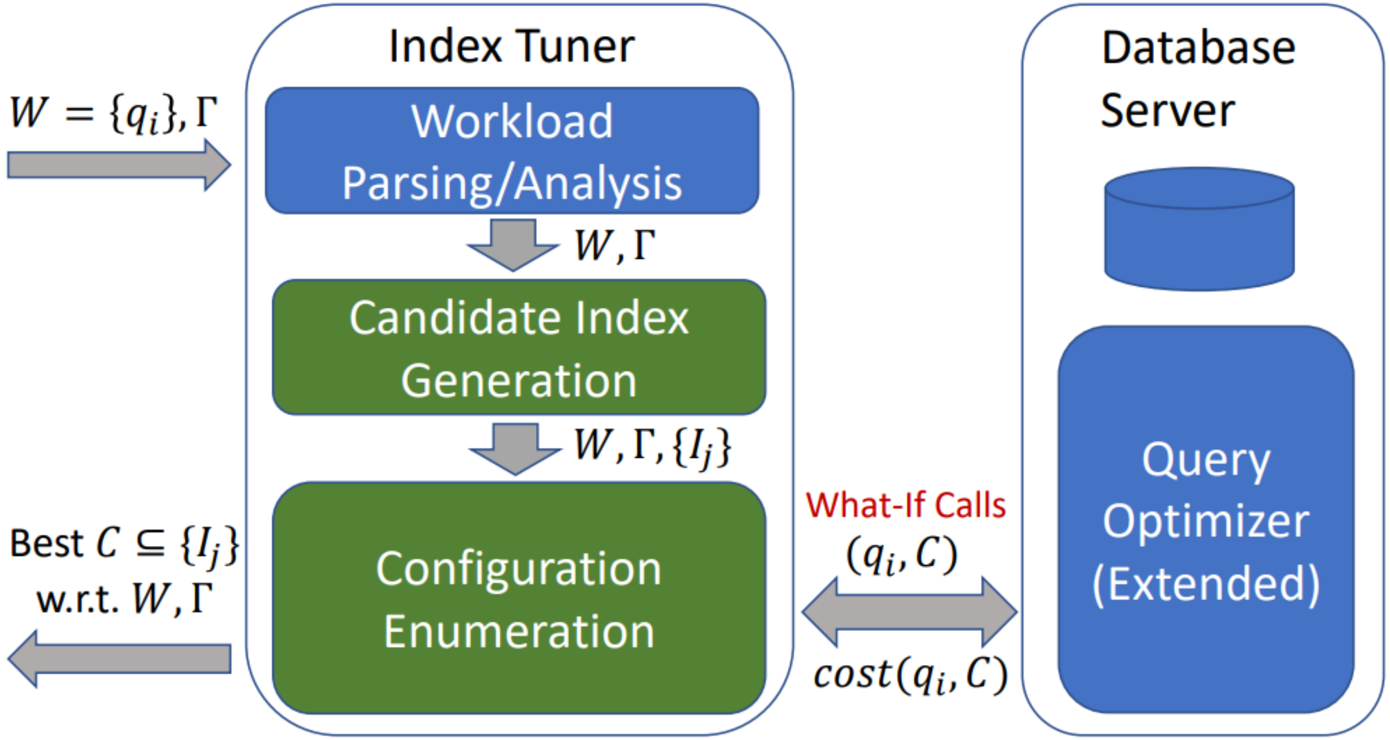
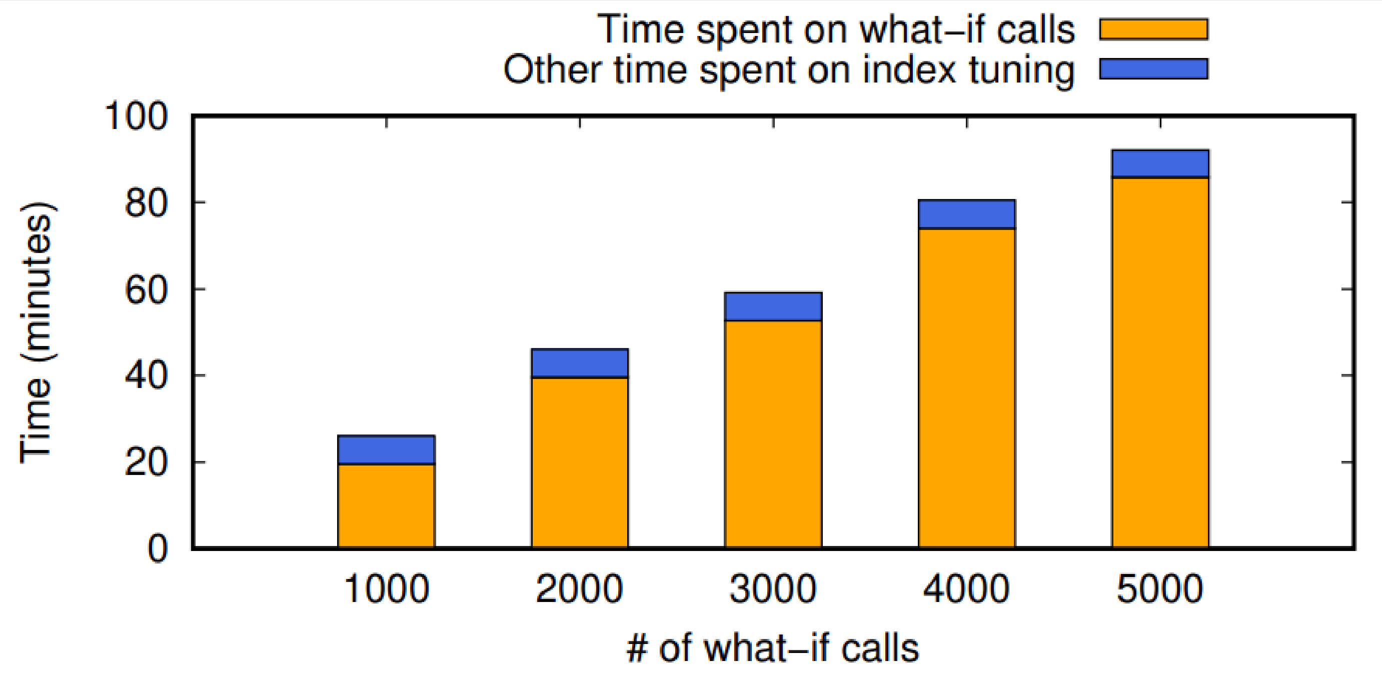
Budget-aware Index Tuning with Reinforcement Learning

- **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



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.

Budget-aware Index Tuning

- 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

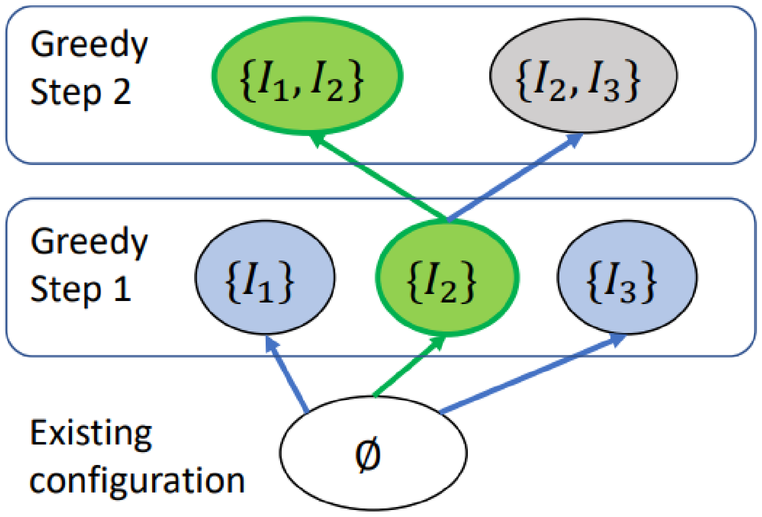
C/q	q_1	q_2	q_3
$\{I_1\}$	X		
$\{I_2\}$		X	X
$\{I_3\}$		X	
$\{I_1, I_2\}$	X		
$\{I_1, I_3\}$			X
$\{I_2, I_3\}$			X
$\{I_1, I_2, I_3\}$			

- 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 Γ)
  - Output: Best configuration C*
  - Budget constraint: The number of cells marked "X" = B

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



(a) Greedy search

C/q	q_1	q_2	q_3
$\{I_1\}$	X	X	X
$\{I_2\}$	X	X	X
$\{I_3\}$	X		
$\{I_1, I_2\}$			
$\{I_1, I_3\}$			
$\{I_2, I_3\}$			
$\{I_1, I_2, I_3\}$			

(b) FCFS

C/q	q_1	q_2	q_3
$\{I_1\}$	X	X	
$\{I_2\}$	X	X	
$\{I_3\}$	X		
$\{I_1, I_2\}$	X		
$\{I_1, I_3\}$			
$\{I_2, I_3\}$	X		
$\{I_1, I_2, I_3\}$			

(c) Two-phase

C/q	q_1	q_2	q_3
$\{I_1\}$	X	X	X
$\{I_2\}$	X	X	
$\{I_3\}$	X	X	
$\{I_1, I_2\}$			
$\{I_1, I_3\}$			
$\{I_2, I_3\}$			
$\{I_1, I_2, I_3\}$			

Atomic Configurations

Non-atomic Configurations

(d) Atomic configuration

Budget-aware Configuration Search using Reinforcement Learning (RL)

An exploration/exploitation trade-off

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

Reinforcement learning

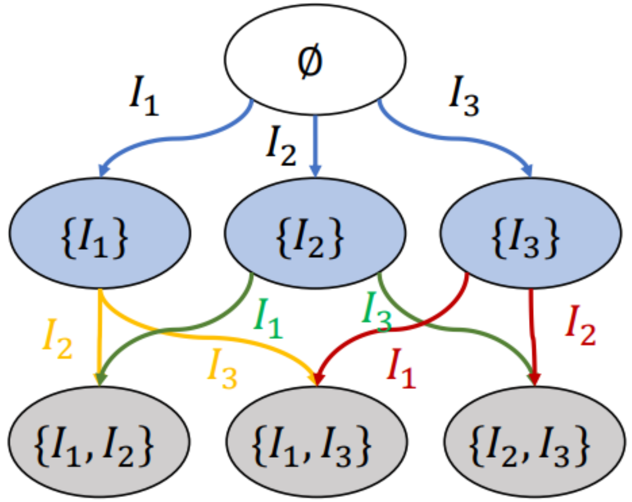
- A principled way of dealing with exploration/exploitation trade-off.

Configuration Search as Markov Decision Process (MDP)

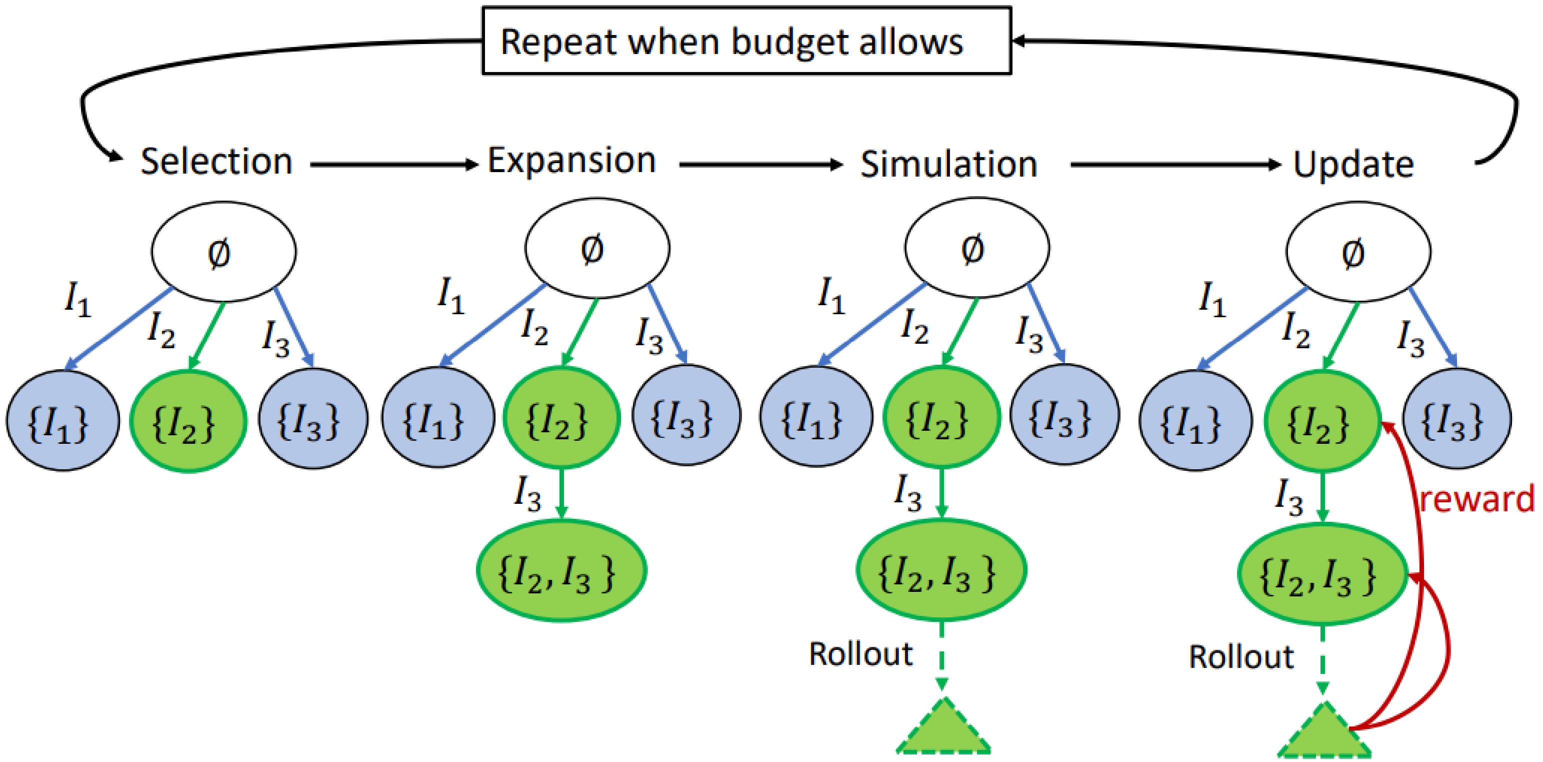
- *State* s: Configuration
- *Action* a: Index (to be included)
- *Transition probability* p: Deterministic
- *Reward* r: Percentage improvement of the workload W over the state/configuration C

$$\eta(W, C) = \left(1 - \frac{cost(W, C)}{cost(W, \emptyset)}\right) \times 100\%$$

Example MDP with $\{I_1, I_2, I_3\}$



Monte Carlo Tree Search



Action Selection Policy

- UCT
  - Pick the action a that maximizes the UCB (upper-confidence bound) score:

$$\underset{a}{\operatorname{argmax}}\left[\hat{Q}(s,a)+\lambda\cdot\sqrt{\frac{\ln N(s)}{n(s,a)}}\right]$$

  - $\hat{Q}(s,a)$ 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.
- ϵ-greedy
  - Pick the action a with respect to the probability: $\Pr(a|s)=\dfrac{\hat{Q}(s,a)}{\sum_{b\in\mathcal{A}(s)}\hat{Q}(s,b)}$

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).

Rollout Policy

- 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 ϵ-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

Name	Size	# Queries	# Tables	Avg. # Joins	Avg. # Filters	Avg. # Scans
JOB	9.2GB	33	21	7.9	2.5	8.9
TPC-H	*sf*=10	22	8	2.8	0.3	3.7
TPC-DS	*sf*=10	99	24	7.7	0.5	8.8
Real-D	587GB	32	7,912	15.6	0.2	17
Real-M	26GB	317	474	20.2	1.5	21.7

- 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)

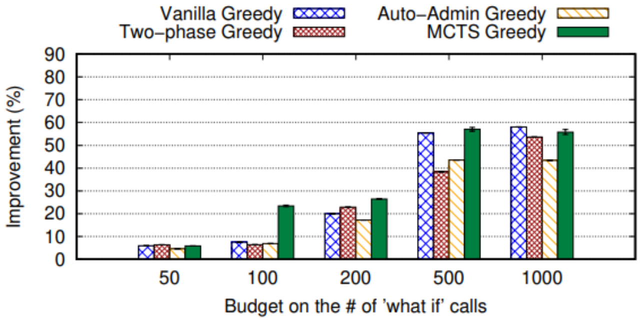
Two-phase greedy

- Two-phase search + FCFS

Auto-admin greedy

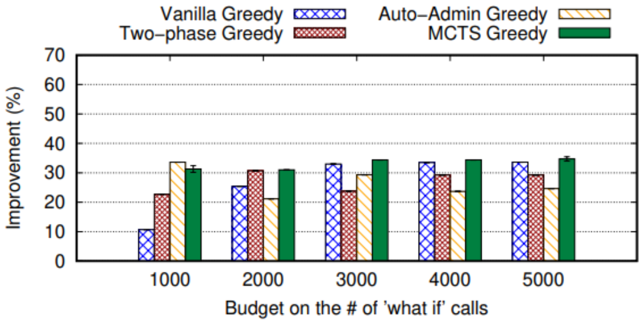
- Two-phase greedy + atomic configuration

Comparison with Budget-aware Greedy (Benchmark Workloads)



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

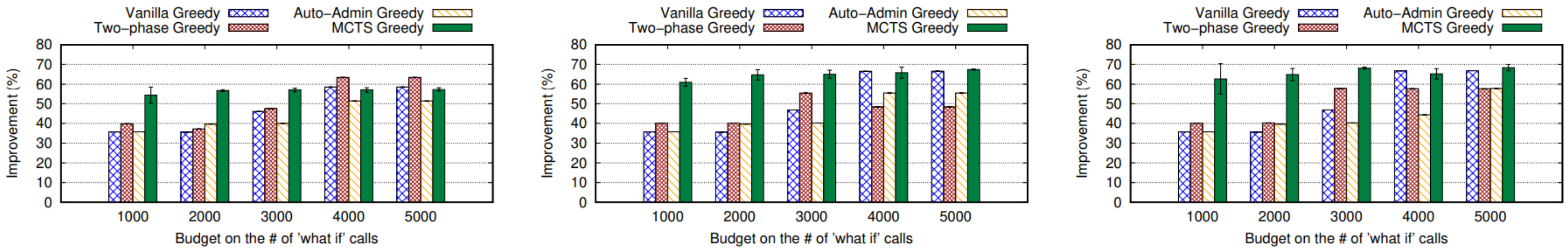
Results on TPC-H



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

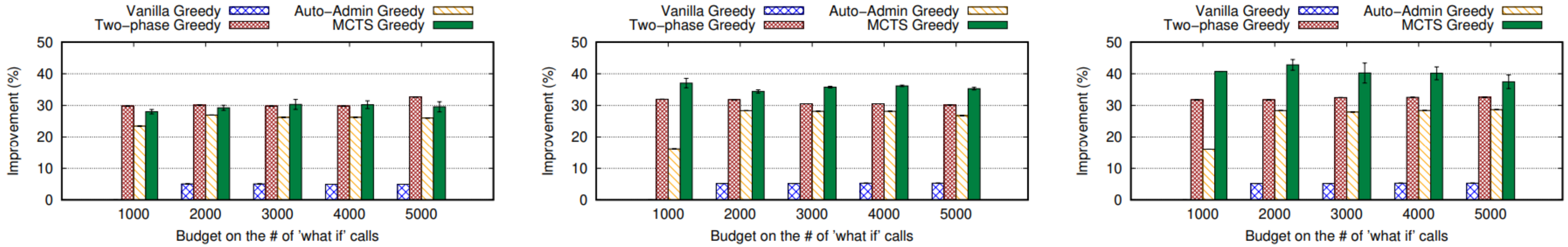
Results on TPC-DS

Comparison with Budget-aware Greedy (Real Workloads)



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

Results on Real-D



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

Results on Real-M

Existing RL Approaches to Index Tuning

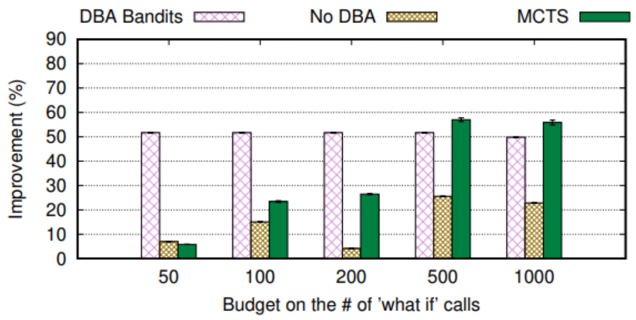
DBA bandits (ICDE 2021)

- Model index selection as a “contextual bandit” problem.
- Customized to make it budget-aware.

No DBA (arXiv 2018)

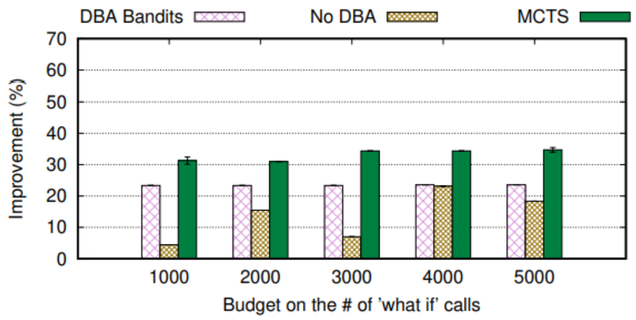
- Solve the index selection problem using deep RL (e.g., deep Q-learning).
- Customized to make it budget-aware.

Comparison with Existing RL (Benchmark Workloads)



(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

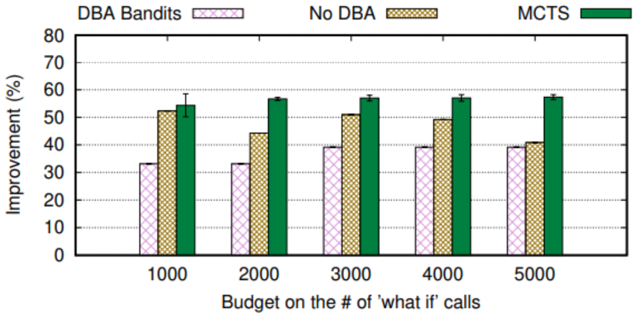
Results on TPC-H



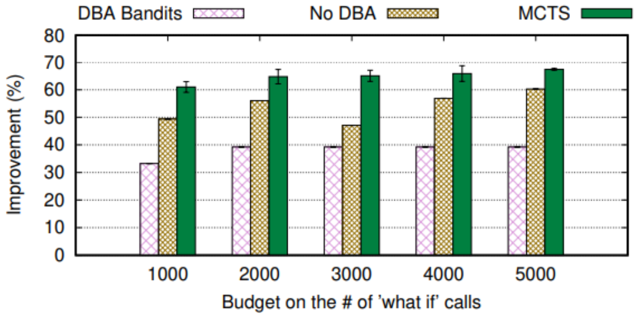
(a) $K = 5$ (b) $K = 10$ (c) $K = 20$

Results on TPC-DS

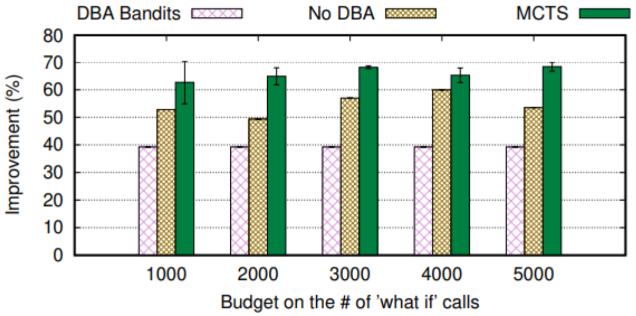
Comparison with Existing RL (Real Workloads)



(a) $K = 5$

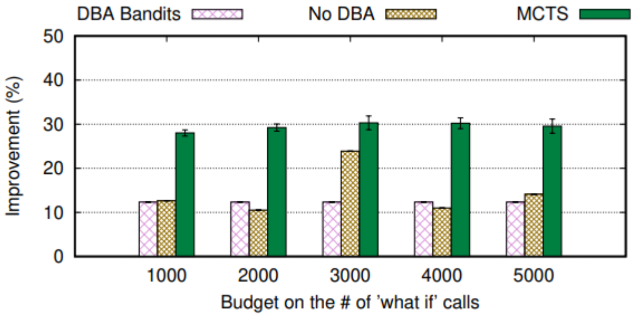


(b) $K = 10$

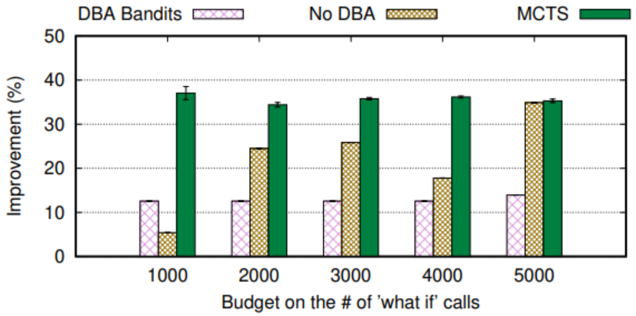


(c) $K = 20$

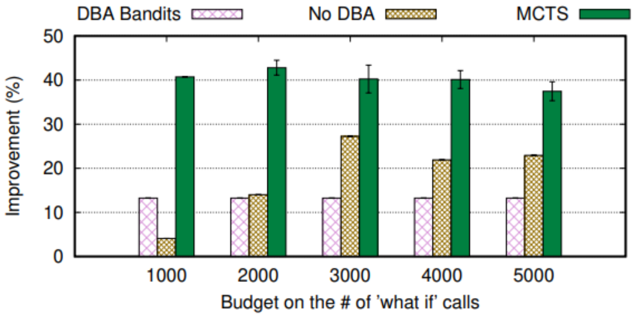
Results on Real-D



(a) $K = 5$



(b) $K = 10$



(c) $K = 20$

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.