No DBA? No regret! Multi-armed bandits for index tuning of analytical and HTAP workloads with provable guarantees

R. Malinga Perera, Bastian Oetomo, Benjamin I. P. Rubinstein, and Renata Borovica-Gajic

Abstract—Automating physical database design has remained a long-term interest in database research due to substantial performance gains afforded by optimised structures. Despite significant progress, a majority of today's commercial solutions are highly manual, requiring offline invocation by database administrators (DBAs). This status quo is untenable: identifying representative static workloads is no longer realistic; and physical design tools remain susceptible to the query optimiser's cost misestimates. Furthermore, modern application environments like hybrid transactional and analytical processing (HTAP) systems render analytical modelling next to impossible. We propose a self-driving approach to online index selection that does not depend on the DBA and query optimiser, and instead learns the benefits of viable structures through strategic exploration and direct performance observation. We view the problem as one of sequential decision making under uncertainty, specifically within the bandit learning setting. Multi-armed bandits balance exploration and exploitation to provably guarantee average performance that converges to policies that are optimal with perfect hindsight. Our comprehensive empirical evaluation against a state-of-the-art commercial tuning tool demonstrates up to 75% speed-up in analytical processing environments and 59% speed-up in HTAP environments. Lastly, our bandit framework outperforms a Monte Carlo tree search (MCTS)-based database optimiser, providing up to 24% speed-up.

Index Terms—Physical Design Tuning, Index Tuning, HTAP, Multi-Armed Bandits, Reinforcement Learning.

1 Introduction

With the growing complexity and variability of database applications and their hosting platforms (e.g., multi-tenant cloud environments), automated physical design tuning, particularly automated index selection, has re-emerged as a contemporary challenge for database management systems. Most database vendors offer automated tools for physical design tuning within their product suites [1], [2], [3]. Such tools form an integral part of broader efforts toward fully automated database management systems which aim to: a) decrease database administration costs and thus total costs of ownership [4], [5]; b) help non-experts use database systems; and c) facilitate hosting of databases on dynamic environments such as cloud-based services [6], [7], [8], [9]. Most physical design tools take an *off-line* approach, where DBAs must decide when to invoke the tool and what representative training workload to provide [10]. Where *online* solutions are provided [8], [11], [12], [13], questions remain: How can tools generalise beyond queries seen to dynamic ad-hoc workloads, where queries are unpredictable and non-stationary? And importantly, is the quality of proposed designs in any way guaranteed?

Modern analytics workloads are dynamic in nature with ad-hoc queries common [14], e.g., data exploration workloads adapt to past query responses [15]. Such ad-hoc workloads hinder automated tuning since: a) inputting representative information to design tools is infeasible under time-evolving workloads; and b) reacting too quickly to changes may result in performance variability, where indices are continuously dropped and created. Any robust automated physical design solution must address such challenges [11].

The situation is further aggravated in HTAP environments, that consist of online transaction processing (OLTP) and online analytical processing (OLAP) workloads. While indices provide (primarily) positive benefits to OLAP queries, they hinder the OLTP performance due to the additional index maintenance overhead. Furthermore, in dynamic settings, workload composition (i.e., analytical to transactional ratio) can vary over time, making it even more challenging to identify useful indices that boost overall workload performance.

To compare alternative physical design structures, automated design tools use a cost model employed by the query optimiser, typically exposed through a "what-if" interface [16], as the sole source of truth. However such cost models make inappropriate assumptions about data characteristics [17], [18]: commercial DBMSs often assume attribute value independence and uniform data distributions when sufficient statistics are unavailable [18], [19], [20]. As a result, estimated benefits of proposed designs may diverge significantly from actual workload performance [8], [9], [20], [21], [22]. Even with more complex data distribution statistics such as single- and multi-column histograms, the issue remains for complex workloads [20]. Moreover, data additions and updates in HTAP environments continuously invalidate statistics, compounding the effect of the optimiser misestimates. Keeping statistics up-to-date in such a setting requires extra effort.

• *R. M. Perera, Bastian Oetomo, Benjamin I.P. Rubinstein and Renata Borovica Gajic are with University of Melbourne, Australia. E-mail:{warnakula,bastian.oetomo,benjamin.rubinstein, renata.borovica}@unimelb.edu.au.*

Manuscript received April 19, 2005; revised August 26, 2015.

In this paper, we demonstrate that even in ad-hoc environments where queries are unpredictable, there are opportunities for index optimisation. We argue that the problem of online index selection under ad-hoc, analytical and HTAP workloads can be efficiently formulated within the multi-armed bandit (MAB) learning setting—a tractable form of Markov decision process. MABs take arms or actions (selecting indices) to maximise cumulative rewards, trading off exploration of untried actions with exploitation of actions that maximise rewards observed so far (see Figure 1). MABs permit learning from observations of actual performance, and need not rely on potentially misspecified cost models. Unlike initial efforts with applying learning for physical design, e.g., more general forms of reinforcement learning [23], bandits offer regret bounds that *guarantee* the fitness of dynamically-proposed indices [24]. In critical production environments, the uncertainties of online learned solutions can create doubts in a DBA's mind, making safety guarantees critical.

The key contributions of the paper can be summarised as:

- We model index tuning as a multi-armed bandit, proposing design choices that lead to a practical, competitive solution.
- Our proposed design achieves a worst-case safety guarantee against any optimal fixed policy, as a consequence of a corrected regret analysis of the C^{2}UCB bandit.
- We introduce a new bandit flavour that extends the existing contextual and combinatorial bandit where structured rewards are observed for each arm, providing additional feedback for the bandit. This bandit variation enjoys a superior regret bound compared to the C^{2}UCB bandit.
- Our comprehensive experiments demonstrate MAB's superiority over a state-of-the-art commercial physical design tool and a deep reinforcement learning agent, with up to 75% speed-up in the former and 58% speed-up in the latter case, under dynamic, analytical workloads.
- We showcase MAB's ability to perform in complex HTAP environments, which are notoriously challenging for index tuning, delivering up to 59% and 24% speed-up over the state-of-the-art commercial design tool and Monte Carlo tree search (MCTS)-based database optimiser, respectively.

2 Problem Formulation

The goal of the *online database index selection problem* is to choose a set of indices (referred to as a *configuration*) that minimises the total running time of a workload sequence within a given memory budget. Neither the workload sequence, nor system run times, are known in advance.

We adopt the problem definition of [13]. Let the *workload* $W = (w_1, w_2, \ldots, w_T)$ be a sequence of *mini-workloads* (e.g., a sequence of individual statements), I the set of *secondary indices*, $C_{mem}(s)$ represent the memory space required to materialise a configuration $s \subseteq I$, and $\mathcal{S} = \{s \subseteq I \,|\, C_{mem}(s) \leq M\} \subseteq 2^I$ be the class of *index configurations* feasible within our total memory allowance M. Our goal is to propose a configuration sequence $S = (s_0, s_1, \ldots, s_T)$, with $s_t \in \mathcal{S}$ as the configuration in round t and $s_0 = \emptyset$ as the starting configuration, which minimises the *total workload time* $C_{tot}(W, S)$ defined as:

$$C_{tot}(W, S) = \sum_{t=1}^{T} C_{rec}(t) + C_{cre}(s_{t-1}, s_t) + C_{exc}(w_t, s_t) \ .$$

Here $C_{rec}(t)$ refers to the *recommendation time* in round t (defined as running time of the recommendation tool) and $C_{cre}(s_{t-1}, s_t)$ refers to the incremental index creation time in transitioning from configuration s_{t-1} to s_t. Finally, $C_{exc}(w_t, s_t)$ denotes the execution time of mini-workload w_t under the configuration s_t, namely the sum of response times of individual statements.

At round t, the system:

1) Chooses a set of indices $s_t \in \mathcal{S}$ in preparation for upcoming workload w_t, without direct access to w_t. s_t only depends on observation of historical workloads $(w_1, \ldots, w_{t-1})$, corresponding sets of chosen indices, and resulting performance;
2) Materialises the indices in s_t which do not exist yet, that is, all indices in the set difference $s_t \backslash s_{t-1}$; and
3) Receives workload w_t, executes all the statements therein, and measures elapsed time of each individual statements and each operator in the corresponding query plan.

3 Contextual Combinatorial Bandits

In this paper, we argue that online index selection can be successfully addressed using multi-armed bandits (MABs) from statistical machine learning, where each arm corresponds to an index. We first present necessary background on MABs, outlining the **essential properties** that we exploit in our work (i.e., bandit context and combinatorial arms) to converge to highly performant index configurations.

We use the following notation. We denote non-scalar values with boldface: lowercase for (by default column) vectors and uppercase for matrices. We also write $[k] = \{1, 2, \ldots, k\}$ for $k \in \mathbb{N}$, and denote the transpose of a matrix or a vector with a prime.

The contextual combinatorial bandit setting under *semi-bandit* feedback involves repeated selections from k possible actions, over rounds $t = 1, 2, \ldots$, in which the MAB:

1) Observes a *context* feature vector (possibly random or adversarially chosen) of each action or *arm* $i \in [k]$, denoted as $\boldsymbol{X}_t = \{\boldsymbol{x}_t(i)\}_{i \in [k]}$, for $\boldsymbol{x}_t(i) \in \mathbb{R}^d$, along with their costs, c_i;
2) Selects or *pulls* a set of arms (referred to as *super arm*) $s_t \in \mathcal{S}_t$, where we restrict the class of possible super arms $\mathcal{S}_t \subseteq \mathcal{S}'_t = \left\{ s \subseteq [k] \,\middle|\, \sum_{i \in s} c_i \leq M \right\} \subseteq 2^{[k]}$; and
3) For each $i_t \in s_t$, observes random *scores* $r_t(i_t)$ drawn from fixed but unknown arm distribution which depends solely on the arm i_t and its context $\boldsymbol{x}_t(i_t)$, whose true expected values are contained in the unknown variable $\boldsymbol{r}^\star_t = \{\mathbb{E}[r_t(i)]\}_{i \in [k]}$.

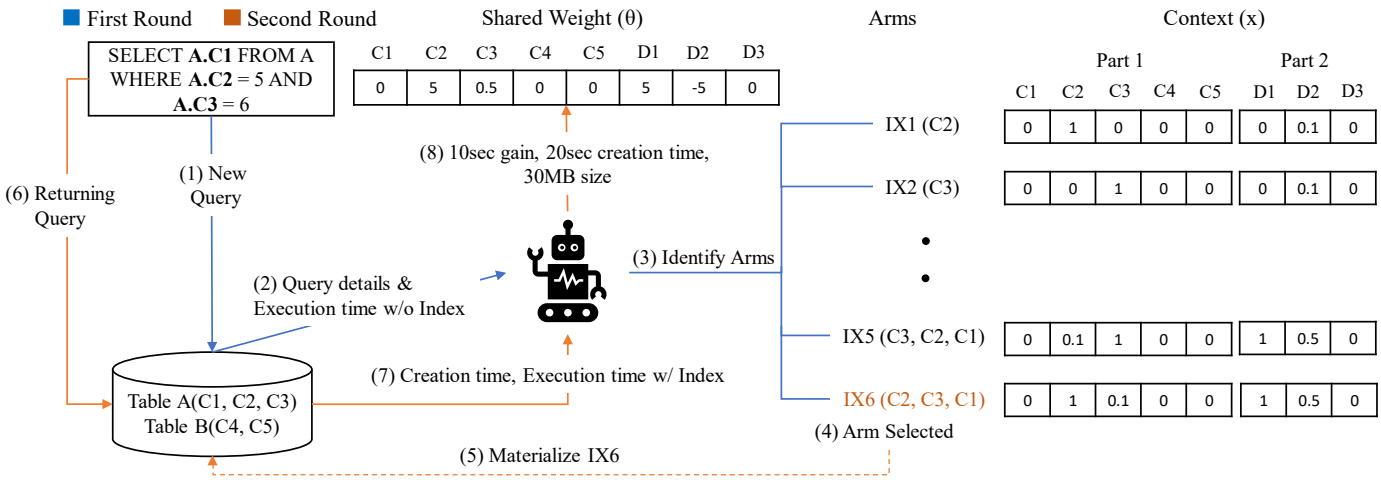

Fig. 1. An abstract view of the proposed bandit learning-based framework for online index selection.

Remark 1. The contextual combinatorial bandit setting is a special case of a Markov decision process, which is solved in general by reinforcement learning (RL). The key difference is that in bandits, state transition is not affected by MAB actions, only rewards are. States (observed via contexts) arrive arbitrarily. This simplicity admits theoretical guarantees for practical MAB learners, where state-of-the-art RL agents regularly have none. When playing in a bandit setting, in practice MAB learners may converge faster than their (typically over parametrised) RL cousins.

A MAB's goal is to maximise the cumulative expected reward $\sum_t \mathbb{E}[R_t(s_t)] = \sum_t g(s_t, \boldsymbol{r}_t^\star, \boldsymbol{X}_t)$ for a known function g. This function g need not be a simple summation of all the scores, however is typically assumed to be monotonic and Lipschitz smooth in the arm scores.

Definition 1. A *monotonic* function $g(s, \mathbf{r}, \mathbf{X})$ is non-decreasing in $\mathbf{r}$: for all $s, \mathbf{X}$, if $\mathbf{r} \preceq \mathbf{r}'$ then $g(s, \mathbf{r}, \mathbf{X}) \leq g(s, \mathbf{r}', \mathbf{X})$.

Definition 2. Function $g(s, \mathbf{r}, \mathbf{X})$ is *C-Lipschitz (uniformly) in* $\mathbf{r}$, if $|g(s, \mathbf{r}, \mathbf{X}) - g(s, \mathbf{r}', \mathbf{X})| \leq C \cdot \|\mathbf{r} - \mathbf{r}'\|_2$, for all $\mathbf{r}, \mathbf{r}'$, $\mathbf{X}, s$.

The core challenge in this problem is that the expected scores for all arms $i \in [k]$ are unknown. Refinement of a bandit learner's approximation for arm i is *generally* only possible by including arm i in the super arm, as the score for arm i is not observable when i is not played. This suggests solutions that balance *exploration* and *exploitation*. Even though at first glance it may seem that each arm needs to be explored at least once, placing practical limits on large numbers of arms, there is a remedy to this as will be discussed shortly.

The C^{2}UCB algorithm. Used to solve the contextual combinatorial bandit problem, the C^{2}UCB Algorithm [24] models the arms' scores as linearly dependent on their contexts: $r_t(i) = \boldsymbol{\theta}' \boldsymbol{x}_t(i) + \varepsilon_t(i)$ for unknown zero-mean (sub-gaussian) random variable ε_t, unknown but fixed parameter $\boldsymbol{\theta} \in \mathbb{R}^d$, and known context $\boldsymbol{x}_t(i)$. It is crucial to notice the implication that, **all learned knowledge is contained in estimates of $\boldsymbol{\theta}$, which is shared between all arms, obviating the need to explore each arm**. Estimation of $\boldsymbol{\theta}$ can be achieved using ridge regression, with $|s_t|$ new data points $\{(\boldsymbol{x}_t(i), r_t(i))\}_{i \in s_t}$ available at round t, further *accelerating the convergence rate* of the estimator $\hat{\boldsymbol{\theta}}$, over observing only one example as might be naïvely assumed.

Point estimates on the expected scores can be made with $\bar{r}_t(i) = \hat{\boldsymbol{\theta}}_t' \boldsymbol{x}_t(i)$, where $\hat{\boldsymbol{\theta}}_t$ are trained coefficients of a ridge regression on observed rewards against contexts. However, this quantity is oblivious to the variance in the score estimation. Intuitively, to balance out the exploration and exploitation, it is desirable to add an *exploration boost* to the arms whose score we are less sure of (i.e., greater estimate variance). This suggests that the upper confidence bound (UCB) should be used, in place of the expected value, and which can be calculated [25] as:

$$\hat{r}_t(i) \quad = \quad \hat{\boldsymbol{\theta}}_t' \boldsymbol{x}_t(i) + \alpha_t \sqrt{\boldsymbol{x}_t(i)' \boldsymbol{V}_{t-1}^{-1} \boldsymbol{x}_t(i)} \ , \qquad (1)$$

where $\alpha_t > 0$ is the exploration boost factor, and $\boldsymbol{V}_{t-1}$ is the positive-definite $d \times d$ scatter matrix of contexts for the chosen arms up to and including round $t-1$. The first term of $\hat{r}_t(i)$ corresponds to arm i's immediate reward, whereas its second term corresponds to its exploration boost, as its value is larger when the arm is sensitive to the context elements we are less confident of (i.e., the underexplored context dimension). Hence, by using $\hat{r}_t(i)$ in place of $\bar{r}_t(i)$, arms with contexts lying in the underexplored regions of context space are more likely to be chosen, as higher scores yield higher g, assuming that g is monotonic increasing in the arm rewards.

Ideally, the super arm $s_t \in \mathcal{S}_t$ is chosen such that $g(s_t, \hat{\boldsymbol{r}}_t, \boldsymbol{X}_t)$ is maximised. However, it is sometimes computationally expensive to find such super arms. In such cases, it is often good enough to obtain a solution via some approximation algorithm where $g(\hat{\boldsymbol{r}}_t, \boldsymbol{X}_t, s_t)$ is *near* maximum. With this criterion in mind, we now define an α-*approximation oracle*.

Definition 3. An α-*approximation oracle* is an algorithm $\mathcal{A}$ that outputs a super arm $\bar{s} = \mathcal{A}(\boldsymbol{r}, \boldsymbol{X})$ with guarantee $g(\bar{s}, \boldsymbol{r}, \boldsymbol{X}) \geq \alpha \cdot \max_s g(s, \boldsymbol{r}, \boldsymbol{X})$, for some $\alpha \in [0, 1]$ and given input $\boldsymbol{r}$ and $\boldsymbol{X}$.

Algorithm 1 The C^{2}UCB Algorithm

Input: $\lambda, \alpha_1, \ldots, \alpha_T$
Initialize $\boldsymbol{V}_0 \leftarrow \lambda \boldsymbol{I}_d$, $\boldsymbol{b}_0 \leftarrow \boldsymbol{0}_d$
for $t \leftarrow 1, \ldots, T$ **do**
  Observe $\mathcal{S}_t$
  $\hat{\boldsymbol{\theta}}_t \leftarrow \boldsymbol{V}_{t-1}^{-1} \boldsymbol{b}_{t-1}$ ▷ estimate via ridge regression
  for $i \in [k]$ **do**
    Observe context $\boldsymbol{x}_t(i)$
    $\hat{r}_t(i) \leftarrow \hat{\boldsymbol{\theta}}_t' \boldsymbol{x}_t(i) + \alpha_t \sqrt{\boldsymbol{x}_t(i)' \boldsymbol{V}_{t-1}^{-1} \boldsymbol{x}_t(i)}$
  end for
  $s_t \leftarrow \mathcal{A}(\hat{\boldsymbol{r}}_t, \boldsymbol{X}_t)$ ▷ using α-approximation oracle
  Play s_t and observe $r_t(i)$ for all $i \in s_t$
  $\boldsymbol{V}_t \leftarrow \boldsymbol{V}_{t-1} + \sum_{i \in s_t} \boldsymbol{x}_t(i) \boldsymbol{x}_t(i)'$ ▷ regression update
  $\boldsymbol{b}_t \leftarrow \boldsymbol{b}_{t-1} + \sum_{i \in s_t} r_t(i) \boldsymbol{x}_t(i)$ ▷ regression update
end for

Note that knapsack-constrained submodular programs are efficiently solved by the greedy algorithm (iteratively select a remaining cost-feasible arm with highest available score) with $\alpha = 1 - 1/e$. C^{2}UCB is detailed in Algorithm 1.

The performance of a bandit algorithm is usually measured by its *cumulative regret*, defined as the total expected difference between the reward of the chosen super arm $\mathbb{E}[R_t(s_t)]$ and an optimal super arm $\max_{s \in \mathcal{S}_t} \mathbb{E}[R_t(s)]$ over T rounds. Such a metric is unfair to C^{2}UCB since its performance depends on the oracle's performance. This suggests assessing C^{2}UCB's performance with a metric using the oracle's performance guarantee as its measuring stick, as follows.

Definition 4. Let $\bar{s}$ be a super arm returned by an α-approximation oracle as a part of the bandit algorithm, and $\boldsymbol{r}_t^\star$ be a vector containing each arms' true expected scores. Then cumulative α-regret is the sum of expected instantaneous regret, $Reg_t^\alpha = \alpha \cdot \max_s g(s, \boldsymbol{r}_t^\star, \boldsymbol{X}_t) - g(\bar{s}_t, \boldsymbol{r}_t^\star, \boldsymbol{X}_t)$.

When g is assumed to be monotonic and Lipschitz continuous, [24] claimed that C^{2}UCB enjoys $\tilde{O}(\sqrt{T})$ α-regret. We have corrected an error in the original proof, as seen in Appendix, confirming the $\tilde{O}(\sqrt{T})$ α-regret. This expression is sub-linear in T, implying that the per-round average cumulative regret approaches zero after sufficiently many rounds. Consequently, online index selection based on C^{2}UCB comes endowed with a *safety guarantee* on worst-case performance: **selections become at least as good as an α-optimal policy (with perfect access to true scores); and potentially much better than any fixed policy**.

4 MAB FOR ONLINE INDEX SELECTION

Performant bandit learning for online index tuning demands arms covering important actions and no more, rewards that are observable and for which regret bounds are meaningful, and contexts and oracle that are efficiently computable and predictive of rewards. Each workload statement is monitored for characteristics such as running time, query predicates, payload, etc. (see Figure 1). These observations feed into generation of relevant arms and their contexts. The learner selects a desired configuration which is materialised. For returning statements, the system identifies benefits of the materialised indices, which are then shaped into the reward signal for learning.

Dynamic arms from workload predicates. Instead of enumerating all column combinations, *relevant* arms (indices) may be generated based on queries: combinations and permutations of query predicates (including join predicates), with and without inclusion of payload attributes from the selection clause. Such workload-based arm generation drastically reduces the action space, and exploits natural skewness of real-life workloads that focus on small subsets of attributes over full tables [15]. Workload-based arm generation is only viable due to dynamic arm addition (reflecting a dynamic action space) and is allowed by the bandit setting: we may define the set of feasible arms for each round at its start.

Context engineering. Effective contexts are predictive of rewards, efficiently computable, and promote generalisation to previously unseen workloads and arms. We form our context in two parts (see Figure 1).

Context Part 1: Indexed column prefix. We encode one context component per column. However unlike a bag-of-words or one-hot representation appropriate for text, similarity of arms depends on having similar column prefixes; common index columns is insufficient. This reflects a ***novel bandit learning aspect of the problem.*** A context component has value m^{-j} where j is the corresponding column's position in the index, *provided* that the column is included in the index and part of the workload. We experimented with values 1, 2, 10 and 100 for m, where 1 represents the one-hot encoding. We observe that smaller values (i.e., 1 and 2) do not provide sufficient differentiation between arms, while the larger values (i.e., 100) only differentiate based on the first column (there is insufficient representation for the rest of the columns). Thus we set m to 10. The value is set to 0 otherwise, including if its presence only covers the payload. Unlike a simple one hot encoding, this context enables the bandit to differentiate between arms with the same set of columns but different ordering, and reward the columns differently based on their position in the index.

Example 1. Under the simplest workload (single query) in Figure 1, our system generates six arms: four using different combinations and permutations of the predicates, two including the payload (covering indices). Index IX5 includes column C1, but the context for C1 is valued as 0, as this column is considered only due to the query payload.

Context Part 2: Derived statistical information. We represent statistical and derived information about the arms and workload, details available during statement execution, and sufficient statistics for unbiased estimates. This statistical information includes: a Boolean indicating a covering index, the estimated size of the index divided by the database size (if not materialised already, 0 otherwise), and the number of times the optimiser has picked this arm in recent rounds. This is shown in Figure 1 under D1, D2 and D3, respectively. Results showed very low sensitivity to the number of rounds considered for the D3 feature, however making it 0 leads to slight increases in creation times.

Reward shaping. As the goal of physical design tuning tools is to minimise end-to-end workload time, we incorporate index creation time and statement execution time into the reward for a workload. We omit index recommendation time, as it is independent of arm selection. However, we measure and report recommendation time of the MAB algorithm in our experiments. Recall that MAB depends only on observed execution statistics from implemented configurations and generalisation of the learned knowledge to unseen arms thereafter. Unlike OLAP workloads, under HTAP workloads, the statement execution time can be negatively impacted by the index maintenance operations, necessitating its inclusion in the reward.

The implementation of the reward for an arm includes the execution time as a *gain* $G_t(i, w_t, s_t)$ for a workload w_t by each arm i under configuration s_t. Indices can impact the execution time in multiple ways. We split the execution time gain into three components: a) data scan gains (G_t^{ds}), b) index maintenance gains (G_t^{im}, usually a negative value), and c) other areas of the query plan which can be difficult to attribute to a single index (unclaimed gains) (G_t^{un}).

$$\begin{aligned} & G_t(i, w_t, s_t) \\ = \;& G_t^{ds}(i, w_t, s_t) + G_t^{im}(i, w_t, s_t) + G_t^{un}(i, w_t, s_t) \;\; . \end{aligned}$$

Data scan gains: The data scan gain by index i for query q is defined as:

$$\begin{aligned} & G_t^{ds}(i, \{q\}, s_t) \\ = \;& \left[C_{tab}(\tau(i), q, \emptyset) - C_{tab}(\tau(i), q, \{i\})\right] \mathbb{1}_{\mathcal{U}(s,q)}(i) \,, \end{aligned}$$

where $\mathcal{U}(s, q)$ denotes the list of indices used by the optimiser in query q under a given configuration s. $C_{tab}(\tau(i), q, \emptyset)$ represents the full table scan time for table $\tau(i)$ and query q, where $\tau(i)$ is the table which i belongs to.[1]

Index maintenance gains: Index maintenance operations can take different forms based on the number of rows updated. The optimiser typically opts for row-wise updates for a small number of rows and index-wise updates otherwise. In the second case, we can easily capture the maintenance gain of an index as each index is updated separately. This is however not straightforward in the case of row-wise updates, where all indices are bulk updated for each row. On these occasions, we compute the total maintenance gain ($G_t^{im}(\mathcal{V}(s, q), \{q\}, s_t)$) for all secondary indices that require maintenance due to a statement q under a given configuration s and equally divide it among the updated indices. $\mathcal{V}(s, q)$ represents the set of indices updated under configuration s by the statement q.

$$G_t^{im}(\mathcal{V}(s, q), \{q\}, s_t) = \left[C_{im}(q, \emptyset) - C_{im}(q, s)\right] \;\; .$$

where $C_{im}(q, \emptyset)$ and $C_{im}(q, s)$ represent the index maintenance time without secondary indices and index maintenance time under configuration s, respectively.

$$\begin{aligned} & G_t^{im}(i, \{q\}, s_t) \\ = \;& \left[G_t^{im}(\mathcal{V}(s, q), \{q\}, s_t) / \left|\mathcal{V}(s, q)\right|\right] \mathbb{1}_{\mathcal{V}(s,q)}(i) \;\; . \end{aligned}$$

Unclaimed gains: Sometimes the impact of the indices can be indirect. For example, while introducing a new index can speed up the data scan, it can require sorting, which can be costly. Therefore, when the overall gain results in a performance regression, MAB needs to take corrective actions to trigger a different query plan. These gains are captured under G_t^{un}. G_t^{un} for a statement is computed by subtracting all the other gains (data scan and index maintenance) from the total gain (G_t^{to}). The total gain (G_t^{to}) can be obtained by using statement running times before and after index creation.

$$G_t^{to}(\{q\}, s_t) = \left[C_{to}(q, \emptyset) - C_{to}(q, s_t)\right] \;\; .$$

Then we equally divide this cost among participating indices ($\mathcal{U}(s, q) \cup \mathcal{V}(s, q)$).

The gain for a workload relates to the gain for individual statement by:

$$G_t(i, w_t, s_t) = \sum_{q \in w_t} G_t(i, \{q\}, s_t) \,.$$

By this definition, gain $G_t(i, w_t, s_t)$ will be 0 if i is not used by the optimiser in the current round t and can be negative if the index creation leads to a performance regression or if the index incurs a maintenance cost.[2] Creation time of i is taken as a negative reward, only if i is materialised in round t, and is 0 otherwise:

$$r_t(i) = G_t(i, w_t, s_t) - C_{cre}(s_{t-1}, \{i\}) \;\; .$$

Minimising the end-to-end workload time, or rather, maximising the end-to-end workload time gained, is the goal of the bandit. As defined earlier, the total workload time C_{tot} is the sum of *execution*, *recommendation* and *creation* times accumulated over rounds. As such, minimising each round's summand is an equivalent problem. Modifying the execution time to the time gain while ignoring the recommendation time yields per-round super arm reward of:

$$\begin{aligned} R_t(s_t) &= C_{exc}(w_t, \emptyset) - \left[C_{exc}(w_t, s_t) + C_{cre}(s_{t-1}, s_t)\right] \\ &\approx \sum_{i \in s_t} G_t(i, w_t, s_t) - \sum_{i \in s_t} C_{cre}(s_{t-1}, \{i\}) \\ &= \sum_{i \in s_t} r_t(i) \;\; . \end{aligned}$$

Selection of the execution plan depends on the query optimiser, and as noted, the query optimiser may resolve to a sub-optimal query plan. As we show, the bandit is nonetheless resilient as it can quickly recover from any such performance regressions. Observed execution times encapsulate real-world effects, e.g., the interaction between statements, application properties, run-time parameters, etc. However, concurrent environments might require modifying the reward design based on specific performance targets (e.g., removing index creation time, or considering total workload run times over query run times). Since the end-to-end workload time includes the index creation and statement execution times, we are indirectly optimising for both efficiency and the quality of recommendations.

1. Due to the reactive nature of multi-armed bandits, we mostly observe a full table scan time for each table $\tau(i)$ and query q. When we do not observe this, we estimate it with the maximum secondary index scan/seek time.

2. The optimiser cost model does not have to agree that MAB choices are optimal. The recommended indices will still be used if the optimiser estimates that recommended MAB indices will provide a positive gain over a full table scan.

Algorithm 2 MAB Simulation for Index Tuning

```
1:  QS ← QueryStore()                          ▷ keeps query information
2:  C²UCB ← InitialiseBandit()                 ▷ A1, L 1-2
3:  while (TRUE) do
4:      queries ← getLastRoundWorkload()
5:      for all queries do
6:          if (isNewTemplate) then
7:              QS.add(query)
8:          else
9:              QS.update(query)
10:         end if
11:     end for
12:     QoI ← QS.getQoI()                      ▷ get queries of interest
13:     arms ← generateArms(QoI)
14:     X ← generateContext(arms, QoI)
15:     s_t ← C²UCB.recommend(arms, X)         ▷ A1, L 4-10
16:     C_cre ← materialise(s_t)
17:     C_exc ← executeCurrentWorkload()
18:     C²UCB.updateWeights(C_cre, C_exc)      ▷ A1, L 12-13
19: end while
```

A greedy oracle for super-arm selection. Recall that C^{2}UCB leverages a near-optimal oracle to select a super arm, based on individual arm scores [24]. As a sum of individual arm rewards, our super-arm reward has a (sub)modular objective function and (as easily proven) exhibits monotonicity and Lipschitz continuity. Approximate solutions to maximise submodular (diminishing returns) objective functions can be obtained with greedy oracles that are efficient and near-optimal [26]. Our implementation uses such an oracle combined with filtering to encourage diversity.

Initially, arms with negative scores are pruned. Then arm selection and filtering steps alternate, until the memory budget is reached. In the selection step, an arm is selected greedily based on individual scores. The filtering step filters out arms that are no longer viable under the remaining memory budget, or those that are already covered by the selected arms based on prefix matching. If a covering index is selected for a query, all other arms generated for that query will be filtered out. Note that filtering is a temporary process that only impacts the current round.

Bandit learning algorithm. Algorithm 2 shows the MAB algorithm, which wraps Algorithm 1 and handles the domain specific aspects of the implementation. We have divided Algorithm 1 into three main parts, initialisation (lines 1-2), arm recommendation (lines 4-10) and weight vector update (lines 12-13). These segments are utilised in Algorithm 2 as C^{2}UCB function calls. After initialising the bandit, Algorithm 2 summarises workload information using templates; these track frequency, average selectivity, first seen and last seen times of the statements which help to generate the best set of arms per round (i.e., QoI). The context is updated after each round based on the workload and selected set of arms. The bandit then selects the round's set of arms, forming a configuration to be materialised within the database. The reward will then be calculated based on observed execution statistics on a new set of statements, and will be used to update the shared weight. To support shifting workloads, where users' interests change over time, the learner may forget learned knowledge depending on the workload shift intensity (i.e., the number of newly introduced statement templates).

In our implementation, we perform bandit updates separately for creation time reward and execution time reward (line 13 of Algorithm 1). At the creation cost update, we temporarily make all context features 0 except for the context feature that is responsible for the index size. This can be viewed as an innovation of independent interest to the bandit community where we decompose the reward into multiple components and want to direct each reward component feedback to a subset of the features. We coin the term *focused update* in reference to this approach. This idea invites a new flavour of bandits elaborated in the next section.

5 Contextual Combinatorial Bandit with Structured Rewards

When rewards can be decomposed into component rewards under two key conditions, we hypothesise that a *focused update* can result in faster convergence: (i) when each reward component is directly related to a small subset of context features we create lower dimensional supervised learning problems; and (ii) when each reward component is directly observed we offer more opportunities for bandit feedback. Under focused updates we use each component of the reward to learn a part of the weight vector (see Figure 2). Indeed for this structured setting we modify the proof of C^{2}UCB to arrive at a tighter regret bound by a factor of $1/\sqrt{n_f}$, where n_f is the number of reward components (i.e., observed number of examples per round).

Our approach to structured rewards is by a reduction to C^{2}UCB. We modify the C^{2}UCB's formulation as if two examples are observed for pulled arm i in round t with respective rewards $\bar{r}_{t,1}(i)$ and $\bar{r}_{t,2}(i)$. Throughout both (sub round) observations, the overall arm reward function is fixed as $r_t(i) = \bar{r}_{t,1}(i) + \bar{r}_{t,2}(i)$. This permits learning at a faster rate, while still coordinating an overall arm reward estimate.

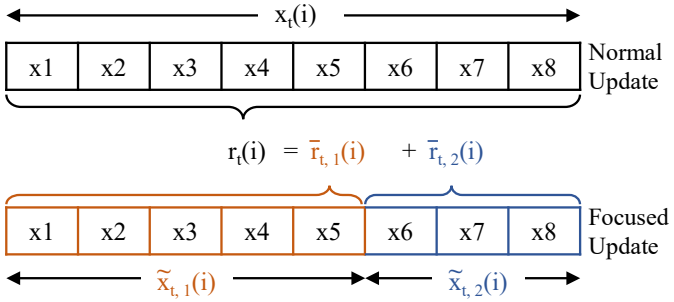

Fig. 2. Regular contextual updates vs focused update.

Example 2. In the bandit setting of this paper, we are motivated by the desire for context part 1 to be completely responsible for execution cost gains—we can learn the negative weight from index creation cost directly into part 2's index size feature. This enables us to switch off the creation cost overhead for already created arms by simply setting the index size context feature to zero.

This ability to use domain knowledge to tailor reward feedback to a subset of context features is a powerful benefit of structured rewards.

Let $\bar{\boldsymbol{x}}_{t,f}(i) \in \mathbb{R}^d$ be the f^{th} context of arm i at round t, such that $r_t(i) = \boldsymbol{\theta}_\star' \boldsymbol{x}_t(i) + \varepsilon_t = \boldsymbol{\theta}_\star'(\bar{\boldsymbol{x}}_{t,1}(i) + \bar{\boldsymbol{x}}_{t,2}(i)) + \varepsilon_{t,1} + \varepsilon_{t,2}$ for two independent zero-mean (subgaussian) noise random variables $\varepsilon_{t,1}$ and $\varepsilon_{t,2}$ with equal variance[3]. To observe the benefit of the focused update, we further assume that $\varepsilon_{t,1}$ and $\varepsilon_{t,2}$ are each $\frac{R}{\sqrt{2}}$-subgaussian, which makes ε_t R-subgaussian.[4] It should be noted that even though we have assumed that the overall context is the sum of the sub-contexts, Equation (1) from Section 3 still holds since the equation is oblivious to how the overall context is obtained from the sub-contexts. We further assume complementary sparse sub-contexts: overall context $\boldsymbol{x}_t(i)$ is the concatenation of $\tilde{\boldsymbol{x}}_{t,1}(i) \in \mathbb{R}^{d_1}$ and $\tilde{\boldsymbol{x}}_{t,2}(i) \in \mathbb{R}^{d_2}$, where $d = d_1 + d_2$ and $\bar{\boldsymbol{x}}_{t,1}(i)' = \begin{bmatrix}\tilde{\boldsymbol{x}}_{t,1}(i)' & \boldsymbol{0}_{1\times d_2}\end{bmatrix}$ and $\bar{\boldsymbol{x}}_{t,2}(i)' = \begin{bmatrix}\boldsymbol{0}_{1\times d_1} & \tilde{\boldsymbol{x}}_{t,2}(i)'\end{bmatrix}$.

To maintain optimal least squares learning given two observations per round with equal variances, at the end of round t we generalise our matrix $\boldsymbol{V}_t$ and b_t updates to:

$$\boldsymbol{V}_t = \lambda \boldsymbol{I} + \sum_{\tau=1}^{t}\sum_{f=1}^{2}\sum_{i\in S_\tau} \bar{\boldsymbol{x}}_{\tau,f}(i)\bar{\boldsymbol{x}}_{\tau,f}(i)'$$
$$\boldsymbol{b}_t = \sum_{\tau=1}^{t}\sum_{f=1}^{2}\sum_{i\in S_\tau} \bar{\boldsymbol{x}}_{\tau,f}(i)\bar{r}_{\tau,f}(i)\ .$$

Notice that this is different from C^{2}UCB's definition of $\boldsymbol{V}_t$ and $\boldsymbol{b}_t$, and hence a new regret analysis is warranted.

We exploit the fact that Theorem 4.2 in [24] holds regardless of the super arm S_t. Therefore, solely for the purpose of modifying the aforementioned theorem, we re-index the context and rewards such that $\bar{\boldsymbol{x}}_{t,f}(i) = \bar{\boldsymbol{x}}_t(i + kf)$ and $\bar{r}_{t,f}(i) = \bar{r}_t(i + kf)$, and we construct the effective super arm $S_t' = \{i' : i' = i + kf, i \in S_t, f \in \{0, 1\}\}$. As such, our definition of $\boldsymbol{V}_t$ and $\boldsymbol{b}_t$ can now be rewritten as:

$$\boldsymbol{V}_t = \lambda \boldsymbol{I} + \sum_{\tau=1}^{t}\sum_{i\in S_\tau'} \bar{\boldsymbol{x}}_{\tau}(i)\bar{\boldsymbol{x}}_{\tau}(i)'$$
$$\boldsymbol{b}_t = \sum_{\tau=1}^{t}\sum_{i\in S_\tau'} \bar{\boldsymbol{x}}_{\tau}(i)\bar{r}_{\tau}(i)\ ,$$

which is syntactically the same as the definition given in [24]. We need to be more careful in concluding the theorem, however, since it contains an intermediate step involving $\det(\boldsymbol{V}_t)$ as defined in [27]. Assuming that $\|\boldsymbol{x}_t(i)\| \le 1$, we bound $\det(\boldsymbol{V}_t)$ as follows:

$$\begin{aligned}
\det(\boldsymbol{V}_t) &\le \left(\frac{\operatorname{tr}(\boldsymbol{V}_t)}{d}\right)^d \\
&= \left(\frac{\operatorname{tr}(\lambda \boldsymbol{I}_d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau'} \bar{\boldsymbol{x}}_{\tau}(i)\bar{\boldsymbol{x}}_{\tau}(i)')}{d}\right)^d \\
&= \left(\frac{\operatorname{tr}(\lambda \boldsymbol{I}_d) + \sum_{\tau=1}^{t}\sum_{i\in S_\tau'} \operatorname{tr}(\bar{\boldsymbol{x}}_{\tau}(i)\bar{\boldsymbol{x}}_{\tau}(i)')}{d}\right)^d \\
&= \left(\frac{\lambda d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau'} \|\bar{\boldsymbol{x}}_{\tau}(i)\|_2^2}{d}\right)^d \\
&= \left(\frac{\lambda d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau} \left(\|\bar{\boldsymbol{x}}_{\tau,1}(i)\|_2^2 + \|\bar{\boldsymbol{x}}_{\tau,2}(i)\|_2^2\right)}{d}\right)^d \\
&= \left(\frac{\lambda d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau} \left(\|\tilde{\boldsymbol{x}}_{\tau,1}(i)\|_2^2 + \|\tilde{\boldsymbol{x}}_{\tau,2}(i)\|_2^2\right)}{d}\right)^d \\
&= \left(\frac{\lambda d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau} \|\boldsymbol{x}_{\tau}(i)\|_2^2}{d}\right)^d \\
&\le \left(\frac{\lambda d + \sum_{\tau=1}^{t}\sum_{i\in S_\tau} 1}{d}\right)^d \\
&\le \left(\frac{\lambda d + tk}{d}\right)^d\ ,
\end{aligned}$$

where we have used the AM-GM Inequality for the first inequality and the fact that $\boldsymbol{x}_\tau(i)' = \begin{bmatrix}\tilde{\boldsymbol{x}}_{\tau,1}(i)' & \tilde{\boldsymbol{x}}_{\tau,2}(i)'\end{bmatrix}$ to arrive at the last equality. Finally, using the fact that $\boldsymbol{V}_{t-1} \preceq \boldsymbol{V}_t$, that the noise is $\frac{R}{\sqrt{2}}$-subgaussian and Theorem 2 from [27], Theorem 4.2 from [24] becomes:

$$\begin{aligned}
\|\hat{\boldsymbol{\theta}}_t - \boldsymbol{\theta}_\star\|_{\boldsymbol{V}_{t-1}} &\le \|\hat{\boldsymbol{\theta}}_t - \boldsymbol{\theta}_\star\|_{\boldsymbol{V}_t} \\
&\le \frac{R}{\sqrt{2}}\sqrt{2\log\left(\frac{\det(\boldsymbol{V}_t)^{1/2}}{\delta \det(\lambda \boldsymbol{I}_d)^{1/2}}\right)} + \lambda^{1/2} S \\
&\le \frac{R}{\sqrt{2}}\sqrt{d\log\left(\frac{1 + tk/\lambda}{\delta}\right)} + \lambda^{1/2} S\ ,
\end{aligned}$$

with probability at least $1 - \delta$, which is the same as that in [24], with the exception of the definition of $\boldsymbol{V}_t$.

Conveniently, Lemma 4.1 from [24] is written in terms of $\boldsymbol{V}_{t-1}$ and α_t, thus the proof follows exactly besides the choice of α_t, rewritten below for convenience:

Lemma 1. If $\alpha_t = \frac{R}{\sqrt{2}}\sqrt{d\log\left(\frac{1+tk/\lambda}{\delta}\right)} + \lambda^{1/2} S$, then we have

$$0 \le \hat{r}_t(i) - r_t^\star(i) \le 2\alpha_t \|\boldsymbol{x}_t(i)\|_{\boldsymbol{V}_{t-1}^{-1}}$$

holds simultaneously for all $t \ge 0$ and $i \in [k]$ with probability at least $1 - \delta$.

We provide the correction of the proof of Lemma 4.2 from [24] in Appendix. This proof can be used by changing

3. Equivariance is without loss of generality. If the two variances were different, the expressions for $\boldsymbol{V}_t$ and $\boldsymbol{b}_t$ would be different. The data with the less variance would be prioritised via larger weight: $\boldsymbol{V}_t = \lambda \boldsymbol{I} + \sum_{\tau=1}^{t}\sum_{f=1}^{2}\sum_{i\in S_\tau} (\frac{\sigma_1\sigma_2}{\sigma_f})^2 \bar{\boldsymbol{x}}_{\tau,f}(i)\bar{\boldsymbol{x}}_{\tau,f}(i)'$ and $\boldsymbol{b}_t = \sum_{\tau=1}^{t}\sum_{f=1}^{2}\sum_{i\in S_\tau} (\frac{\sigma_1\sigma_2}{\sigma_f})^2 \bar{\boldsymbol{x}}_{\tau,f}(i)\bar{r}_{\tau,f}(i)$. The hyperparameter λ would need different adjustments since $\lambda = \sigma_1^2\sigma_2^2/\gamma^2$. The value of γ^2 stays the same, serving as the variance for the prior of $\boldsymbol{\theta}$.

4. For independent r.v.'s X R_1-subgaussian and Y R_2-subgaussian, $X + Y$ must be $\sqrt{R_1^2 + R_2^2}$-subgaussian.

the definition of the matrix $\boldsymbol{X}_T$ into:

$$\boldsymbol{X}'_T = \begin{bmatrix} \bar{\boldsymbol{x}}'_{T,1}(s_{(1,T)}) \\ \vdots \\ \bar{\boldsymbol{x}}'_{T,1}(s_{(|S_T|,T)}) \\ \bar{\boldsymbol{x}}'_{T,2}(s_{(1,T)}) \\ \vdots \\ \bar{\boldsymbol{x}}'_{T,2}(s_{(|S_T|,T)}) \end{bmatrix} .$$

Then we rewrite:

$$\begin{aligned} \det(\boldsymbol{V}_T) &= \det\left(\boldsymbol{V} + \sum_{t=1}^{T}\sum_{f=1}^{2}\sum_{i\in s_t} \bar{\boldsymbol{x}}_{t,f}(i)\bar{\boldsymbol{x}}_{t,f}(i)'\right) \\ &= \det\left(\boldsymbol{V} + \sum_{t=1}^{T-1}\sum_{f=1}^{2}\sum_{i\in s_t} \bar{\boldsymbol{x}}_{t,f}(i)\bar{\boldsymbol{x}}_{t,f}(i)' + \right. \\ &\qquad \left. \sum_{f=1}^{2}\sum_{i\in s_T} \bar{\boldsymbol{x}}_{T,f}(i)\bar{\boldsymbol{x}}_{T,f}(i)'\right) \\ &= \det\left(\boldsymbol{V}_{T-1} + \boldsymbol{X}_T\boldsymbol{X}'_T\right) \ , \end{aligned}$$

and the rest of the proof follows very similarly, with slight difference in the dimension of $\boldsymbol{X}_T$, changing from $|s_T|$ into $2|s_T|$. Finally, the third last equality requires us to find the trace of the matrix of interest, which is $\mathrm{tr}(\boldsymbol{X}'_T\boldsymbol{V}_{T-1}^{-1}\boldsymbol{X}_T) = \sum_{f=1}^{2}\sum_{i\in s_T}\boldsymbol{x}_{T,f}(i)\boldsymbol{V}_{T-1}^{-1}\boldsymbol{x}_{T,f}(i)$, which in turn gives us our new determinant inequality:

$$\det(\boldsymbol{V}_T) \geq \det(\mathbf{V}_{T-1})\left(1 + \sum_{f=1}^{2}\sum_{i\in s_T}\|\boldsymbol{x}_{T,f}(i)\|^2_{\boldsymbol{V}_{T-1}^{-1}}\right) .$$

Therefore, we have our modification of Lemma 4.2 of [24] (Lemma 1 in Appendix) as follows:

Lemma 2. Let $\boldsymbol{V} \in \mathbb{R}^{d\times d}$ be a positive definite matrix, $s_t \subseteq \{1,\cdots,k\}$ where $|s_t| \leq \ell$ for $t = 1,2,\ldots,$ and $\boldsymbol{V}_T = \boldsymbol{V} + \sum_{t=1}^{T}\sum_{f=1}^{2}\sum_{i\in s_t}\bar{\boldsymbol{x}}_{t,f}(i)\bar{\boldsymbol{x}}_{t,f}(i)'$. Then, if $\forall t, i\ \lambda \geq \ell$ and $||\boldsymbol{x}_t(i)||_2 \leq 1$ for concatenated context $\boldsymbol{x}_t(i) = \bar{\boldsymbol{x}}_{t,1}(i) + \bar{\boldsymbol{x}}_{t,2}(i)$ where $\bar{\boldsymbol{x}}_{t,1}(i)' = \begin{bmatrix}\tilde{\boldsymbol{x}}_{t,1}(i)' & \mathbf{0}_{1\times d_2}\end{bmatrix}$ and $\bar{\boldsymbol{x}}_{t,2}(i)' = \begin{bmatrix}\mathbf{0}_{1\times d_1} & \tilde{\boldsymbol{x}}_{t,2}(i)'\end{bmatrix}$, we have

$$\begin{aligned} \sum_{t=1}^{T}\sum_{i\in s_t}\|\boldsymbol{x}_t(i)\|^2_{\boldsymbol{V}_{t-1}^{-1}} &= \sum_{f=1}^{2}\sum_{t=1}^{T}\sum_{i\in s_t}\|\boldsymbol{x}_{t,f}(i)\|^2_{\boldsymbol{V}_{t-1}^{-1}} \\ &\leq 2\log\det\boldsymbol{V}_T - 2\log\det\boldsymbol{V} \\ &\leq 2d\log((\mathrm{tr}(\boldsymbol{V}) + T\ell)/d) - \\ &\qquad 2\log\det\boldsymbol{V} \ . \end{aligned}$$

Since there is no modification to the objective function, and since all the theorems and lemma required to arrive at the final regret bound are the same, the regret bound for the modified C^{2}UCB stays the same, as stated in [24]:

$$\sum_{t=1}^{T} Reg_t^{\alpha} \leq C\frac{R}{\sqrt{2}}\sqrt{8Td\log\left(1+\frac{Tk}{d\lambda}\right)}\cdot \left(\sqrt{d\log\left(\frac{1+Tk/\lambda}{\delta}\right)} + \sqrt{\lambda}S\right) .$$

Notice that in cases where there are n_f examples per arm in each round instead of only two, the regret will generalise into:

$$\sum_{t=1}^{T} Reg_t^{\alpha} \leq C\frac{R}{\sqrt{n_f}}\sqrt{8Td\log\left(1+\frac{Tk}{d\lambda}\right)}\cdot \left(\sqrt{d\log\left(\frac{1+Tk/\lambda}{\delta}\right)} + \sqrt{\lambda}S\right) ,$$

which has a factor of $\frac{1}{\sqrt{n_f}}$, $n_f \in \mathbb{N}$ compared to the original C^{2}UCB where $n_f = 1$.

6 Experimental Methodology

We evaluate our MAB framework across a range of widely used analytical and HTAP industrial benchmarks, comparing it to a state-of-the-art physical design tool shipped with a commercial database product referred to as the Physical Design Tool (PDTool). This is a mature product, proven to outperform other physical design tools available on the market [21], [28]. As a representative of the most recent studies that successfully use Monte Carlo tree search (MCTS) to tune indices [29], [30], [31], we test our framework against a database optimiser that can tune indices using MCTS, called UDO [29]. UDO is originally designed to work with analytical queries only, which we extend to work with HTAP workloads (we refer to this baseline as MCTS hereafter).

Benchmarks. For HTAP performance testing we use CH-BenCHmark [32], [33], TPC-H benchmark (with uniform distribution) [34] and TPC-H Skew benchmark [35] with Zipfian factor 4. CH-BenCHmark provides a complex mixed workload, combining TPC-C [36] and TPC-H benchmarks. The CH-BenCHmark schema comprises an unmodified TPC-C schema and three tables (Supplier, Region, Nation) from TPC-H. Its workload is composed of TPC-C transactional workload and modified 22 TPC-H [34] queries adapted to the CH-BenCHmark schema.

While CH-BenCHmark provides a uniform dataset, we are unaware of any HTAP benchmarks with skewed data generation. While there are some OLTP benchmarks with skewed datasets [33], they do not provide the required level of OLAP complexity for the index selection problem to be interesting. Due to the limitations of existing benchmarks, we decided to extend the TPC-H skew benchmark to include INSERT, DELETE and UPDATE statements to mimic a skewed HTAP benchmark [37]. The TPC-H skew data generation tool already provides the functionality to generate data for inserts and deletes. In our extension, we additionally perform updates on existing records using the same generated data. To highlight the impact of skewness on the overall performance, we also report comparable HTAP results on the original TPC-H database.

For analytical experiments, we use five publicly available benchmarks: TPC-DS [38], a complex benchmark resulting in a large number of candidate configurations; SSB [39] with easily achievable index benefits; Join Order Benchmark (JOB) with IMDb dataset (a real-world dataset) [18] (henceforth referred to as IMDb), a challenging workload for index recommendations with index overuse leading to performance regressions; and finally, TPC-H and TPC-H skew benchmarks.

Unless stated otherwise, all experiments use scale factor (SF) 10, resulting in approximately 10GB of data per workload, except in the case of the IMDb dataset, which has a fixed size of 6GB.[5] We consider two broad types of workloads, allowing us to compare different aspects of the recommendation process:

1) *Static*: The workload sequence is known in advance, and repeating over time (modelling workloads used for reporting purposes). In the absence of dynamic environment complexities, this simpler setting allows us to single-out the effectiveness (the ability to find a good configuration) and the efficiency (the search overhead) of the MAB search strategy.
2) *Dynamic*: The region of interest shifts over time from one group of queries to another (modelling data exploration). For HTAP experiments, interest shifts through workloads with different transaction and analytical compositions, ranging from fully analytical workloads to transaction heavy workloads. Dynamic workloads are used to evaluate adoption speed, cost of exploration and memory efficiency in dynamic environments.

Across experiments, each group of templates is invoked over rounds, producing different instances. For static experiments, we invoke the PDTool at the start of the second round giving the first round workload as the representative workload. For dynamic workloads, we invoke the PDTool soon after the workload shift since this workload will become representative of future rounds. This setting is somewhat unrealistic and favourable for PDTool, since in real-life the PDTool will seldom truly have knowledge of the representative workload (i.e., what is yet to arrive in the future), advantaging the PDTool in our experiments. However, it presents a viable comparison against the workload-oblivious MAB. Bandits do not use any workload information ahead of time, but instead observe a workload sequence and react accordingly.

Physical design tuning parameters. Both PDTool and MAB are given a memory budget approximately equal to the size of the data (1x; 10GB for SF 10 datasets and 6GB for IMDb dataset) for the creation of secondary indices. We have experimented with different memory budgets ranging from 0.25x to 2x (since benefits of additional memory seem to diminish beyond a 2x limit) under TPC-H and TPC-H skew benchmarks, and observed the same patterns throughout that range.[6] We have naturally picked the middle of the active region (1x) as our default memory budget. All these workloads come with original primary and foreign keys that influence the choice of indices. We grant the aforementioned memory budget on top of this.

In search of the best possible design, we do not constrain the running time of PDTool.All proposed indices are materialised and workload invoked over the same commercial DBMS in both cases (MAB and PDTool).

5. CH-BenCHmark does not scale with the SF parameter like most of the other benchmarks we use. It uses a number of warehouses (similar to the TPC-C benchmark) as a scaling parameter. For our experiments, we use 137 warehouses which generates approximately a 10GB dataset.

6. Both tools converge to the same execution cost by the final round, when enough memory was given to fit the entire useful configuration.

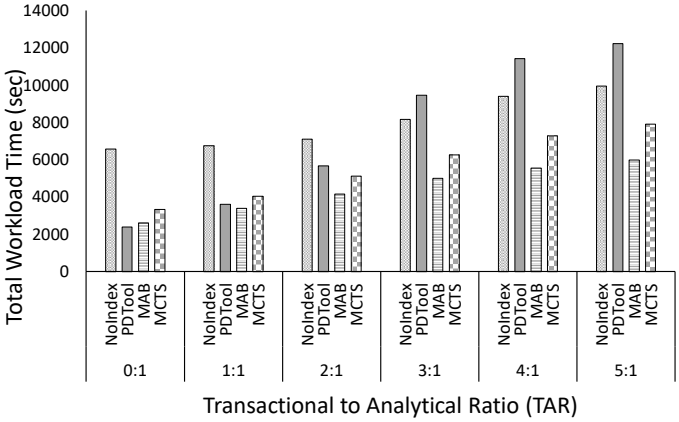

Fig. 3. MAB vs. PDTool vs. MCTS total workload time under CH-BenCHmark for *static* workloads with a range of different TARs

Hardware. All experiments are performed on a server equipped with 2x 24 Core Xeon Platinum 8260 at 2.4GHz, 1.1TB RAM, and 50TB disk (10K RPM) running Windows Server 2016. We report cold runs, clearing database buffer caches prior to every query execution.

7 Experimental Results

This section reports empirical comparisons of MAB against PDTool and MCTS on HTAP workloads and summarises results under analytical workloads, reported earlier [40]. We report the total workload time broken down by recommendation, index creation, and workload execution times. Total workload time captures all the costs incurred from the start of the experiment to the end. Index deletion cost is negligible compared to creation and execution costs and does not have an observable growth with index size.[7] Therefore, we ignore the index deletion cost. For HTAP experiments, execution time is further divided into analytical and transactional components. In addition, we present original statement times without any secondary indices (denoted as NoIndex). We present summary graphs with total end-to-end workload time and convergence graphs with total workload and execution times per round. Finally, we present results against a well-tuned reinforcement learning agent.

7.1 MAB vs PDTool Under HTAP Workloads

7.1.1 Static HTAP Workloads

To illustrate the impact of transactional statements, we use a series of CH-BenCHmark static experiments varying the transactional to analytical ratio. In each of these experiments, we keep the analytical component constant with 22 adapted TPC-H queries (*a set of analytical queries*) while changing the size of the transactional component.

The transactional component of the workload is composed of 5 transactions (new-order, payment, order-status, delivery and stock-level) with a pre-specified transaction mixture (44%, 44%, 4%, 4% and 4%, respectively). The smallest transactional workload adhering to the specified TPC-C transaction mixture comprises 11 new order transactions, 11

7. When tested with indices of different sizes and complexities, we observed sub-millisecond deletion costs in all cases.

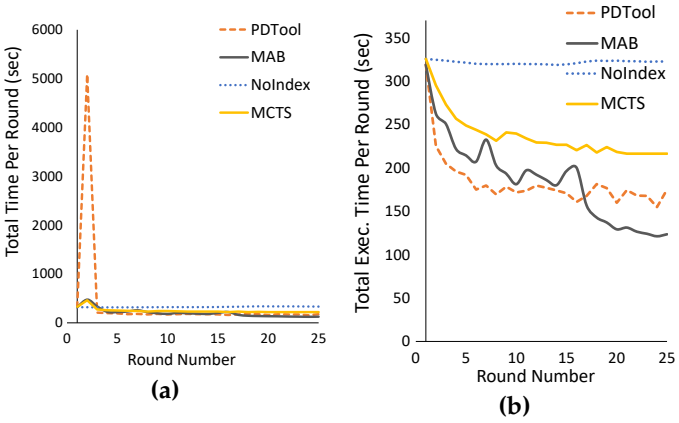

Fig. 4. MAB vs. PDTool vs MCTS convergence under CH-BenCHmark for *static* workloads with 3:1 TAR: (a) End-to-end workload time, (b) Total execution time.

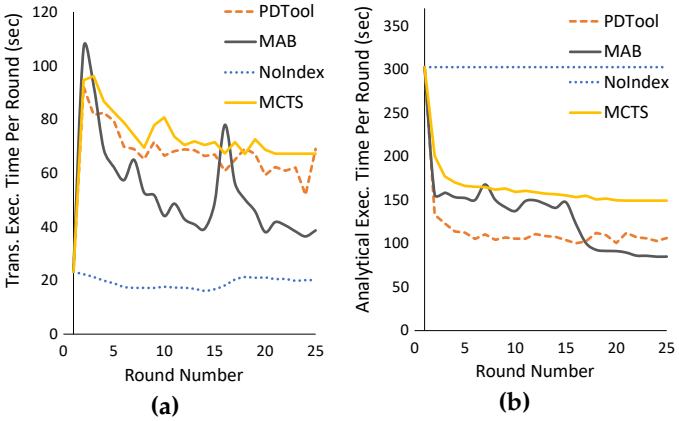

Fig. 5. MAB vs. PDTool vs MCTS convergence under CH-BenCHmark for *static* workloads with 3:1 TAR: (a) Transactional Execution cost, (b) Analytical Execution cost.

payment transactions, an order-status transaction, a delivery transaction and a stock-level transaction (approximately 650 statements). This smallest transactional workload is henceforth referred to as *a set of transactional statements*. We define the transactional to analytical ratio (henceforth referred to as *TAR*) as the ratio between transactional and analytical statement sets. As an example, 5:1 TAR is composed of one analytical set (22 TPC-H queries) and 5 transactional sets (i.e., 55 new order transactions, 55 payment transactions, 5 order-status transactions, 5 delivery transactions and 5 stock-level transactions, resulting in approximately 3300 transactional statements per round).

As evident from Figure 3, in transaction-heavy workloads, MAB performs much better than PDTool providing up to 51% speed-up (5:1) in total workload time, whereas PDTool performs better in the fully analytical workload (0:1) providing up to 8% speed-up. Static workloads over uniform datasets are the best case for offline physical design tools, as a pre-determined workload sequence may perfectly represent future statements. We will look into analytical workloads in more detail in Section 7.3. At 1:1 TAR, the first ratio that introduces the transactional component, MAB starts to take the lead providing a 4% performance gain in total workload time. MAB reaches the same last round execution time as PDTool; however, due to better execution times in early rounds, PDTool provides 7.7% total execution time speed up over MAB. From 2:1 TAR onwards, MAB dominates the PDTool providing 26%, 47%, 51% and 51% total workload time speed-up over PDTool, under 2:1, 3:1, 4:1, 5:1 TARs, respectively. PDTool struggles to perform better than NoIndex from 3:1 TAR onwards due to the heavy recommendation costs incurred by PDTool, yet PDTool is still superior to NoIndex in execution cost.

The MCTS-based approach performs much better than NoIndex but records lower performance in total workload cost compared to the PDTool and MAB. MAB managed to outperform MCTS by 21.7%, 16.3%, 18.7%, 20%, 23.7% and 24.1% under 0:1, 1:1, 2:1, 3:1, 4:1 and 5:1 TARs, respectively. Furthermore, MCTS requires longer training outside the total workload time (e.g., MCTS used 1h on average for training across the experiments). On the downside, MCTS action space grows like $O(2^k)$ where k is the number of arms, which limits its candidate indices to unique column subsets and not the permutation of those columns.

To further understand the results, we dive into the 3:1 TAR experiment, which provides a good balance of transactional and analytical statements to demonstrate the importance of both analytical gain and transactional overhead. As shown in the convergence graphs in Figure 4(b), MAB converges to a better configuration providing 29.4% and 42.8% faster execution time by the last round compared to PDTool and MCTS, respectively. Figure 4 also explains PDTool's higher total workload time compared to NoIndex. While PDTool consumes a high recommendation time in the first round, it results in a better per round total workload and execution times than NoIndex.

Balancing the configuration fitness between transactional and analytical workloads is the prime concern of index tuning in HTAP environments. How tools achieve this balance can be better understood by breaking the execution cost into analytical and transactional components. As Figure 5 demonstrates, MAB configuration provides better execution time for both analytical and transactional workload components compared to both PDTool and MCTS. MAB provides 19.9% and 42.5% better execution times by the 25th round for analytical and transactional workloads, respectively, compared to the second best option (MCTS in the transactional and PDTool in the analytical cost). In initial rounds, MAB performance is inferior to PDTool in transactional execution time, but it quickly learns the negative impact of indices on the transactional workload. By the 4th round, MAB surpasses PDTool in transactional execution time by dropping indices with negative rewards. While removing the unnecessary indices, MAB makes sure not to impact the analytical execution times by keeping the high reward indices intact. MAB performs several configuration changes in rounds 15–17, which results in a sudden oscillation in transactional execution time, but these configuration changes allow the bandit to find a superior configuration in both analytical and transactional execution costs. There is some variability in transactional execution costs even with the same number of transactions in each round, as the number of statements per round can be different (e.g., different new orders can have a different number of items in a transaction, leading to a different number of statements).

MCTS performance under the static experiments are less satisfactory due to its limitations in the action space and longer training times. These issues will be compounded

TABLE 1
HTAP: Total workload time breakdown for HTAP workloads (in min): the best choice is in bold text.

Workload	Recommendation		Index Creation		Execution		Analytical		Transactional		Total	
	MAB	PDT	MAB	PDT	MAB	PDT	MAB	PDT	MAB	PDT	MAB	PDT
CH	**0.1**	79.15	6.67	**1.86**	**76.6**	76.72	55.02	**48.74**	**21.58**	27.98	**83.37**	157.73
TPC-H	**0.14**	14.09	9.41	**6.43**	87.41	**81.82**	57.42	**53.4**	29.99	**28.42**	**96.96**	102.35
TPC-H Sk.	**0.15**	13.92	14.79	**12.74**	**77.29**	102.29	**37.19**	65.45	40.11	**36.84**	**92.22**	128.94

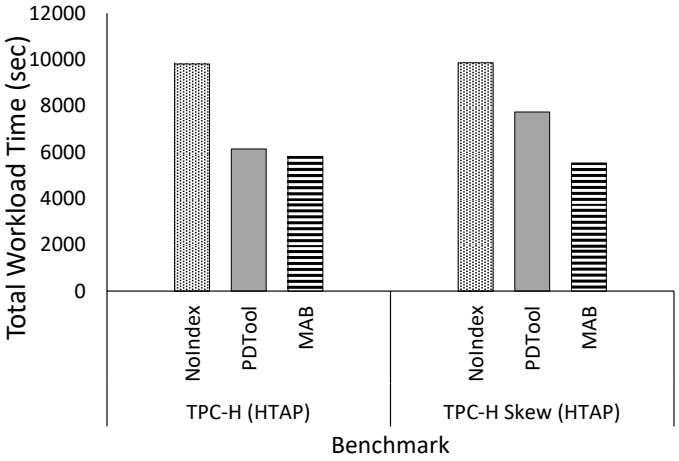

Fig. 6. MAB vs. PDTool vs. NoIndex total end-to-end workload time for static TCP-H and TPC-H skew HTAP workloads.

with dynamic workloads which would require multiple training sessions. Therefore, in the remaining experiments, we compare MAB against PDTool (the strongest competitor).

Further analysis of additional RL approaches under analytical workloads can be found in [40]. Those experiments demonstrate that deep RL's randomised exploration of the vast state-action space and challenging hyperparameter tuning contributes to the solution volatility, whereas MAB typically provides better convergence and simpler implementation.

7.1.2 The Impact of Data Skew in HTAP Workloads

We now experiment with TPC-H and TPC-H Skew HTAP workloads to demonstrate the impact of the addition of transactional statements to well-known OLAP benchmarks.

We experiment with a similar number of transactional statements as in the 3:1 TAR CH-BenCHmark experiment. The OLTP part of the workload is composed of 6 templates (two insert templates, two delete templates and two update templates). We use the original data generation tools to generate the INSERT, DELETE and UPDATE statements for ORDER and LINEITEM tables. The transactional workload used here is less complex than CH-BenCHmark but sufficient to demonstrate the impact of HTAP workloads on the overall performance.

As shown in Figure 6, MAB performs better in both TPC-H and TPC-H skew HTAP workloads. Interestingly, MAB achieves 5.2% better total workload time under the TPC-H benchmark, which is usually favourable to PDTool. MAB converges to a similar performant configuration, comparing the final round execution cost. However, with MAB's longer execution times in the first few rounds, PDTool achieves a 6% better total execution time. Due to the higher recommendation time of PDTool, it has a higher total workload

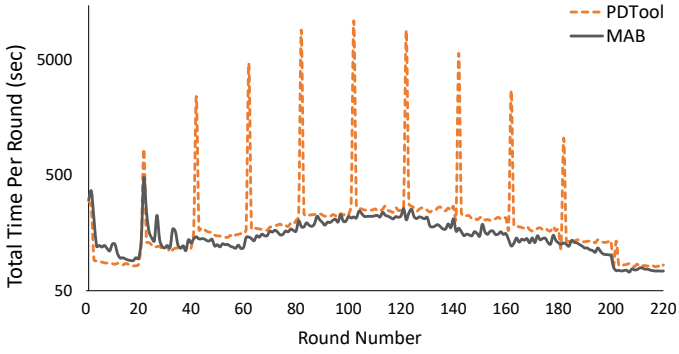

Fig. 7. MAB vs. PDTool total-workload time convergence under CH-BenCHmark for *dynamic* workloads with different transaction levels **(log y-axis)**

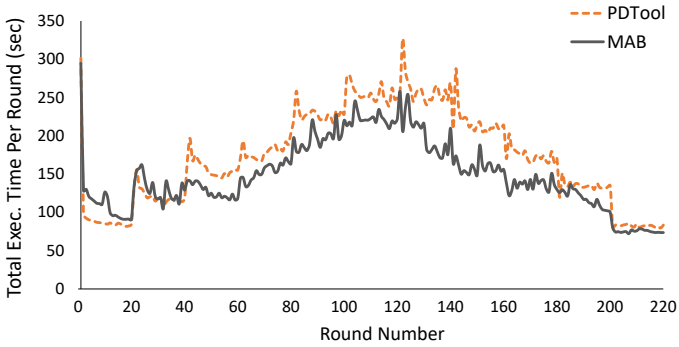

Fig. 8. MAB vs. PDTool *total execution time* convergence under CH-BenCHmark for dynamic workloads with different transaction levels

time. In TPC-H Skew, MAB dominates the PDTool across all dimensions, having 28.4% better total workload time and 24% better execution time. All HTAP results are summarised in Table 1.

7.1.3 Dynamic HTAP Workloads

This experiment gradually increases and decreases the transactional workload component over the rounds. We start with 0:1 TAR, which is purely analytical, and then we add transactional workload sets one by one till we reach 5:1 TAR. Afterwards, we gradually reduce the transactional workload sets one by one to reach 0:1 TAR again. We run 20 rounds in each TAR.

After each workload change, PDTool is invoked with the new workload from the previous round, which is a good representation of the next 19 rounds. It is essential to provide the workload from at least one complete round because PDTool considers the transactional to analytical ratio when making recommendations. However, as observable from Figure 7, each invocation of PDTool takes a substantial amount of time for larger workloads in higher transaction levels.

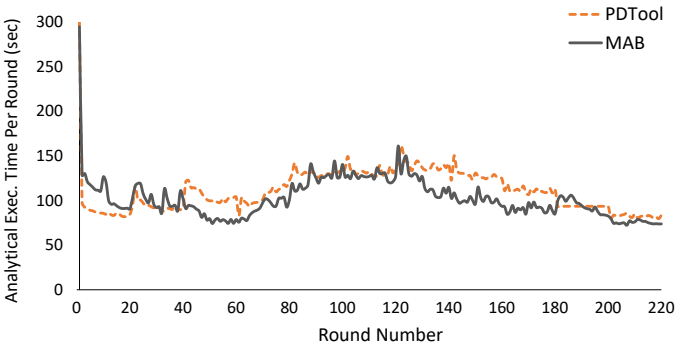

Fig. 9. MAB vs. PDTool *analytical execution time* convergence under CH-BenCHmark for dynamic workloads with different transaction levels

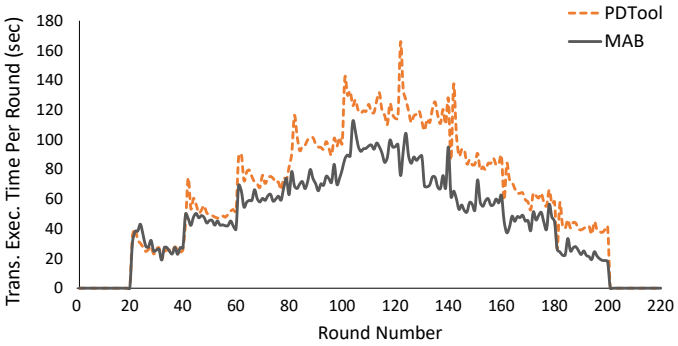

Fig. 10. MAB vs. PDTool *transactional execution* time convergence under CH-BenCHmark for dynamic workloads with different transaction levels

MAB performs most of the configuration changes at the start of the experiment and then after the first workload change (0:1 → 1:1). Understandably, MAB needs to explore extensively at the start of the experiment. However, a similar level of exploration for the first workload change might not be as intuitive, given that we do not see such exploration from MAB for the rest of the experiment. In round 21, MAB is exposed to transactional statements for the first time in this experiment. As a result, MAB performs more exploration and learns the negative impact of indices in these rounds and thus performs much better after round 40. Ultimately, MAB provides a 57.8% speed up in the total workload time compared to PDTool. A significant portion of this speed-up is attributed to MAB's lower recommendation cost compared to PDTool.

To compare the different configurations proposed by the tools, we plot execution time over the rounds in Figure 8. After the first two transaction levels (i.e., after round 40), MAB always manages to lock into a superior configuration providing faster execution time. As a result, MAB provides an 11% speed-up in total execution cost.

In the entire experiment, we go through the same TAR two times (except for 5:1 TAR), which results in similar workloads. However, from Figure 8, it is noticeable that PDTool reaches higher execution costs in the descending part of the experiment (after round 120) compared to the ascending part of the experiment (rounds 1 to 120). Configurations proposed by the PDTool depend on the existing secondary indices present in the system and the underlying data. Continuous additions and deletions change the underlying data, partially invalidating the statistics used by the optimiser. Furthermore, at each PDTool invocation, the system has a different starting set of secondary indices, impacting PDTool's recommendations. Therefore the recommendations proposed for similar workloads in ascending and descending parts of the graph are different. For example, rounds 80–100 and 120–140 run a 4:1 TAR workload, whereas PDTool proposes two very different configurations for these two sections. While it proposes only 18 indices at round 81, 27 are proposed in round 121, with only 11 indices being shared across 2 configurations. On the other hand, MAB converges quickly in the later experiment rounds, taking advantage of the already obtained knowledge.

To further analyse MAB's gain in the dynamic experiment, we need to break down the execution time into analytical and transactional components. As one can observe from Figures 9 and 10, MAB obtains the gain mainly from the transactional workload. MAB provides 4.5% better analytical execution cost and 22.6% better transactional execution cost compared to the PDTool. As observable from Figure 9, MAB is obtaining a noticeable analytical gain in 2:1 and 3:1 TARs. In the analytical heavy workloads (0:1, 1:1 TARs), PDTool records a better or similar analytical execution time. MAB opts for the transactional friendly configuration for transactional heavy workloads (4:1, 5:1 TARs), reducing thereby the analytical execution time gain. On the transactional end, MAB leads in almost all workloads. As expected, transactional execution cost gain increases for the transactional heavy workloads.

Impact of data skewness on dynamic HTAP workloads: This section explores the compound effect of data skewness and dynamic workloads. We experiment with a dynamic HTAP TPC-H skew workload to demonstrate this effect. We run a shifting workload with different TARs similar to the dynamic ch-BenCHmark, where each shift runs for 15 rounds. There are three shifts (0.5×, 1×, 2× TARs) with 45 rounds in total. Here 1× represents the TAR used in the static TPC-H skew HTAP experiment. However, in this experiment, we shift the analytical workload as well. All analytical templates are divided into three almost equal size sets and used one set per shift. In this experiment, MAB provided an 82.51% gain in total workload time and a 22.77% gain in total execution time.

7.1.4 Space Savings Under HTAP Workloads

In the case of index tuning of HTAP workloads, more indices can result in higher running time for transactional components of the workload. Consequently, a minimal index set can be optimal for a transaction-heavy workload. While such a configuration might be suboptimal for the analytical component of the workload, a minimal configuration can result in a better total workload execution time due to the significant savings obtained from avoiding index maintenance activities stemming from transactional statements. Our experiments observed that PDTool usually exploits the entire given memory budget, resulting in a higher transactional execution cost.[8]

8. We have observed that PDTool sometimes goes over the given budget due to errors in index size estimations. On the other hand, MAB initially estimates the index size based on the statistics and corrects the estimate after indices are materialised for the first time and therefore does not suffer from the same issue.

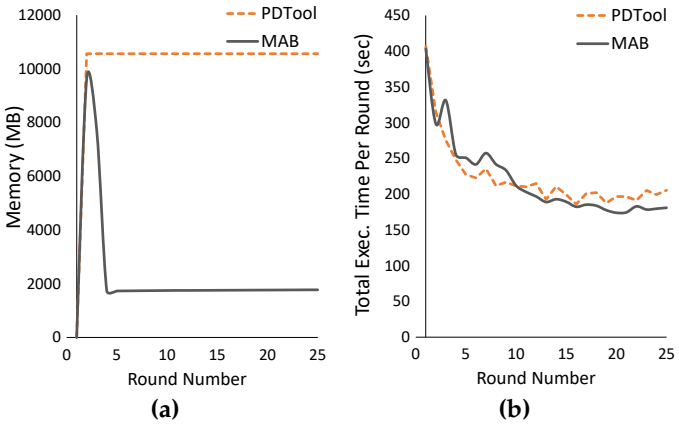

Fig. 11. MAB vs. PDTool convergence under CH-BenCHmark for *static* workloads with 5:1 TAR: (a) Memory use, (b) Total execution time.

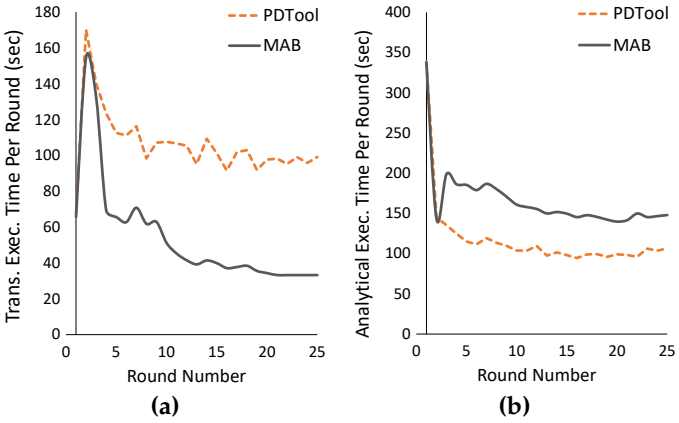

Fig. 12. MAB vs. PDTool convergence under CH-BenCHmark for *static* workloads with 5:1 TAR: (a) Transactional Execution cost, (b) Analytical Execution cost.

On the flip side, MAB learns the negative impact of indices on transactional statements and dynamically adjusts the configuration. However, this behaviour was not visible with fully analytical workloads. The index size context feature typically carries a negative weight due to negative rewards from index creation operations and forces the bandit to choose the smaller arms that provide the best gains in execution cost.

At 5:1 TAR, MAB provides a configuration that yields an 83% memory saving while achieving an 8.8% execution time gain by the last round (see Figure 11). This execution cost gain is smaller than the gain we observed under transaction level 3, as the usefulness of indices reduces when the workload becomes transactions heavy.

While it might be counter-intuitive for an index tuning tool to use less memory to provide better performing configurations, it can be easily understood by observing the analytical and transactional execution times of both tools by the last round (see Figure 12). Comparing the last round configurations of both tools, PDTool creates an analytical friendly configuration that provides a 27% speed-up in analytical execution time (around 40 second gain per round). On the other hand, MAB locks into a smaller configuration that is more suitable for transactional statements providing a 60% speed-up in transactional execution cost (around 60 second gain per round). Ultimately MAB provides an 8.8% speed-up in total execution time while offering a significant memory saving.

7.1.5 Impact of restricted recommendation times under HTAP workloads

In all experiments, we run with unrestricted running times for PDTool in search of the best recommendation quality. Some tuning tools, like PDTool, can restrict the tuning session to a time provided by the user [41]. While this can negatively impact the recommendation quality (and in turn execution time), this restriction can reduce the total workload time.

To shed some light on this issue, we experiment with CH-BanCHmark 3:1 TAR workload providing different tuning times to the PDTool (See Table 2). We notice an apparent increase in execution time when we reduce the recommendation time, impacting the total workload time in the long run. However, in our 25-round experiments, the reduction in recommendation time leads to better total workload times for the PDTool. Nevertheless, MAB's total workload time remains superior.

As observable in Table 2, when given a tuning time less than the minimum, PDTool will end without providing any recommendations. Unfortunately, identifying the minimum time requirement for the tuning session is next to impossible (other than through the trial and error). For example, the 3:1 TAR experiment gives the first recommendation at 30 minutes, whereas under the 5:1 TAR workload, PDTool took 60 minutes.

7.1.6 Impact on Recommendation Quality of Including Index Maintenance Time in Rewards

This section tests the impact of including the index maintenance time in the reward. We experiment against the OLAP version of the bandit, which only considers the index creation time and query execution time. While the OLAP version focuses on improving the data scan gains entirely, the HTAP version tries to balance the gain from the data scans and the negative impact of the index maintenance. We run a CH-BenCHmark 5:1 TAR workload for 25 rounds. In this experiment, we notice that the OLAP version provides a 5.33-minute gain in analytical execution time, while causing a 13.02-minute loss in transactional execution time. Overall, the HTAP version provides a 7.68-minute gain in total execution time. This experiment shows that HTAP reaches a better balance by considering both data scan gains and index maintenance overheads.

TABLE 2
HTAP: Total workload time breakdown for CH-BenCHmark 3:1 TAR workloads (in min)

Component	Recommend	Creation	Execution	Total
MAB	0.1	6.67	76.6	83.37
PDT (15 mins)	No Recommendations			
PDT (30 mins)	30.64	2.34	84.74	117.72
PDT (60 mins)	60.67	2.11	81.17	143.95
PDT (original)	79.15	1.86	76.72	157.73

TABLE 3
Total time breakdown for analytical workloads (in min): the best choice is in bold text.

Workload		Recommendation		Creation		Execution		Total	
		PDTool (#)	MAB	PDTool	MAB	PDTool	MAB	PDTool	MAB
Static	SSB	0.34 (0.34)	**0.02**	**0.95**	1.86	**12.9**	13.15	**14.19**	15.03
	TPC-H	0.6 (0.6)	**0.08**	**2.45**	5.66	**46.35**	55.64	**49.4**	61.38
	TPC-H Sk.	0.58 (0.58)	**0.11**	**8.37**	19.82	54.17	**32.06**	63.12	**51.99**
	TPC-DS	44.86 (44.86)	**1.53**	**1.45**	5.94	302.63	**242.15**	348.94	**249.62**
	IMDB	0.34 (0.34)	**0.31**	**1.1**	1.3	11.01	**9.42**	12.41	**11.03**
Dynamic	SSB	1.28 (0.32)	**0.05**	**1.5**	2.21	**5.42**	5.69	8.2	**7.95**
	TPC-H	1.55 (0.32)	**0.12**	**9.36**	9.74	26.35	**25.14**	37.25	**35**
	TPC-H Sk.	1.65 (0.41)	**0.16**	**14.98**	20.96	85.49	**21.44**	102.11	**42.56**
	TPC-DS	11.13 (2.78)	**1.66**	**6.08**	16.48	187.08	**155.65**	204.29	**173.79**
	IMDB	3.09 (0.77)	**0.29**	**1.59**	2.24	11.21	**7.93**	15.89	**10.46**
Random	SSB	2.83 (0.57)	**0.02**	**1.77**	2.37	26.59	**16.83**	30.85	**19.22**
	TPC-H	7.55 (1.51)	**0.08**	14.68	**7.06**	84.14	**80.43**	106.37	**87.57**
	TPC-H Sk.	3.3 (0.66)	**0.08**	**31.74**	34.68	48.71	**39.44**	83.75	**74.2**
	TPC-DS	310.22 (62.04)	**1.4**	**8.23**	19.81	323.57	**227.02**	642.01	**248.24**
	IMDB	14.74 (2.94)	**0.28**	2.72	**1.14**	48.55	**14.47**	66.01	**15.89**

The average time of a single PDTool invocation

TABLE 4
Total end-to-end workload time for static analytical workloads under different database sizes (in min)

Workload	SF	PDTool	MAB
TPC-H	1	**2.02**	2.03
	10	**49.4**	61.38
	100	891.01	**793.40**
TPC-H Skew	1	4.17	**3.83**
	10	63.12	**51.99**
	100	2640.64	**1219.33**

TABLE 5
Total time breakdown for analytical TPC-H Skew workloads under different round sizes (in min)

Round size	Rec.	Creation	Execution	Total
Single Query	1.11	27.77	30.16	59.04
0.5x	0.13	22.39	30.39	52.92
1x	0.11	19.82	32.06	51.99
2x	0.08	12.66	43.53	56.27

7.2 MAB vs PDTool Under Analytical Workloads

In addition to the static and dynamic workloads, we incorporate random workloads for analytical testing.[9] Random experiments test the delicate balance between swift and careful adaptation under returning workloads, which can lead to unwanted index oscillations. As presented in Table 3, under all three analytical settings (static, dynamic, and random, MAB achieves a faster total workload time, except for SSB and TPC-H static analytical workloads. Consistent with the HTAP experiments, fully analytical workloads on uniform datasets work as the best case for offline tuning tools. However, when underlying data is skewed or dynamic, recommendations based on a pre-determined workload alone can have unfavourable outcomes.

The main experiments used skewed datasets with Zipfian factor 4 (and uniform datasets with Zipfian factor 0). To further investigate the impact of the degree of data skew, we experiment with different Zipfian factors ranging from 1 to 3 with analytical workloads. As shown in Figure 13 under Zipfian factors 2 and 3, MAB demonstrates over 51% and 58% performance gain against PDTool, respectively. Whereas under Zipfian factor 1, PDTool outperforms MAB by 16%. PDTool missing the index on $Orders.O\_custkey$ appears to be more costly with Zipfian factors 2 and 3, mainly affecting Q22. An in-depth analysis of the solution fitness on analytical workloads can be found in [40].

7.2.1 The Impact of Database Size

To examine the impact of database size, we run TPC-H uniform and TPC-H Skew benchmarks with static analytical workloads on SF 1, 10 and 100 databases. Under SF 10, MAB performs better in the case of TPC-H Skew and PDTool performs better on TPC-H (see Table 4). The impact of sub-optimal index choices is even more evident for larger databases, leading to a huge gap between total workload times of MAB and PDTool for TPC-H Skew (44 hours in the former vs 20 hours in the latter case). In TPC-H, PDTool results in a higher total workload time (14.8 hours vs. 13.2 hours for MAB). This is mainly due to sub-optimal optimiser decisions, where the optimiser favours the usage of indices (coupled with nested loops joins) when alternative plans would be a better option. For instance, under the recommended indices from PDTool, some instances of Q5 run longer than 8 minutes (using index nested loops join), whereas others finish in 1.5 minutes (using a plan based on hash joins). We notice that, with larger database sizes, execution time dominates contributing more than 91% to the total workload time. We observe faster and more accurate convergence of MAB under larger databases, due to a clear difference between rewards for different arms, highlighting MAB's excellent potential for larger databases.

7.2.2 Hypothetical Index Creation vs Actual Index Creation

Managing the exploration-exploitation balance under a large number of candidate indices, with an enormous number of combinatorial choices, is non trivial. PDTool explores using the "what-if" analysis, which comes under the tool's recommendation time, whereas MAB explores using index creations.

Comparing the total of recommendation and index creation times (henceforth referred to as *exploration cost*) between MAB and PDTool presents a clear picture about these two exploration methods. From Table 3 we can observe that, in most cases (9 out of 15) MAB archives a better exploration cost compared to PDTool when running analytical workloads. However when the workload is small (e.g., dynamic shifting) PDTool tends to perform better. TPC-DS, with the highest number of candidate indices among these benchmarks (over 3200 indices), provides a great test case for exploration efficiency. Under TPC-DS, MAB exploration cost is significantly lower in shifting and random settings, and marginally higher in the static setting. Despite the efficient exploration, MAB does not sacrifice recommendation quality in any way (achieving faster execution times in 12

9. A query sequence is chosen entirely at random (modelling more dynamic settings, such as cloud services)

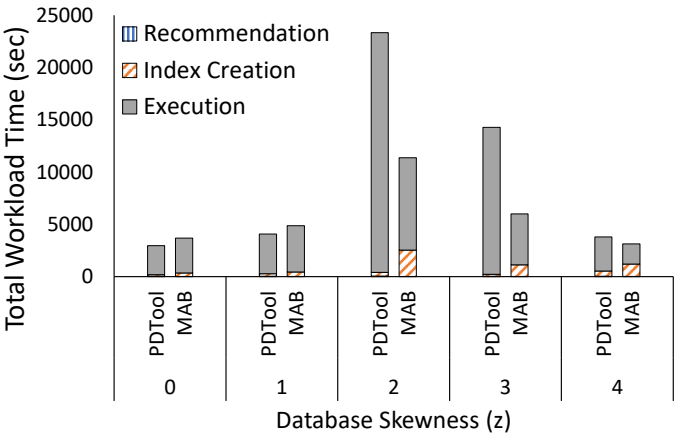

Fig. 13. MAB vs. PDTool total end-to-end workload time under TPC-H skew *static* analytical workloads with different Zipfian factors (z)

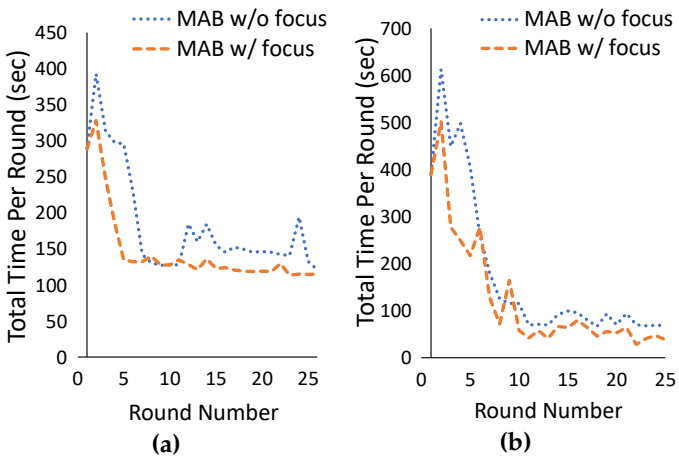

Fig. 14. MAB end-to-end workload time convergence with and without focus update for *static* analytical workloads: (a) TPC-H, (b) TPC-H Skew

out of 15 cases, with significantly faster execution across all TPC-DS experiments).

This efficient exploration is promoted by the linear reward-context relationship along with C^{2}UCB's weight sharing (Section 3), resulting in a small number of parameters to learn. An arm's identity becomes irrelevant and context (Section 4) becomes the sole determining factor of each arm's expected score, which allows MAB to predict the UCB of a newly arriving arm with known context ***without* trying it even once**.

7.2.3 The Impact of Round Size

In the original TPC-H Skew static analytical experiment (1x), each bandit round includes all the benchmark templates (22 queries). To analyse the impact of the round size (bandit invocation frequency), we conduct experiments with single-query (1 query), 0.5x (11 queries) and 2x (44 queries) round sizes using the TPC-H Skew analytical workload. All three round sizes converge to the same performant configurations by the last round. We observe a faster convergence with small round sizes, resulting in lower execution costs in the first few rounds. While the execution cost gain from 1x to 0.5x is noticeable, dividing the round further (single query) does not provide a considerable benefit compared to the added creation and recommendation overhead. With larger round sizes, we observe lower creation costs due to less frequent bandit updates (see Table 5). MAB performs better under all round-sizes compared to PDTool. A DBA can decide on the round size (bandit invocation frequency) based on the application and DBA's primary goal (faster convergence vs lower creation cost). We leave auto-tuning of this parameter as an interesting future work avenue.

7.2.4 The Impact of Focused Updates

This section analyses the impact of focused updates on the convergence speed. As shown in Figure 14, under both TPC-H and TPC-H Skew analytical workloads, there is a clear improvement in convergence speed. Faster convergence results in 17% and 20% gain in total execution time and 21% and 28% gain in total workload time under TPC-H and TPC-H skew benchmarks with focused updates, respectively.

7.3 Experimental Results Summary

This section summarises the experimental results. In an experiment against a uniform dataset and 6 TARs, MAB showed a 28.9% and 20.8% gain on average compared to PDTool and MCTS, respectively. Diving deeper into the 3:1 TAR, we noticed that MAB provides improvements in both transactional execution cost and analytical execution cost. Execution cost gain from MAB peaks with more balanced TARs. With workloads that are at the extreme ends of the spectrum (either purely analytical or purely transactional queries), both PDTool and MAB provide similar execution times. However, large PDTool recommendation times were observed with transactional heavy workloads. Besides better performance, we noticed that MAB also provides remarkable memory saving with transactional heavy workloads (up to 83%).

Using learned knowledge, MAB performed much better than PDTool in dynamic experiments. On the other hand, PDTool's performance degraded in the dynamic setting when index recommendations had to consider existing indices. Furthermore, we demonstrated the superiority of MAB-based PDS tuning over different bandit update frequencies and with large databases. Finally, we demonstrated that under analytical workloads, MAB outperforms PDTool in 13/15 experiments, showcasing the robustness of the MAB framework.

8 Related Work

HTAP. HTAP workloads are composed of online transaction processing (OLTP) workloads and online analytical processing (OLAP) workloads. While most existing analytical systems depend on data pipelines writing to a separate data warehouse for OLAP queries, such an approach limits the users from running analytics on fresh operational data. Research has targeted hybrid environments that can cater to OLTP statements and OLAP queries. The last few years have witnessed the emergence of HTAP focused database architectures, platforms and databases [42], [43], [44], [45], [46], [47], [48], commercial tools [49], [50], [51], and benchmarks [32], [33], [52]. This rapid growth of research and commercial interest in HTAP environments highlights an important point of efficiently processing analytical and transactional statements over the same dataset.

Automated physical design tuning. Most commercial DBMS vendors nowadays offer physical design tools in their products [1], [2], [3]. These tools rely heavily on the query optimiser to compare benefits of different design structures without materialisation [16]. Such an approach is ineffective when base data statistics are unavailable, skewed, or change dynamically [10]. In these dynamic environments, the problem of physical design is aggravated: a) deciding *when* to call a tuning process is not straightforward; and b) deciding *what* is a representative training workload is a challenge.

Online physical design tuning. Several research groups have recognised these problems and have offered lightweight solutions to physical design tuning [11], [12], [13]. While such solutions are more flexible and need not know the workload in advance, they are typically limited in terms of applicability to new unknown workloads (generalisation beyond past), and do not come with theoretical guarantees that extend to actual runtime conditions. Moreover, by giving the optimiser a central role, the tools remain susceptible to its mistakes [9]. [8] extends [1] with the use of additional components, in a narrowed scope of index selection to mimic an online tool. This takes corrective actions against the optimiser mistakes through a validation process.

Adaptive and learning indices. Another dimension of online physical design tuning is database cracking and adaptive indexing that smooth the creation cost of indices by piggybacking on query execution [53], [54]. Recent efforts have gone a step further and proposed replacing data structures with learned models that are smaller in size and faster to query [55], [56]. Such approaches are complementary to our efforts: once the data structures (or models) are materialised inside a DBMS, the MAB framework can be used to automate the decision making as to which data structure should be used to speed-up query analysis.

Learning approaches to optimisation and tuning. Recent years have witnessed new machine learning approaches to automate decision-making processes within databases. For instance, reinforcement learning approaches have been used for query optimisation and join ordering [57], [58], [59], [60]. In [9], regression has been used to successfully mitigate the optimiser's cost misestimates as a path toward more robust index selection. [9] shows promising results when avoiding query regressions. However, this classifier incurs up to 10% recommendation time, impacting recommendation cost in all cases, especially where recommendation cost already dominates the cost for PDTool (e.g., TPC-DS, IMDb).

When it comes to tuning, the closest approaches employ variants of RL for index selection or partitioning [23], [30], [31], [61], [62] or configuration tuning [5], [29]. [62] describes RL-based index selection, which depends solely on the recommendation tool for query-level recommendations and is affected by decision combinatorial explosion, both issues addressed in our work. Unlike its more general counterpart (RL), MABs have advantages of faster convergences, simpler implementation, and theoretical guarantees [40]. There has also been recent interest in using bandits for database tasks such as monitoring, query optimisation and join ordering [63], [64], [65].

Use of learned cost/cardinality estimators and cost models. Learned cost estimators and models [66], [67] allow accurate and faster cost estimations and provide better execution plans. Better estimations and models can be particularly beneficial for avoiding estimation errors in offline optimiser-based tools like PDTool. Even learned systems like our MAB system can benefit from such cost models to avoid the cold start problem [68]. These learned models and estimations will require using 'optimiser hints', forcing the optimiser to use a different query plan than its original choice. However, when used externally with commercial systems, flexibility in optimiser hints will be limited. Furthermore, such learned solutions suffer from long training times [66], which will be problematic given that PDTool already suffers from long recommendation times.

Workload compression. Large complex workloads have been a challenge for index tuning tools. This is visible from high recommendation times in PDTool and high creation times in HMAB under CH-BenCHmark. Workload compression [69], [70], [71] can alleviate these challenges by efficiently identifying a small subset of queries that can be used for index tuning.

Workload forecasting. Both PDtool and MAB have limited visibility into the future. Understanding what a future workload might look like would allow these tools to provide better recommendations. We acknowledge the progress in workload forecasting [72] as a complementary research direction to PDS tuning.

9 Conclusions

This paper develops a multi-armed bandit learning framework for online index selection. This framework does not depend on the DBA and the (error-prone) query optimiser for index selection and learns the benefits of indices through strategic exploration and observation. We justify our choice of MAB over general reinforcement learning for online index tuning, comparing MAB against DDQN, a popular RL algorithm based on deep neural networks, demonstrating significantly faster convergence of the MAB. Furthermore, our extensive experimental evaluation demonstrates advantages of MAB over an existing commercial physical design tool (up to 75% speed up, and 23% on average), and exemplifies robustness to data skew, unpredictable ad-hoc workloads and complex HTAP environments.

Acknowledgments

We gratefully acknowledge support from the Australian Research Council Discovery Project DP220102269 as well as Discovery Early Career Researcher Award DE230100366.

References

[1] S. Agrawal, S. Chaudhuri, L. Kollár, A. P. Marathe, V. R. Narasayya, and M. Syamala, "Database tuning advisor for Microsoft SQL Server 2005," in *VLDB*, 2004, pp. 1110–1121.

[2] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. J. Storm, C. Garcia-Arellano, and S. Fadden, "DB2 design advisor: Integrated automatic physical database design," in *VLDB*, 2004, pp. 1087–1097.

[3] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zaït, and M. Ziauddin, "Automatic SQL tuning in oracle 10g," in *VLDB*, 2004, pp. 1098–1109.

[4] D. Zilio, S. Lightstone, K. Lyons, and G. Lohman, "Self-managing technology in IBM DB2 universal database," in *ACM CIKM*, 2001, pp. 541–543.

[5] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. C. Mowry, M. Perron, I. Quah *et al.*, "Self-driving database management systems," in *CIDR*, 2017.

[6] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational cloud: a database service for the cloud," in *CIDR*, 2011, pp. 235–240.

[7] V. R. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri, "SQLVM: Performance isolation in multi-tenant relational database-as-a-service," in *CIDR*, 2013.

[8] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri, "Automatically indexing millions of databases in microsoft azure sql database," in *SIGMOD*, 2019, pp. 666–679.

[9] B. Ding, S. Das, R. Marcus, W. Wu, S. Chaudhuri, and V. R. Narasayya, "AI Meets AI: leveraging query executions to improve index recommendations," in *SIGMOD*, 2019.

[10] S. Chaudhuri and V. Narasayya, "Self-tuning database systems: A decade of progress," in *VLDB*, 2007, pp. 3–14.

[11] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis, "On-Line index selection for shifting workloads," in *ICDEW*, 2007, pp. 459–468.

[12] K.-U. Sattler, I. Geist, and E. Schallehn, "QUIET: Continuous querydriven index tuning," in *VLDB*. Elsevier, 2003, pp. 1129–1132.

[13] N. Bruno and S. Chaudhuri, "An online approach to physical design tuning," in *ICDE*, 2007, pp. 826–835.

[14] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou, "The researcher's guide to the data deluge: Querying a scientific database in just a few seconds," *VLDB*, vol. 4, no. 12, pp. 1474–1477, 2011.

[15] I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki, "NoDB: Efficient query execution on raw data files," in *SIGMOD*, 2012, pp. 241–252.

[16] S. Chaudhuri and V. Narasayya, "AutoAdmin "what-if"; index analysis utility," in *SIGMOD*, 1998, pp. 367–378.

[17] S. Christodoulakis, "Implications of certain assumptions in database performance evaluation," *TODS*, vol. 9, no. 2, pp. 163–186, 1984.

[18] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" *VLDB*, vol. 9, no. 3, pp. 204–215, Nov. 2015.

[19] A. Aboulnaga and S. Chaudhuri, "Self-tuning histograms: Building histograms without looking at data," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 181–192, 1999.

[20] R. Borovica-Gajic, S. Idreos, A. Ailamaki, M. Zukowski, and C. Fraser, "Smooth scan: Robust access path selection without cardinality estimation," *The VLDB Journal*, vol. 27, no. 4, pp. 521–545, 2018.

[21] R. Borovica, I. Alagiannis, and A. Ailamaki, "Automated physical designers: What you see is (not) what you get," in *DBTest*, 2012, p. 9.

[22] K. E. Gebaly and A. Aboulnaga, "Robustness in automatic physical database design." in *EDBT*, vol. 261. ACM, 2008, pp. 145–156.

[23] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," 2018, unpublished.

[24] L. Qin, S. Chen, and X. Zhu, "Contextual combinatorial bandit and its application on diversified online recommendation," in *SDM*, 2014, pp. 461–469.

[25] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *WWW*, 2010, pp. 661–670.

[26] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions–i," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[27] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits," in *NIPS*, vol. 11, 2011, pp. 2312–2320.

[28] J. Kossmann, S. Halfpap, M. Jankrift, and R. Schlosser, "Magic mirror in my hand, which is the best in the land? an experimental evaluation of index selection algorithms," *VLDB*, vol. 13, no. 12, pp. 2382–2395, 2020.

[29] J. Wang, I. Trummer, and D. Basu, "Udo: universal database optimization using reinforcement learning," *VLDB*, vol. 14, no. 13, pp. 3402–3414, 2021.

[30] W. Wu, C. Wang, T. Siddiqui, J. Wang, V. Narasayya, S. Chaudhuri, and P. A. Bernstein, "Budget-aware index tuning with reinforcement learning," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1528–1541. [Online]. Available: https://doi.org/10.1145/3514221.3526128

[31] X. Zhou, L. Liu, W. Li, L. Jin, S. Li, T. Wang, and J. Feng, "Autoindex: An incremental index management system for dynamic workloads," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2196–2208.

[32] R. Cole, F. Funke, L. Giakoumakis, W. Guy, A. Kemper, S. Krompass, H. Kuno, R. Nambiar, T. Neumann, M. Poess *et al.*, "The mixed workload ch-benchmark," in *DBTest*, 2011, pp. 1–6.

[33] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *VLDB*, vol. 7, no. 4, pp. 277–288, 2013.

[34] TPC, "TPC-H benchmark," http://www.tpc.org/tpch/.

[35] Microsoft, "TPC-H skew benchmark," https://www.microsoft.com/en-us/download/details.aspx?id=52430.

[36] TPC, "TPC-C benchmark," http://www.tpc.org/tpcc/.

[37] R. M. Perera, "Tpc-h and tpc-h skew htap workload," https://github.com/malingaperera/TPC_H_HTAP, 2021.

[38] R. O. Nambiar and M. Poess, "The making of tpc-ds," in *VLDB*. VLDB Endowment, 2006, p. 1049–1058.

[39] P. O. Neil, B. O. Neil, and X. Chen, "Star schema benchmark," 2009, unpublished.

[40] R. M. Perera, B. Oetomo, B. I. Rubinstein, and R. Borovica-Gajic, "Dba bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees," in *ICDE*. IEEE, 2021, pp. 600–611.

[41] S. Chaudhuri and V. R. Narasayya, "Anytime algorithm of database tuning advisor for microsoft sql server." https://www:microsoft:com/en-us/research/publication/anytime-algorithm-of-databasetuning-advisor-for-microsoft-sql-server, 2020.

[42] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki, "The case for heterogeneous htap," in *CIDR*, 2017.

[43] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *SIGMOD*, 2016, p. 583–598.

[44] M. Athanassoulis, K. S. Bøgh, and S. Idreos, "Optimal column layout for hybrid workloads," *VLDB*, vol. 12, no. 13, p. 2393–2407, Sep. 2019.

[45] D. Makreshanski, J. Giceva, C. Barthels, and G. Alonso, "Batchdb: Efficient isolated execution of hybrid oltp+olap workloads for interactive applications," in *SIGMOD*. Association for Computing Machinery, 2017, p. 37–50.

[46] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, W. Wei, C. Liu, J. Zhang, J. Li, X. Wu, L. Song, R. Sun, S. Yu, L. Zhao, N. Cameron, L. Pei, and X. Tang, "Tidb: A raft-based htap database," *VLDB*, vol. 13, no. 12, p. 3072–3084, Aug. 2020.

[47] J. Ramnarayan, B. Mozafari, S. Wale, S. Menon, N. Kumar, H. Bhanawat, S. Chakraborty, Y. Mahajan, R. Mishra, and K. Bachhav, "Snappydata: A hybrid transactional analytical store built on spark," in *SIGMOD*, 2016, p. 2153–2156.

[48] A. Kemper and T. Neumann, "Hyper: A hybrid oltp olap main memory database system based on virtual memory snapshots," in *ICDE*, 2011, pp. 195–206.

[49] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T.-H. Lee, J. Loaiza, N. Macnaughton, V. Marwah, N. Mukherjee, A. Mullick, S. Muthulingam, V. Raja, M. Roth, E. Soylemez, and M. Zait, "Oracle database in-memory: A dual format in-memory database," in *ICDE*, 2015, pp. 1253–1258.

[50] P.-r. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos, "Real-time analytical processing with sql server," *VLDB*, vol. 8, no. 12, p. 1740–1751, Aug. 2015.

[51] J. Yang, I. Rae, J. Xu, J. Shute, Z. Yuan, K. Lau, Q. Zeng, X. Zhao, J. Ma, Z. Chen, Y. Gao, Q. Dong, J. Zhou, J. Wood, G. Graefe, J. Naughton, and J. Cieslewicz, "F1 lightning: Htap as a service," *VLDB*, vol. 13, no. 12, p. 3313–3325, Aug. 2020.

[52] F. Coelho, J. a. Paulo, R. Vilaça, J. Pereira, and R. Oliveira, "Htap-bench: Hybrid transactional and analytical processing benchmark," in *ICPE*, 2017, p. 293–304.

[53] S. Idreos, M. L. Kersten, and S. Manegold, "Database cracking," in *CIDR*, 2007, pp. 68–78.
[54] G. Graefe and H. A. Kuno, "Self-selecting, self-tuning, incrementally optimized indexes." in *EDBT*, 2010, pp. 371–381.
[55] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The case for learned index structures," in *SIGMOD*, 2018, pp. 489–504.
[56] A. Galakatos, M. Markovitch, C. Binnig, R. Fonseca, and T. Kraska, "Fiting-tree: A data-aware index structure," in *SIGMOD*, 2019.
[57] T. Kaftan, M. Balazinska, A. Cheung, and J. Gehrke, "Cuttlefish: A lightweight primitive for adaptive query processing," 2018, unpublished.
[58] I. Trummer, S. Moseley, D. Maram, S. Jo, and J. Antonakakis, "SkinnerDB: Regret-bounded query evaluation via reinforcement learning," *VLDB*, vol. 11, no. 12, pp. 2074–2077, 2018.
[59] A. Kipf, T. Kipf, B. Radke, V. Leis, P. A. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," in *CIDR*, 2019.
[60] R. Marcus and O. Papaemmanouil, "Towards a hands-free query optimizer through deep learning," in *CIDR*, 2019.
[61] B. Hilprecht, C. Binnig, and U. Röhm, "Towards learning a partitioning advisor with deep reinforcement learning," in *aiDM*, 2019.
[62] D. Basu, Q. Lin, W. Chen, H. T. Vo, Z. Yuan, P. Senellart, and S. Bressan, "Regularized cost-model oblivious database tuning with reinforcement learning," in *TLDKS*. Springer, 2016, pp. 96–132.
[63] H. Grushka-Cohen, O. Biller, O. Sofer, L. Rokach, and B. Shapira, "Using bandits for effective database activity monitoring," in *PAKDD*, H. W. Lauw, R. C.-W. Wong, A. Ntoulas, E.-P. Lim, S.-K. Ng, and S. J. Pan, Eds., 2020, pp. 701–713.
[64] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making learned query optimization practical," in *SIGMOD*, 2020.
[65] V. Ghadakchi, M. Xie, and A. Termehchy, "Bandit join: Preliminary results," in *aiDM-SIGMOD*, 2020.
[66] H. Lan, Z. Bao, and Y. Peng, "A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration," *Data Science and Engineering*, vol. 6, pp. 86–101, 2021.
[67] G. Li, X. Zhou, and L. Cao, *AI Meets Database: AI4DB and DB4AI*. New York, NY, USA: Association for Computing Machinery, 2021, p. 2859–2866. [Online]. Available: https://doi.org/10.1145/3448016.3457542
[68] B. Oetomo, R. M. Perera, R. Borovica-Gajic, and B. I. Rubinstein, "Cutting to the chase with warm-start contextual bandits," in *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2021, pp. 459–468.
[69] S. Chaudhuri, A. K. Gupta, and V. Narasayya, "Compressing sql workloads," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 488–499.
[70] S. Deep, A. Gruenheid, P. Koutris, J. F. Naughton, and S. Viglas, "Comprehensive and efficient workload compression," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 418–430, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol14/p418-deep.pdf
[71] T. Siddiqui, S. Jo, W. Wu, C. Wang, V. Narasayya, and S. Chaudhuri, "Isum: Efficiently compressing large and complex workloads for scalable index tuning," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 660–673. [Online]. Available: https://doi.org/10.1145/3514221.3526152
[72] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon, "Query-based workload forecasting for self-driving database management systems," in *SIGMOD*, 2018, pp. 631–645.

R. Malinga Perera Malinga Perera is a PhD candidate in the School of Computing and Information Systems at the University of Melbourne. His research interests include machine learning and databases. He was part of the team designing multiple large scale big-data systems which won national-level awards in Sri Lanka (1st runner up 'Best Technology or Framework Innovation' in SLASSCOM Innovation Awards 2019). He completed his bachelor's in the University of Moratuwa, Sri Lanka (CS and Eng).

Bastian Oetomo Bastian Oetomo is a PhD candidate in the School of Computing and Information Systems at the University of Melbourne. His research is on multi-armed bandit applied to databases. He completed his DMathSc (Applied Mathematics), BSc (Mechanical Systems) and MEng (Electrical) at the University of Melbourne. During his studies, he received multiple student awards for his performance.

Benjamin I. P. Rubinstein Benjamin Rubinstein is a Professor in the School of Computing and Information Systems, at the University of Melbourne. His research interests span machine learning, security & privacy, and databases. He has been part of teams that have: analysed privacy of products at the Australian Bureau of Statistics, the financial industry, and Transport for NSW; robustness of translation systems to data poisoning attacks with Meta; helped identify and plug side-channel attacks against the Firefox browser; deanonymised Victorian Myki transport data and an unprecedented Australian Medicare data release; developed scalable Bayesian approaches to record linkage tested by U.S. Census; and shipped production systems for entity resolution in Bing and the Xbox360. Rubinstein completed a BSc (Pure Maths), BEng (Software Hons.), MCompSci (Research) at the University of Melbourne, and a PhD (CS) at UC Berkeley in 2010.

Renata Borovica-Gajic Renata Borovica-Gajic holds a position of Senior Lecturer in Data Analytics in the School of Computing and Information Systems at The University of Melbourne. Dr Borovica-Gajic received her Ph.D. degree in Computer Science from Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland. Renata's research focuses on solving data management problems when storing, accessing and processing massive data sets, enabling faster, more predictable, and cheaper data analysis as a result. She is also interested in the topics of scientific data management, data exploration, query optimization, physical database design, and hardware-software co-design. Her work has repeatedly appeared in premier data management outlets, including SIGMOD, VLDB, ICDE and VLDB Journal, and she is a recipient of the SIGMOD 2022 Test-of-Time Award as well as Australian Research Council (ARC) DECRA Fellowship.