Hybrid Cost Modeling for Reducing Query Performance Regression in Index Tuning

Wentao Wu

Abstract—Autonomous index tuning ("auto-indexing" for short) has recently started being supported by cloud database service providers. Index tuners rely on query optimizer's cost estimates to recommend indexes that can minimize the execution cost of an input workload. Such cost estimates can often be erroneous that lead to significant query performance regression. To reduce the chance of regression, existing work primarily uses machine learning (ML) technologies to build prediction models to improve query execution cost estimation using actual query execution telemetry as training data. However, training data collection is typically an expensive process, especially for index tuning due to the significant overhead of creating/dropping indexes. As a result, the amount of training data can be limited in auto-indexing for cloud databases. In this paper, we propose a new approach named "hybrid cost modeling" to address this challenge. The key idea is to limit the ML-based modeling effort to the *leaf operators* such as table scans, index scans, and index seeks, and then combine the ML-model predicted costs of the leaf operators with optimizer's estimated costs of the other operators in the query plan. We conduct theoretical study as well as empirical evaluation to demonstrate the efficacy of applying hybrid cost modeling to index tuning, using both industrial benchmarks and real workloads.

Index Terms—Query optimization, cost modeling, autonomous index tuning

1 Introduction

There has been extensive research in the area of index tuning over the past decades (see [40] for a recent survey). Major commercial database systems, including Oracle [14], IBM DB2 [45], and Microsoft SQL Server [11], are all equipped with index tuning tools. Existing tools adopt a cost-based approach that selects an index *configuration* (i.e., a subset) from a number of candidates that results in the minimum query optimizer's estimated cost for a given workload with multiple queries [11]. A basic yet critical step involved in this process is therefore to estimate the cost for a query over a candidate index configuration. While implementations vary, most existing systems rely on the so-called "what-if" utility that allows the optimizer to generate query plans and estimate their costs for a given pair of query and index configuration by creating the indexes as "hypothetical indexes" that are not materialized [12].

Figure 1 outlines this cost-based index tuning architecture. The index tuner sends a query Q with the description of a candidate index configuration C to the query optimizer. The "what-if" utility of the query optimizer simulates C by creating the hypothetical indexes contained by C. That is, it generates all metadata and statistics information about these indexes and makes them visible to the query optimizer. The query optimizer utilizes this information to estimate costs for query plans that use the indexes in C. The best plan P chosen by the query optimizer under the configuration C, along with the query optimizer's cost estimate for P, is returned to the index tuner.

In recent years, cloud database providers have started offering *autonomous* index tuning service ("auto-indexing" for short), during which indexes are recommended and applied directly to production database servers in a continuous manner by monitoring the database query workloads [15]. Since the indexes are recommended based on query optimizer's estimated costs, query performance regression (QPR) is likely to happen after the recommended indexes are materialized, due to errors from various sources such as cardinality estimation and cost modeling [17]. Such QPR can often be significant and, to fix QPR, the created indexes have to be dropped [15]. Therefore, the time and resource spent on auto-indexing is often wasted due to QPR. To reduce the chance of QPR, most existing solutions (e.g., [17, 37, 54]) rely on building machine learning (ML) models to improve query execution time prediction by leveraging query execution feedback. Compared to earlier technologies on improving query optimizer's estimated costs, such as calibration of cost model constants (e.g., [18, 30]), or improving cardinality estimation with execution feedback (e.g., [9]) or sampling (e.g., [50, 51]), ML-based cost modeling has the privilege of optimizing for the cost estimation errors directly (instead of reducing errors in individual components of cost modeling) and therefore typically results in better cost estimates.

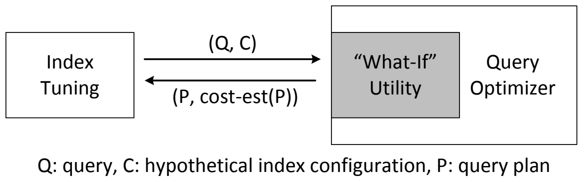
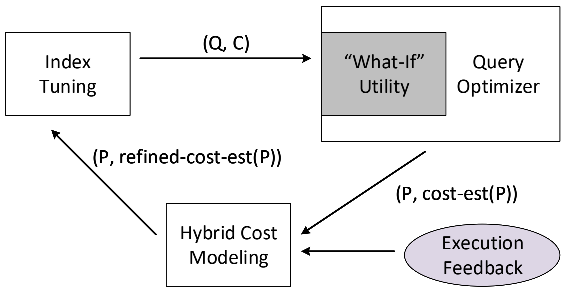


Fig. 1. Architecture of cost-based index tuning.

One major challenge faced when applying these learning-based cost modeling techniques in practice is the collection of training data, which is an expensive process in

- *Wentao Wu is with Microsoft Research, Redmond, WA, USA. E-mail: wentao.wu@microsoft.com.*



Q: query, C: hypothetical index configuration, P: query plan

Fig. 2. Revised architecture of index tuning using query execution feedback and hybrid cost modeling.

general [46]. This challenge becomes even more severe when it comes to index tuning [17], due to the significant overhead of creating/dropping indexes. As a result, in the cloud auto-indexing scenario it is not feasible to proactively collect training data from the production server, which would result in dramatic interruption or slowdown of customer workloads. One proposal in the literature is to use a B-instance (i.e., a replica) that mirrors the production database server but does not serve customer workloads at real time [29]. Nonetheless, the operational cost of using B-instance is often prohibitive for cloud database service providers [15]. Therefore, in reality one could only hope for limited amount of training data to be *passively* obtained during auto-indexing by monitoring query execution, which becomes a bottleneck for training ML-based cost models.

In this paper, we propose "hybrid cost modeling" to address the challenge of limited execution data for training ML models, in the context of auto-indexing. Our main idea is to only train ML models for *leaf* operators in query execution plans such as table scans, index scans, and index seeks. We then develop a simple yet principled approach to *combine* the ML model predictions for the execution costs of the leaf operators and the query optimizer's estimated execution costs of the other operators in the query plan. We conduct both theoretical analysis and experimental evaluation to demonstrate the efficacy of our proposed approach in improving index tuning, using both industrial benchmarks and real workloads.

Figure 2 presents the revised index tuning architecture that leverages query execution feedback and hybrid cost modeling. After the query optimizer returns the best plan P, we refine its cost estimate using hybrid cost modeling on top of available query execution feedback. We then send the refined cost estimate, instead of the original cost estimate from the query optimizer, to the index tuner. Note that our approach is *passive* rather than *proactive*: We do not use hybrid cost modeling inside query optimizer to affect its plan choice, which could be another option but also require surgical change to the query optimizer. As a result, if the query optimizer ends up with choosing a poor execution plan because of bad cost estimates, we cannot bail it out. However, by refining cost estimates afterwards, we increase the chance of *detecting* such bad plans during index tuning and therefore avoiding corresponding disastrous index configurations in future tuning sessions that may lead to serious query performance regression [17].

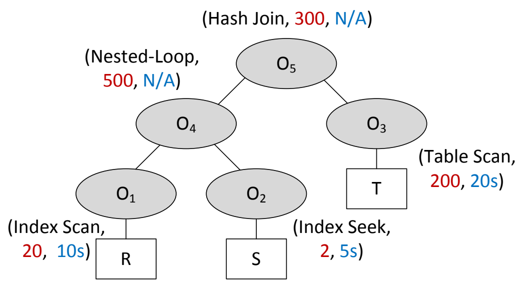


Fig. 3. A running example of hybrid cost modeling.

Cost Unit	Default Value
seq_page_cost	1.0
random_page_cost	4.0
cpu_tuple_cost	0.01
cpu_index_tuple_cost	0.005
cpu_operator_cost	0.0025
parallel_tuple_cost	0.1

TABLE 1
Cost units used by PostgreSQL's query optimizer [2].

(Paper Organization) The rest of the paper is organized as follows. We propose hybrid cost modeling in Section 2 and present an analysis of its efficacy in Section 3. In Section 4, we present experimental evaluation results of applying hybrid cost modeling in auto-indexing. We summarize related work in Section 5 and conclude in Section 6.

2 Hybrid Cost Modeling

The idea of having ML-model predicted execution costs for the leaf operators and query optimizer's estimated costs for the other operators *simultaneously* in the same query plan raises a new challenge of combining two different types of cost estimates. To understand the issue of hybrid operator-level cost estimates better, Figure 3 presents an annotated query execution plan as a running example.

Here R, S, and T are tables, whereas O_1 to O_5 are physical operators. We annotate each operator with its type and estimated cost. In this example, the cost estimates for the three *leaf* operators O_1, O_2, and O_3 come from ML-based cost models, which indicate the estimated CPU execution time of these operators. On the other hand, the cost estimates for the two *internal* operators O_4 and O_5 are made by the optimizer, which also measure the 'execution time' of these operators but are based on the built-in *cost units* of the query optimizer. For the purpose of illustration, Table 1 presents some examples of such cost units used by PostgreSQL's query optimizer [2]. The semantics of these cost units are optimizer-specific. While their semantics are consistent (and therefore can be combined) inside the query optimizer's own cost modeling system, they may not be aligned with ML-model predicted costs. Therefore, it may be invalid to directly combine (e.g., sum up) model predictions with query optimizer's estimated costs.

2.1 Combining Mixed Types of Cost Estimates

We propose *hybrid cost modeling*, a simple yet principled approach to combining different types of operator-level cost estimates that coexist in the same query plan, which can be

Algorithm 1: Hybrid cost modeling.

```
Input: $P$, a query plan; $\mathcal{O}$, the set of operators with sufficient execution feedback; $\mathcal{M}$, the operator-level ML-based cost models built with execution feedback of $\mathcal{O}$.
Output: $\text{cost}(P)$, estimated cost of $P$.
1  $\text{cost}(P) \leftarrow 0$;
2  $o^{\text{pivot}} \leftarrow PickPivot(\mathcal{O})$; // Find the "pivot" operator (see Algorithm 2 for details).
3  foreach operator $o \in P$ do
4      if there is a model $M \in \mathcal{M}$ for $o$ then
5          $\text{ml-cost}(o) \leftarrow M(o)$;
6          $\text{cost}(P) \leftarrow \text{cost}(P) + \frac{\text{ml-cost}(o)}{\text{act-cost}(o^{\text{pivot}})} \cdot \text{opt-cost}(o^{\text{pivot}})$;
7      else
8          $\text{cost}(P) \leftarrow \text{cost}(P) + \text{opt-cost}(o)$;
9      end
10 end
11 return $\text{cost}(P)$;
```

Notation	Description
o	An operator in the query plan
$\text{opt-cost}(o)$	The query optimizer's estimated cost of o
$\text{act-cost}(o)$	The actual execution cost of o
$\text{ml-cost}(o)$	The cost estimate of o from ML-based cost models
o^{pivot}	The pivot operator

TABLE 2
Terminology and notation used by Algorithm 1.

viewed as a generalized form of cost-unit calibration [18, 30, 49, 50, 51, 53]. Algorithm 1 presents the details. We summarize the notation used by Algorithm 1 in Table 2.

Our main idea is the following. We choose one "pivot" operator o^{pivot} from the operators with execution feedback (line 2). We use the execution cost of o^{pivot} as a baseline, and compute the *relative* cost

$$\text{rel-cost}(o) = \frac{\text{ml-cost}(o)}{\text{act-cost}(o^{\text{pivot}})} \quad (1)$$

for any operator o in the given plan P where we have an ML-based operator-level cost model. We then scale the relative cost back using $\text{opt-cost}(o^{\text{pivot}})$. For any operator in P without an ML-based cost model, we simply use the query optimizer's cost estimate for it (lines 3 to 10).

Example 1: Continue with the running example in Figure 3. Given the three operators O_1 to O_3 with available execution feedback, suppose that we choose O_3 as the pivot operator. Assume that the ML-based cost models for table scans, index scans, and index seeks are perfect, i.e., for any such operator o we would have $\text{ml-cost}(o) = \text{act-cost}(o)$. The relative costs of O_1, O_2, and O_3 are then

$$\text{rel-cost}(O_1) = \frac{\text{ml-cost}(O_1)}{\text{act-cost}(O_3)} = \frac{\text{act-cost}(O_1)}{\text{act-cost}(O_3)} = \frac{10}{20} = 0.5,$$

$$\text{rel-cost}(O_2) = \frac{\text{ml-cost}(O_2)}{\text{act-cost}(O_3)} = \frac{\text{act-cost}(O_2)}{\text{act-cost}(O_3)} = \frac{5}{20} = 0.25,$$

and

$$\text{rel-cost}(O_3) = \frac{\text{ml-cost}(O_3)}{\text{act-cost}(O_3)} = \frac{\text{act-cost}(O_3)}{\text{act-cost}(O_3)} = \frac{20}{20} = 1.$$

Notation	Description
$\mathcal{L}$	Leaf operators
$\mathcal{I}$	Internal operators
P	Plan CPU time
L	Leaf CPU time
I	Internal CPU time
α	$\rho(L, I)$, Pearson CC between L and I
σ_L	Standard deviation of L
σ_I	Standard deviation of I
η	$\eta = \frac{\sigma_L}{\sigma_I}$
P'	Estimated plan cost
L'	Estimated leaf cost
I'	Estimated internal cost
$\sigma_{L'}$	Standard deviation of L'
$\sigma_{I'}$	Standard deviation of I'
η'	$\eta = \frac{\sigma_{L'}}{\sigma_{I'}}$
β	$\rho(L, I')$, Pearson CC between L and I'
γ	$\rho(I, I')$, Pearson CC between I and I'
ρ	$\rho(P, P')$, Pearson CC between P and P'

TABLE 3
Notation used in problem formulation and analysis.

Meanwhile, the scaling factor is $\text{opt-cost}(O_3) = 200$. Consequently, the adjusted estimated costs for O_1, O_2, and O_3 are $0.5 \times 200 = 100$, $0.25 \times 200 = 50$, and $1 \times 200 = 200$. Therefore, the final estimated cost for the example plan P is $\text{cost}(P) = 100 + 50 + 200 + 500 + 300 = 1150$.

2.2 Application to Index Tuning

Note that Algorithm 1 remains generic as we have not yet specified the input set of operators $\mathcal{O}$. While $\mathcal{O}$ can be arbitrary in theory, it is typically application-driven. In the context of index tuning, we propose to focus on *leaf* operators, including table scans, index scans, index seeks, and so on (depending on the operator types supported by the database system), for the following reasons:

- First, the operators in $\mathcal{O}$ should have sufficient amount of execution feedback (as auto-indexing proceeds).
- Second, the operators should cover significant amount of work performed by a query plan.
- Third, the operators should reflect the impact of auto-indexing on a query plan.

In the rest of the paper, we call the operators in $\mathcal{O}$ the *backbone* operators, with the understanding that they refer to the leaf operators when applied to auto-indexing. Moreover, while Algorithm 1 assumes that the ML-based cost models are already built and frozen, this is not necessary in reality. As we accumulate more and more execution feedback during auto-indexing, it makes sense to train the ML models again with the new execution data collected.

2.3 Selection of Pivot Operator

Note that Algorithm 1 does not specify how to select the pivot operator o^{pivot}. In theory, we could pick *any* operator with execution feedback. However, it is clear that the choice of o^{pivot} has impact on the estimated cost, because we use $\text{opt-cost}(o^{\text{pivot}})$ as the scaling factor when combining with query optimizer's cost estimates. To better understand this impact, we need a quantitative metric to measure the efficacy of hybrid cost modeling.

Since we primarily target auto-indexing as the application scenario, *we are not very interested in the accuracy* *of the estimated costs returned by hybrid cost modeling*. Instead, we care more about whether we can *compare query plans based on the estimated costs*. Therefore, we are satisfied as long as we can distinguish good plans from bad ones by using the estimated costs. For example, we perhaps only need to know that one plan is 20% better/cheaper than the other one, if we use 20% as the threshold of query performance regression. This suggests that we should consider the *correlation* between the estimated costs and actual execution costs of the plans. Therefore, we use the well-known Pearson correlation coefficient (Pearson CC) as our metric to quantify the efficacy of hybrid cost modeling.

Below we start with a problem formulation, followed by a correlation analysis of hybrid cost modeling. Table 3 summarizes the notation that we will use in our analysis. We then present a simple mechanism for selecting the pivot operator by maximizing the correlation.

2.3.1 Problem Formulation

We use P, L, and I to represent the total CPU time spent on the entire plan, the leaf operators, and the internal operators, respectively. Clearly, $P = L + I$, where $L = \sum_{o\in\mathcal{L}} \text{act-cost}(o)$ and $I = \sum_{o\in\mathcal{I}} \text{act-cost}(o)$.

Similarly, we can represent the estimated plan cost as $P' = L' + I'$, where, by Algorithm 1,

$$L' = \sum_{o\in\mathcal{L}} \frac{\text{ml-cost}(o)}{\text{act-cost}(o^{\text{pivot}})} \cdot \text{opt-cost}(o^{\text{pivot}}) \tag{2}$$

and

$$I' = \sum_{o\in\mathcal{I}} \text{opt-cost}(o). \tag{3}$$

To simplify our analysis, assume that the ML-based cost models are perfect, namely, $\text{ml-cost}(o) = \text{act-cost}(o)$ for any $o \in \mathcal{L}$. The impact of cost modeling errors can be easily incorporated. Moreover, define a constant

$$\lambda = \frac{\text{opt-cost}(o^{\text{pivot}})}{\text{act-cost}(o^{\text{pivot}})}. \tag{4}$$

By Equation 2, it follows that

$$L' = \lambda \cdot \sum_{o\in\mathcal{L}} \text{act-cost}(o) = \lambda \cdot L. \tag{5}$$

2.3.2 Correlation Analysis

We are interested in the Pearson correlation coefficient $\rho(P, P')$ between the actual plan execution cost P and the estimated plan cost P' using hybrid cost modeling. Based on the formulation in Section 2.3.1, we have

$$\rho = \rho(P, P') = \rho(L+I, L'+I') = \rho(L+I, \lambda\cdot L+I'). \tag{6}$$

With the notation in Table 3, we have the following lemma.

Lemma 2: ρ only depends on η, η', α, β, and γ. Specifically, we have the following relationship:

$$\rho = \frac{\eta\eta' + \alpha\eta' + \beta\eta + \gamma}{\sqrt{\eta^2 + 2\alpha\eta + 1}\cdot\sqrt{(\eta')^2 + 2\beta\eta' + 1}}. \tag{7}$$

Proof: By Equation 6, we have

$$\rho = \rho(L+I, \lambda L + I') = \frac{\text{Cov}(L+I, \lambda L + I')}{\sigma_{L+I}\cdot\sigma_{\lambda L+I'}}.$$

By the definition of covariance,

$$\text{Cov}(L+I, \lambda L+I') = \text{E}[(L+I) - \text{E}(L+I)][(\lambda L+I') - \text{E}(\lambda L+I')].$$

Using simple arithmetic calculation, we can obtain

$$\text{Cov}(L+I, \lambda L+I') = \lambda\sigma_L^2 + \lambda\cdot\text{Cov}(L,I) + \text{Cov}(L,I') + \text{Cov}(I,I').$$

On the other hand, by the definition of variance, we have

$$\sigma_{L+I}^2 = \text{E}[(L+I) - \text{E}(L+I)]^2 = \sigma_L^2 + 2\cdot\text{Cov}(L,I) + \sigma_I^2.$$

Similarly, we have

$$\sigma_{\lambda L+I'}^2 = \lambda^2\sigma_L^2 + 2\lambda\cdot\text{Cov}(L,I') + \sigma_{I'}^2.$$

Using the relationships

$$\text{Cov}(L,I) = \rho(L,I)\sigma_L\sigma_I = \alpha\sigma_L\sigma_I,$$
$$\text{Cov}(L,I') = \rho(L,I')\sigma_L\sigma_{I'} = \beta\sigma_L\sigma_{I'},$$
$$\text{Cov}(I,I') = \rho(I,I')\sigma_I\sigma_{I'} = \gamma\sigma_I\sigma_{I'},$$

it then follows that

$$\text{Cov}(L+I, \lambda L+I') = \lambda\sigma_L^2 + \lambda\alpha\sigma_L\sigma_I + \beta\sigma_L\sigma_{I'} + \gamma\sigma_I\sigma_{I'},$$
$$\sigma_{L+I}^2 = \sigma_L^2 + 2\alpha\sigma_L\sigma_I + \sigma_I^2,$$
$$\sigma_{\lambda L+I'}^2 = \lambda^2\sigma_L^2 + 2\lambda\beta\sigma_L\sigma_{I'} + \sigma_{I'}^2.$$

As a result, we have

$$\rho = \frac{\lambda\sigma_L^2 + \lambda\alpha\sigma_L\sigma_I + \beta\sigma_L\sigma_{I'} + \gamma\sigma_I\sigma_{I'}}{\sqrt{\sigma_L^2 + 2\alpha\sigma_L\sigma_I + \sigma_I^2}\cdot\sqrt{\lambda^2\sigma_L^2 + 2\lambda\beta\sigma_L\sigma_{I'} + \sigma_{I'}^2}}.$$

Dividing both the numerator and the denominator by $\sigma_I\sigma_{I'}$,

$$\rho = \frac{\lambda\frac{\sigma_L}{\sigma_I}\frac{\sigma_L}{\sigma_{I'}} + \lambda\alpha\frac{\sigma_L}{\sigma_{I'}} + \beta\frac{\sigma_L}{\sigma_I} + \gamma}{\sqrt{\left(\frac{\sigma_L}{\sigma_I}\right)^2 + 2\alpha\frac{\sigma_L}{\sigma_I} + 1}\cdot\sqrt{\lambda^2\left(\frac{\sigma_L}{\sigma_{I'}}\right)^2 + 2\lambda\beta\frac{\sigma_L}{\sigma_{I'}} + 1}}.$$

Since $\eta = \frac{\sigma_L}{\sigma_I}$ and $\eta' = \frac{\sigma_{L'}}{\sigma_{I'}} = \frac{\lambda\sigma_L}{\sigma_{I'}}$, it follows that

$$\rho = \frac{\eta\eta' + \alpha\eta' + \beta\eta + \gamma}{\sqrt{\eta^2 + 2\alpha\eta + 1}\cdot\sqrt{(\eta')^2 + 2\beta\eta' + 1}}.$$

This completes the proof of the lemma. □

We have several interesting observations by Lemma 2. First, we have the following *lower bounds* for ρ that only depend on η and η' (Theorem 3 and Corollary 4).

Theorem 3: Define a function

$$f(\eta,\eta') = \frac{\eta\eta' - \eta' - \eta - 1}{(\eta+1)(\eta'+1)} = \frac{1 - \frac{1}{\eta} - \frac{1}{\eta'} - \frac{1}{\eta\eta'}}{(1+1/\eta)(1+1/\eta')}.$$

For any $0 \le \eta, \eta' < \infty$, we have $\rho \ge f(\eta,\eta')$.

Proof: We have $-1 \le \alpha, \beta, \gamma \le 1$. By Equation 7,

$$\eta\eta' + \alpha\eta' + \beta\eta + \gamma \ge \eta\eta' - \eta' - \eta - 1,$$
$$\sqrt{\eta^2 + 2\alpha\eta + 1} \le \sqrt{\eta^2 + 2\eta + 1} = \eta + 1,$$
$$\sqrt{(\eta')^2 + 2\beta\eta' + 1} \le \sqrt{(\eta')^2 + 2\eta' + 1} = \eta' + 1.$$

As a result, it follows that $\rho \ge \frac{\eta\eta'-\eta'-\eta-1}{(\eta+1)(\eta'+1)} = f(\eta,\eta')$. This completes the proof the theorem. □

Algorithm 2: Selection of the pivot operator.

```
Input: $\mathcal{O}$, the set of backbone operators.
Output: $o^{\text{pivot}}$, the pivot operator.
1 $o^{\text{pivot}} \leftarrow$ nil; $\lambda \leftarrow 0$;
2 foreach $o \in \mathcal{O}$ do
3     $\lambda_o \leftarrow \frac{\text{opt-cost}(o)}{\text{act-cost}(o)}$;
4     if $\lambda_o > \lambda$ then
5         $\lambda \leftarrow \lambda_o$;
6         $o^{\text{pivot}} \leftarrow o$;
7     end
8 end
9 return $o^{\text{pivot}}$;
```

Corollary 4: Define a function

$$g(\eta, \eta') = \frac{\eta}{\eta+1} \cdot \frac{\eta'}{\eta'+1} = \frac{1}{1+1/\eta} \cdot \frac{1}{1+1/\eta'}.$$

If $0 \leq \alpha, \beta, \gamma \leq 1$, then $\rho \geq g(\eta, \eta')$.

The proof is very similar to that of Theorem 3 and thus omitted. Clearly, $g(\eta, \eta') > f(\eta, \eta')$. Intuitively, positive α, β, and γ suggest positive correlations between L, I, and I', which is the typical case in real workloads (see Section 4.1). Based on Theorem 3 and Corollary 4, we immediately have the following important observation:

Observation 5: If $\eta \gg 1$ and $\eta' \gg 1$, we have $f(\eta, \eta') \approx 1$ and $g(\eta, \eta') \approx 1$. As a result, $\rho \approx 1$.

That is, *when both η and η' are sufficiently large*, we should expect very strong correlation between the estimated cost (using Algorithm 1) and the actual cost of a plan. More generally, both $f(\eta, \eta')$ and $g(\eta, \eta')$ are increasing functions with respect to η and η'. This implies that we need to *increase both η and η' to improve ρ.*

2.3.3 Maximizing Correlation with Pivot Operator

Recall that $\eta = \frac{\sigma_L}{\sigma_I}$ whereas $\eta' = \frac{\lambda \sigma_L}{\sigma_{I'}}$. σ_L and σ_I are the standard deviations of the *actual* leaf and internal operator CPU time, whereas $\sigma_{I'}$ is the standard deviation of the query optimizer's estimated internal cost. All three are constants for a given workload, so we cannot change η. On the other hand, η' depends on λ as well, *which depends on our choice of the pivot operator*. Because η' increases as λ increases, it suggests that we should pick the pivot operator that *maximizes* λ, as defined by Equation 4. Algorithm 2 presents the details of our selection strategy for the pivot operator based on this observation.

3 Theoretical Analysis

In this section, we present an in-depth analysis of hybrid cost modeling to understand its efficacy. We focus our analysis on the impact of η and η' over the correlation coefficient ρ between estimated cost with hybrid modeling and true execution cost. By Observation 5, if both η and $\eta's$ are very large, then ρ is close to 1 as well, which would be the best case for hybrid modeling. Nonetheless, different workloads have different values of η and η'. Therefore, it is natural to ask the question of what we should expect in practice. Based on our empirical study on real workloads (see Section 4.1), we observe that η' is typically much larger than η in reality. As Figure 9 shows, the median value of η' observed on the real workloads is 6.8×10^3, whereas the median value of η is 18.8. This motivates us to organize our analysis into two parts: (1) the case when η' is very large (Section 3.1), which is typical in practice; and (2) the case when η' is not large (Section 3.2), which still happens in practice and is more challenging to analyze.

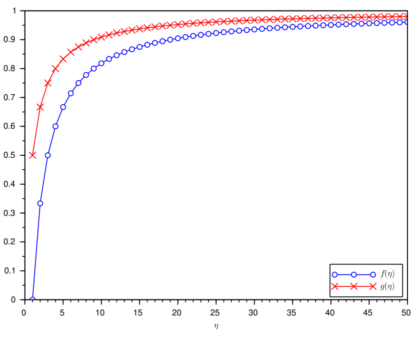


Fig. 4. Plots of $f(\eta)$ and $g(\eta)$ with the growth of η.

3.1 The Case When η' Is Very Large

We can assume that $\eta' \gg 1$ and therefore $\frac{1}{\eta'} \approx 0$.

3.1.1 Impact on Lower Bounds

We start by revisiting the lower bounds $f(\eta, \eta')$ in Theorem 3 and $g(\eta, \eta')$ in Corollary 4 of the correlation coefficient ρ. We have the following approximations:

Observation 6: If $\eta' \gg 1$, we have $1/\eta' \approx 0$. As a result, $f(\eta, \eta') \approx \frac{1-1/\eta}{1+1/\eta}$ and $g(\eta, \eta') \approx \frac{1}{1+1/\eta}$.

Define $f(\eta) = \frac{1-1/\eta}{1+1/\eta}$ and $g(\eta) = \frac{1}{1+1/\eta}$. Figure 4 depicts these two functions with η increasing from 1 to 50. We observe that both functions increase quickly when η grows. For example, when $\eta = 10$, $f(\eta) = 0.81$ and $g(\eta) = 0.91$. When $\eta = 18.8$ (i.e., the median we observed on our workloads), we have $f(\eta) = 0.90$ and $g(\eta) = 0.95$.

3.1.2 Impact of η on Correlation Coefficient ρ

We next present a more detailed analysis regarding the impact of η on ρ, which makes one step further than the analysis in Section 3.1.1 that targets the lower bounds of ρ. Dividing the numerator and denominator in Equation 7 by η' and using $\frac{1}{\eta'} \approx 0$, we obtain the following:

$$\rho \approx \frac{\eta + \alpha}{\sqrt{\eta^2 + 2\alpha\eta + 1}} = \frac{1 + \alpha/\eta}{\sqrt{1 + 2\alpha/\eta + (1/\eta)^2}}. \quad (8)$$

Again, if η is sufficiently large, then $1/\eta \approx 0$ and thus $\rho \approx 1$. We next view ρ as a function of η and α.

Lemma 7: Assume that Equation 8 holds and $\eta + \alpha > 0$. For a given $0 < \epsilon < 1$, there exists some η_0 s.t. if $\eta > \eta_0$ then $\rho > 1 - \epsilon$. Specifically,

$$\eta_0 = \sqrt{\frac{1 - \alpha^2}{1/(1-\epsilon)^2 - 1}} - \alpha. \quad (9)$$

Proof: Using Equation 8, $\rho > 1 - \epsilon$ implies

$$\eta + \alpha > (1 - \epsilon) \cdot \sqrt{\eta^2 + 2\alpha\eta + 1}.$$

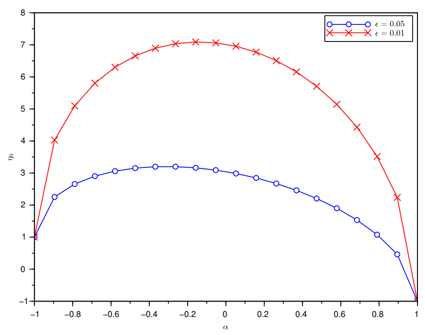


Fig. 5. Plots of η_0 as a function of α.

Given that $\eta + \alpha > 0$, it follows that

$$(\eta+\alpha)^2 > (1-\epsilon)^2 \cdot \left((\eta+\alpha)^2 + (1-\alpha^2)\right).$$

Since $0 < 1-\epsilon < 1$, we have $1/(1-\epsilon)^2 > 1$. As a result,

$$(\eta+\alpha)^2 > \frac{1-\alpha^2}{1/(1-\epsilon)^2 - 1}.$$

Since $\eta + \alpha > 0$, taking the square root of both sides of the above inequality completes the proof. □

Lemma 7 suggests that there is a minimum η_0 such that ρ can be sufficiently high as long as $\eta > \eta_0$. We present two concrete examples below:

- Set $\epsilon = 0.05$, i.e., we want to have $\rho > 1-\epsilon = 0.95$. As a result, $\eta_0 = \sqrt{(1-\alpha^2)/0.108} - \alpha$.
- Set $\epsilon = 0.01$, i.e., we want to have $\rho > 1-\epsilon = 0.99$. As a result, $\eta_0 = \sqrt{(1-\alpha^2)/0.0203} - \alpha$.

Figure 5 plots the η_0 as a function of $-1 \le \alpha \le 1$ in the above two examples. We observe that η_0 has a maximum $\eta_0^{\max}$ within $-1 \le \alpha \le 1$. As long as $\eta > \eta_0^{\max}$, we have $\rho > 1-\epsilon$ regardless of α. In fact, this is easy to prove using Equation 9. Specifically we have the following theorem.

Theorem 8: η_0 achieves its maximum $\eta_0^{\max}$ when $\alpha = -\sqrt{1-(1-\epsilon)^2}$. In more detail, we have

$$\eta_0^{\max} = \frac{1}{\sqrt{1-(1-\epsilon)^2}}. \quad (10)$$

Proof: We can view η_0 as a function of α, i.e., $\eta_0 = \eta_0(\alpha)$. Define a constant $C = \frac{1}{\sqrt{1/(1-\epsilon)^2-1}}$. By Equation 9, $\eta_0(\alpha) = C\sqrt{1-\alpha^2} - \alpha$. Taking derivatives of $\eta_0(\alpha)$,

$$\eta_0'(\alpha) = -\frac{C\alpha}{\sqrt{1-\alpha^2}} - 1 \text{ and } \eta_0''(\alpha) = -\frac{C}{(1-\alpha^2)^{3/2}}.$$

Since $C > 0$ and $|\alpha| \le 1$, we have $\eta_0''(\alpha) < 0$. Therefore, $\eta_0(\alpha)$ is a *concave* function that achieves its maximum when $\eta_0'(\alpha) = 0$. Setting $\eta_0'(\alpha) = 0$ yields

$$\alpha = -\frac{1}{\sqrt{C^2+1}} = -\sqrt{1-(1-\epsilon)^2}. \quad (11)$$

Substituting Equation 11 into Equation 9 gives

$$\eta_0^{\max} = \sqrt{1-(1-\epsilon)^2} + \frac{(1-\epsilon)^2}{\sqrt{1-(1-\epsilon)^2}} = \frac{1}{\sqrt{1-(1-\epsilon)^2}}.$$

This completes the proof of the theorem. □

Continuing with the previous two concrete examples, by Theorem 8 we have

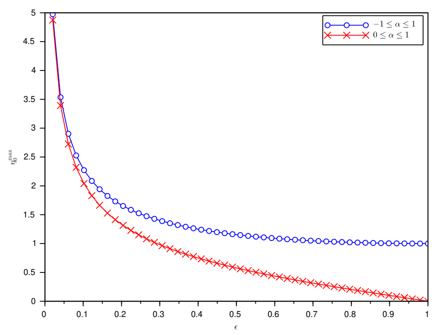


Fig. 6. Plots of $\eta_0^{\max}$ as a function of ϵ.

- For $\epsilon = 0.05$, $\eta_0^{\max} = 3.2$ when $\alpha = -0.31$;
- For $\epsilon = 0.01$, $\eta_0^{\max} = 7.1$ when $\alpha = -0.14$.

These results can be easily verified in Figure 5. Moreover, by Equation 11, we have $\alpha \to 0$ as $\epsilon \to 0$. Meanwhile, $\eta_0^{\max}$ increases as ϵ decreases. In particular, $\eta_0^{\max} \to \infty$ as $\epsilon \to 0$. Figure 6 further plots $\eta_0^{\max}$ with respect to ϵ.

3.1.2.1 The Case of Positive α: So far we have focused on the general case where $-1 \le \alpha \le 1$. In practice, it is reasonable to assume a positive α, i.e., $0 \le \alpha \le 1$. For the workloads that we studied in Section 4.1, we observed only one workload with a negative $\alpha = -0.09$. Therefore, similar to Corollary 4, we can further improve the result given by Theorem 8 for the case when $0 \le \alpha \le 1$.

Corollary 9: If $0 \le \alpha \le 1$, η_0 achieves its maximum $\eta_0^{\max,p}$ when $\alpha = 0$ (the superscript p means a *positive* α):

$$\eta_0^{\max,p} = \frac{1-\epsilon}{\sqrt{1-(1-\epsilon)^2}}. \quad (12)$$

Proof: By the proof of Theorem 8, we have

$$\eta_0'(\alpha) = -\frac{C\alpha}{\sqrt{1-\alpha^2}} - 1, \quad \text{where } C > 0.$$

If $\alpha \ge 0$, we have $\eta_0'(\alpha) < 0$. Hence, $\eta_0(\alpha)$ is a decreasing function of α. As a result, η_0 achieves its maximum when $\alpha = 0$. Setting $\alpha = 0$ in Equation 9 gives Equation 12. □

Comparing Equation 12 with Equation 10 suggests that $\eta_0^{\max,p} < \eta_0^{\max}$. This means that in the (practically common) case of a positive α, *one only needs a smaller value of η to expect a high ρ*. Figure 6 illustrates this difference. When $\epsilon \to 0$, however, we have $\eta_0^{\max,p} \to \eta_0^{\max}$.

3.1.2.2 Analysis of η When $\eta \le \eta_0$: We now study the case of $\eta \le \eta_0$. By Equation 8, for a given fixed α, we can further view ρ as a function of η, namely, $\rho = \rho(\eta)$. We have the following simple result.

Lemma 10: Assume that Equation 8 holds. For a given $-1 \le \alpha \le 1$, ρ is a *non-decreasing* function of η.

Proof: Taking the derivative for ρ (Equation 8) gives

$$\rho'(\eta) = \frac{1-\alpha^2}{w^3}, \quad \text{where } w = \sqrt{\eta^2 + 2\alpha\eta + 1}.$$

Since $|\alpha| \le 1$ and $w > 0$, $\rho'(\eta) \ge 0$. Therefore ρ is a non-decreasing function of η. □

Theorem 11: If Equation 8 holds, then $\alpha \le \rho \le 1$.

Proof: By Lemma 10, $\rho(\eta)$ is a non-decreasing function of η. Given that $0 \le \eta < \infty$, we have $\rho(0) \le \rho(\eta) \le$

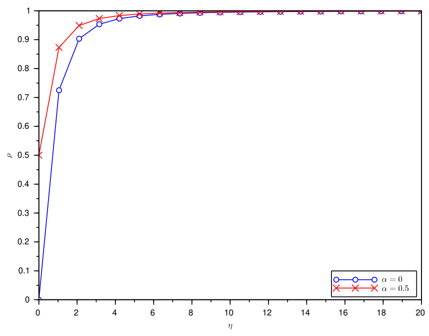


Fig. 7. Plots of ρ as a function of η for a fixed α.

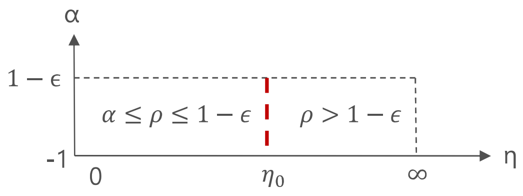


Fig. 8. Summary of the analysis when $\frac{1}{\eta'} \approx 0$.

$\rho(\infty)$. By Equation 8, $\rho(0) = \alpha$ whereas $\rho(\infty) = 1$. This completes the proof of the theorem. □

In particular, when $|\alpha| < 1$, ρ is a strictly increasing function of η. When $\alpha = 1$, $\rho = 1$; when $\alpha = -1$, $\rho = 1$ if $\eta \ge 1$, otherwise $\rho = -1$. Theorem 11 holds in all these cases. Figure 7 further plots the two functions $\rho(\eta) = \frac{\eta}{\sqrt{\eta^2+1}}$ when $\alpha = 0$ and $\rho(\eta) = \frac{\eta+0.5}{\sqrt{\eta^2+\eta+1}}$ when $\alpha = 0.5$. It is clear that $\rho \ge \alpha$ in both cases.

3.1.2.3 Summary: Figure 8 summarizes our analysis. In the presence of a very large η', ρ only depends on η and α. Given a desired threshold $0 < \epsilon < 1$, for a given $-1 \le \alpha \le 1-\epsilon$, along the spectrum $0 \le \eta < \infty$ there exists some η_0 such that $\rho \ge 1-\epsilon$ when $\eta > \eta_0$. On the other hand, if $\eta \le \eta_0$, then a weaker bound for ρ is $\alpha \le \rho \le 1-\epsilon$. Note that the condition $\alpha \le 1-\epsilon$ is necessary for $\eta_0 \ge 0$ (see Equation 9). We have two remarks in order. First, it is straightforward to extend the analysis to the general case of backbone operators (not just leaf operators) versus the rest. Second, so far we have assumed that there are no cost modeling errors for backbone operators, which is unlikely the case in practice. It is straightforward to extend the analysis to incorporate cost modeling errors, though the analytic formulas will become more complicated.

3.2 The Case When η' Is Not Large

So far we have focused ourselves on the case when η' is very large (more precisely, $\frac{1}{\eta'} \approx 0$), which is typical in practice. One may be also interested in the case when this does not hold. In the following, we study this case in more detail. The techniques used in our analysis are similar to those used in Section 3.1, though the analytic results obtained are more complicated.

By Equation 7, we can also view ρ as a function of η':

$$\rho = \rho(\eta') = \frac{A(B\eta' + C)}{\sqrt{(\eta')^2 + 2\beta\eta' + 1}}, \quad (13)$$

where $A = \frac{1}{\sqrt{\eta^2+2\alpha\eta+1}}$, $B = \eta + \alpha$, and $C = \beta\eta + \gamma$. As was in Lemma 7, we assume $\eta + \alpha \ge 0$. Note that this automatically holds if $\eta \ge 1$. Taking the derivative gives

$$\rho'(\eta') = \frac{ABv^2 - u(\eta' + \beta)}{v^3}, \quad (14)$$

where $u = A(B\eta' + C)$, $v = \sqrt{(\eta')^2 + 2\beta\eta' + 1}$. Note that the derivative of v satisfies $v'(\eta') = \frac{\eta'+\beta}{v}$.

Now let $\rho'(\eta') = 0$. We obtain

$$\eta'_0 = \frac{\beta C - B}{\beta B - C} = \frac{1-\beta^2}{\gamma - \alpha\beta}\eta + \frac{\alpha - \beta\gamma}{\gamma - \alpha\beta}. \quad (15)$$

Using the relation $ABv^2|_{\eta'=\eta'_0} = u|_{\eta'=\eta'_0}(\eta'_0 + \beta)$ gives

$$Bv^2|_{\eta'=\eta'_0} = (B\eta'_0 + C)(\eta'_0 + \beta),$$

it then follows that

$$\rho(\eta'_0) = \frac{A(B\eta'_0 + C)}{v|_{\eta'=\eta'_0}} = \sqrt{\frac{(\eta+\alpha)^2 + \frac{(\gamma-\alpha\beta)^2}{1-\beta^2}}{(\eta+\alpha)^2 + (1-\alpha^2)}}. \quad (16)$$

Furthermore, we have

$$\rho''(\eta') = \frac{AB(\beta - C)v^3 - 3v(\eta'+\beta)[ABv^2 - u(\eta'+\beta)]}{v^6}.$$

Again, using the relation

$$ABv^2|_{\eta'=\eta'_0} = u|_{\eta'=\eta'_0}(\eta'_0 + \beta),$$

it follows that

$$\rho''(\eta'_0) = \frac{AB(\beta - C)}{v^3|_{\eta'=\eta'_0}} = \frac{AB[(1-\eta)\beta - \gamma]}{v^3|_{\eta'=\eta'_0}}. \quad (17)$$

Assume $\beta > 0$ where $\beta = \rho(L, I')$ (see Table 3). Therefore, if $\eta > 1 - \frac{\gamma}{\beta}$, we have $\rho''(\eta'_0) < 0$ and thus $\rho(\eta')$ attains its maximum at η'_0. On the other hand, if $\eta < 1 - \frac{\gamma}{\beta}$, $\rho''(\eta'_0) > 0$ and thus $\rho(\eta')$ attains its minimum at η'_0.

Moreover, if $\gamma > \beta$ where $\gamma = \rho(I, I')$ (see Table 3), then $1 - \frac{\gamma}{\beta} < 0$. Thus $\eta > 1 - \frac{\gamma}{\beta}$ always holds and $\rho(\eta')$ attains its maximum at η'_0. On the other hand, if $0 < \gamma \le \beta$, then $0 \le 1 - \frac{\gamma}{\beta} < 1$. If $\eta \ge 1$ then $\rho(\eta')$ still attains its maximum at η'_0. Otherwise we need to compare η and $1 - \frac{\gamma}{\beta}$. Hence, we have proved the following result:

Theorem 12: If $\eta \ge 1$ and $0 < \beta, \gamma < 1$, then $\rho(\eta')$ attains its maximum $\rho_{\max} = \rho(\eta'_0)$ (Equation 16) at η'_0, and $\rho(\eta')$ attains its minimum $\rho_{\min}$ at either

$$\rho(0) = AC = \frac{\beta\eta + \gamma}{\sqrt{\eta^2 + 2\alpha\eta + 1}}$$

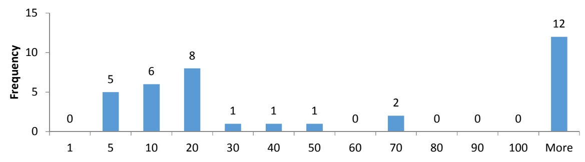
or

$$\rho(\infty) = AB = \frac{\eta + \alpha}{\sqrt{\eta^2 + 2\alpha\eta + 1}}.$$

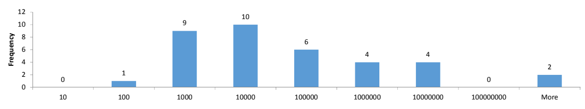
We have $\rho_{\min} \le \rho \le \rho_{\max}$.

4 Experimental Evaluation

We start by an empirical study of the two quantities η and η' that play crucial roles in our correlation analysis (see Section 2.3.2) using real workloads. Following that, we further present empirical evaluation results by applying hybrid cost modeling (Algorithm 1) to index tuning.



(a) Distribution of η: mean = 369.4, median = 18.8.



(b) Distribution of η': mean = 1.2×10^7, median = 6.8×10^3.

Fig. 9. The distributions of η and η' on real workloads.

4.1 Empirical Study of η and η' on Real Workloads

While Section 3 provides an in-depth analysis that characterizes the connection between ρ, η, and η', it remains unclear what we should expect in practice. We thus studied 36 real workloads in the context of index tuning with various physical design (e.g., both row store and column store with necessary indexes) and with at least 10 queries. Figure 9 presents the distributions of η and η' over these workloads. We computed η' by using Algorithm 2 to pick the pivot operator and therefore λ. We observe that η' is much larger than η, which has motivated us to focus our theoretical analysis on the case when η' is very large (Section 3.1). There remains one workload with relatively small η', i.e., $\eta' < 100$ as shown in Figure 9(b), and we have also analyzed this case in Section 3.2.

As was shown in Figure 9, there is huge variance in the distribution of η on the real workloads that we studied. Although 25 out of the 36 workloads have $\eta \geq 10$, there are still 11 workloads with relatively small η. So a natural question is that how large ρ is over these real workloads. In Figure 10, we present the distributions of both Pearson CC and Spearman CC on the 36 real workloads. Spearman CC is the rank-based version of Pearson CC. Compared to Pearson CC, Spearman CC is more robust when there are outliers, but it ignores the relative differences between costs. Compared with the optimizer's cost estimates, the hybrid cost estimates returned by Algorithm 1 improve the correlation coefficients from 0.55 to 0.80 on average.

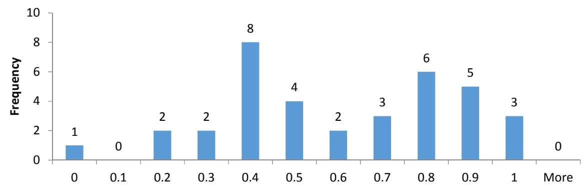
4.2 Evaluation of End-to-end Index Tuning

We now evaluate the efficacy of hybrid cost modeling (Algorithm 1) when applied to end-to-end index tuning.

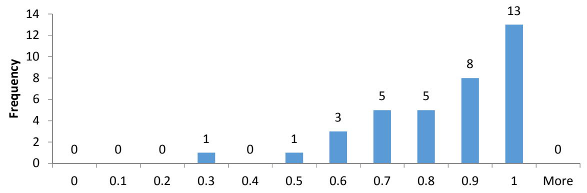
4.2.1 Experimental Settings

The effectiveness of Algorithm 1 relies on the following factors: (1) the backbone operators $\mathcal{O}$; (2) the operator-level cost models $\mathcal{M}$; and (3) the execution feedback $\mathcal{F}$. For (1), as we have discussed in Section 2.2, we use leaf operators as backbone operators; For (2), we use the operator-level modeling approach presented in [28], as it represents the state of the art to the best of our knowledge; For (3), we assume sufficient amount of execution feedback is available for leaf operators (see Sections 4.2.2 and 4.2.3).

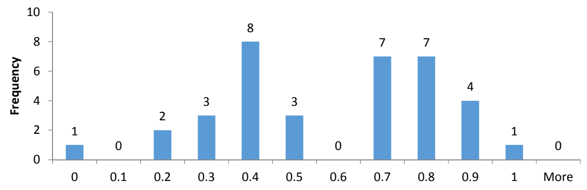
We present some implementation details of the operator-level cost models. For each physical operator that appears in the query plans collected as execution feedback, we train



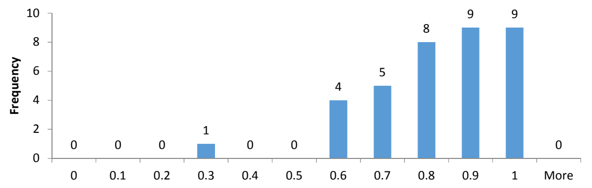
(a) Distribution of Pearson CC using query optimizer's estimates: mean = 0.54, median = 0.56.



(b) Distribution of Pearson CC using hybrid cost modeling in Algorithm 1 (i.e., ρ): mean = 0.81, median = 0.82.



(c) Distribution of Spearman CC using query optimizer's estimates: mean = 0.53, median = 0.62.



(d) Distribution of Spearman CC using hybrid cost modeling in Algorithm 1: mean = 0.78, median = 0.80.

Fig. 10. The distributions of Pearson CC and Spearman CC on real workloads using query optimizer's estimates vs. hybrid cost modeling in Algorithm 1.

Name	Description
C_OUT	Number of output tuples
S_OUT_AVG	Average width of output tuples
S_OUT_TOT	Total number of output bytes
C_IN	Number of input tuples (per child)
S_IN_AVG	Average width of input tuples (per child)
S_IN_TOT	Total number of input bytes (per child)
OUT_USAGE	Type of parent operator
DEG_PARALLEL	Degree of parallelism
EST_CPU_COST	Optimizer estimated CPU cost
EST_IO_COST	Optimizer estimated I/O cost
EST_OP_COST	Optimizer estimated operator cost

TABLE 4
"Global" features that are used by all operators.

an ML model based on the features extracted from each operator. Table 4 summarizes the "global" features that are shared by all operators, whereas Table 5 further summarizes the "local" features that are opreator-specific. We then train a *boosted regression tree* (BRT) model for each operator.

We used both synthetic and real database workloads in our evaluation. For synthetic data, we used both TPC-H and TPC-DS benchmarks with scaling factor of 10; for real data, we used three customer workloads Real-1,

Name	Description	Operator
T_SIZE	Size of input table in tuples	Scan/Seek
PAGES	Size of input table in pages	Scan/Seek
T_COL	Number of columns in a tuple	Scan/Seek
IDX_LVL	Levels of index in access path	Seek
C_S_COL	Number of sort columns	Sort
MIN_COMP	# tuples×# sort columns	Sort
HAGG_AVG	# hashing operations per tuple	Hash Agg.
HAGG_TOT	HASH_AVG×# tuples	Hash Agg.
HJ_AVG_B	# hashing op. per tuple (build)	Hash Join
HJ_TOT_B	HASH_AVG_B×# tuples (build)	Hash Join
HJ_AVG_P	# hashing op. per tuple (probe)	Hash Join
HJ_TOT_P	HASH_AVG_P×# tuples (probe)	Hash Join
C_I_TBL	# tuples in the inner table	Nested Loop
S_IN_SUM	# input bytes from all children	Merge Join

TABLE 5
"Local" features that are operator-specific.

Name	DB Size	#Queries	η	η'
TPC-DS	10GB	99	56.6	3.1×10^4
TPC-H	10GB	22	51.2	4.8×10^3
Real-1	40GB	12	1.7	4.5×10^3
Real-2	60GB	20	429.9	5.5×10^5
Real-3	100GB	40	217.9	2.4×10^6

TABLE 6
Workloads used in end-to-end index tuning evaluation.

Real-2, and Real-3. Table 6 presents the details of the workloads that we used and their characteristics. η' is very large over all of these workloads. We implemented the AutoAdmin index tuner [11] that has demonstrated state-of-the-art performance in recent benchmark study [25] and has been integrated into Microsoft's Database Tuning Advisor [10]. We focused on tuning single-query workloads (i.e., the index tuner was invoked for each query in a given workload), which is common in cloud auto-indexing practice [15], and we conducted experiments using a workstation configured with Intel 2.6GHz CPUs and 192GB main memory. We used Microsoft SQL Server 2017 running on top of Windows Server 2019. It remains future work to investigate the efficacy of hybrid cost modeling on top of other database systems and index tuners (e.g., Dexter [23, 24] for PostgreSQL).

4.2.2 Initial Index Configuration

Since index tuning needs to start from an initial configuration, we generated various initial configurations for our experiments in the following way. For each query q in the workload, we generated different index configurations by limiting the number of indexes recommended by the index tuner (without using execution feedback). Specifically, we keep asking the index tuner to return the next best index until it runs out of recommendations. Suppose that the indexes recommended subsequently are $i_1, ..., i_n$. We then have n configurations $I_1 = \{i_1\}$, $I_2 = \{i_1, i_2\}$, ..., and $I_n = \{i_1, i_2, ..., i_n\}$. We used each of these n configurations as a different initial configuration.

4.2.3 Execution Feedback

We generate execution feedback in the following manner. For each initial configuration, we run the query and collect its execution time. For this purpose, we enable the

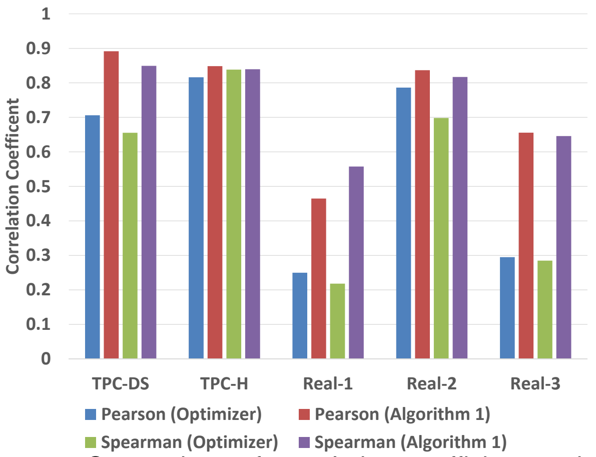


Fig. 11. Comparison of correlation coefficients using hybrid cost estimates from Algorithm 1 over using query optimizer's default cost estimates.

"statistics XML" utility [1] provided by Microsoft SQL Server to track operator-level execution information. We then randomly pick one query plan from each query into the execution feedback repository $\mathcal{F}$.

4.2.4 Performance Metrics

We evaluate both the effectiveness of Algorithm 1 and the overall improvement of index tuning when execution feedback is utilized, with the following metrics:

(Effectiveness of Algorithm 1) We use both the Pearson and Spearman correlation coefficients.

(Overall Improvement) We measure the *relative* improvement of the index configuration I^{new} returned by index tuning over the original index configuration I^{old}, defined as follows. Let $c(q, I)$ and $a(q, I)$ be the estimated and actual execution costs of q over a configuration I, respectively. We define the *estimated improvement* of I^{new} over I^{old} as

$$c(I^{\text{old}}, I^{\text{new}}) = \big(c(q, I^{\text{old}}) - c(q, I^{\text{new}})\big)/c(q, I^{\text{old}}) \\ = 1 - c(q, I^{\text{new}})/c(q, I^{\text{old}}).$$

We also define the *actual improvement* of I^{new} over I^{old}:

$$a(I^{\text{old}}, I^{\text{new}}) = \big(a(q, I^{\text{old}}) - a(q, I^{\text{new}})\big)/a(q, I^{\text{old}}) \\ = 1 - a(q, I^{\text{new}})/a(q, I^{\text{old}}).$$

We use the actual improvement as our metric, whereas the estimated improvement is useful for controlling the recommendation from the index tuner, as we will see.

4.2.5 Evaluation Results

Figure 11 presents the correlation coefficients between estimated costs and actual CPU times. We compare the correlation coefficients using hybrid cost estimates produced by Algorithm 1 against ones using query optimizer's default cost estimates. We observe significant improvement over four of the five workloads. This implies that hybrid cost modeling is considerably better than using optimizer's default cost estimates for all operators in the query plan.

In Figures 12, 13, and 14, we present the distributions of the actual improvement (over all tested configurations) for TPC-DS queries by using execution feedback and hybrid

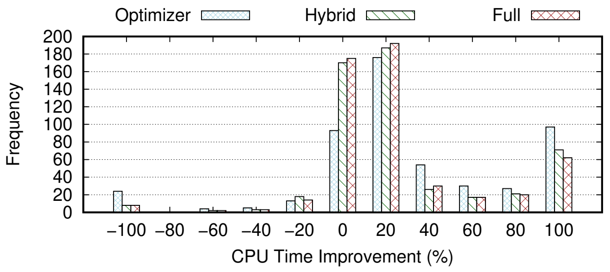


Fig. 12. The distributions of CPU time improvement over TPC-DS queries (improvement threshold $\tau = 0$).

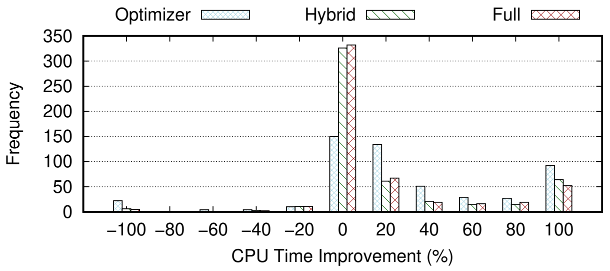


Fig. 13. The distributions of CPU time improvement over TPC-DS queries (improvement threshold $\tau = 0.1$).

cost modeling (denoted as 'Hybrid' in the plots) compared with index tuning without feedback (i.e., by using query optimizer's cost estimates, denoted as 'Optimizer' in the plots). In index tuning, there is usually a threshold τ for *estimated improvement* and an index configuration is recommended only if its estimated improvement is above the threshold. In our experiments, we varied τ from 0 to 0.2 (i.e., 20% estimated improvement). We have the following observations. First, using hybrid cost modeling in index tuning significantly reduces the chance of query performance regression. The number of cases with 20% regression (i.e., -20% actual improvement) is reduced from 24 to 8 (66.7% reduction) when $\tau = 0$, is reduced from 22 to 6 (72.7% reduction) with $\tau = 0.1$, and is reduced from 22 to 4 (81.8% reduction) with $\tau = 0.2$.

Second, when increasing τ, the chance of performance regression decreases for both index tuning with query optimizer's cost estimates and hybrid cost modeling. However, the chance reduces much faster for index tuning with hybrid cost modeling. This implies that, while using hybrid cost modeling can still estimate the performance improvement incorrectly, the estimation error is much smaller compared to using query optimizer's default cost estimates.

Third, by comparing index tuning with query optimizer's cost estimates and hybrid cost modeling, we also observe that actual improvement is diminished in more cases when using hybrid cost modeling—notice that there are more cases in the bin with less than 20% actual improvement. However, cases with more significant improvement ($\geq$40%) are less impacted. In other words, cases falling into the bins with 0% to 40% improvement tend to be moved into the bins with 0% to 20% improvement. Therefore, if performance improvement is indeed significant, index tuning with hybrid cost modeling is unlikely to dismiss it. To shed some light on this phenomenon, in Table 7 we further compare the overall query execution CPU time by using query optimizer's cost estimates and hybrid cost

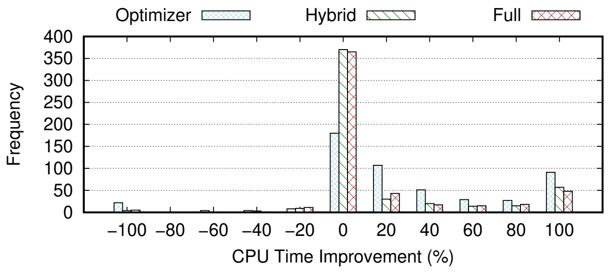


Fig. 14. The distributions of CPU time improvement over TPC-DS queries (improvement threshold $\tau = 0.2$).

modeling. We observe that, compared to the CPU time spent on the initial configuration, which is 320 seconds with $\tau = 0$, hybrid cost modeling brings it down to 28 seconds (i.e., 91.3% improvement) whereas using query optimizer's cost estimates requires 54 seconds instead (i.e., 83%). Comparing the CPU time given by hybrid cost modeling over using query optimizer's cost estimates, i.e., 28 seconds vs. 54 seconds, we see a relative improvement of 48.1%. This reinforces our observation that hybrid cost modeling does not miss significant improvement; moreover, by performing much better in terms of reducing QPR, it can significantly outperform using query optimizer's cost estimates in terms of overall improvement.

We have observed similar results on the other workloads that we tested, though query performance regression is not as significant as we see on the TPC-DS workload.

(Comparison with Full-fledged ML Models) As a demonstration of the motivation of this work, we further compare hybrid cost modeling with using ML models for all operators (instead of just for backbone/leaf operators). Again, Figures 12, 13, and 14 present the distributions of the actual improvement (over all tested configurations) for TPC-DS queries (denoted as 'Full' in the plots). Compared with hybrid cost modeling, using full-fledged ML models performs similarly in terms of the number of regressions avoided. However, it also dismisses more improved cases, especially those significant ones. For example, with $\tau = 0.1$, the number of most significantly improved cases (i.e., with actual improvement between 80% and 100%) is reduced from 64 by using hybrid cost modeling to 52 by using full-fledged ML models. As a result, the improvement on overall CPU time drops from 91.2% to only 36.5% w.r.t. to the initial configuration, as shown in Table 7. With insufficient execution feedback on the internal operators (e.g., joins), the trained ML models are more likely to overfit and thus result in higher generalization errors. Last but not least, one extra benefit of hybrid cost modeling compared to using full-fledged ML models is the significantly reduced training time. For example, for TPC-DS, the time spent on model training 74 seconds for the full-fledged approach vs. 24 seconds for hybrid cost modeling (i.e., a 67.6% reduction).

5 Related Work

(Index Tuning and Query Performance Regression) The problem of autonomous index tuning (a.k.a., auto-indexing) has been studied for decades, e.g., [6, 8, 10, 11, 16, 24, 26, 35, 36, 39, 41, 45, 47, 48, 52]. Cloud database

τ	Time Initial (s)	Time Optimizer (s)	Time Hybrid (s)	Time Full (s)	Impr. Optimizer (%)	Impr. Hybrid (%)	Impr. Full (%)
0	320	54	28	202	83.0	91.3	36.8
0.1	320	52	28	203	83.9	91.2	36.5
0.2	320	52	27	204	83.9	91.5	36.4

TABLE 7
Comparison of overall CPU time improvement of query execution on TPC-DS.

providers recently started offering auto-indexing as a service to customers [15]. The problem of query performance regression becomes a challenge in this context, and there has been work on using ML technologies to address this challenge [17, 37, 54]. The success of these approaches requires collection of substantial amount of training data, which is typically infeasible on cloud database servers running production workloads due to the significant overhead and disruption caused by creating/dropping indexes.

(Query Execution Cost Modeling) In recent years, there has been substantial effort that aims to provide more accurate estimate for query execution cost [4, 5, 19, 20, 21, 28, 32, 34, 38, 43, 49, 50, 53, 54]. Unlike early work that mainly focuses on improving cost estimates inside the optimizer (prominently, via improved cardinality estimates), this line of work constructs cost models by using machine learning (ML) technologies and actual query execution data collected at runtime. The effectiveness of these learned cost models has been demonstrated in various applications such as admission control [44], query scheduling [3], query optimization [31, 51], and index tuning [17, 37, 40].

(Query Execution Feedback) The usage of execution feedback goes beyond the scope of building query execution cost models using ML technologies. Another area that has been extensively explored in the literature is to improve query optimization by using exact cardinality observed in query execution (e.g., [22, 33]), statistics built on top of observed cardinality (e.g. [7]), or sampling (e.g., [27, 51]). While this line of work also improves query plan cost estimates as a by-product (by leveraging more accurate cardinality estimates), its ultimate goal is to impact the decision made by the query optimizer so that it may return a different, perhaps better execution plan. This is different from our goal of using execution feedback in this work to improve index tuning, where we do not want to modify the query optimizer. Rather, the execution feedback is consumed by the index tuner to avoid proposing bad index configurations (and therefore bad query plans generated by the query optimizer using the "what-if" utility).

(Inconsistent Cost Estimates) One noticeable problem when leveraging execution feedback, as documented in the literature [5, 13, 42], is that *partial* execution feedback may result in *inconsistent* cost estimates that mislead the query optimizer. That is, if some plans receive improved cost estimates whereas the others do not, then the plan returned by the optimizer might be even worse. One reason is that, although the query optimizer can estimate costs appropriately for query plans with execution feedback, it may underestimate costs for plans without execution feedback. Again, we avoid this inconsistency problem by not using hybrid cost modeling inside the query optimizer.

6 Conclusion

In this paper, we proposed hybrid cost modeling to address the challenge of limited amount of training data in the context of cloud database auto-indexing, which is a simple yet principled approach that combines ML-based model predictions for execution costs of leaf operators and query optimizer's default cost estimates for other internal operators. We presented both theoretical analysis and experimental evaluation of hybrid cost modeling, using both industrial benchmarks and real workloads. Our theoretical analysis reveals that cost estimates made by hybrid cost modeling can result in much higher correlation with actual query execution time, compared to that of using query optimizer's default cost estimates. Moreover, empirical results demonstrate not only the validity of the theoretical analysis but also the efficacy of applying hybrid cost modeling in index tuning, which significantly reduces the chance of query performance regression while retaining significant query performance improvement at the same time.

Acknowledgement: We thank the anonymous reviewers, Sudipto Das, Bailu Ding, Surajit Chaudhuri, and Vivek Narasayya for their valuable feedback on this work.

References

[1] The "statistics xml" utility of microsoft sql server. https://learn.microsoft.com/en-us/sql/t-sql/statements/set-statistics-xml-transact-sql?view=sql-server-ver16, 2023.
[2] Cost constants used by postgresql's query planner/optimizer. https://www.postgresql.org/docs/current/runtime-config-query.html, 2024.
[3] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report-generation workloads. *The VLDB Journal*, 20:589–615, 2011.
[4] M. Ahmad, S. Duan, A. Aboulnaga, and S. Babu. Predicting completion times of batch query workloads using interaction-aware models and simulation. In *EDBT*, 2011.
[5] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401, 2012.
[6] M. Brucato, T. Siddiqui, W. Wu, V. R. Narasayya, and S. Chaudhuri. Wred: Workload reduction for scalable index tuning. *Proc. ACM Manag. Data*, 2(1):50:1–50:26, 2024.
[7] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *SIGMOD*, 2002.
[8] N. Bruno and S. Chaudhuri. Automatic physical database tuning: A relaxation-based approach. In *SIGMOD*, pages 227–238, 2005.
[9] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *SIGMOD*, pages 211–222, 2001.
[10] S. Chaudhuri and V. Narasayya. Anytime algorithm of database tuning advisor for microsoft sql server, June 2020.
[11] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *VLDB*, pages 146–155, 1997.
[12] S. Chaudhuri and V. R. Narasayya. Autoadmin 'what-if' index analysis utility. In *SIGMOD*, pages 367–378, 1998.

[13] S. Chaudhuri, V. R. Narasayya, and R. Ramamurthy. A pay-as-you-go framework for query execution feedback. *PVLDB*, 1(1):1141–1152, 2008.

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

[15] 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.

[16] D. Dash, N. Polyzotis, and A. Ailamaki. Cophy: A scalable, portable, and interactive index advisor for large workloads. *Proc. VLDB Endow.*, 4(6):362–372, 2011.

[17] 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.

[18] W. Du, R. Krishnamurthy, and M.-C. Shan. Query optimization in a heterogeneous dbms. In *VLDB*, 1992.

[19] J. Duggan, U. Çetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In *SIGMOD*, 2011.

[20] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, 2009.

[21] B. Hilprecht and C. Binnig. Zero-shot cost models for out-of-the-box learned cost prediction. *Proc. VLDB Endow.*, 15(11):2361–2374, 2022.

[22] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD*, pages 106–117, 1998.

[23] A. Kane. The automatic indexer for postgres. https://github.com/ankane/dexter, June 2017.

[24] A. Kane. Introducing dexter, the automatic indexer for postgres. https://medium.com/@ankane/introducing-dexter-the-automatic-indexer-for-postgres-5f8fa8b28f27, June 2017.

[25] 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. *Proc. VLDB Endow.*, 13(11):2382–2395, 2020.

[26] J. Kossmann, A. Kastius, and R. Schlosser. SWIRL: selection of workload-aware indexes using reinforcement learning. In *EDBT*, pages 2:155–2:168, 2022.

[27] P. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD*, pages 175–186, 2007.

[28] J. Li, A. C. König, V. R. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for sql queries using statistical techniques. *PVLDB*, 5(11):1555–1566, 2012.

[29] L. Ma, B. Ding, S. Das, and A. Swaminathan. Active learning for ml enhanced database systems. In *SIGMOD*, pages 175–191, 2020.

[30] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *VLDB*, pages 149–159, 1986.

[31] R. C. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11), 2019.

[32] R. C. Marcus and O. Papaemmanouil. Plan-structured deep neural network models for query performance prediction. *Proc. VLDB Endow.*, 12(11):1733–1746, 2019.

[33] V. Markl, V. Raman, D. E. Simmen, G. M. Lohman, and H. Pirahesh. Robust query processing through progressive optimization. In *SIGMOD*, pages 659–670, 2004.

[34] D. Paul, J. Cao, F. Li, and V. Srikumar. Database workload characterization with query plan encoders. *Proc. VLDB Endow.*, 15(4):923–935, 2021.

[35] R. M. Perera, B. Oetomo, B. I. P. Rubinstein, and R. Borovica-Gajic. DBA bandits: Self-driving index tuning under ad-hoc, analytical workloads with safety guarantees. In *ICDE*, pages 600–611. IEEE, 2021.

[36] R. Schlosser, J. Kossmann, and M. Boissier. Efficient scalable multi-attribute index selection using recursive strategies. In *ICDE*, pages 1238–1249, 2019.

[37] J. Shi, G. Cong, and X. Li. Learned index benefits: Machine learning based index performance estimation. *Proc. VLDB Endow.*, 15(13):3950–3962, 2022.

[38] T. Siddiqui, A. Jindal, S. Qiao, H. Patel, and W. Le. Cost models for big data query processing: Learning, retrofitting, and our findings. In *SIGMOD*, pages 99–113. ACM, 2020.

[39] 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 *SIGMOD*, pages 660–673, 2022.

[40] T. Siddiqui and W. Wu. Ml-powered index tuning: An overview of recent progress and open challenges. *SIGMOD Rec.*, 52(4):19–30, 2023.

[41] T. Siddiqui, W. Wu, V. R. Narasayya, and S. Chaudhuri. DISTILL: low-overhead data-driven techniques for filtering and costing indexes for scalable index tuning. *Proc. VLDB Endow.*, 15(10):2019–2031, 2022.

[42] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. ISOMER: consistent histogram construction using query feedback. In *ICDE*, page 39, 2006.

[43] J. Sun and G. Li. An end-to-end learning-based cost estimator. *Proc. VLDB Endow.*, 13(3):307–319, 2019.

[44] S. Tozer, T. Brecht, and A. Aboulnaga. Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *ICDE*, 2010.

[45] G. Valentin, M. Zuliani, D. C. Zilio, G. M. Lohman, and A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *ICDE*, pages 101–110, 2000.

[46] F. Ventura, Z. Kaoudi, J. Quiané-Ruiz, and V. Markl. Expand your training limits! generating training data for ml-based data management. In *SIGMOD*, pages 1865–1878, 2021.

[47] X. Wang, W. Wu, C. Wang, V. R. Narasayya, and S. Chaudhuri. Wii: Dynamic budget reallocation in index tuning. *Proc. ACM Manag. Data*, 2(3):182, 2024.

[48] K. Whang. Index selection in relational databases. In *Foundations of Data Organization*, pages 487–500, 1985.

[49] W. Wu, Y. Chi, H. Hacigümüs, and J. F. Naughton. Towards predicting query execution time for concurrent and dynamic database workloads. *PVLDB*, 6(10):925–936, 2013.

[50] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *ICDE*, 2013.

[51] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In *SIGMOD*, pages 1721–1736, 2016.

[52] W. Wu, C. Wang, T. Siddiqui, J. Wang, V. Narasayya, S. Chaudhuri, and P. A. Bernstein. Budget-aware index tuning with reinforcement learning. In *SIGMOD*, 2022.

[53] W. Wu, X. Wu, H. Hacigümüs, and J. F. Naughton. Uncertainty aware query execution time prediction. *PVLDB*, 7(14):1857–1868, 2014.

[54] Y. Zhao, G. Cong, J. Shi, and C. Miao. Queryformer: A tree transformer model for query plan representation. *Proc. VLDB Endow.*, 15(8):1658–1670, 2022.

Wentao Wu received the B.S. and M.S. degrees in computer science from Fudan University in 2007 and 2010, respectively, and the Ph.D. degree from the University of Wisconsin-Madison in 2015. He is currently a principal researcher with the Data Systems group, Microsoft Research, Redmond. His research interest includes database management systems, machine learning systems, big data processing and analytics, data mining, and etc.