Learning-based Query Performance Modeling and Prediction

Mert Akdere, Uğur Çetintemel, Matteo Riondato, Eli Upfal, Stanley B. Zdonik

Brown University, Providence, RI, USA
{makdere,ugur,matteo,eli,sbz}@cs.brown.edu

***Abstract*— Accurate query performance prediction (QPP) is central to effective resource management, query optimization and query scheduling. Analytical cost models, used in current generation of query optimizers, have been successful in comparing the costs of alternative query plans, but they are poor predictors of execution latency. As a more promising approach to QPP, this paper studies the practicality and utility of sophisticated learning-based models, which have recently been applied to a variety of predictive tasks with great success, in both static (i.e., fixed) and dynamic query workloads.**

We propose and evaluate predictive modeling techniques that learn query execution behavior at different granularities, ranging from coarse-grained plan-level models to fine-grained operator-level models. We demonstrate that these two extremes offer a tradeoff between high accuracy for static workload queries and generality to unforeseen queries in dynamic workloads, respectively, and introduce a hybrid approach that combines their respective strengths by selectively composing them in the process of QPP. We discuss how we can use a training workload to (i) pre-build and materialize such models offline, so that they are readily available for future predictions, and (ii) build new models online as new predictions are needed. All prediction models are built using only static features (available prior to query execution) and the performance values obtained from the offline execution of the training workload.

We fully implemented all these techniques and extensions on top of PostgreSQL and evaluated them experimentally by quantifying their effectiveness over analytical workloads, represented by well-established TPC-H data and queries. The results provide quantitative evidence that learning-based modeling for QPP is both feasible and effective for both static and dynamic workload scenarios.

I. Introduction

Modern database systems can greatly benefit from query performance prediction (QPP), i.e., predicting the execution latency of a query plan on a given hardware and system configuration. For example, resource managers can utilize QPP to perform workload allocation such that interactive behavior is achieved or specific QoS targets are met. Optimizers can choose among alternative plans based-on expected execution latency instead of total work incurred.

Accurate QPP is important but also challenging: database systems are becoming increasingly complex, with several database and operating system components interacting in sophisticated and often unexpected ways. The heterogeneity of the underlying hardware platforms adds to this complexity by making it more difficult to quantify the CPU and I/O costs. Analytical cost models predominantly used by the current generation of query optimizers cannot capture these interactions and complexity; in fact, they are not designed to do so. While they do a good job of comparing the costs of alternative query plans, they are poor predictors of plan execution latency. Recent work [1] showed this result for TPC-DS [17], and this paper does same for TPC-H [8] data and queries.

In this paper, we utilize learning-based modeling and prediction techniques to tackle QPP for analytical workloads. Data-driven, learning-based modeling is fast emerging as an essential ingredient of both user-facing applications, such as predictive analytics, and system-facing applications, such as autonomic computing and self-management. Prior work reported evidence that such techniques can also be used effectively for QPP, at least in constrained settings (e.g., in static query workloads [1]). Our study substantially improves and generalizes these results in a number of new directions, arguing that learning-based techniques tailored to database query execution are generally applicable to and can be highly effective for QPP.

One of our key contributions is to show that queries can be modeled at different granularities, each offering different tradeoffs involving predictive accuracy and generality. If a representative workload is available for training purposes, we can make highly accurate predictions using coarse-grained, plan-level models [1]. Such models, however, do not generalize well, performing poorly for unseen or changing workloads. For these cases, fine-grained, operator-level modeling performs much better due to its ability to capture the behavior of arbitrary plans, although they do not perform as well as plan-level models for fixed workloads. We then propose a hybrid approach that selectively composes plan- and operator-level models to achieve high accuracy without sacrificing generality.

All these modeling techniques require a training query workload to be executed, so that appropriate feature and performance values are extracted and logged. Models can then be built (i.e., trained) over these logs in offline mode, online mode, or in conjunction. The main advantage of pre-building and materialization is that the models are immediately ready for use in predictions whenever needed. The challenge, however, is to decide which models to pre-build, since it is clearly not feasible to build all possible models in advance. To guide this decision, we propose heuristics that rely on estimates for additional accuracy yields and use frequencies. The online approach, on the other hand, allows for a custom (and potentially more accurate) model to be built for a specific

prediction task, but delays the prediction until an appropriate model is built. Note that online building proceeds over the already available feature data, and does not require new sample query runs. Finally, online and offline modeling can be seamlessly combined, with the decision of which online models to create influenced by the pre-built models. We note that these techniques require only static features (i.e., compile-time features which are available prior to query execution) for performance prediction.

Finally, we describe how all these techniques can be used in combination to provide progressively improved predictions. When a new QPP is needed, we can immediately use the pre-built models to come up with an initial prediction, which we can then continue to improve over time by building better models online optionally with run-time features.

While we study the utility of learning-based models for query execution latency as the performance metric of interest, the proposed techniques are general, and thus can be used in the prediction of other metrics such as throughput. We should also note that this paper does not consider QPP in the presence of concurrent execution, which is an important and challenging problem to address, but is outside the scope of this paper.

We fully implemented these techniques and report experimental results that quantify their cost and effectiveness for a variety of usage scenarios on top of PostgreSQL/TPC-H. The results reveal that our novel learning-based modeling techniques can serve as an effective QPP solution for analytical query workloads.

The rest of the paper is organized as follows: we start with background information on data-driven model-based prediction in Section II. In Section III, we first describe our general approach to using statistical learning techniques for QPP. Plan and operator -level performance prediction methods are described in Section III-A and Section III-B, respectively. Next, in Section III-D we introduce the hybrid prediction method. Online modeling techniques which build prediction models at query execution time are discussed in Section IV. We present experimental results using the TPC-H query workload in Section V. We then end the paper with related work and conclusion remarks in Sections VI and VII.

II. Background: Model-based Prediction

We use the term model to refer to any predictive function such as Linear and Multiple Regression, Bayesian Nets, and Support Vector Machines. The main property of a predictive function is learning: to generalize from the given examples in order to produce useful outputs for new inputs.

In the learning approach, an input data set, called the training data, is initially used to tune the parameters of a prediction model. This process is called training (or learning). In general, model training involves selecting (i) a training data set and (ii) the training features (i.e., attributes/variables), a subset of all attributes in the data set, in addition to the learning operation.

In some cases, a domain expert can manually specify the training features. In other cases, this step is trivial as the prediction attribute(s) directly determine the explanatory features, e.g., in auto-regressive models. Alternatively, the training features can be learned automatically via *feature selection*; however, given a set of n attributes, trying the power set is prohibitively expensive if n is not small or training is expensive [2], [3], [4] thereby requiring heuristic solutions.

Most approaches rank the candidate attributes (often based on their correlation to the prediction attribute(s) using metrics such as information gain or correlation coefficients) and use this ranking to guide a heuristic search [4] to identify the most predictive attributes tested over a disjoint test data set. In this paper, we use a similar *Forward Feature Selection* algorithm based on linear correlation coefficients [4]. This algorithm performs a best-first search in the model space. It starts with building models using small number of high-correlation features and iteratively builds more complex and accurate models by using more features. The features are considered based on their correlation ranks with the target/prediction attribute(s).

Once a prediction model is trained, it can then be used for predicting the unknown values of the target attributes given the values of the explanatory attributes. Hypothesis testing and confidence interval estimations are two common techniques for determining the accuracy of predictions [2]. One form of hypothesis testing that is commonly used is K-Fold Cross Validation (K-CV). K-CV divides the training data up into k non-overlapping partitions. One of the partitions is used as validation data while the other $k-1$ partitions are used to train the model and to predict the data in the validation interval. In this study, we use K-CV to estimate the accuracy of our prediction models.

III. Modeling Query Executions

In this study, we describe QPP methods based on data-driven statistical learning models. As is usual in most learning approaches, all of our modeling techniques consist of two main phases: training and testing. The high-level operations involved in these phases are explained in Figure 2.

In the training phase, prediction models are derived from a training data set that contains previously executed queries (i.e., training workload) and the observed performance values (i.e., execution times). In this phase, queries are represented as a set of features with corresponding performance values. The goal in training is to create an accurate and concise operational summary of the mapping between the feature values and the observed performance data points. The prediction models are then used to predict the performance of unforeseen queries in the test phase. In more complex QPP methods, the training and testing phases can be performed continuously for improved accuracy and adaptivity.

Our approach to QPP relies on models that use only static, compile-time features, which allow us to produce predictions before the execution of queries. There are several static information sources, such as the query text and execution plans, from which query features can be extracted prior to execution. In this study, we use features that can be obtained from the

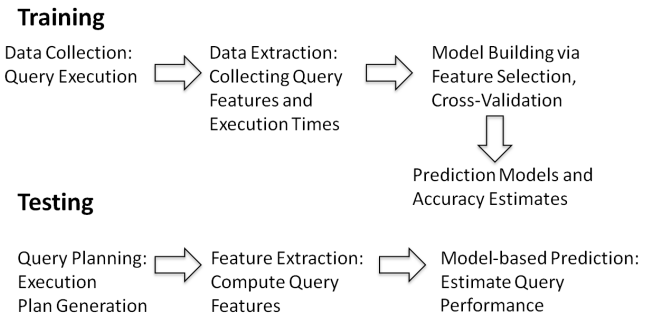


Fig. 1. **Statistical Modeling Approach to QPP.**

information provided by the query optimizer. Many DBMS provide optimizer calls that expose query-plan information and statistical estimates such as the optimized query-plan structure and operator selectivities (for example, `EXPLAIN` in PostgreSQL and `EXPLAIN PLAN` in Oracle).

This paper shows that it is possible to create models at varying granularities for query performance prediction. As in [1], one coarse modeling method is to create a single, plan-level prediction model that utilizes query plan features for modeling the execution times of queries. We discuss this approach in Section III-A. A finer grained approach would be to model each operator type separately and use them collectively through selective composition to model entire query plans. We describe this method in Section III-B and compare the relative advantages and drawbacks of the two approaches in Section III-C. Next, in Section III-D, we introduce a "hybrid" modeling approach that combines the fine and coarse grained modeling methods to form a highly accurate and general QPP approach.

A. Plan-level Modeling

In the plan-level modeling approach, the performance of a query is predicted using a single prediction model. We use the features presented in Table III-A for building plan-level models. This set of features contains query optimizer estimates such as operator cardinalities and plan execution costs together with the occurrence count of each operator type in the query plan.

As mentioned before, we need to address two main challenges when using model-based learning techniques. The first problem, *model selection*, is the process of picking the right prediction model for the given task and data set. In general, it is not possible to identify the most accurate prediction model without training and testing multiple models. In our study, we show results with two types of prediction models for plan-level modeling: a regression variant of Support Vector Machines (SVMs) [5] and Kernel Canonical Correlation Analysis (KCCA) [6], [7]. KCCA models were recently used in [1] for plan-level QPP over the TPC-DS analytical query workload. Both model types provided high accuracy in our experiments. We note that all of the approaches we present here are model-agnostic and can readily work with different model types.

The second problem, *feature selection*, deals with the issue of choosing the most predictive features from the available set

Feature Name	Description
p_tot_cost	Estimated total plan cost
p_st_cost	Estimated plan start cost
p_rows	Estimated number of output tuples
p_width	Estimated average size of an output tuple (in bytes)
op_count	Number of query operators in the plan
row_count	Estimated total number of tuples input and output to/from each operator
byte_count	Estimated total size (in bytes) of all tuples input and output
<operator_name>_cnt	The number of <operator_name> operators in the query
<operator_name>_rows	The total number of tuples output from <operator_name> operators

TABLE I

Features for plan-level models – p_st_cost refers to the cost of query execution until the first output tuple. <operator_name> refers to query operators (e.g., Limit, Materialize and Sort).

of features. Feature selection does not need to be performed for all types of models. For instance, in our case we perform feature selection for SVMs but not for KCCA as it performs dimensionality reduction as part of its operation. However, for many model types feature selection is an important problem. In our experiments, we frequently observed that SVM models using the full set of features given in Table III-A performed less accurately than models with smaller number of features. We use the best-first search based, forward feature selection algorithm [4], described in Section II to perform feature selection. This algorithm starts by building models using a small number of features, and iteratively creates more complex and accurate models by adding features in order of correlation with the target variable (i.e., query execution time).

Once a plan-level prediction model is built and stored (i.e., materialized), it can then be used to estimate the performance of new incoming queries based on the query-plan feature values that can be obtained from the query optimizer without executing the query.

B. Operator-level Modeling

We now introduce a finer-grained operator-level modeling approach. Unlike the plan-level approach, which uses a single prediction model, the operator-level technique relies on a collection of models that are selectively composed for end-to-end query performance prediction. In the operator-level modeling approach, two separate prediction models are built for each query operator type:

- **A start-time model** is used for estimating the time spent during the execution of an operator (and in the sub-query plan rooted at this operator) until it produces its first output tuple. This model captures the (non-)blocking behavior of individual operators and their interaction with pipelined query execution.
- **A run-time model** is used for modeling the total execution time of query operators (and the sub-plans rooted at these operators). Therefore, the run-time estimate of the root operator of a given query plan is the estimated execution time for the corresponding query.

To illustrate the semantics and the use of the start-time model, we consider the *Materialize* operator, which materializes its input tuples either to disk or memory. Assume that in a query tree, the Materialize operator is the inner child operator of a *Nested Loop* join. Although the materialization operation is performed only once, the join operator may scan the materialized relation multiple times. In this case, the start-time of the Materialize operator would correspond to the actual materialization operation, whereas the run-time would represent the total execution time for the materialization and scan operations. In this manner, the parent Nested Loop operator can use the start-time and run-time estimates to form an accurate model of its own execution time. This technique also allows us to transparently and automatically capture the cumulative effects of blocking operations and other operational semantics on the execution time.

We used a single, fixed collection of features to create models for each query operator. The complete list of features is given in Table II. This list includes a generic set of features that are applicable to almost all query operators. They can also be easily acquired from most, if not all, existing DBMSs. As in the case of plan-level modeling approach, we use the forward feature selection algorithm, to build accurate prediction models with the relevant set of features. We used multiple linear regression (MLR) models for the query operators. In addition to performing accurately in our experiments, MLR models are intuitive and easily interpretable similar to analytic cost models.

Feature Name	Description
np	Estimated I/O (in number of pages)
nt	Estimated number of output tuples
nt1	Estimated number of input tuples (from left child operator)
nt2	Estimated number of input tuples (from left right operator)
sel	Estimated operator selectivity
st1	Start-time of left child operator
rt1	Run-time of left child operator
st2	Start-time of right child operator
rt2	Run-time of right child operator

TABLE II

Features for the operator-level models – Start time refers to the time spent in query execution until the first output tuple.

The individual operator models are collectively used to estimate the execution latency of a given query by selectively composing them in a hierarchical manner akin to how optimizers derive query costs from the costs of individual operators. That is, by appropriately connecting the inputs and outputs of prediction models following the structure of query plans, it is possible to produce predictors for arbitrary queries.

In Figure 2, we illustrate this process for a simple query plan consisting of three operators. The performance prediction operation works in a bottom-up manner: each query operator uses its prediction models and feature values to produce its start-time and run-time estimates. The estimates produced by an operator are then fed to the parent operator, which uses them for its own performance prediction.

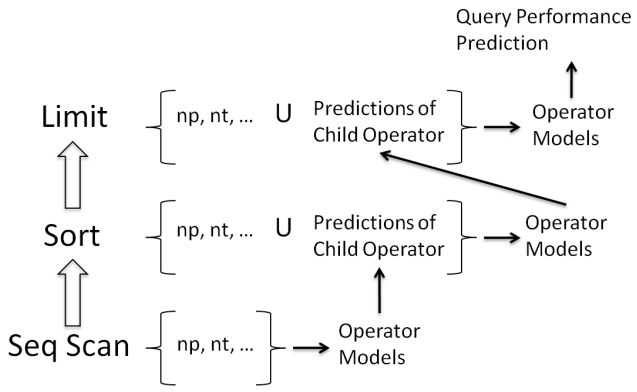


Fig. 2. **Operator-level query performance prediction: operator models use operator-level features together with the predictions of child operators for performance prediction.**

C. Plan vs. Operator -level Modeling

The premise of the plan-level approach is that queries with similar feature vectors will have similar query plans and plan statistics, and therefore are likely to exhibit similar behavior and performance. Such an approach is specifically targeted to *static workload* scenarios where the queries in the training and test phases have similar execution plans (e.g., generated from the same query templates or from the same user program).

Furthermore, this approach is based on the correlation of the query plans and statistics with the query execution times. This correlation is used directly in mapping query-plan based features to execution performance. The high-level modeling approach used in this case therefore offers the ability to capture the cumulative effects of a set of hidden lower level factors, such as operator interactions during query processing, on the query execution times with a single, low complexity model.

The plan-level approach, however, is prone to failure in some common real-world scenarios. A significant problem exists in the case of *dynamic workloads* where queries with unforeseen execution plans are frequently observed. Even worse, there can also be problems with static workloads. As the feature values only represent a limited view of a query plan and its execution, it is possible that different queries can be mapped to very similar feature values and therefore be inaccurately modeled. While it is unlikely for completely different queries to be mapped to identical features, similar queries can sometimes have different execution performance. For instance, increasing the number of time consuming aggregate operations in a query will not significantly change its feature vector, but may highly increase its execution time. Adding more features (e.g., number of aggregates and constraints) to the model would alleviate such issues, however, each added feature would also increase the size of the required training data.

By using multiple prediction models collectively in a hierarchical manner, the operator-level prediction method is able to produce performance predictions for arbitrary queries. Therefore, it is a more general approach compared to the plan-level method and has the potential to be more effective for dynamic query workloads where unforeseen query plan structures are common.

On the downside, the operator-level prediction method may suffer from drawbacks similar to those that affect analytical cost estimation methods (as both methods rely on low-level operator-based models). A key problem is that the prediction errors in the lower levels of a query plan are propagated to the upper levels and may significantly degrade the end prediction accuracy.

Another potential problem is that concurrent use of multiple resources such as CPU and disk may not be correctly reflected in the operator-level (or the analytical) models. For instance, a query could be simply performing an aggregate computation on the rows of a table that it sequentially scans from the disk. If the per-tuple processing takes less time than reading a tuple from the disk, then the query execution time is approximately the same as the sequential scan time. However, if the processing of a tuple takes longer than reading it from the disk, then the execution time will be closer to the processing time. As such, the interactions of the query execution system and the underlying hardware/software platforms can get quite complex. In such cases, simple operator-level modeling approaches may fall short of accurately representing this sophisticated behavior. Therefore, in static query workloads where training and testing queries have similar plan structures we expect the high-level information available in the plan-level approach to result in more accurate predictions.

D. Hybrid Modeling

In hybrid modeling, we combine the operator and plan level techniques to obtain an accurate and generally applicable QPP solution. As discussed, this is a general solution that works for both static and dynamic workloads. We note that as long as the predictive accuracy is acceptable, operator-level modeling is effective. However, for queries with low operator-level prediction accuracy, we learn plan-level models for the inaccurately modeled query sub-plans and compose both types of models to predict the performance of entire query plans. We argue, and later also experimentally demonstrate, that this hybrid solution indeed combines the relative benefits of the operator and plan level approaches by not only retaining the generality of the former but also yielding predictive accuracy values comparable or much better than those of the latter.

Hybrid QPP Example: To illustrate the hybrid method, we consider the performance prediction of an example TPC-H query (generated from TPC-H template-13), whose execution plan is given in Figure 3. This plan is obtained from a 10GB TPC-H database installed on PostgreSQL. As we describe in detail in the Experiments section, we build operator-level models on a training data set consisting of example TPC-H query executions. When we use the operator-level models for performance prediction in this example query, we obtain a prediction error (i.e., $|$*true value - estimate*$|$ */ true value*) of 114%. Upon analysis of the individual prediction errors for each operator in the query plan, we realized that the sub-plan rooted at the *Materialize* operator (highlighted sub-plan in the figure) is the root cause of the prediction errors in the upper level query operators. The operator-level model based prediction error for the materialization sub-plan is 97%.

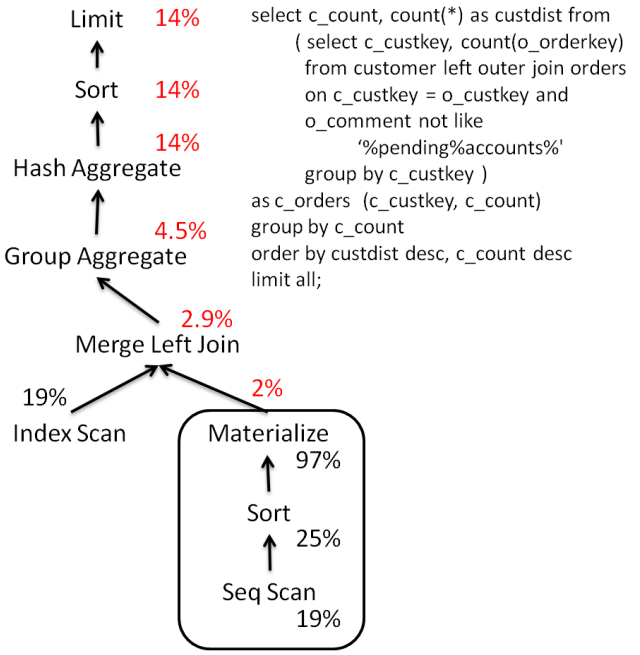


Fig. 3. **Hybrid QPP example: plan-level prediction is used for the highlighted sub-plan together with operator-level prediction for the rest of the operators to produce the end query performance prediction.**

In the hybrid approach, we build a separate plan-level model for the highlighted sub-plan. The model is trained using the occurrences of the highlighted sub-plan in the training data. The hybrid method uses the plan-level model to directly predict the execution performance of the materialization sub-plan, while the rest of the prediction operations is unchanged, i.e., performed with the operator-level models. The prediction errors obtained with the hybrid approach are shown with the red values in the figure. The new overall prediction error for this example query drops down to 14%.

Given a training data set consisting of example query executions, the goal of the hybrid method is to accurately model the performance of all queries in the data set using operator-level models together with a minimal number of plan-level models. In this way, we maximize the applicability of the operator-level models in QPP and maintain high prediction accuracy with the integration of plan-level models.

The hybrid performance prediction method is described in Algorithm 1. The algorithm starts by building prediction models for each query operator based on the provided training data. The accuracy of operator-level prediction is then estimated by application on the training data (e.g., either through cross-validation or holdout test data). Next, the algorithm tries to increase the performance prediction accuracy by building plan-level models.

Each plan-level model is used for directly modeling the performance of a separate query plan (or sub-plan). In a query

plan with N operators, there is a maximum of $N-1$ sub-plans (e.g., in a chain of operators) for plan-level modeling. Then a training data set with M queries can have $O(MN)$ candidate sub-plans for modeling.

In theory, we could build and test plan-level models for each distinct sub-plan (with at least a minimum number of occurrences in the training data set) and try to find a minimal subset of these models for which the prediction accuracy is sufficiently high. However, this would require a large amount of time since (i) we need to build and test models for all candidate sub-plans, and (ii) the prediction accuracy of each subset of models (in increasing sizes) needs to be separately estimated with testing.

Instead, we propose heuristics that iteratively build a collection of plan-level models to maximize the expected predictive accuracy. In each iteration, a new plan-level model is built, tested and added to the model set, if it improves the overall prediction accuracy (by more than a threshold value, ϵ). The models are chosen, built and tested according to *plan ordering strategies*. We consider the following strategies for the hybrid approach:

- **Size-based:** order the plans by size (in increasing *number of operators*).
  The size-based strategy considers generating models for smaller plans before larger ones. This strategy is based on the fact that smaller plans occur more frequently (since by definition all sub-plans of a large plan are at least as frequent) in any data set, and therefore models for smaller plans are more likely to appear in future queries. In case of a tie, the more frequently occurring plan is given priority.
- **Frequency-based:** order the plans in decreasing *occurrence frequency*.
  The frequency-based strategy is similar to the size-based strategy except that it directly uses the occurrence frequency of a plan from the training data for ranking. In case the occurrence count is the same for two plans, smaller plans are considered first. An important difference from the size-based strategy is that when a large plan has a high occurrence frequency, the frequency-based strategy will consider modeling its sub-plans sequentially before considering other plans.
- **Error-based:** order the plans in decreasing value of *occurrence frequency* $\times$ *average prediction error*.
  The error-based strategy considers plans with respect to their total prediction error across all queries in the training data. The assumption is that more accurate modeling of such high error plans will more rapidly reduce the overall prediction error.

In all of the above strategies, the plans for which (i) the average prediction accuracy with the existing models is already above a threshold, or (ii) the occurrence frequency is too low are not considered in model generation.

In order to create the list of candidate plans (i.e., `candidate_plans`) for modeling, we traverse the plans of all queries in the training data in a depth-first manner in function `get_plan_list`. During the traversal, this function builds a hash-based index using keys based on plan tree structures. In this way, all occurrences of a plan structure are hashed to the same value and metrics required by the heuristic strategies such as the occurrence frequency and average prediction error can be easily computed.

Algorithm 1 Hybrid Model Building Algorithm

Input: *data* = example query executions
Input: *strategy* = plan selection strategy
Input: *target_accuracy* = target prediction accuracy
Output: *models* = prediction models
Output: *accuracy* = estimated prediction accuracy

```
1.  models = build_operator_models(data)
2.  [predictions, accuracy] = apply_models(data, models)
3.  candidate_plans = get_plan_list(strategy, data, predictions)
4.  while accuracy ≤ target_accuracy and not stop_cond() do
5.     plan = get_next(strategy, candidate_plans)
6.     plan_model = build_plan_model(data, plan)
7.     [predictions, new_accuracy] = apply_models(data, mod-
       els ∪ plan_model)
8.     if new_accuracy − ϵ ≤ accuracy then
9.        candidate_plans.remove(plan)
10.    else
11.       models = models ∪ plan_model
12.       candidate_plans.update(predictions, plan_model)
13.       accuracy = new_accuracy
```

When a new plan-level model is added to the set of chosen models (i.e., `models`), the candidate plan list is updated with the new prediction errors and occurrence frequencies for all plans. The occurrence frequency of a plan p will change with the addition of a model when the plan for the added model contains p as a sub-plan (since such occurrences of p are consumed by the newly added model).

We can efficiently identify the set of plans for which the prediction errors or the occurrence frequencies might change with the addition of a model as follows: In the hash-based index built by the get_plan_list function, we also store the identifiers for the corresponding queries (which own the plans). As such, when a new model is added, the only plans that need to be updated are the plans that can be applied to one or more of the queries that the newly added plan is also applicable.

Finally, in cases where the target accuracy is unachievable, a maximum number of iterations can be used as a stop condition (i.e., `stop_cond()`) to terminate the algorithm. Other variations for the stopping condition, such as setting a maximum number of iterations without accuracy improvement, are also possible but not evaluated in this study.

IV. Online Model Building

In dynamic query workloads, where queries with unforeseen plan structures are present in the test data, the plan-level QPP method performs poorly due to lack of good training data. On the other hand, operator-level and hybrid methods are designed

to be much more applicable to unforeseen plan structures. In addition, the hybrid method will utilize its plan-level models as much as possible to provide accuracy levels higher than those achievable through pure operator-level modeling.

The prediction accuracy of the hybrid approach in dynamic workload scenarios depends on the applicability of its plan-level models to future queries. As a case study, we analyze execution plans of TPC-H queries on a 10GB TPC-H database running on PostgreSQL. In Figure 4(b), we show the most common sub-plans within the execution plans of queries generated from 14 TPC-H templates for which we could use operator-level prediction techniques in our experiments (see Section V). Our key observations for this data set include:

(1) Smaller sub-plans are more common across the TPC-H query plans (see Figure 4(a)).
(2) The plans for the queries of each TPC-H template (except template-6) share common sub-plans with the plans of queries of at least one other TPC-H template (see Figure 4(c)).

These observations suggest that for the TPC-H workload: (i) it is possible to create plan-level models based on the execution plans for the queries of a TPC-H template and utilize them in the performance prediction of queries from other TPC-H templates, and (ii) the size-based plan ordering strategy discussed in Section III-D will likely achieve higher applicability compared to the other strategies in the dynamic workload case.

However, the hybrid approach may fail to increase the prediction accuracy for dynamic workloads in some cases. For example, the prediction errors for some unforeseen queries may not originate from the common sub-plans, and as a result, plan-level models from the training data cannot reduce the error. In other cases, the common sub-plans could actually be the source of prediction errors, but the plan-ordering strategies may not choose to build plan-level models for them. For instance, some applicable plan-level models may be discarded, because they did not improve the accuracy in training.

To address these issues, in the online modeling technique, we build new plan-level models for QPP at run-time upon the receipt of a query. We initially produce predictions with the set of existing models, and then update our results after new plan-level models are built for the received query.

Online model building is performed similarly to offline model building described for the hybrid method. However, in the online case, the set of candidate plans are generated based on the set of sub-plans of the execution plan for the newly received query. The online building of plan-level models guarantee that if the execution plan for a test query has a common sub-plan (with high prediction error) with the queries in the training data, then a plan-level model will be built and used for its prediction (if a plan-level model with higher accuracy than the operator-level prediction method exists).

V. Experiments

A. Setup

In our experiments, we use the TPC-H decision support benchmark [8] implemented on top of PostgreSQL. The details are presented below.

DBMS. We use an instrumented version of PostgreSQL 8.4.1. The instrumentation code monitored features and performance metrics from query executions; i.e., for each query, the execution plan, the optimizer estimates and the actual values of features as well as the performance metrics were logged.

Data sets and workload. We created 10GB and 1GB TPC-H databases according to the specification. The primary key indices as indicated in the TPC-H specification were created for both databases. We enforced a limit of one hour execution time (per query) to keep the overall experimentation duration under control. This resulted in 18 of the 22 TPC-H templates being used, as the remaining 4 templates always took longer than 1 hour to execute in the 10GB case.

There are approximately 55 queries from each template in both databases. With the 1GB database, all queries finish under an hour and the data set contains 1000 queries. On the other hand, with the 10GB database only 17 of the queries from template-9 finished within an hour, so we have 17 template-9 queries in the 10GB data set. Thus, the resulting 10GB data set we used contains 960 queries.

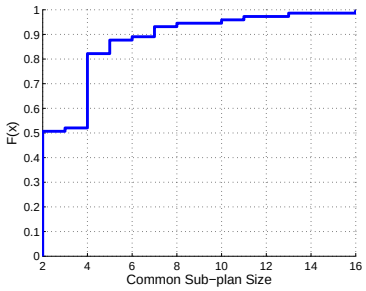
Hardware. Unless stated otherwise, all queries were executed on a single commodity desktop with 4GB RAM running Linux 2.6.28 and the database buffer pool size was set to 1GB (25% of the total RAM as the rule of thumb). All queries were executed sequentially with cold start (i.e., both filesystem and DB buffers were flushed before the start of each query).

Predictive models. For plan-level modeling, we used Support Vector Machines (from libsvm [5]) with the nu-SVR kernel for regression and the Kernel Canonical Correlation Analysis (KCCA) method. We implemented KCCA using GSL, the GNU Scientific Library. For operator-level QPP we used multiple linear regression based models (from Shark [9]). All models were integrated to the database as user defined functions. Our algorithms were implemented as a combination of C-based user-defined functions in PostgreSQL and as external applications written in C++ and Python. The feature selection algorithm, described in Section II, was used to build accurate models using a small number of features.

Metrics and validation. We use the mean relative error as our error metric:

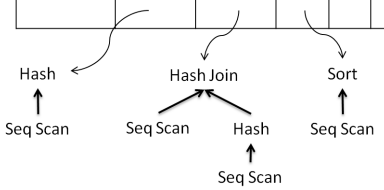
$$\frac{1}{N}\sum_{i=1}^{N}\frac{|actual_i - estimate_i|}{actual_i}$$

This metric is useful when we would like to minimize the relative prediction error in all queries regardless of their execution time. Non-relative error metrics such as the mean-square error would be better for minimizing the absolute difference (or its square) in actual and predicted execution times. Other popular metrics include R^2 and predictive risk [1]. These

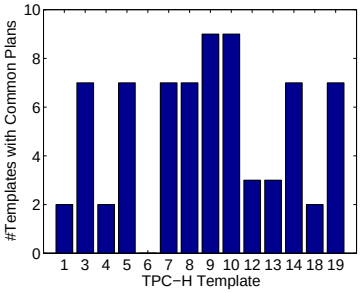


(a) CDF for common sub-plan sizes (#query operators)

Templates	3, 5, 7, 10, 8, 14, 19, 9	3, 5, 10, 8, 14, 19, 9	13, 12, 10, 9	3, 5, 8	3, 5, 8	1, 4, 18
Plan Size	2	4	2	5	7	2



(b) 6 most common sub-plans across queries of 14 TPC-H Templates



(c) #templates the queries of a TPC-H template shares common sub-plans with

Fig. 4. **Analysis of common sub-plans for the execution plans of queries generated from 14 TPC-H Templates.**

metrics measure the performance of the estimates with respect to a point estimate (i.e., the mean). As such, in many cases, they can have deceptively low error values even though the actual estimates have high error, as these metrics depend on the scale and statistical characteristics of the entire data set.

Our results, except for the dynamic workload cases, are based on 5-fold cross validation. That is, the data is divided into 5 equal-sized parts, 4 of which are used to build models for prediction on the remaining part. This process is repeated 5 times, i.e., all parts are used in testing. The reported prediction accuracy is the average of the accuracy values from the testing of each cross-validation part. We used *stratified sampling* for dividing the data into parts to ensure that each part contains roughly equal number of queries from each template.

B. Prediction with Optimizer Cost Models

We start with results showing predictions on top of analytical cost models used by conventional optimizers are non-starters for QPP. Specifically, we built a linear regression model to predict the query execution times based on the query optimizer cost estimates. Overall, the maximum relative error is 1744%, the minimum relative error is 30% and the mean relative error is 120%[1].

To provide more intuition into the reasons, we show the optimizer costs versus the query execution times for a subset of the queries (a stratified sample) on the 10GB TPC-H data set in Figure 5. Observe that the lower left and lower right data points correspond to queries with roughly the same execution times, even though their cost estimates have a magnitude of difference. Similarly, the points on the lower and upper right corners are assigned roughly identical plan costs by the optimizer but differ by two orders of magnitude in their execution times.

In this setup, most queries are I/O intensive. We expect this to be the ideal case for predicting with analytical cost models. The reason is that optimizer cost models generally rely on the assumption that I/O is the most time consuming operation. Therefore, for CPU intensive workloads, we would expect to see even lower accuracy values.

[1]In this case, the predictive risk [1] is about .93, which is close to 1. This result suggests that it performs much better compared to a point estimate, although the actual relative errors per query as we reported are high.

As a concrete example, consider TPC-H template-1, which includes an aggregate over numeric types. We observed that evaluating aggregates over numeric types can easily become the bottleneck, because arithmetic operations are performed in software rather than hardware. As such, introducing additional aggregates to a query can significantly alter the execution time even though the volume of I/O (and hence the predictions with the cost model) remains approximately constant.

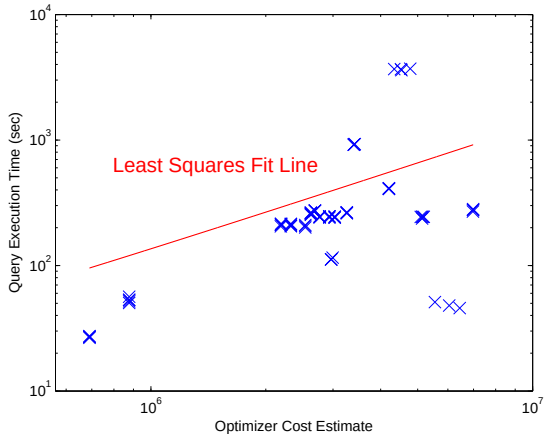


Fig. 5. **Optimizer Cost vs Query Execution Time (log-log plot)**

C. Predicting for Static Workloads

Results for the plan-level and operator-level methods are given in Figure 6 both for the 10GB and 1GB TPC-H scenarios. These results were obtained using estimate-based features both in training and testing. The use of actual (observed) values as features is discussed in Section V-C.3.

1) Plan-level Modeling: Plan-level prediction is performed on all the 18 TPC-H templates. Overall, using SVMs we obtained on average 6.75% and 17.43% prediction errors for the 10GB and 1GB databases, respectively (Figure 6(a)-(c)). The prediction errors with KCCA modeling in the same scenarios were 2.1% and 3.1% (Figure 6(d)-(f)). The high accuracy results imply that plan-level modeling can be very effective for static workloads.

To understand the difference in accuracy between the two model types, here we briefly describe their characteristics.

With SVMs, the general approach is to map the query features to a high dimensional space and perform regression in that space. In KCCA, the query features and the target values are projected to separate subspaces such that their projections are maximally correlated. Prediction with KCCA is then performed using a nearest neighbor strategy. As such the predictions of KCCA use all of the training data, whereas SVM results are only based on a subset of the points (support vectors). In addition, it is important to note that for SVMs we perform a heuristic based search resulting in `hard` feature selection (i.e., choose a fixed subset of features and discard the other features), whereas KCCA as part of its execution, performs `soft` feature selection (i.e., all features are represented in the projected feature subspace).

With SVM-based modeling, queries from template-9 stand out as the worst predicted queries. We note that template-9 queries take much longer than the queries of the other templates. As the number of instances of template 9, and therefore of longer running queries, is relatively few in both data sets, the prediction models may not fit well. To alleviate this problem, we built a separate SVM-based model for template-9 for the 10GB case, which reduced its error to 7%.

2) Operator-level Modeling: We now discuss operator-level prediction results on 14 of the 18 TPC-H templates[2].

For the 10GB case, in 11 of the 14 templates the operator-level prediction method performed better than 20% error (Figure 6(g)). For these 11 templates the average error is 7.30%. The error, however, goes up to 53.92% when we consider all the 14 templates, a significant degradation.

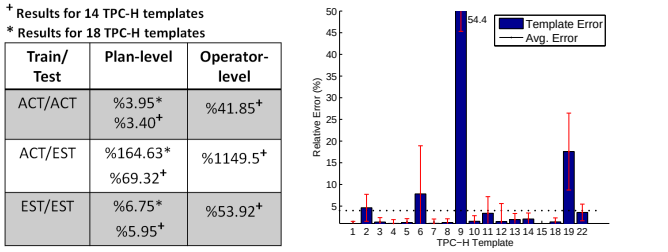
For the 1GB scenario, we show the results of operator-level prediction for the 14 TPC-H templates in Figure 6(i). In this case, for 8 of the templates the average error is below 25% and the mean error is 16.45%. However, the mean error for all the 14 TPC-H templates is 59.57%.

We see that operator-level prediction produces modest errors for many cases, but also does perform poorly for some. We analyzed the set of templates that belongs to the latter case, and noticed that they commonly exhibit one or more of the following properties:

- (Estimation errors) the optimizer statistic estimates are significantly inaccurate.
- (I/O-compute overlap) there is significant computation and I/O overlap in the query. The end-effect of such concurrent behavior on execution time is difficult to capture due to pipelining.
- (Operator interactions) The operators of the same query heavily interact with each other (e.g., multiple scans on the same table that use the same cached data).

Next, we discuss the practical impact of statistics estimation errors on accuracy. We then turn to the latter two issues that represent the fundamental limitations of operator-level modeling; such models learn operator behavior "in isolation" without representing the context within which they occur.

3) Impact of Estimation Errors: We tried all the combinations of actual and estimate feature values for training and testing for (SVM-based) plan-level and operator-level prediction. The results are given in Figure 7(a) for the 10GB scenario. For further detail, we also show the prediction errors grouped by TPC-H templates in Figure 7(b) for the actual/actual case and plan-level prediction (over the 10GB scenario). These results are to be compared with those in Figure 6(a).



+ Results for 14 TPC-H templates
* Results for 18 TPC-H templates

Train/ Test	Plan-level	Operator-level
ACT/ACT	%3.95* %3.40+	%41.85+
ACT/EST	%164.63* %69.32+	%1149.5+
EST/EST	%6.75* %5.95+	%53.92+

(a) Prediction with Actual Values vs Estimates

(b) Plan-level Modeling with Actual Values (10GB)

Fig. 7. **Impact of Estimation Errors on Prediction Accuracy in Static Workload Experiments**

Unsurprisingly, the best results are obtained in the actual/actual case (i.e., training/testing with actual feature values), which is not a viable option in practice due to the unavailability of the actual values without running the queries. The next best results are obtained with the estimate/estimate option, the option that we used in the rest of the paper. Finally, the results obtained with actual/estimate (i.e., training on actual values and testing on estimates) are much worse than the other two, primarily due to optimizer estimation errors that are not taken into account during training.

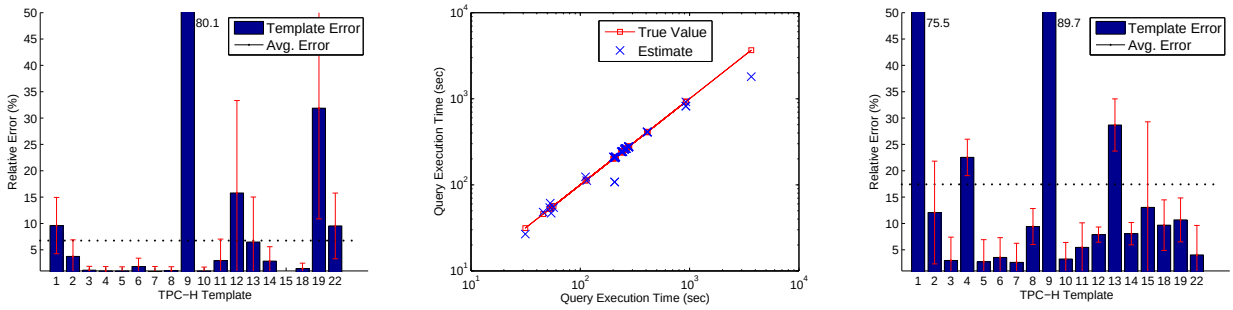
To provide a sense of the magnitude of the estimation errors made by the optimizer, consider template-18, which is one of the templates that exhibit the biggest error in operator-level prediction with actual/estimate model building. Instances of template-18 include the following `group by` clause on table `lineitem`:

group by l_orderkey having sum(l_quantity) > 314

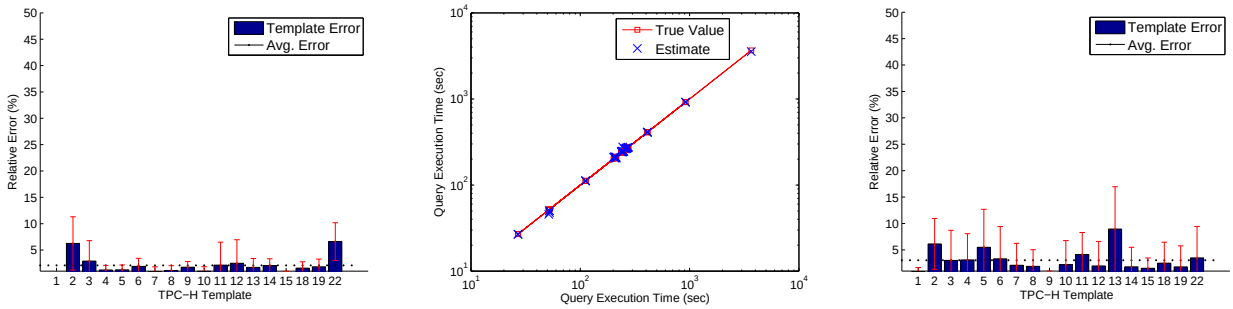
There are 15 million distinct `l_orderkey` values in `lineitem` (out of approximately 60 million tuples). The estimated number of groups satisfying `sum(l_quantity)` $>$ 314 is 399521, whereas the actual number is 84. The PostgreSQL query optimizer computes this estimate using histograms (with 100 bins) for each column based on the attribute independence assumption. The results are later fed into a Hash-Semi-Join, whose cost estimate is correspondingly very much off the mark.

Comparing the actual/actual against the estimate/estimate results, we observe that optimization estimate errors lead to, perhaps surprisingly, only a modest degradation in prediction accuracy. This result is due to the ability of the models to also integrate error corrections during learning. Thus, while better

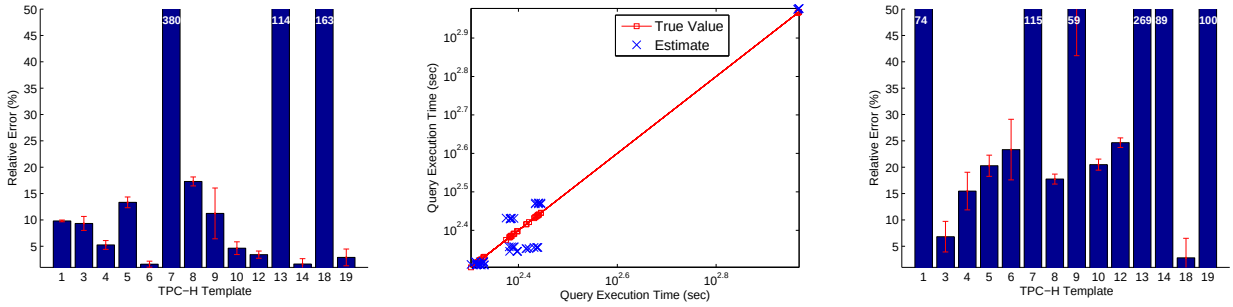
[2]The execution plans for the queries of the remaining 4 templates contain PostgreSQL-specific structures, namely INITPLAN and SUBQUERY, which lead to non-standard (i.e., non tree-based) execution plans with which our current operator-level models cannot cope at present.



(a) SVM-based plan-level modeling, errors by template (10GB) (b) SVM-based plan-level prediction (10GB) (c) SVM-based plan-level modeling, errors by template (1GB)



(d) KCCA-based plan-level modeling, errors by template (10GB) (e) KCCA-based plan-level prediction (10GB) (f) KCCA-based plan-level modeling, errors by template (1GB)



(g) Operator-level, Errors by Template (10GB) (h) Operator-level Prediction (10GB) (i) Operator-level, Errors by Template (1GB)

Fig. 6. **Static workload experiments with plan-level (using SVMs and KCCA models) and operator-level modeling in 1GB and 10GB TPC-H databases. Error values in bar-plots are capped at 50%. Larger error values are printed next to the corresponding bars.**

estimations generally mean better results, it is possible to produce highly accurate predictions even with rather mediocre estimations (as in the case of PostgreSQL).

4) Hybrid Prediction Method: We now present comparative results of the three plan ordering strategies (see Section III-D) discussed for offline hybrid model selection. The results, shown in Figure 8, were obtained with the 14 TPC-H templates used in operator-level modeling and the 10 GB database.

As described earlier, we first create an ordered list of query sub-plans based on the chosen plan ordering strategy, leaving out sub-plans with average error lower than a given threshold (.1 in this experiment) for the size-based and frequency-based strategies. Then, at each iteration (x-axis), we create a (SVM-based) model for the next plan in the ordered list, add this model to the current model set and then re-evaluate prediction error on the test workload (y-axis). The step behavior is observed when a newly created model decreases the error.

We observe that the size-based and error-based strategies quickly reduce the error rate. The size-based strategy takes longer to reach the minimum error level, as in some cases larger sub-plans should be modeled for reducing the error and it takes time for this strategy to reach those plans.

The frequency-based strategy initially takes longer to reduce the error. The reason is that this strategy can easily get stuck in a relatively large sub-plan that has a high occurrence rate, since it needs to explore all the sub-plans involved in the larger sub-plan (starting from the smallest sub-plan) until it decreases the error rate. As discussed earlier, all such sub-plans are by definition at least as frequent, hence need to be explored with this heuristic. Overall, the error-based strategy provides a well balanced solution, quickly and dramatically reducing the prediction errors only with a small number of additional models. We also note that the final accuracy obtained with the hybrid-method approaches to that of the KCCA-based model.

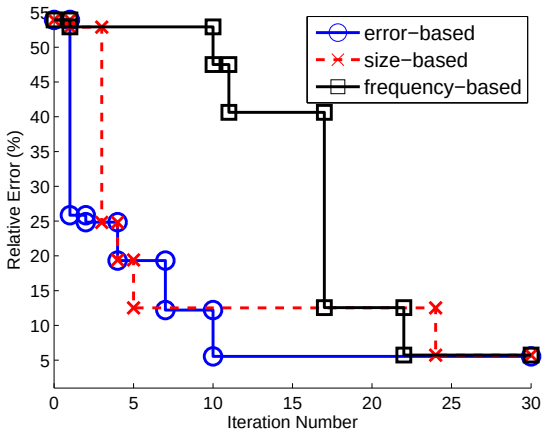


Fig. 8. **Hybrid Prediction Plan Ordering Strategies**

D. Predicting for Dynamic Workload

The results so far have shown that for known, static workloads, plan-level modeling performs well. They also revealed that hybrid models offer similar accuracy to plan-level models for static workloads. Next, we present results demonstrating that plan-level modeling has serious limitations for dynamic workloads, whereas hybrid modeling still continues to provide high accuracy. We also report comparative results for online model building (Section IV) that creates custom hybrid models for a given query from the training data.

For this experiment, we used the 12 templates shown in Figure 9. For each template we build and test separate prediction models using only the training data from the other templates. The two other TPC-H templates were excluded because they include specific operators exclusively found in those templates, and thus cannot be modeled with our current setup. We show results for KCCA and SVM -based plan-level, operator-level, hybrid (with error-based and size-based strategies) and online modeling algorithms.

As expected, plan-level models perform poorly across the board and thus do not offer much value in dynamic workloads. However, SVM-based plan models perform better than the KCCA-based approach. We also observe that the online (hybrid) modeling algorithm performs best in all cases, except for template-7. Further investigation reveals that the training data lacks a specific sub-plan that is the root cause of the error on template-7. These results confirm the ability of online modeling to identify the models that are very likely to help by utilizing the knowledge of a given query plan. Such models can be eliminated by offline strategies if they do not help improve training accuracy.

Another interesting observation is that the size-based hybrid strategy performs somewhat better than the error-based strategy in these experiments. This can be explained by the ability of the former to favor models for smaller sub-plans that are more likely to occur in unseen queries.

E. Platform Independence

In this experiment, we apply our QPP techniques on a different hardware platform to demonstrate its applicability in

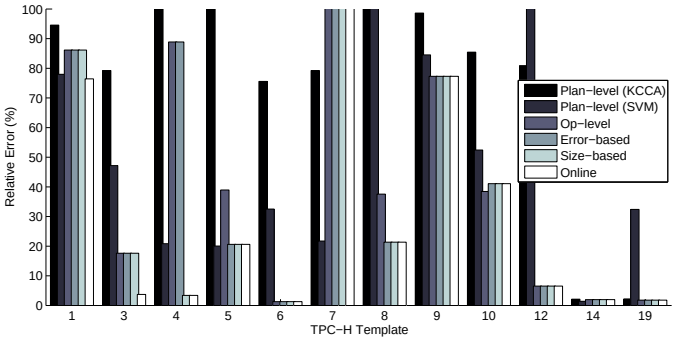


Fig. 9. **Dynamic Workload Prediction Results**

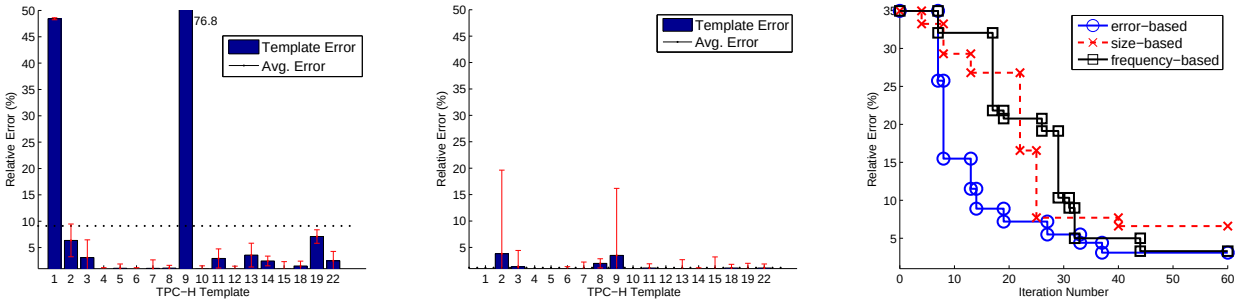
different environments. We show results from a 10GB TPC-H experiment executed on a 2.8GHz machine with 8GB RAM. The database buffer pool size was set to 2GB. We used the same queries that were used in the 10GB experiment. Plan-level prediction results (following a static workload scenario) using SVMs and KCCA are shown in Figures 10(a)-(b).

The average prediction errors are slightly lower than the errors obtained in the previous 10GB experiment. On this new platform the TPC-H queries execute faster than on the previous platform. For instance, template-9 queries finish under 20 minutes instead of an hour. As such the query run times are less divergent. This is because of the higher disk speed in the new platform, 83MB/sec sustained read rate (versus the 55MB/sec read rate before) as well as the faster CPU and increased RAM size. In addition, there is much less variance on the execution times of queries in this setup. We attribute this to the higher disk speed and to the increased RAM size in the new platform. Since 80% of the database can fit into memory, the number of duplicate reads from the disk are significantly reduced (due to the filesystem and database caches). The lower variance of query execution times observed in this data set enables improved accuracy on the predictions.

In Figure 10(c), we show QPP results (based on the static workload scenario) using hybrid modeling with different plan ordering strategies. As before, the error-based strategy reduces the prediction error faster than the other strategies. While the error and frequency -based strategies converge to the same prediction error, the size-based method resulted in a slightly higher error value. This is possible since each plan-ordering strategy considers plan-level models in different order and the initially chosen models may cause the later plan-level models to be discarded. Finally, we note that the accuracy obtained with the hybrid method (using SVM-based plan-level models) is higher than the accuracy of SVM-based plan-level modeling and close to the accuracy of KCCA-based model.

VI. Related Work

Plan-level predictions have recently been studied [1]. In [1], authors consider KCCA-based plan-level QPP for the following static workloads: the TPC-DS query benchmark and a query workload obtained from a customer database. They report that they can predict execution times within 20% of the actual time for 85% of their test queries. In addition to execution time, estimation of other performance metrics such



(a) SVM-based plan-level modeling, errors by template (10GB)

(b) KCCA-based plan-level modeling, errors by template (10GB)

(c) Hybrid QPP

Fig. 10. **QPP Platform Independence**

as disk I/O and message bytes is also considered.

In previous work, machine learning techniques have been used in the context of query optimizers [10], [11], [12]. In the learning optimizer (LEO) [10], [11] project, model-based techniques are used to create a self-tuning query optimizer. The goal in [10], [11] is to produce better cost estimates for use in query optimization. The approach taken is to compare the estimates of the query optimizer with the actual values observed during query execution to repair the inaccurate estimates. In [12], a statistical technique called *transform regression* is used to create cost models for XML operators.

Recently, there have been successful applications of learning techniques in self-managing systems. In [13], authors present a statistics-driven modeling framework for data-intensive *Cloud* applications. KCCA-based techniques are used for predicting the performance of Hadoop jobs. In [16], a statistics-driven workload generation framework is presented for the purpose of identifying suggestions (e.g., scheduling and configuration) to improve the energy efficiency of MapReduce systems.

In [14], [15] authors describe an experimental modeling approach for capturing interactions in *query mixes*, i.e., concurrently running queries. Given a query workload, the goal is to come up with an execution schedule that minimizes the total execution time. The query interactions are modeled using statistical models based on selectively chosen sample executions of query mixes.

VII. Conclusions

This paper studied techniques for learning-based QPP over analytical workloads. We proposed novel modeling techniques and demonstrated their general applicability and effectiveness with implementation on PostgreSQL and TPC-H data and queries. We provide the most comprehensive work on this topic to date, and show highly accurate and general results.

Learning-based QPP is a fertile research area, with many open opportunities and challenges to be explored. One immediate idea is to supplement the static models studied in this paper with additional run-time features. The values for such features can be obtained during the early stages of query execution, and used to create richer models that yield higher predictive accuracy with modest delays in prediction.

As mentioned earlier, this paper does not address QPP in the presence of concurrent query execution. There is already some promising work addressing this problem [14], [15], [20], and we believe the techniques proposed here can be extended to provide an alternative perspective to this challenge. As yet another direction, our techniques can be adapted to work in platforms such as MapReduce/Hadoop [18] and Dryad [19].

References

[1] Ganapathi, A. et al. Predicting multiple performance metrics for queries: Better decisions enabled by machine learning. ICDE 2009.

[2] Makridakis, S., Wheelwright S., and Hyndman, R. Forecasting Methods and Applications. Third Edition. John Wiley & Sons, Inc. 1998.

[3] M. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. IEEE Trans. on Knowledge and Data Engineering, 15(3), Nov. 2003.

[4] I. H. Witten and E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, second edition, June 2005.

[5] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. http://www.csie.ntu.edu.tw/cjlin/libsvm.

[6] S. Akaho. A Kernel Method For Canonical Correlation Analysis. IMPS2001.

[7] Francis R. Bach, Michael I. Jordan. Kernel Independent Component Analysis. Journal of Machine Learning Research, 3, pp.1-48, 2002.

[8] TPC-H benchmark specification, http://www.tpc.org/tpch/

[9] Christian Igel, Verena Heidrich-Meisner, and Tobias Glasmachers. Shark. Journal of Machine Learning Research 9, pp. 993-996, 2008.

[10] Volker Markl, G. M. Lohman and V. Raman. LEO: An Autonomic Query Optimizer for DB2. IBM Systems Journal Special Issue on Autonomic Computing, January 2001.

[11] M. Stillger, G. M. Lohman, V. Markl and M. Kandil. LEO - DB2's LEarning Optimizer. VLDB 2001.

[12] Zhang, N., et al. Statistical learning techniques for costing XML queries. VLDB 2005.

[13] Ganapathi, A., Yanpei Chen, Fox, A., Katz, R., Patterson, D. Statistics-driven workload modeling for the Cloud. Data Engineering Workshops (ICDEW), 2010 pp.87-92, 1-6 March 2010.

[14] Ahmad, M., Aboulnaga, A., Babu, S., and Munagala, K. Modeling and exploiting query interactions in database systems. In Proceeding of the 17th ACM Conference on Information and Knowledge Management (Napa Valley, California, USA, October 26 - 30, 2008). CIKM '08. ACM, New York, NY, 183-192.

[15] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Qshuffler: Getting the query mix right. In ICDE, 2008.

[16] Chen, Y., Ganapathi, A. S., Fox, A., Katz, R. H., Patterson, D. A. Statistical workloads for energy efficient mapreduce. Tech. Rep. UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[17] R. Othayoth and M. Poess, The making of tpc-ds, in VLDB 06: Proceedings of the 32nd international conference on Very large data bases. VLDB Endowment, 2006, pp. 10491058.

[18] Hadoop MapReduce web site. http://hadoop.apache.org/mapreduce/

[19] Michael Isard et al. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. EuroSys, Portugal, March 21-23, 2007.

[20] Modeling and Prediction of Concurrent Query Performance. J. Duggan, U. Cetintemel, O. Papaemmanouil, E. Upfal. In SIGMOD'11.