Learning-based Query Performance Modeling and Predection

Mert Akdere and Ugur Cetintemel

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-11-01
January 2011

Learning-based Query Performance Modeling and Prediction

Mert Akdere
Brown University
makdere@cs.brown.edu

Uğur Çetintemel
Brown University
ugur@cs.brown.edu

ABSTRACT

Accurate query performance prediction (QPP) is central to effective resource management, query optimization and user experience management. Analytical cost models, which are commonly used by optimizers to compare candidate plan costs, 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.

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 and generality, 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.

1. 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 [6] 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 [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 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 fea-

tures (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 workloads, substantially improving upon the existing solutions.

The rest of the paper is organized as follows: we start with background information on data-driven model-based prediction in Section 2. In Section 3, we first describe our general approach to using statistical learning techniques for QPP. Plan and operator -level performance prediction methods are described in Section 3.1 and Section 3.2, respectively. Next, in Section 3.4 we introduce the hybrid prediction method. Online modeling techniques which build prediction models at query execution time are discussed in Section 4. We present experimental results using the TPC-H query workload in Section 5. We then end the paper with related work and conclusion remarks in Sections 6 and 7.

2. BACKGROUND: MODEL-BASED PREDICTION

We use the term model to refer to any predictive function such as Multiple Regression, Bayesian Nets, and Support Vector Machines. Training a model involves using historical data sets to determine the best model instance that explains the available data. For example, fitting a function to a time series may yield a specific polynomial instance that can be used to predict future values.

Model training (or building) requires selecting (i) the feature attributes, a subset of all attributes in the data set, and (ii) a prediction model, e.g., Linear Regression and Support Vector Machines (SVMs), to be used for modeling. In general, we cannot know which model type and feature set will produce the most accurate model for a given data set without building and testing multiple models. In some cases, a domain expert can manually specify the feature attributes. In other cases, this step is trivial as the prediction attribute(s) directly determine the feature attribute(s), e.g., in autoregressive models. Alternatively, feature attributes can be learned automatically; however, given a set of n attributes, trying the power set is prohibitively expensive if n is not small or training is expensive [4, 3, 2] thereby requiring heuristic solutions.

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

While we use a feature selection algorithm for building accurate models using relevant features, we do not consider building multiple models of different types for solving the model selection problem. Instead in each one of our experiments we use a single type of prediction model, either Linear Regression or SVMs, that performs well.

Hypothesis testing and confidence interval estimations are two common techniques for determining predictive accuracy [2]. As mentioned, it is not possible to estimate a priori what model would be most predictive for a given data set without training/testing it. One form of hypothesis testing that is commonly used is K-Fold Cross Validation (K-CV). K-CV divides the observed 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 will use cross-validation techniques to estimate the accuracy of our prediction models.

3. MODELING QUERY EXECUTIONS

In this study, we describe QPP methods based on 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. In this phase, queries are represented as a set of features (i.e., predictive variables) with corresponding performance values (i.e., target variables) and the goal 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.

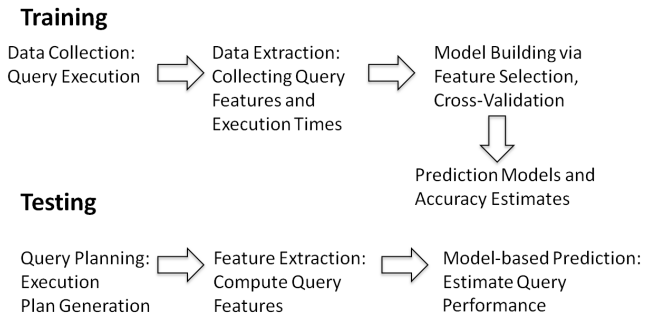


Figure 1: Statistical Modeling Approach to Query Performance Prediction.

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 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).

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 3.1. 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 3.2 and compare the relative advantages and drawbacks of the two approaches in Section 3.3. Next, in Section 3.4, we introduce a "hybrid" modeling approach that combines the fine and coarse grained modeling methods to form a highly accurate and general QPP approach.

3.1 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 1 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.

Feature Name	Description
p_tot_cost	Estimated plan total cost
p_st_cost	Estimated plan start cost
p_rows	Estimated number of output tuples
p_width	Estimated average width 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 1: Features for plan-level models. p_st_cost refers to the cost of query execution until the first output tuple. <operator_name> refers to the query operators such as *Limit, Materialize* and *Sort.*

As mentioned in Section 2, we need to address two challenges when using model-based learning techniques. The first problem, *feature selection*, deals with the issue of choosing the most predictive features from the available set of features. In our experiments, we frequently observed that models using the full set of features given in Table 1 performed less accurately than with smaller number of features. We use a best-first search based, forward feature selection algorithm [4], described in Section 2 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).

The second problem is *model selection*, the process of picking the right prediction model for the given task and data set. As discussed in Section 2, it is not possible in general to identify the most accurate prediction model without training and testing multiple models. We use a regression variant of Support Vector Machines (SVMs) [5] for plan-level modeling, which provided high accuracy in our experiments. However, we note that all of the approaches we present here are model-agnostic and can readily work with different model types.

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.

3.2 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 prediction model* is used for estimating the time spent during the execution of an operator (and in the subquery 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 prediction 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 2. 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.

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 2: 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.

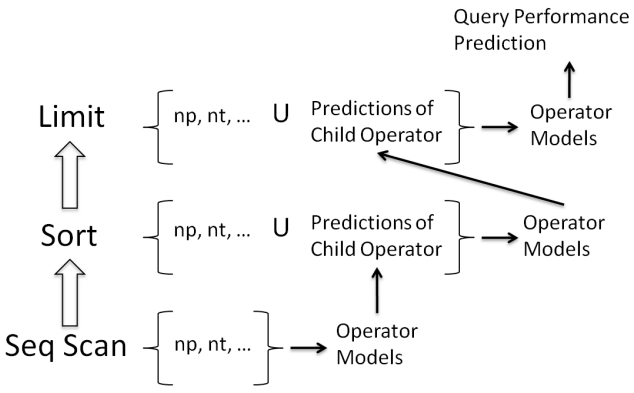


Figure 2: Operator-level query performance prediction: operator models use operator-level features together with the predictions of child operators for performance prediction.

3.3 Plan- versus 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 scenarios in which 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 execution 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 lower level underlying factors, such as operator interactions during query processing, on the execution time for each distinct query plan (in the training data) with a 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 query workloads where queries with unforeseen execution plans are frequently observed. Even worse, there can be problems even 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 an 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 the 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.

3.4 Hybrid Modeling

In hybrid query performance prediction, we combine the operator- and plan- level modeling 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. Thus, as long as the predictive accuracy is acceptable, operator-level modeling is effective. For queries with low operator-level prediction accuracy, on the other hand, we learn models for specific query sub-plans using plan-level modeling and compose both types of models to predict the performance of the entire plan. We argue, and later also experimentally demonstrate, that this hybrid solution indeed combines the relative benefits of the operator-level 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%.

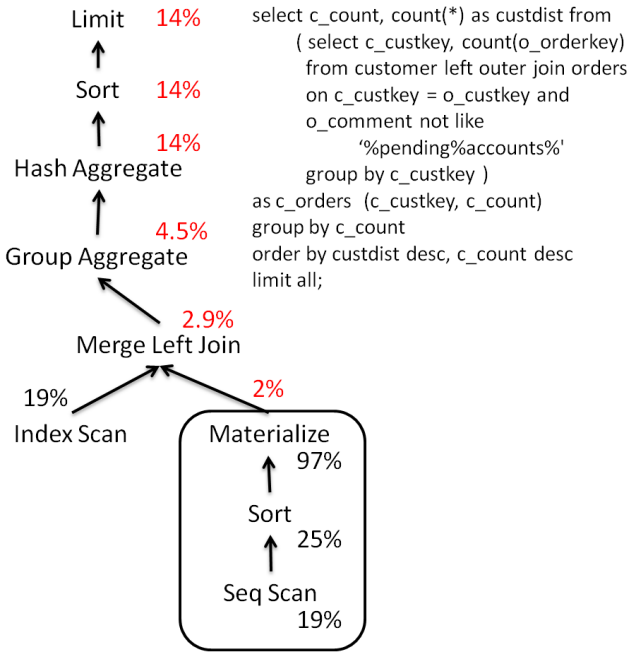


Figure 3: Hybrid QPP example: plan-level prediction is used for the highlighted sub-plan together with operator-level prediction for the rest of the operators to produce the end query performance prediction.

In the hybrid approach, we build a separate plan-level model for this 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 query performance prediction 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 hold-out 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 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 involving two plans with the same size, the more frequently occurring plan is given priority.

- `Frequency-based`: order the plans in decreasing *occurrence frequency* across the queries

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 frequency 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 switching to 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 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.

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 do
5.     plan = get_next(strategy, candidate_plans)
6.     plan_model = build_plan_model(data, plan)
7.     [predictions, new_accuracy] = apply_models(data, models
       ∪ 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
```

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.

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: For each plan-level model, we need to keep track of the set of queries to which they can be applied, and in turn, for each query we need to keep track of the set of applicable plan-level models. 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 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.

4. ONLINE MODEL BUILDING

In dynamic query workloads, where queries with unforeseen plan structures are present, the plan-level performance prediction method performs poorly due to lack of good training data. The operator-level and the hybrid prediction 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 much 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 in future queries. As a case study, we analyze the generated execution plans for the TPC-H query workload 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 Experiments Section for more details.). Our key observations for this data set include:

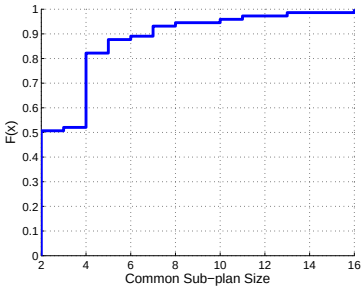
(1) Smaller sub-plans are more common across the TPC-H query plans (see Figure 4(a)).

(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 3.4 will likely achieve higher applicability compared to the other strategies in the dynamic workload case.

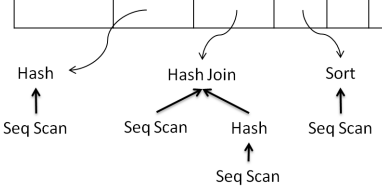
However, the hybrid approach may fail to increase the prediction accuracy for dynamic workloads in some cases. For example, the prediction errors for some unforeseen query plans may not originate from the common sub-plans, and as a result, plan-level models from the training data cannot reduce the error.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

In the online modeling technique, we build new plan-level models for performance prediction at run-time upon the receipt of a query. We initially produce predictions with the set of existing models,

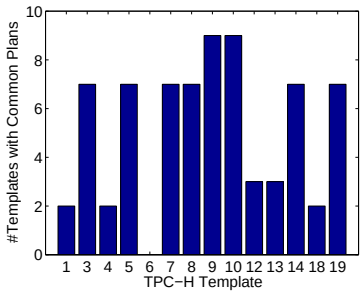


(a) CDF for common sub-plan sizes (#query operators)

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

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

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 better estimated accuracy than the operator-level prediction method exists).

5. EXPERIMENTS

5.1 Setup

Our experimental study uses the TPC-H decision support benchmark [6] implemented on top of PostgreSQL. The details are presented below.

Database Management System. 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 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. The queries were executed on a single commodity server with 4GB RAM running Ubuntu with kernel 2.6.28. 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. We used Support Vector Machines (available from the libsvm library [5]) with the nu-SVR kernel for support-vector based regression and linear regression models (available from the Shark machine learning library [7]) for plan- and operator-level modeling, respectively. Both 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. A forward feature selection algorithm, described in Section 2, was used to build accurate prediction 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 error 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 square error would be better for minimizing the absolute difference (or its square) in actual and predicted execution times. Other types of metrics include R^2 or predictive risk [1]. These 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 parts, and 4 parts 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 individual accuracy values from the testing of each cross-validation part. We used *stratified sampling* for dividing the data into 5 parts to ensure that each part contains roughly equal number

of queries from each template.

5.2 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.

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.

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

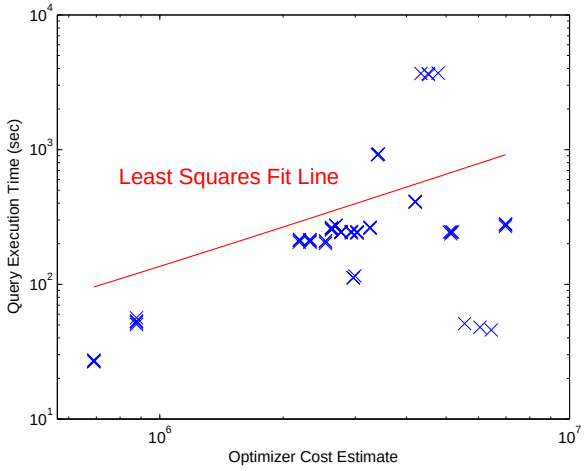


Figure 5: Optimizer Cost vs Query Execution Time

5.3 Predicting for Static Workloads

Results for the plan-level and operator-level prediction methods are given in Figure 6 both for the 10GB and 1GB TPC-H scenarios. These results were obtained using estimate-based features for building models in training and for prediction in testing. The use of actual (observed) values for features is discussed in Section 5.3.3.

5.3.1 Plan-level Modeling

The plan-level prediction results contain values for all the 18 TPC-H templates. Overall, the average relative prediction errors are 6.75% and 17.43% for the 10GB and 1GB databases, respectively (Figure 6(a)-(c)), implying that plan-level modeling can be very effective for static workloads. The difference between the results for the two databases can be explained by the respective ratios of the standard deviation to the average execution time of queries, which is about 2.63 times greater in the 1GB database case. This characteristics makes the 1GB case fundamentally more difficult to predict.

In both cases, queries from template-9 stand out as the worst predicted set of 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 do not fit well. To alleviate this problem, we built a separate prediction model for template-9 for the 10GB case, which reduced its error down to 7%.

5.3.2 Operator-level Modeling

We now show 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(d)). 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.

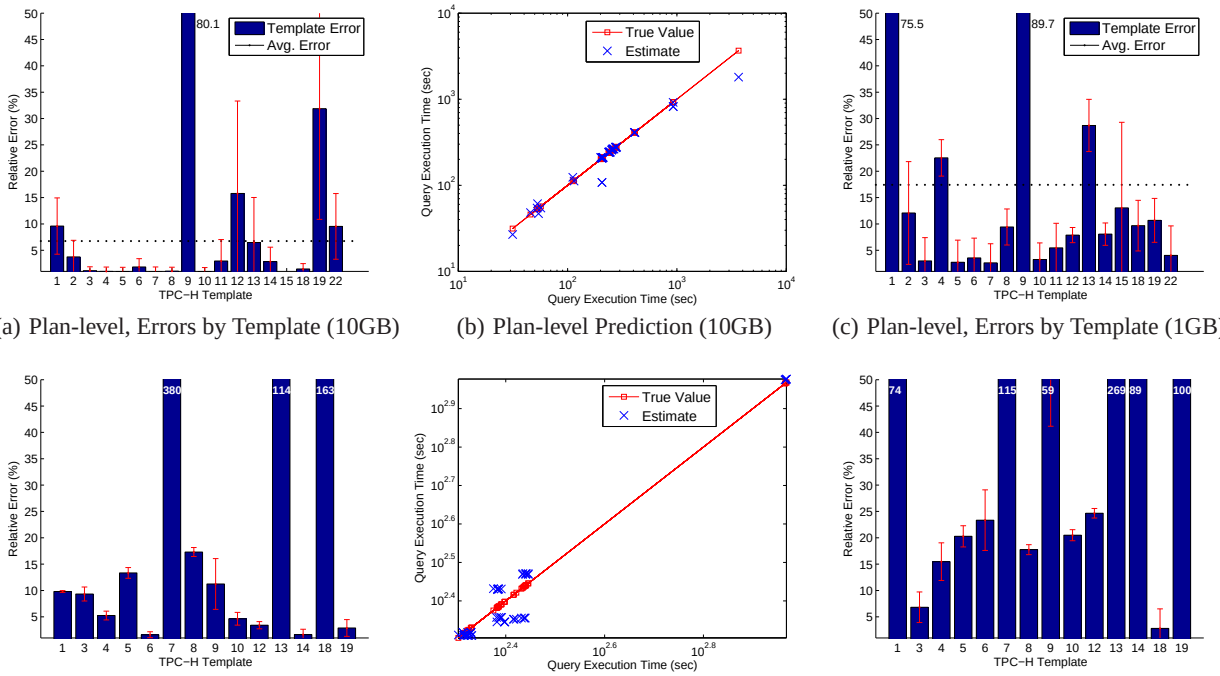
For the 1GB scenario, we show the results of operator-level prediction for the 14 TPC-H templates in Figure 6(f). In this case, for 8 of the templates the average error is below 25% and the mean error is 16.45%. However, the mean error for all the 14 TPC-H templates is 59.57% (slightly larger than the 10GB case).

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).

[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.

[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) Plan-level, Errors by Template (10GB) (b) Plan-level Prediction (10GB) (c) Plan-level, Errors by Template (1GB)

(d) Operator-level, Errors by Template (10GB) (e) Operator-level Prediction (10GB) (f) Operator-level, Errors by Template (1GB)

Figure 6: Static workload experiments with plan-level and operator-level prediction methods in 1GB and 10GB TPC-H databases. The error values in bar-plots are capped at 50. Error values beyond the limits of the plots are printed next to their corresponding bars.

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

5.3.3 Impact of Estimation Errors

We tried all the combinations of actual and estimate feature values for training and testing for 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).

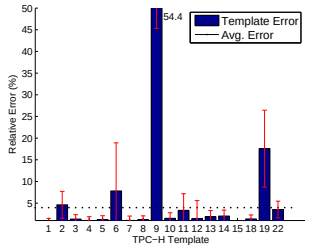
Unsurprisingly, the best results are obtained in the actual/actual case (i.e., training and testing with actual feature values), which is not a viable option in practice due to the unavailability of the actual feature values without running the queries. The next best results are obtained with the estimate/estimate option (i.e., training and testing with estimated feature values), 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

$^{+}$ 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)

Figure 7: Impact of Estimation Errors on Prediction Accuracy in Static Workload Experiments

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

5.3.4 Hybrid Prediction Method

We now present comparative results of the three plan ordering strategies (see Section 3.4) 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.

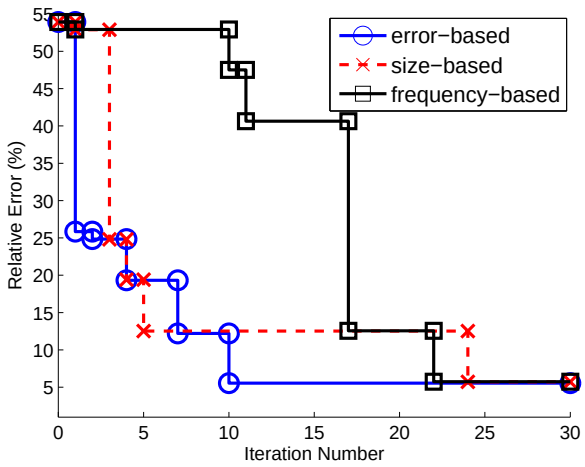


Figure 8: Hybrid Prediction Plan Ordering Strategies

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 model for the next plan in the ordered list, add this model to the current model set and then re-evaluate predictive 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.

5.4 Predicting for Dynamic Workload

The results so far have shown that for known, static workloads, plan-level modeling performs well. They have also revealed that hybrid models offer similar, and sometimes even better accuracy than plan-level models. Next, we present results demonstrating that plan-level modeling has serious limitations for unknown or changing workloads, whereas hybrid modeling still continues to provide high accuracy. We also report comparative results for online model building (Section 4) that creates custom hybrid models for a given query from the available training data.

For this experiment, we used the 12 templates shown in Figure 9, with 11 of them used in training and the remaining for testing. That is for each template we build and test separate prediction models based on the training data of 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 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 the presence of dynamic workloads. 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.

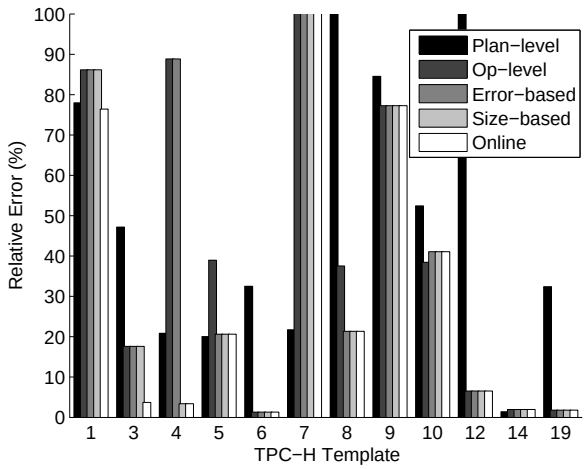


Figure 9: Dynamic Workload Prediction Results

6. RELATED WORK

Query-plan-level predictions have recently been studied [1]. In [1], authors consider plan-level query performance prediction for the following static query workloads: the TPC-DS query benchmark and a query workload obtained from a customer database. They report that they can predict individual query execution times within 20% of the actual time for 85% of their test queries. In addition to the query execution time, estimation of other performance metrics such as disk I/O and message bytes is also considered. In this paper, we focused on the execution time performance metric. While we can apply our techniques separately for each performance metric, we plan to consider the extension to joint prediction of multiple metrics in future work.

In previous work, machine learning techniques have been used in the context of the query optimizer [8, 9, 10]. In the learning optimizer (LEO) [8, 9] project, model-based techniques are used to create a self-tuning database query optimizer. The goal in [8, 9] 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 [10], a statistical modeling technique called *transform regression* is used to create cost models for XML query operators. In addition, new training data can be efficiently integrated into their existing cost models for adapting to changing workloads.

Recently, there have been successful applications of machine learning techniques in system self-management problems. In [11], authors present a statistics-driven modeling framework for data-intensive *Cloud* applications. Kernel Canonical Correlation Analysis (KCCA) modeling techniques are used to make predictions for the execution performance of Hadoop jobs. In [14], 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 [12, 13] authors describe a successfull experimental modeling approach for capturing interactions in *query mixes*, i.e., sets of concurrently running queries. Given a query workload, the goal is to come up with a query execution schedule (in terms of query mixes) that minimizes the total execution time. The query interactions are modeled using statistical models based on selectively chosen sample executions of query mixes. In our study, we have not yet considered performance prediction in concurrent query workloads.

Finally, there has also been work on query progress indicators [15, 16]. Query progress indicators provide estimations for the completion degrees of running queries. Such studies assume that the work done by individual query operators are transparent, i.e., externally visible. While these studies are also related to query execution performance, they do not provide predictions for the execution time of queries.

7. CONCLUSIONS

This paper studied techniques for learning-based modeling of query execution for QPP over analytical workloads. We proposed novel query 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 results that significantly improve upon the existing solutions in terms of generality and predictive accuracy.

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. A generalization of this approach will lead us to an online, progressive prediction model, where predictions are continually updated during query execution, in a manner similar to online aggregation [20].

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 [12, 13], 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 for other data processing platforms such as MapReduce/Hadoop [18] and Dryad [19].

As database systems and the underlying software-hardware platforms become increasingly sophisticated, it is becoming increasingly infeasible to manually develop and maintain accurate models for system behavior. As such, learning-based modeling will become increasingly more useful, prevalent, and eventually indispensable. Our work and results form another promising step towards facilitating this vision.

8. REFERENCES

[1] Ganapathi, A., Kuno, H., Dayal, U., Wiener, J.,Fox, A., Jordan, M., and Patterson, D. Predicting multiple performance metrics for queries: Better decisions enabled by machine learning. In International.Conf. on Data Engineering (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. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

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

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

[8] 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.

[9] M. Stillger, G. M. Lohman, V. Markl and M. Kandil. LEO - DB2's LEarning Optimizer. Proc. Intl. Conf. Very Large Data Bases (VLDB). 2001.

[10] Zhang, N., Haas, P. J., Josifovski, V., Lohman, G. M., and Zhang, C. 2005. Statistical learning techniques for costing XML queries. In Proceedings of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 - September 02, 2005). Very Large Data Bases. VLDB Endowment, 289-300.

[11] 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.

[12] 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.

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

[14] 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.

[15] Chaudhuri, S., Narasayya, V., and Ramamurthy, R. Estimating progress of execution for SQL queries. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data (Paris, France, June 13 - 18, 2004). SIGMOD '04

[16] Luo, G., Naughton, J. F., Ellmann, C. J., and Watzke, M. W. Increasing the Accuracy and Coverage of SQL Progress Indicators. In Proceedings of the 21st international Conference on Data Engineering (April 05 - 08, 2005). ICDE.

[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. 1049Ð1058.

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

[19] Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly European Conference on Computer Systems (EuroSys), Lisbon, Portugal, March 21-23, 2007

[20] "Online Aggregation." J. M. Hellerstein, P. J. Haas, and H. J. Wang. Proc. 1997 ACM SIGMOD Intl. Conf. Management of Data, 171-182.