Predicting Completion Times of Batch Query Workloads Using Interaction-aware Models and Simulation

Mumtaz Ahmad
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
m4ahmad@uwaterloo.ca

Songyun Duan*
IBM T.J.Watson Research Center
Hawthorne, New York, USA
sduan@us.ibm.com

Ashraf Aboulnaga
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
ashraf@cs.uwaterloo.ca

Shivnath Babu
Department of Computer Science
Duke University
Durham, North Carolina, USA
shivnath@cs.duke.edu

ABSTRACT

A question that database administrators (DBAs) routinely need to answer is how long a batch query workload will take to complete. This question arises, for example, while planning the execution of different report-generation workloads to fit within available time windows. To answer this question accurately, we need to take into account that the typical workload in a database system consists of mixes of concurrent queries. Interactions among different queries in these mixes need to be modeled, rather than the conventional approach of considering each query separately. This paper presents a new approach for estimating workload completion times that takes the significant impact of query interactions into account. This approach builds performance models using an experiment-driven technique, by sampling the space of possible query mixes and fitting statistical models to the observed performance at these samples. No prior assumptions are made about the internal workings of the database system or the cause of query interactions, making the models robust and portable. We show that a careful choice of sampling and statistical modeling strategies can result in accurate models, and we present a novel interaction-aware workload simulator that uses these models to estimate workload completion times. An experimental evaluation with complex TPC-H queries on IBM DB2 shows that this approach consistently predicts workload completion times with less than 20% error.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Modeling and prediction; H.2.m [**Database Management**]: Miscellaneous

*This work was done while the author was at Duke University.

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

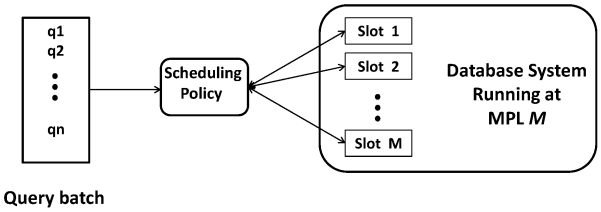

Figure 1: Problem definition

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Data warehouses and the Business Intelligence (BI) workloads that they run are an important and growing segment of the database market [5, 8, 12, 15]. A large fraction of BI workloads are long-running batch query workloads that are run repeatedly [20]. For example, enterprises run report-generation workloads on a frequent basis to analyze customer and sales activity. Such batch query workloads are critical for operational and strategic planning, so they have to be run and managed efficiently.

Figure 1 illustrates a question that arises frequently in batch workload management. A database system has to process a batch of queries q_1, q_2, ..., q_n. The *multi-programming level (MPL)* of the system is set to M, so the system can run M queries concurrently at any point in time. Whenever a query q finishes among the M queries running currently, a new query q' from the batch will be scheduled in its place based on a given scheduling policy. (First-in-First-out, or *FIFO*, and Shortest-Job-First are two popular scheduling policies.) Given the query batch, scheduling policy, and MPL, can we predict (ahead of time) how long the database system will take to process the entire batch of queries?

Automated tools to answer this question with good accuracy and efficiency are needed in a number of workload-management tasks:

- Database administrators (DBAs) may need to plan the execution of different report-generation workloads to fit within available time windows.
- Accurate prediction can be used to give data analysts continu-

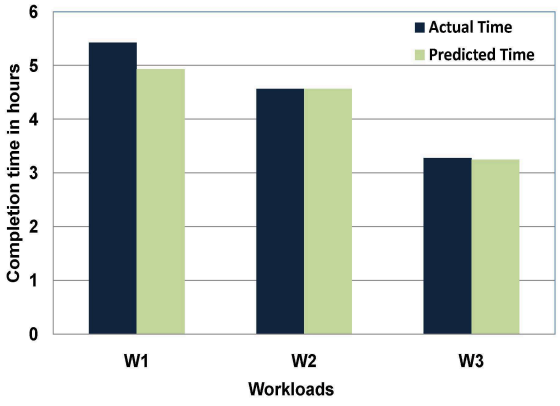

Figure 2: Completion times for workloads running different query mixes

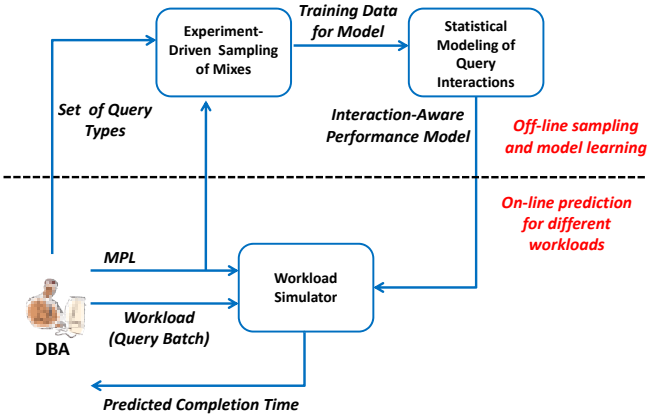

Figure 3: Solution overview

ous feedback on the progress of running workloads.

- Such tools form *what-if modules* [10, 19] to determine which scheduling policy to use for a workload or the resources needed to complete a high-priority workload within a given deadline.
- Estimating workload completion time can be used as a what-if module to partition a query workload across multiple database instances in a parallel database system.
- As we show shortly, such a tool can enable query reordering for increased efficiency in systems that use FIFO scheduling.

A major challenge in predicting the completion time of batch query workloads in database systems is that the queries in a workload *interact* with each other when they execute concurrently. When a *mix* of concurrently running queries is executed as part of a batch workload, the queries in the mix will affect the performance of each other, and this effect may be negative or positive. Not taking such interactions into account, as in previous work such as [13], can lead to inaccurate predictions for BI report-generation workloads where both queries and query interactions can be complex. We demonstrate the effect of query interactions on workload completion time using a real example based on the widely-used TPC-H decision-support benchmark [32].

In this example, we are given a batch of 60 TPC-H queries to be run on an IBM DB2 database system with MPL 10 running a TPC-H 10GB database.[1] The scheduling policy is FIFO, which is the default (and often only) policy in most commercial database systems. The FIFO policy schedules queries in the order in which they are added to the batch. We created three separate workloads, denoted W_1-W_3. In all three workloads, the queries being executed are exactly the same, but we change the arrival order of the queries. The different orderings of the queries results in different concurrent mixes getting scheduled in W_1-W_3. Figure 2 shows the actual completion times of W_1-W_3. Note that the times vary from 3.3 to 5.4 hours; a significant variation in both absolute and relative terms.

While the same batch of queries was processed in all three cases, the *query mixes* that executed on the database system in each case were different. A query mix consists of a set of queries that execute concurrently with each other, and we can view the execution of a workload as a *sequence of query mixes*. Queries executing concurrently in a mix interact with each other. For example, a query q_1 can bring data into the buffer pool that, in turn, enables a concurrently running query q_2 to reduce its I/O (an example of *positive* interaction). Alternatively, q_1 and q_2 could interfere with each other on hardware resources such as CPU or memory, or on internal database system resources such as latches or locks (all examples of *negative* interaction).

The results in Figure 2 show the considerable impact query interactions can have on performance. Queries that compete for resources get executed concurrently in W_1, resulting in negative interactions and poor performance. In W_2 and W_3, the interactions are less negative, and occasionally even positive, where queries that help each other get executed concurrently. The only difference between these three workloads is in the query interactions that arise. It is therefore clear that ignoring interactions when predicting workload completion times can lead to very inaccurate predictions.

Surprisingly, there are no research or industry-strength automated tools for predicting batch query workload completion times in a general way. In this paper, we address this limitation and present an interaction-aware solution for predicting workload completion times.[2] Figure 2 also shows the predictions of our solution for workloads W_1-W_3, and it can be seen that the predictions are quite accurate. The defining feature of our solution is that it treats a batch query workload as a sequence of query mixes, and accounts for the query interactions that arise in these mixes. Our solution does not rely on predefined knowledge of the internal workings of the database system or the cause of query interactions, making it robust and portable across database systems.

1.1 Anatomy of an Interaction-aware Predictor

We begin with an overview of our interaction-aware predictor of batch workload completion times. The predictor comprises a *simulator* that can simulate the execution of query mixes in a given database system. The simulator performs this nontrivial task using *interaction-aware performance models* that can estimate the running time of queries executing with other queries in a mix.

Figure 3 shows the overall workflow of the predictor, which consists of a predominantly off-line learning component and an on-line prediction component. The workflow is invoked by a database administrator (DBA) when she identifies a context where batch workloads are executed repeatedly, and predictions of workload completion times can be useful. As input to the off-line phase, the DBA provides a set of *query types* (or templates) such that each query

[1]The details of our experimental setup are given in Section 8.

[2]A short version of this paper appears in [4].

in the batch workloads of interest can be mapped to one of these types. A number of tools are available that can help the DBA to extract templates from query logs (e.g., [22]). We make two key observations:

1. Sampling and modeling constitute the *off-line* training phase. Once models are trained for a given set of T query types, these models can be used *on-line* by the simulator to estimate the completion time of different workloads composed of any number of query instances of these T types.
2. Model training is completely independent of the size or completion time of the workload (e.g., the number of instances of each query type in the workload), or the number of distinct workloads for which predictions are needed.

Sampling and Modeling (Off-line Phase): Query interactions exhibit complex patterns that are hard to model analytically. Instead, we propose a new approach that designs and conducts *experiments* to sample the space of possible interactions. Each experiment runs a chosen query mix. The database monitoring data collected from all experiments is used to train a statistical model that captures the significant query interactions that can arise in the workloads of interest. This approach requires no prior assumptions about the internal workings of the database system or the nature or cause of query interactions, making it portable across systems. We address a number of challenges to make this approach practical: (i) choosing an appropriate model, (ii) developing an algorithm that selects a minimal and representative set of experiments, and (iii) developing an incremental sampling technique to interleave sampling and modeling with on-line simulation of workload execution.

Simulating Workload Execution (On-line Phase): The simulator uses a recurrence relation in conjunction with interaction-aware performance models to simulate the execution of the workload as a sequence of query mixes. This approach overcomes a major disadvantage of conventional analytical modeling and simulation, where domain experts have to spend many hours developing the simulator and the analytical models that it uses, only for it to become inaccurate when database internals are modified. We believe that simulation is a very powerful approach to database administration and tuning. However, very few works in this area (e.g., [24]) have harnessed the power and flexibility of simulation. Another noteworthy feature of our simulator is that it incorporates the simulation of the scheduling policy as a pluggable component. While we used this feature to support two common scheduling policies, FIFO and Shortest-Job-First, a variety of sophisticated scheduling policies can be supported if needed.

Roadmap: This paper is organized as follows. Section 2 presents an overview of related work. Section 3 demonstrates the significant impact that query interactions can have, and why modeling interactions is nontrivial. The simulator is described in Section 4. Our algorithms for sampling and modeling are presented in Sections 5 and 6, respectively. Section 7 discusses some design choices and possible extensions. Section 8 presents an experimental study using TPC-H queries on DB2, demonstrating the accuracy of our predictions for different workloads, scheduling policies, and data distributions. We present our conclusions in Section 9.

2. RELATED WORK

We are not aware of any work focusing on predicting the completion time of BI workloads, particularly in an interaction-aware manner. Overall, there is very little work that deals in a general way with the performance of query mixes and the interactions among concurrently executing queries within these mixes. In our prior work [2, 3], we showed that the presence of query interactions can cause highly suboptimal database performance if interaction-oblivious query schedulers are used. We developed two new query scheduling algorithms that significantly improve performance by choosing the appropriate query mixes to schedule based on performance models that capture the effect of query interactions. In [31], interaction-aware techniques are used to avoid overload in three-tier transactional systems. In these works, it was sufficient to distinguish between good and bad query mixes to be able to obtain good schedules or avoid overload. In this paper, we focus on predicting the completion time of a batch query workload, which is a far more challenging problem. For accurate prediction of workload completion time, the sequence of query mixes that will execute when the batch query workload is run, and the completion time of these mixes, need to be predicted with high accuracy. This required developing new sampling and modeling techniques to address the problem of predicting completion times with sufficiently high accuracy. We also develop a simulator to track the progress of the workload, and the novel sampling and modeling techniques that we propose help the simulator achieve good accuracy.

Symbol	Description
M	Multiprogramming level
T	Number of query types
Q_j	Query type j
q_j	An instance of query type j
$m_i = \langle N_{i1}, N_{i2}, \ldots, N_{iT} \rangle$	A query mix, m_i, with N_{ij} instances of each query type Q_j
t_j	Average execution time of a Q_j query instance when running alone
A_{ij}	Average completion time of a Q_j query instance when running in mix m_i
$\hat{A}_{ij}$	Estimated completion time of a Q_j query instance when running in mix m_i
l_i	Length of phase i in workload simulator
wc_{ij}	Fraction of q_j's work completed in its phases 1 to i in workload simulator

Table 1: Notation used in the paper

Other prior work on concurrently running query mixes generally falls into two categories: work on multi-query optimization (e.g., [25]), and work on sharing scans in the buffer pool (e.g., [23]). Both of these categories try to induce positive interactions between queries based on detailed knowledge of database system internals, but are fairly restricted in the types of interactions considered. In contrast, our work focuses on capturing different kinds of both positive and negative query interactions, regardless of their underlying cause; and without explicit knowledge of database system internals.

Some recent papers have employed the concept of *transaction mixes* in different application areas. These papers define a transaction mix as all the transactions of different types that run during a time interval or monitoring window without considering which of these transactions ran concurrently. This is fundamentally different from our notion of a *concurrent* query mix. Like our work, these papers use statistical techniques to learn models to estimate performance metrics for transaction mixes. Transaction mix models have been used for performance prediction, capacity planning, and detecting anomalies in performance [18, 30, 35, 34]. However, unlike our work, none of these papers consider interactions caused by the concurrent execution of transactions.

Machine learning and statistical performance modeling is gaining acceptance as a way to build robust performance models for complex problems in database systems. It was used in [13] to

Query Type	$Q1$	$Q7$	$Q9$	$Q13$	$Q18$	$Q21$
Run Time t_j (sec)	10.07	5.76	9.66	6.12	7.12	7.3

Table 2: Average run time, t_j, of different TPC-H query types on a 1GB database

	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
Mix	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}
m_1	11	143.9	8	144.6	3	211.2	2	97.8	2	149.8	4	127.5
m_2	2	361.7	8	298.6	1	476.0	18	121.2	0	0.0	1	231.2

Table 3: A_{ij} (in seconds) for different query types in query mixes on a 1GB database

Query Type	$Q1$	$Q7$	$Q9$	$Q13$	$Q18$	$Q21$
Run Time t_j (sec)	294.61	102.06	578.61	101.27	554.56	570.37

Table 4: Average run time, t_j, of different TPC-H query types on a 10GB database

	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
Mix	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}
m_3	1	1897.4	2	72.7	5	2919.3	0	0.0	2	1904.1	0	0.0
m_4	4	538.0	0	0.0	0	0.0	0	0.0	1	539.3	0	0.0
m_5	0	0.0	4	264.5	0	0.0	0	0.0	1	3413.7	0	0.0

Table 5: A_{ij} (in seconds) for different query types in query mixes on a 10GB database

predict performance metrics for database queries. The proposed techniques are able to make performance predictions for individual query types with less than 20% error for 85% of the test cases. The authors point out that statistical learning techniques can be harnessed to great advantage in the area of database performance modeling. However, their work focuses exclusively on single query types and does not consider interactions and query mixes, which are our focus in this paper. By using interaction-aware techniques, we are able to achieve prediction accuracy similar to [13] for batch BI workloads with complex interacting queries. Machine learning techniques are used in [16] to predict the range of execution time for a query in a database system. Similarly, machine learning is employed for database provisioning in [9, 14]. Experiment-driven performance modeling has also been used for tuning database configuration parameters [6, 11, 29]. In [11], the authors propose a tool that helps in finding better configuration parameter settings for a database system by planning experiments corresponding to different parameter values. In [29], the authors propose techniques to better tune the CPU and memory allocations for database workloads running inside virtual machines.

3. QUERY MIXES AND INTERACTIONS

We begin with a brief illustration of the complex patterns of query interactions and their impact on database performance. We use the TPC-H decision-support benchmark with two data sizes, 1GB and 10GB. The TPC-H benchmark defines 22 *query types*. Each query type is a template with parameter markers that are instantiated with values to generate query instances in the batch workload. The workloads are then run on a DB2 database system. Q_1-Q_T denote the T query types specified by the DBA. A batch workload W consists of zero or more instances of each of these types. A query mix m_i can be represented as a vector $\langle N_{i1}, N_{i2}, \ldots, N_{iT}\rangle$, where N_{ij} is the number of instances of query type Q_j in m_i, and $\sum_{j=1}^{T} N_{ij}$=M. M is the MPL of the database system. Table 1 presents a summary of the notation used in this paper.

To study the impact of query interactions, we need to measure the completion time of different queries in a mix. We also need this information to train the performance models that we use to predict workload completion times. We use the following procedure to obtain this information for a given query mix $m_i = \langle N_{i1}, N_{i2}, \ldots, N_{iT}\rangle$. All queries in the mix are started at the same time, t_s. The different queries in the mix have different run times and they do not finish at the same time. So in order to make sure that we are running the same query mix m_i throughout the measurement interval, as soon as a query of type Q_j finishes, we start another instance of Q_j. The longest running query instance of the initial set of query instances started at t_s finishes at some time t_f. We consider m_i to be finished at t_f and we do not start any further instance after t_f. Thus the longest running query type in the mix will have a completion time of $t_f - t_s$, while other query types in the mix will have a completion time in the range $[0-(t_f-t_s)]$. We always sample query mixes on a database with a warm buffer pool. To warm up the buffer pool, we run some random query mixes before collecting any samples.

We use t_j to denote the average completion time of queries of type Q_j when run alone in the system at $M = 1$. We use A_{ij} to denote the average completion time of queries of type Q_j when run in mix m_i. Interactions among queries that run concurrently in mixes can be negative or positive. The effect of any significant interaction involving Q_j in m_i, regardless of the cause of the interaction, will be observed in A_{ij}. We say that a query of type Q_j has negative interactions in mix m_i if $A_{ij} > t_j$, i.e., an instance of Q_j is expected to run slower in the mix than when run alone. On the other hand, $A_{ij} < t_j$ indicates positive interactions.

Performance Impact: Table 2 shows the run time of the 6 longest running TPC-H queries on a 1GB database when they run alone in the system (i.e., the t_j values). Table 3 shows two mixes, m_1 and m_2, consisting of these 6 query types. For each mix, the table shows N_{ij} and A_{ij} values. The large differences in A_{ij} values for the same query type illustrate the impact of query interactions. The following observations can be made about m_1 and m_2:

- In both m_1 and m_2, all A_{ij} values are much higher than the corresponding t_j values shown in Table 2. Thus, all queries are impacted negatively in these mixes.
- Both mixes have the same number ($N_{ij} = 8$) of instances of Q_7 and the same total number of queries ($M = 30$). However, A_{ij} for Q_7 in m_2 is almost twice the A_{ij} for Q_7 in m_1. Thus, instances of Q_7 are expected to run twice as slow in m_2 as they

run in m_1.

Next, we turn our attention to the same 6 query types on a 10GB database. Table 4 shows the run time of these query types when they are alone in the system. Table 5 shows three query mixes for this setting. A number of observations can be made about these tables, illustrating the high impact and complex patterns of query interactions:

- The A_{ij} value of Q_7 in mix m_3 is 72.7 seconds, which is lower than Q_7's t_j of 102.06 seconds. Thus, m_3 generates positive interactions for Q_7: instances of Q_7 run faster on average in mix m_3 than when they run alone in the system.
- Mix m_4 generates mild positive interactions for Q_{18}. Other interactions are strongly negative.
- Contrast Q_{18}'s performance in m_4 and m_5. Both mixes have the same total number of queries ($M = 5$), and one instance each of Q_{18}. Q_{18} benefits from positive interactions in m_4, but suffers a 6x slowdown in m_5. Thus, Q_{18} interacts positively with Q_1, but negatively with Q_7; which we may not expect from the performance of Q_1 and Q_7 in isolation (Table 4).

All the mixes used in these examples were run repeatedly to verify consistency and statistical significance. We observed the same performance patterns across runs, and the standard deviation of the completion times was always less than 4% of the mean for all query types in all the mixes reported here.

From these examples we see that: (a) Interactions in query mixes impact query run times significantly, sometimes by orders of magnitude. (b) Interactions are fairly complex in nature. Rules of thumb or simple intuitions cannot always explain query behavior in mixes. (c) The performance of a query Q cannot be predicted unless we are able to model the effect of other queries running concurrently with Q. Thus, it is important to develop mix-based reasoning about query workloads to better manage the performance of database systems. Further demonstration of the impact of query interactions can be found in [1].

4. SIMULATING WORKLOAD EXECUTION

We now describe the simulation of workload execution. Apart from the inputs in Figure 1—query batch, MPL, and scheduling policy—the simulator gets two inputs from the workflow in Figure 3: (i) the list of T query types, and (ii) interaction-aware performance models of these query types generated by experiment-driven sampling and modeling. For extensibility, the scheduling policy is itself a simulator with a standard API. The main API calls include returning the initial query mix to be scheduled, and returning the next query to be scheduled when a running query finishes. In FIFO, for example, the initial query mix $\langle N_{i1},\ldots,N_{iT}\rangle$ consists of the first M queries added to the batch, with later queries scheduled in FIFO order.

The execution of the workload is simulated as the execution of a sequence of query mixes. The execution of each mix is called a *workload phase*. A phase change happens when a running query finishes and another one starts. The goal of the simulator is to simulate the execution of workload phases and the transitions among them, and to estimate how long each phase will take. The predicted workload completion time is the total time taken by all phases.

The simulator tracks the *fraction of total work completed* by each query in each workload phase. Consider a query instance q_j of type Q_j. This query instance will start its execution at the start of some workload phase, which we call *query phase 1* for this query instance. The query will execute through different workload phases until it completes all the work that it needs to perform.

Let wc_{ij} be the fraction of q_j's work completed in its query phases 1 to i. When q_j starts, its wc_{ij}=0, and q_j is done when its wc_{ij}=1. The following recurrence relation tracks wc_{ij} through all the query phases:

$$\begin{aligned} wc_{0j} &= 0 \\ wc_{ij} &= wc_{(i-1)j} + \frac{l_i}{\hat{A}_{ij}} \end{aligned} \quad (1)$$

$wc_{(i-1)j}$ is the fraction of q_j's total work completed up to query phase $i-1$. The term $\frac{l_i}{\hat{A}_{ij}}$ is the fraction of q_j's total work that gets completed during query phase i. Here, l_i is the running time of phase i and $\hat{A}_{ij}$ is the *estimated run time* of q_j when it runs in the query mix of phase i. That is, $\hat{A}_{ij}$ is the estimated time for a query instance of type Q_j to execute, from start to finish, if it executes solely in this mix . Since phase i runs only for time l_i, q_j will complete only a fraction $\frac{l_i}{\hat{A}_{ij}}$ of its work in this phase.

Estimating $\hat{A}_{ij}$ and l_i: Let m_i=$\langle N_{i1}, \ldots, N_{iT}\rangle$ be the mix that runs in q_j's query phase i. The simulator uses interaction-aware performance models to estimate $\hat{A}_{ij}$ and l_i in Equation 1. We show in Section 6 that these models can estimate the run time $\hat{A}_{ij}$ of any instance of a query type Q_j in m_i. Recall that $wc_{(i-1)j}$ is the fraction of q_j's total work completed up to its query phase i-1. Since 1-$wc_{(i-1)j}$ represents the fraction of work remaining for q_j, (1-$wc_{(i-1)j}$)$\hat{A}_{ij}$ represents the *remaining time to completion* of q_j in m_i. Thus, l_i, the length of phase i, is the minimum (1-$wc_{(i-1)j}$)$\hat{A}_{ij}$ over all query instances in this phase.

Phase i ends when the query instance with minimum remaining time to completion in it finishes, at which point the simulator transitions to phase i+1. When a phase ends, the simulator updates the state of the simulation, namely, the wc_{ij} values of all unfinished queries. The simulator then transitions to the next phase by adding the next query (if any) given by the scheduling policy simulator. This process continues until all queries in the workload are completed. The estimated completion time of the workload is the sum of the lengths (l_i) of all distinct phases encountered during the simulation.

The simulator has a special case for handling the last phase of the workload. At the start of this phase, there are M running queries, but as these queries finish, they will not be replaced by other queries. The simulator makes a simplifying assumption and estimates the length of this phase as the maximum (1-$wc_{(i-1)j}$)$\hat{A}_{ij}$ over all query instances in this phase, ignoring the change in query mix as queries finish. This simplifying assumption enables the simulator to use the same performance model (with the same MPL) for all workload phases without affecting prediction accuracy.

5. SAMPLING

Interaction-aware performance models form key building blocks of our overall solution. For each query type Q_j, the workload simulator needs an interaction-aware performance model that can take as input the query mix $m_p = \langle N_{p1}, N_{p2}, \ldots, N_{pT}\rangle$ running in a phase, and return an estimate $\hat{A}_{pj}$ of the completion time of an instance of Q_j in m_p.

We map the problem of generating an interaction-aware performance model for Q_j as the problem of training a *regression model*. The model is trained from a set of n samples, where sample $s_i, 1 \leq i \leq n$, has the form $s_i = \langle m_i, A_{ij}\rangle = \langle N_{i1}, \ldots, N_{iT}, A_{ij}\rangle$. Sample s_i denotes an observation that an instance of type Q_j, when run in mix m_i, is completed in time A_{ij}. An appropriate type of regression model *model*$_j$ can be fitted to the n samples $s_1, \ldots, s_n$ to predict the completion time $\hat{A}_{pj}$ for an instance of Q_j when run

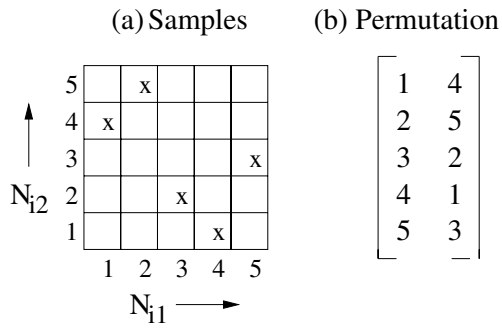

Figure 4: Space-filling sampling from the space of possible mixes via Latin Hypercube Sampling (LHS)

in any mix m_p. The model will have the form: $\hat{A}_{pj}$ = *model*$_j(m_p)$ = *model*$_j(N_{p1}, \ldots, N_{pT})$. Two key questions arise:

- *Sampling question:* How to efficiently generate a representative set of samples from which to train the model?
- *Modeling question:* What type of regression model gives the best accuracy in estimating query completion times in mixes?

The rest of this section considers the sampling question. The modeling question is considered in Section 6.

5.1 Latin Hypercube Sampling (LHS)

One straightforward sampling technique is to sample randomly from the space of possible mixes. Each selected sample would be a query mix that is run to observe its performance. However, random sampling is inefficient from the modeling perspective. The number of observed mixes required to learn a good model through random sampling can be very large, especially when the number of query types (i.e., the dimensionality of the mix space) is large; the reason being that mixes from the same local space may be repeated unnecessarily. The family of space-filling designs contains more efficient sampling techniques. *Latin Hypercube Sampling (LHS)* comes from this family and performs well in practice [17]. LHS selects n mixes from a space of T query types as follows:

S_1: The range of the possible number of instances of each query type is broken into n equal subranges.

S_2: n mixes are selected from the space such that each subrange of each query type has one and only one selected mix in it, and the minimum distance between selected mixes is maximized.

Intuitively, given the same number of mixes that can be selected, LHS gives better coverage of the mix space than random sampling.

Figure 4 shows an example where we have $T = 2$ query types, and we select $n = 5$ mixes using LHS. The two dimensions N_{i1} and N_{i2} denote the number of instances of each query type in a mix. LHS divides each of these dimensions into 5 equal subranges and selects the mixes to sample such that each subrange in each dimension has one sample. The "x" symbols in Figure 4 denote the set of mixes that LHS selects.

LHS samples are very efficient to generate because of their similarity to *permutation matrices* from matrix theory. Generating n LHS samples involves generating T independent permutations of $1, \ldots, n$, and joining the permutations on a position-by-position basis. The $T = 2$ permutations $\{1, 2, 3, 4, 5\}$ and $\{4, 5, 2, 1, 3\}$ were combined to generate the $n = 5$ LHS samples in Figure 4.

Although LHS is efficient and gives good coverage of the mix space, it cannot be applied directly in our setting for two reasons:

1. Database systems that process batch BI workloads predominantly run at a fixed MPL M: most actual mixes that run will have M query instances, and no mix will have $> M$ instances. Thus, it is more productive to sample query mixes that have close to, but no more than, M query instances.
2. As Section 5.2 shows, to build accurate models for prediction, the training samples should cover all possible *interaction levels (ILs)* of mixes. *The interaction level of a mix m is defined as the number of distinct query types in m.* More query types in a mix lead to more distinct kinds of query interactions.

Training	Test	Q_1	Q_7	Q_9	Q_{13}	Q_{18}	Q_{21}
TR_1	TT_1	21%	28%	26%	21%	20%	41%
TR_1	TT_2	62%	230%	101%	79%	54%	122%
TR_2	TT_1	32%	50%	45%	36%	31%	69%
TR_2	TT_2	15%	25%	19%	16%	15%	25%

Table 6: Prediction errors for different test sets of mixes

MPL and IL can be seen as two important meta-properties of a mix. A fundamental insight from our work is that these two meta-properties play a critical role in sampling and modeling, so they have to be treated specially. However, conventional LHS cannot handle our requirements regarding MPL and IL. Section 5.3 describes our new sampling algorithm that addresses this problem.

5.2 Making Sampling IL-Aware is Important

We present an empirical result to show that covering all ILs of mixes is important for accurate prediction on real workloads. The hypothesis tested is: Given a *test set* of mixes for performance prediction, *training* on the subspace of mixes with the same ILs as the test mixes can produce a more accurate prediction model than training on the subspace of mixes with ILs different from those in the test mixes.

We took two sets of TPC-H query mixes: (i) Set I contains mixes with IL $\in [1, 2, 3]$, and (ii) Set II contains mixes with IL $\in [4, 5, 6]$. We randomly picked 500 mixes from Set I as training mixes TR_1, and picked another disjoint 300 mixes from Set I as test mixes TT_1. Similarly, from Set II we created training mix set TR_2 with the same size as TR_1, and test mix set TT_2 with the same size as TT_1.

Table 6 shows the prediction errors—measured using the *mean relative error (MRE)* metric—for the four different combinations of training and test sets. MRE is defined as $\frac{|predicted - actual|}{actual} \times 100\%$ averaged over all the test mixes, where *actual* and *predicted* are the actual and predicted values of the run time of a given query in a test mix. The main observations from Table 6 are:

- When training mixes and test mixes come from similar sets of interaction levels, prediction is more accurate (i.e., lower error in Table 6).
- In both cases where training mixes and test mixes are from different ILs (the second and third rows in Table 6), the prediction errors are relatively high. However, using TR_2 to predict for TT_1 is much better than using TR_1 to predict for TT_2. Recall that TR_2 and TT_2 are both from IL $\in [4, 5, 6]$ which partially capture query interactions in mixes from IL $\in [1, 2, 3]$. So the model learned from TR_2 covers more query interactions than the model learned from TR_1.

5.3 IL-Aware LHS Algorithm

The above empirical exercise shows that we need to sample mixes that cover different ILs in order to make accurate predictions for real workloads. Meanwhile, there is a hard constraint that the number of concurrent query instances in a sampled mix should not exceed the fixed MPL M. The original LHS technique cannot handle either of these requirements. Therefore, we adapt LHS in order to make it *IL-aware* and also to satisfy the condition on MPL. Our modified, IL-aware LHS algorithm works as follows:

Goal: Sample n mixes for the given T query types and MPL=M:

- For a given IL=k, generate a permutation matrix P_k of size $n \times T$ using standard LHS. Each row in P_k represents a mix.
- For each mix (row) in P_k, randomly set the values of T-k of its columns to 0 to make its IL equal to k. Then, uniformly scale the nonzero values to make them integers such that their sum is as close to, but does not exceed, the given MPL M.
- Since our budget is n samples and we want to cover each IL, the number of mixes that are allowed for IL=k is $\texttt{ceil}(\frac{n}{num\_ILs})$ where *num_ILs* = $min(T, M)$. We randomly pick this many mixes from the n rows of P_k.

Section 8 shows empirically that our IL-aware LHS sampling generates training samples that produce good prediction models.

5.4 Incremental IL-Aware Sampling

As we demonstrate empirically in Section 8, very good predictions can be obtained by sampling around $10T$ mixes. At the same time, we also observe that $5T$ training samples or less are often good enough. Basic LHS as well as its IL-aware version are both designed to collect a batch of samples of a predetermined size. This batch nature of sampling creates a difficult decision problem for the DBA who has to decide how much time and resources to dedicate to sample generation before any predictions can be made. To simplify this decision-making process, we developed an *incremental* version of the IL-aware sampling algorithm that works as follows:

1. First, select the T samples with IL=1 which correspond to the "corner points" of the mix space of the form $\langle M, 0, 0, ..., 0\rangle$, $\langle 0, M, ..., 0\rangle$, . . ., $\langle 0, 0, ..., M\rangle$ (M=MPL). These samples cover the case where there are no significant interactions across different query types. Also, if the model has to be trained from a small number of samples, this step guarantees that every query type has at least one sample where its instances are nonzero.
2. Use the IL-aware LHS algorithm from Section 5.3 to plan a set S of n=$9T$ training samples to collect from the feasible ILs>1.
3. Generate a sequential list of samples by sampling randomly without replacement from S. Since the $10T$ samples from Steps 1 and 2 are distributed uniformly across all ILs, this step maintains a roughly uniform distribution across ILs as more samples are collected (a desired property from Section 5.2).
4. The DBA can collect samples incrementally based on the sequential list from Step 3. She can suspend the sampling process any time, and resume it later if the need for more accuracy is felt, e.g., based on comparing predicted workload completion times with the actual run times seen on the production system.

6. MODELING

We now turn to the question of which type of regression model to train from the collected samples. The machine-learning literature provides many candidate model types, some of which have been used in previous work on predicting single query completion times (e.g., [13]). Our selection of the model type is driven by two key differences between our problem scenario and previous ones. First, we cannot rely on the features of individual queries alone since we have to capture query interactions. Second, large numbers of training samples (a few 1000s) were considered available in previous problem settings. In contrast, we expect few 10s-100s of training samples per model because of the higher cost of sample generation, and we seek techniques that work well in these regimes. As a rule of thumb in machine learning, more complex model types can be more accurate if (and only if) trained well, but need more samples for accurate training. We have to walk this line carefully.

We tried conventional models like linear and quadratic regression, and found them suboptimal for predicting query completion times in mixes. Such models attempt to fit a predefined "structure" to the distribution of query completion times in the training samples, and then make predictions based on this fitted structure. This approach is hit or miss: the predictions of the model are accurate only if the true structure of the data distribution matches the assumed structure of the model.

The above observations motivated us to consider a different approach called *instance-based learning* [33]. The basic idea here is to infer the performance of a mix m_p of interest from the performance of mixes in the training set that are similar to m_p. (Similarity metrics can be derived, e.g., from the popular L_2 norm.) Intuitively, instance-based learning focuses on the "local space" around m_p at prediction time rather than using a structure learned previously. A simple type of instance-based learning is a *1-nearest neighbor (1-NN)* model. When asked to estimate A_{pj}, the average running time of query type Q_j in mix m_p, a 1-NN model returns the average running time of Q_j in the mix that is most similar to m_p among the training samples.

A *k-nearest neighbor (k-NN)* model is a more robust alternative to 1-NN models. A k-NN model finds the top k mixes most similar to m_p among the training samples—denoted $m_1, \ldots, m_k$ without loss of generality—and returns the estimate $\hat{A}_{pj} = \frac{\sum_{i=1}^{k} A_{ij}}{k}$. While k-NN is great in theory, it is hard to tune k for good accuracy because the density of samples can differ across regions of the full space. The parameter k degrades to a coarse and unpredictable tuning knob as k-NN focuses on the local space in dense regions and becomes more global in less dense regions.

An elegant alternative is to use all n training samples available, $\langle m_i, A_{ij}\rangle, 1 \leq i \leq n$, to estimate A_{pj} as a weighted combination of A_{ij} values from the individual training mixes. The weight of A_{ij} will be a measure of the correlation (similarity) between mixes m_i and m_p. We adopt this approach, and we leverage *Gaussian processes* from machine learning [33] to develop a new model called the *Gaussian Process model for query Mixes (GPM)*.

To estimate A_{pj} as a weighted combination of the A_{ij} values from the training samples, GPM needs a weight for each A_{ij} that reflects the correlation (similarity) between mixes m_i and m_p. There can be different choices for the correlation function, and we use the following function for GPM:

$$corr(m_i, m_p) = \Pi_{x=1}^{T} \texttt{exp}(-\theta_x |N_{ix} - N_{px}|^{\delta_x}) \qquad (2)$$

This function captures the intuitive phenomenon that the correlation between mixes m_i and m_p decreases as m_i and m_p become more dissimilar in terms of the number of instances of each query type. The parameters $\theta_x \geq 0$ and $\delta_x > 0$ are constants specific to query type Q_x. These constants capture the fact that each query type Q_x may have its own rate at which the performance impact changes as the number of instances of Q_x vary. The values of these constants are not predetermined, and are learned from the training samples using maximum likelihood estimation. Next, we describe GPM and how it uses the correlation functions.

GPM uses a simple trick to combine the best of conventional regression models and Gaussian processes. GPM models the *residuals* R_{pj}=$A_{pj} - \vec{f}^{t}(m_p)\vec{\beta}$ that denote the difference between true and estimated values after fitting a regression model (represented by $\vec{f}^{t}(m_p)\vec{\beta}$). Here, $\vec{f}(m_p)$=[$f_1(m_p)$,$f_2(m_p)$,. . .,$f_h(m_p)$]^t is a vector of basis functions for regression, and $\vec{\beta}$ is the corresponding $h \times 1$ vector of regression coefficients. The t notation represents the matrix transpose operation. An example is: $\vec{f}^{t}(m_p)\vec{\beta} = 0.1 + 3N_{p1} - 2N_{p1}N_{p2} + N_{p2}^2$. In this case, $\vec{f}(m_p)$=[1,N_{p1},

$N_{p2},\ N_{p1}N_{p2},\ N_{p1}^2,\ N_{p2}^2]^t$, and $\vec{\beta}$=$[0.1, 3, 0, -2, 0, 1]^t$. Given the actual training samples $\langle m_i, A_{ij}\rangle$ for a query type Q_j, we can train the $\vec{f}^t(m_p)\vec{\beta}$ model using popular techniques like *least-squares estimation* [33], and then generate the corresponding samples $\langle m_i, R_{ij}\rangle$. GPM models the residuals as Gaussian-distributed random variables, as described in the following definition.

DEFINITION 1. **Gaussian Process model for query Mixes (GPM):** *For any set of L query mixes m_i, $1 \leq i \leq L$, and corresponding residuals R_{ij} for a query type Q_j, the vector of random variables $[R_{1j}, R_{2j}, \ldots, R_{Lj}]$ has a multivariate Gaussian (MVG) distribution. (Note that any MVG can be described fully by its mean vector and matrix of pairwise covariances.) The mean vector of $[R_{1j}, \ldots, R_{Lj}]$'s MVG consists of all zeros. The covariance matrix of the MVG is given by $Cov(R_{xj}, R_{yj}) = \alpha^2 corr(m_x, m_y)$, $x, y \in [1, \ldots, L]$. α is a constant and $corr(m_x, m_y)$ is as per Equation 2.* □

GPM has the following property that we use for prediction [27].

PROPERTY 1. **Prediction using GPM:** *We are given training samples $\langle m_{ij}, A_{ij}\rangle$, $1 \leq i \leq n$, for a query type Q_j. For any mix m_p, GPM generates an estimate of A_{pj} that is Gaussian distributed with mean $\bar{A}_{pj}$ and variance v_{pj}^2. When a prediction ($\hat{A}_{ij}$) of A_{pj} is needed, we return a sample from this distribution. $\bar{A}_{pj}$ and v_{pj}^2 are defined as follows:*

$$\bar{A}_{pj} = \vec{f}^t(m_p)\vec{\beta} + \vec{c}^t\mathbf{C}^{-1}(\vec{A}_j - \mathbf{F}\vec{\beta}) \tag{3}$$

$$v_{pj}^2 = \alpha^2[1 - \vec{c}^t\mathbf{C}^{-1}\vec{c} + \frac{(1 - \vec{1}^t\mathbf{C}\vec{c})^2}{\vec{1}^t\mathbf{C}\vec{1}}] \tag{4}$$

where $\vec{c} = [corr(m_p, m_1), \ldots, corr(m_p, m_n)]^t$, $\mathbf{C}$ is an $n \times n$ matrix with element i, j equal to $corr(m_i, m_j)$, $\vec{A}_j = [A_{1j}, \ldots, A_{nj}]^t$, and $\mathbf{F}$ is an $n \times h$ matrix with the ith row composed of $\vec{f}^t(m_i)$. $\vec{1}$ is an $n \times 1$ vector of all 1s. □

Note that $\vec{f}^t(m_p)\vec{\beta}$ in Equation 3 is simply a plug in of mix m_p into the conventional regression model used in the GPM model. The second term in Equation 3 is an adjustment of the prediction based on the errors (residuals) seen at the training samples, i.e., $A_{ij} - \vec{f}^t(m_i)\vec{\beta}$, $1 \leq i \leq n$. Intuitively, the prediction at mix m_p can be seen as a combination of the estimate from the conventional regression model with a *correction term* computed as a weighted sum of the residuals for the training samples, where the weights are determined by the correlation function from Equation 2. The weights ensure that the prediction at m_p is affected more by the residuals of nearby mixes than by the residuals of far-away mixes.

7. DISCUSSION AND EXTENSIONS

We now discuss some design decisions as well as avenues for future extensions. Given the T query types that can appear in workloads, we use an IL-aware algorithm to generate sample query mixes, run the mixes to generate training data, and fit a GPM to this data. It is important to note that once this training is done (a one-time off-line process), the model and simulator can be used to estimate the completion time of any workload composed of any number of queries of these T types. New query types can be added efficiently to an existing set because we can generate new samples incrementally to update the GPM models.

It is also important to note that our technique does not place an unduly heavy burden on the DBA. The role of the DBA is explained in the workflow presented in Figure 3. The DBA needs to identify the scenarios where repeated prediction is needed for workloads consisting of a set of query types. To obtain these predictions, the DBA needs to identify the different query types. We describe elsewhere [3] automatic techniques for identifying query types given a query log, which can itself be collected using automated tools. Once the query types have been identified, our workflow generates training mixes, runs experiments for collecting performance data for these mixes, and trains performance models on the collected data. All these steps are off-line steps that are performed automatically and only once. The role of the DBA is mainly to schedule the sampling experiments in times where they would not interfere with the normal operation of the system as outlined in Section 5.4. Once the models are trained, the on-line simulator can be invoked by the DBA to predict the completion times of workloads, and this invokation is a very simple process.

Another role of the DBA is to decide when to retrain the performance model. New query types can be added to the model incrementally by collecting sample mixes that contain these query types. However, the question is: When is a complete retraining of the model required? Strictly speaking, any significant change in hardware, software configuration, or database characteristics should require model retraining. But in reality, models can remain accurate even in the face of such changes. One simple approach to decide when to retrain is for the DBA to monitor the accuracy of the completion time predictions, and decide to retrain the model only if this accuracy becomes unacceptable. More elaborate techniques are possible, but we leave them for future work. For example, we can use techniques such as those described in [21] to evolve models collected for one system and database configuration so that these models are accurate for another system and database configuration.

We have chosen to capture and simulate interactions in a database workload at the granularity of query types. A range of other options exist. A simpler scheme than ours is to categorize queries into CPU or I/O bound, so that two or more concurrent CPU (I/O) bound queries will interact negatively. A more complex scheme than ours involves capturing interactions at the level of different execution phases of physical operators in database query execution plans (e.g., hash join and sort). Note that our overall methodology of statistical modeling based on samples from planned experiments applies regardless of the modeling granularity. Our empirical results indicate that query-level modeling provides good accuracy. A possible avenue for future work is to use machine-learning techniques like *crossvalidation* [33] on the samples to determine the best modeling granularity automatically.

The sampling techniques from Section 5 are proactive in the sense that they deliberately execute a set of selected mixes to generate a representative sample set. An alternative approach, which we call *passive sampling*, relies on monitoring the execution of query mixes as part of the regular workload on the database system. Each time an instance of Q_j runs from start to finish in a mix m_i, an $\langle m_i, A_{ij}\rangle$ sample is obtained. It is possible to supplement proactive samples with passive samples that are available almost for free. Passive sampling by itself cannot guarantee a representative set of samples because passive sampling is at the mercy of the workloads seen and scheduling decisions made.

A number of solutions have been proposed recently to support proactive sampling techniques in a production environment without affecting the user-facing workload. Oracle 11g provides such a service called *Test-Execute* in the database system [7]. This service is used to test new plans for poorly-tuned queries. Systems to collect samples on standby databases and on virtualized resources in data centers are proposed in [11] and [36], respectively.

8. EXPERIMENTS

Our empirical evaluation was done using the TPC-H benchmark on a DB2 version 8.1 database server. The server has dual 3.4GHz Intel Xeon CPUs with 4GB RAM, and runs Windows Server 2003. TPC-H scale factors 1 and 10, denoted by 1GB and 10GB, respectively, were used. The buffer pool size of the database was set to 400MB for the 1GB database, and 2.4GB for the 10GB database. Configuration parameters and indexes were tuned for the TPC-H workloads using the DB2 Configuration Advisor and the DB2 Design Advisor.

Query Types: We consider all the 22 TPC-H query types except for Q_{15}. We omit Q_{15} which creates and drops a view for ease of implementation (and not because of a limitation of our prediction technique). In addition to query batches that include all the T=21 query types, we also considered batches containing instances of only the top 12 and the top 6 longest-running query types (when queries run alone in the system as shown in Tables 2 and 4). Thus, our batch workloads have $T = 6$, $T = 12$, or $T = 21$.

Scheduling policies: We consider the FIFO and Shortest-Job-First (SJF) scheduling policies. Since FIFO depends on the order in which queries are added to the batch, we systematically vary this order to create a number of different workloads. Each workload is defined by an *initialization parameter IQ* and an *arrival order skew* B. To construct a query batch, we first go through the given list of query types and add IQ instances of each type. Going through the list again in a round-robin fashion, we keep adding B instances of each query type to the batch until all the queries are added. Intuitively, as B increases, more queries of the same type are scheduled together by FIFO. For SJF scheduling, the average run time of instances of a query type running alone in the system is computed as shown in Tables 2 and 4, and then used as the estimated run time of all instances of that query type.

Workloads: For the 1GB database, we use $IQ = 10$ and $B = 5$, 25, and 50. We use 60 instances of each query type. Thus, batches with $T = 21$ query types consist of 1260 query instances, batches with $T = 12$ query types consist of 720 query instances, and batches with $T = 6$ query types consist of 360 query instances. Query instances are generated by instantiating TPC-H query templates with different parameter values chosen as per TPC-H rules. For these workloads, we use MPL $M \in \{5, 10, 20, 30, 40, 50\}$.

For the 10GB database, we use $IQ = 0$ and $B = 2$, 5, and 10. We use $T = 6$ and 10 instances of each query type. When running these workloads, we use MPL $M \in \{5, 10\}$. We limit the workload sizes and MPL for the 10GB database due to the long run times of queries on this database. Recall from Figure 2 that 60-query workloads on the 10GB database can take more than 5 hours to complete.

Performance Models: To build the performance models required by our workload simulator, we collect samples using the IL-aware LHS sampling algorithm (Section 5) and train a GPM from the collected samples (Section 6). We used the Weka data mining toolkit [33] to implement GPMs.

Methodology: By systematically varying the choice of query types, size of query batch, and scheduling policy, the parameter B, and MPL M, we generated 120 distinct and varied workloads. We run these workloads with a warm buffer pool and measure their completion times. These workloads have actual completion times ranging from 30 minutes to more than 5 hours. For each workload W, we compare W's actual completion time *act* with W's completion time predicted by our technique, *pred*. Our error metric is the *prediction error* computed as:

$$Prediction\ error = \frac{|pred - act|}{act} \times 100\%$$

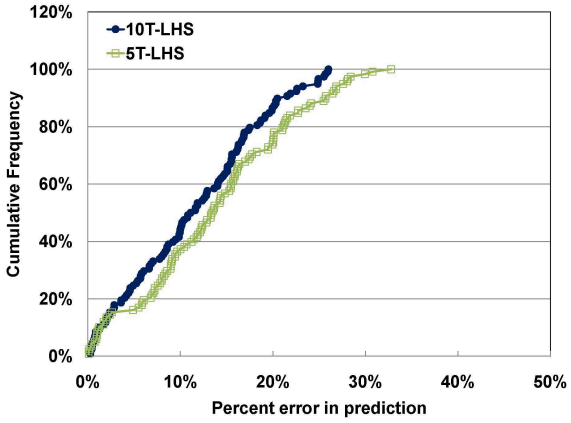

Figure 5: Prediction error across all workload runs for $5T$ and $10T$ samples

Section 8.1 analyzes the cumulative distribution of the prediction error across all 120 workloads. Section 8.2 studies the robustness and scalability of our solution.

8.1 Overall Accuracy

Our first experimental result demonstrates the overall accuracy of our predictions. Figure 5 shows the cumulative frequency distribution of the relative prediction error in all our 120 workload runs.[3] We consider two cases that correspond to the two plots in Figure 5. In one case, we use GPM models trained on $5T$ samples collected by our IL-aware LHS algorithm. In the other case, we use GPM models trained on $10T$ samples. (Recall that T denotes the number of query types.) Thus, for T=21 TPC-H query types, in the case of $5T$ and $10T$, we sample no more than 105 and 210 query mixes, respectively, to train the models. All samples are collected with a warm buffer pool.

Figure 5 shows that, in the case of $5T$ samples, about 75% of the time the prediction errors are less than 20%. If the database administrator (DBA) has a larger sampling budget and is willing to collect up to $10T$ samples, then the overall accuracy improves to the point where more than 85% of the time the prediction errors are less than 20%. These end-to-end results show that our sampling, modeling, and workload simulation algorithms result in accurate and robust predictions across a wide range of workloads. The DBA can now make accurate predictions for the future workloads in her database by collecting a small number of samples just once (which can be done along with initial system setup and workload tuning).

Our empirical observations indicate that $5T$ training samples provide sufficient accuracy for most cases. If the DBA is unsure about the time and resources to invest upfront for sampling, then she can use our incremental sampling algorithm from Section 5.4. Figure 6 shows the results for the incremental sampling algorithm. The plots with "-inc" suffix show how the prediction error drops as incremental sampling brings in $3T$, $5T$, and $7T$ samples over time. From the figure, we can see that even in case of $3T$ samples about 40% of the time the errors are less than 20%, and more than 60% of the time the errors are less than 30%. These accuracies may be enough for the workloads that a DBA is seeing, and she needs to go no further. The effectiveness of the approach is clear in that this case requires only 63 samples for 21 TPC-H query types (as opposed to 210 samples for $10T$). $5T$ incremental samples give

[3] An error line towards the upper-left corner represents lower error (higher accuracy) than one towards the lower-right corner.

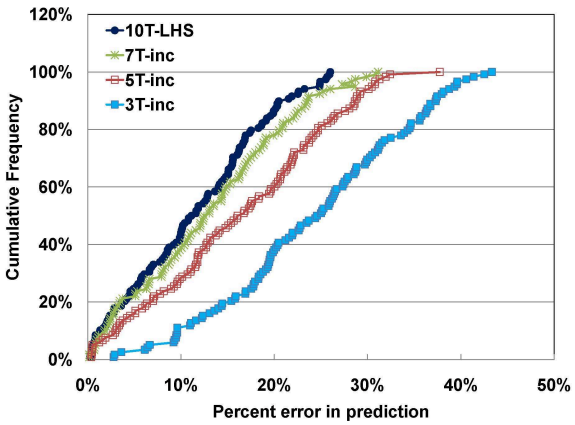

Figure 6: Prediction error across all workload runs for incremental sampling

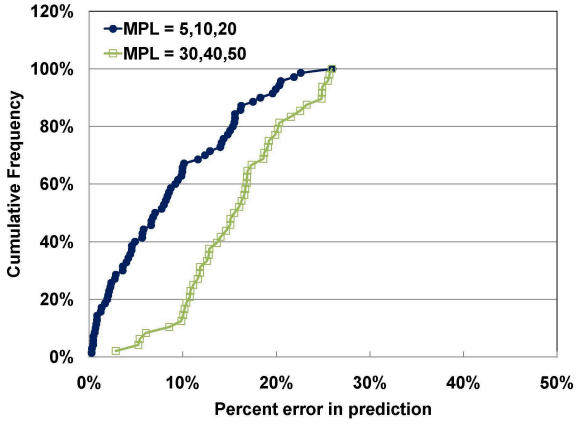

Figure 7: Prediction error across different MPL settings

Prediction scenario (given by DBA)	Training time	Ratio of training time to avg workload run time
M=5,T=21,1GB	0.6 hours	0.9 (1260-query workloads)
M=30,T=21,1GB	4.3 hours	3.9 (1260-query workloads)
M=10,T=6,10GB	15 hours	4 (60-query workloads)

Table 7: Training overhead of representative prediction tasks

us errors lower than 25% almost 80% of the time. The figure also shows that the results for $7T$ are very close to $10T$, so the DBA will often not need to go all the way up to $10T$.

The time required to obtain these predictions consists of the off-line time to collect samples and build the model, and the on-line time used by the simulator. The simulator run time is minimal: for the largest workload consisting of 1260 queries, the simulator takes less than 5 seconds overall. For workloads with fewer queries, the simulator run time is typically a fraction of a second. Table 7 shows the training overhead of our solution for a representative subset of prediction scenarios. The first column is the prediction scenario identified by the DBA. The second column shows the absolute training time: the time needed by our incremental sampling algorithm to generate samples and learn a model that gives $\leq$20% error for $\geq$80% of the time. Absolute training times do not tell the full story. The third column shows relative training time: the ratio of the absolute training time to the average time to run a workload corresponding to that scenario in our experiments. Notice that the training time is equal to the time to run a very small number of workloads. Our solution offers three important advantages:

- Recall that once the models are trained, they can be reused to give predictions for any number of workloads that match the prediction scenario.
- The relative training time drops as the number of query instances $|W|$ in the batch workload increases, since absolute training time is independent of $|W|$. Once we train the models, we can give predictions for workloads of any size. In practice, batch workloads can be large in size. For the 10GB case, a typical 60-query workload can take more than 5 hours to run, and a 1260-query workload can take more than 100 hours. In comparison, our training time is 15 hours.
- Incremental sampling gives the DBA the flexibility to collect more samples as and when resources are available. The training overhead can be spread out over time, and there is no need to allocate a contiguous block of (absolute) training time.

8.2 Robustness and Scalability

Next, we drill down into the accuracy data presented in Figure 5 to study different aspects of the performance of our techniques. In all the figures in this section, we use $10T$ training samples and omit other cases due to lack of space. The trends that we note in the following discussion were observed for all sample sizes.

Effect of MPL: Recall that to study the accuracy of our approach in different settings, we vary MPL from 5 to 50. Figure 7 shows the cumulative frequency distribution of error for the above 120 runs partitioned by MPL. The figure shows that in the case of lower MPLs $(5, 10, 20)$, around 95% of the predictions have error less than 20%. On the other hand, for higher MPLs, around 80% of the predictions have error less than 20%.

The increase in prediction error is due to two factors. At high MPLs, there is more load on the system. Thus, the variance in query completion times is higher, making prediction harder. (In practice, DBAs are very careful in setting MPL values to avoid regimes with high variance in performance.) The second factor that leads to reduced accuracy at higher MPLs is that the space of possible query mixes (from which we have to sample) grows substantially with increasing MPL. For MPL M and number of query types T, the total size of this space is the number of ways we can select M objects from T object types, unordered and with repetition. The number of possible selections is given by $S(T, M) = \binom{M+T-1}{M}$ [26]. Even for the simple case of 6 query types, we get $S(6, 20) = 53130$ and $S(6, 50) = 3478761$; a 65-fold increase in the size of the space to be modeled. However, in our experiments, we did not increase the number of samples, always collecting the same number of samples for every MPL for a given database size. Despite the high system load and explosion in the modeling space, we still maintain good prediction accuracy, which demonstrates the robustness of our approach. The current state of art offers no tools that DBAs can use to predict workload completion times with such degrees of accuracy even at low loads.

Sensitivity to workload parameters and scheduling policy: We now turn our attention to varying different parameters that can affect prediction accuracy, such as the number of query types, scheduling policy, and workload size. We study the change in error distribution as these settings vary. Results are shown across all MPLs.

Since our workload simulator makes predictions based on changing query mixes, we want to see how well it performs when the workload size changes. When there are more queries to schedule, the simulator will encounter more distinct mixes; so there is

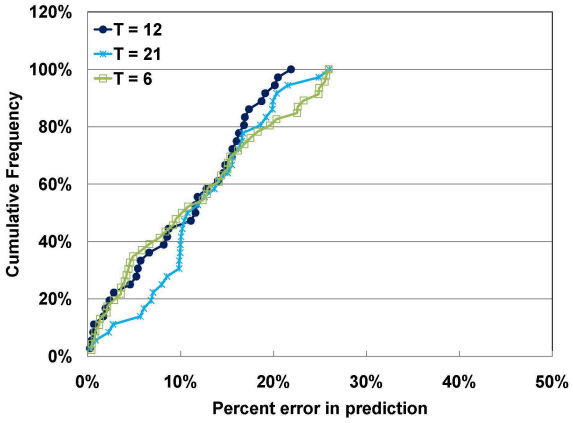

Figure 8: Varying workload size and number of query types

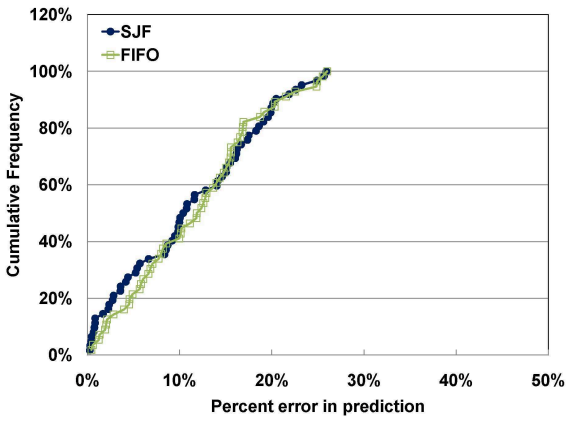

Figure 10: Prediction error for different scheduling policies

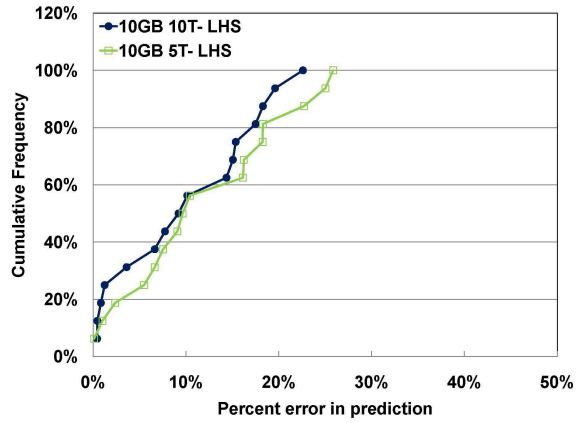

Figure 9: Prediction error for workloads on a 10GB database

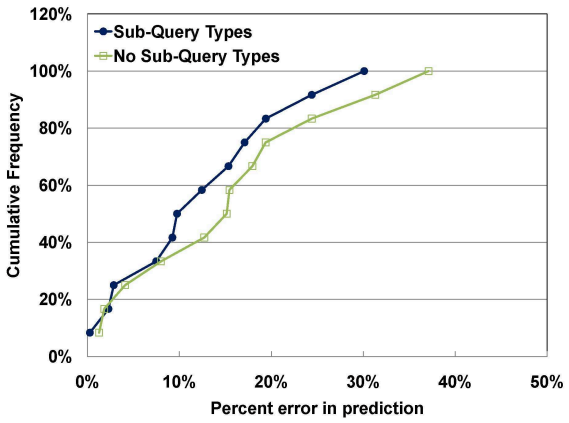

Figure 11: Prediction error for skewed data

a higher likelihood that it accumulates error in its prediction of workload completion time. To study the effect of workload size on accuracy, Figure 8 shows the error distribution for three types of workloads: (i) 1GB workloads that consist of 1260 queries picked from $T = 21$ distinct query types, (ii) 1GB workloads that consist of 720 queries picked from $T = 12$ distinct query types, and (iii) 1GB workloads that consist of 360 queries picked from $T = 6$ distinct query types and 10GB workloads that consist of 60 queries picked from $T = 6$ distinct query types.

The figure clearly shows the robustness and scalability of our approach when the number of query types is increased. In fact, we see slightly better accuracy for increased number of query types and larger workloads. Increasing the number of query types increases the sampling space. However, recall that when we use fewer query types, we use the longest-running query types in the system. When the workload consists solely of long-running queries that significantly interact with each other (in particular at high MPLs), there is more variance in completion time as compared to a workload where these long-running queries are separated by more predictable short-running queries.

Figure 9 shows the error distribution of the workloads that run on a 10GB database. These are long-running workloads, on average taking more than 4 hours to finish. The figure shows that even when we limit ourselves to only $5T$ samples, the prediction accuracy remains good, which shows that our approach is robust in the case of long-running queries even with a small number of samples.

Figure 10 shows prediction errors for the two different scheduling policies that we consider, FIFO and SJF. The errors are very similar for both policies, illustrating that our approach is robust to variations in the scheduling policy.

Sensitivity to data skew: So far we considered each of the 21 TPC-H query templates to be a distinct query type. These templates are parameterized, and for the same query type, the workload contains query instances with different values of the corresponding parameters. Because TPC-H has uniform data distribution by default, all instances of the same query type exhibit similar performance. However, in the presence of skewed data distributions, we may have to partition a query template into more than one query type to maintain the property that instances of the same query type have similar performance. We decide whether a query template needs such partitioning by (i) generating a large number of query instances with different parameter instantiations using the template, (ii) obtaining the corresponding query plans and their costs from the query optimizer, and (iii) clustering the plans based on the optimizer's estimated cost. This process is efficient because the plans need not be run. Full details are given in [3].

We report one empirical result to show the effectiveness of our approach. We used the skewed TPC-D/H database generator [28] to generate a 1GB TPC-H database that follows a Zipfian distribution with skew parameter $z = 1$. On this database, the skew-aware approach for identifying query types partitioned the Q_9 query template into 4 *sub-query* types. Figure 11 shows the error distributions

for workloads on this skewed database with 360 query instances comprising the 6 longest-running TPC-H query types. The figure shows the error for the skew-aware algorithm that partitions Q_9 into sub-query types, and for the skew-oblivious algorithm that does not partition Q_9. Both algorithms have good accuracy, but the skew-aware algorithm is more accurate, illustrating that our techniques can effectively handle data skew.

Overall, we see that the combination of our sampling, modeling, and workload simulation algorithms result in accurate and robust predictions across a wide range of workload settings.

9. CONCLUSION

It is important for a DBA in a business intelligence setting to be able to predict the completion time of different batch workloads. Such predictions are required, for example, to plan the execution of workloads in a batch window or to answer what-if tuning and capacity-planning questions. The state of the art does not offer tools or techniques that a DBA can use for this task. In this paper, we demonstrated that accurate prediction of workload completion times necessarily requires reasoning about interacting query mixes, and not individual query types. We presented an interaction-aware approach for predicting workload completion times that relies on: (1) principled interaction-aware sampling of the space of possible query mixes, (2) modeling the performance of queries in different mixes based on the collected samples using an instance-based statistical learning technique suited to the problem, and (3) a workload simulator that uses the performance models to simulate the query mixes that will be encountered during workload execution, and thereby predict the completion time of the workload. An experimental evaluation with the TPC-H benchmark running on IBM DB2 showed that our interaction-aware approach can predict workload completion times with high accuracy.

10. REFERENCES

[1] M. Ahmad, A. Aboulnaga, and S. Babu. Query interactions in database workloads. In *Proc. Int. Workshop on Testing Database Systems (DBTest)*, 2009.

[2] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Modeling and exploiting query interactions in database systems. In *Proc. ACM Conf. on Information and Knowledge Management (CIKM)*, 2008.

[3] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report generation workloads. *The VLDB Journal*, 2011. (to appear).

[4] M. Ahmad, S. Duan, A. Aboulnaga, and S. Babu. Interaction-aware prediction of business intelligence workload completion times. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2010. (short paper).

[5] Aster Data. http://www.asterdata.com/.

[6] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala. Automated experiment-driven management of (database) systems. In *Proc. Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[7] P. Belknap, B. Dageville, K. Dias, and K. Yagoub. Self-tuning for SQL performance in Oracle database 11g. In *Proc. Int. Workshop on Self Managing Database Systems (SMDB)*, 2009.

[8] G. Candea, N. Polyzotis, and R. Vingralek. A scalable, predictable join operator for highly concurrent data warehouses. *Proc. VLDB Endowment (PVLDB)*, 2(1), 2009.

[9] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In *Proc. Int. Conf. on Autonomic Computing (ICAC)*, 2006.

[10] U. Dayal, H. A. Kuno, J. L. Wiener, K. Wilkinson, A. Ganapathi, and S. Krompass. Managing operational business intelligence workloads. *Operating Systems Review*, 43(1), 2009.

[11] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *Proc. VLDB Endowment (PVLDB)*, 2(1), 2009.

[12] D. Feinberg and M. A. Beyer. Magic quadrant for data warehouse database management systems. Gartner Research Note, 2008. mediaproducts.gartner.com/reprints/microsoft/vol3/article7/article7.html.

[13] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Fox, M. Jordan, and D. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2009.

[14] S. Ghanbari, G. Soundararajan, J. Chen, and C. Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *Proc. Int. Conf. on Autonomic Computing (ICAC)*, 2007.

[15] Greenplum. http://www.greenplum.com/.

[16] C. Gupta, A. Mehta, and U. Dayal. PQR: Predicting query execution times for autonomous workload management. In *Proc. Int. Conf. on Autonomic Computing (ICAC)*, 2008.

[17] C. R. Hicks and K. V. Turner. *Fundamental Concepts in the Design of Experiments*. Oxford University Press, 1999.

[18] T. Kelly. Detecting performance anomalies in global applications. In *Proc. Workshop on Real, Large Distributed Systems*, 2005.

[19] S. Krompass, H. A. Kuno, J. L. Wiener, K. Wilkinson, U. Dayal, and A. Kemper. Managing long-running queries. In *Proc. Int. Conf. on Extending Database Technology (EDBT)*, 2009.

[20] A. Mehta, C. Gupta, and U. Dayal. BI Batch Manager: A system for managing batch workloads on enterprise data warehouses. In *Proc. Int. Conf. on Extending Database Technology (EDBT)*, 2008.

[21] M. P. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Relative fitness modeling. *Comm. ACM*, 52(4), 2009.

[22] MySQL log profiler and analyzer. http://myprofi.sourceforge.net.

[23] K. O'Gorman, A. El Abbadi, and D. Agrawal. Multiple query optimization in middleware using query teamwork. *Software - Practice and Experience*, 35(4), 2005.

[24] O. Ozmen, K. Salem, M. Uysal, and M. H. S. Attar. Storage workload estimation for database management systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2007.

[25] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. *SIGMOD Record*, 29(2), 2000.

[26] H. J. Ryser. *Combinatorial Mathematics*. The Mathematical Association of America, 1963.

[27] T. J. Santner, B. J. Williams, and W. Notz. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[28] Skewed TPC-D data generator. ftp://ftp.research.microsoft.com/users/viveknar/ TPCDSkew/.

[29] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2008.

[30] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Proc. European Conference on Computer Systems (EuroSys)*, 2007.

[31] S. Tozer, T. Brecht, and A. Aboulnaga. Q-cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In *Proc. Int. Conf. on Data Engineering (ICDE)*, 2010.

[32] TPC-H. http://www.tpc.org/tpch/.

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

[34] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Middleware*, 2007.

[35] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proc. Int. Conf. on Autonomic Computing (ICAC)*, 2007.

[36] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner. JustRunIt: Experiment-based management of virtualized data centers. In *Proc. USENIX Annual Technical Conference*, 2009.