Query Interactions in Database Workloads

Mumtaz Ahmad
University of Waterloo

Ashraf Aboulnaga
University of Waterloo

Shivnath Babu
Duke University

ABSTRACT

Database workloads consist of *mixes* of queries that run concurrently and interact with each other. In this paper, we demonstrate that query interactions can have a significant impact on database system performance. Hence, we argue that it is important to take these interactions into account when characterizing workloads, designing test cases, or developing performance tuning algorithms for database systems. To capture and model query interactions, we propose using an experimental approach that is based on sampling the space of possible interactions and fitting statistical models to the sampled data. We discuss using such an approach for database testing and tuning, and we present some opportunities and research challenges.

1. INTRODUCTION

Characterizing a database workload requires understanding the impact of this workload on all aspects of the system. Such workload characterization is required, for example, when designing workloads and test cases to test database system features or performance, when tuning a database system in deployment, or when analyzing the performance of a production system. To this end, many benchmark workloads exist that consist of queries and transactions that try to stress different aspects of the database system [15].

The typical workload in a database system consists of *mixes* of queries of different types running concurrently and interacting with each other. The interaction among queries can have a significant impact on their performance, and this impact can be positive or negative. For example, a query Q_1 can bring data into the buffer pool that is then used by a concurrently running query Q_2. 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. In this paper, we demonstrate the impact of query interactions and discuss their implications on database testing and performance tuning.

To illustrate the need for reasoning about query interactions, consider the TPC-H benchmark, which is widely used to test decision support systems. Studying the performance characteristics of individual TPC-H query types gives us insights into database system performance, but it is not enough to answer questions such as the following: If we run 5 concurrent TPC-H queries, namely, 2 instances of Q_1 with 3 instances of Q_{13}, then what would be the performance of the system? How would a single instance of Q_1 behave compared to the case of running 5 concurrent instances of Q_1? Looking at Q_1 alone or Q_{13} alone does not allow us to accurately answer these questions. We need to incorporate interactions among concurrent queries into our understanding of workload characteristics and database system performance.

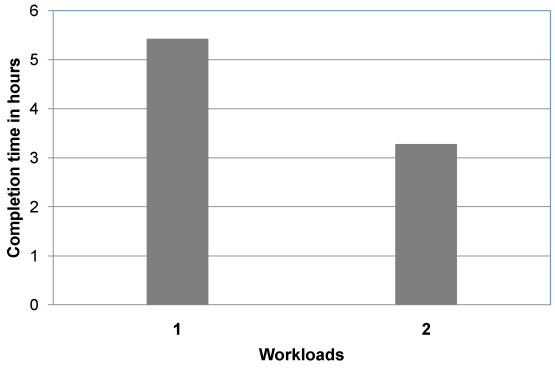

Figure 1: Workload completion time for different arrival orders.

As an example of the effect of query interactions on performance, consider the two workloads shown in Figure 1. Both these workloads consist of exactly the same 60 instances of TPC-H queries running on a 10GB database on DB2 (details of our experimental setup are presented in the next section). The database physical design and the tuning parameters of DB2 are exactly the same for both workloads. The only difference between the two workloads is the arrival order of the queries, which results in *different query mixes* being executed by the system. This simple change results in the completion time changing from 3.3 hours to 5.4 hours. In Workload 1, queries that compete for resources get executed concurrently, resulting in negative interactions. In Workload 2, queries that help each other get executed together, resulting in positive interactions. The 2.1 hour difference in performance is completely attributable to different query

DBTest'09, June 29, 2009, Providence, Rhode Island, USA.

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 1: Run time, t_j, of different TPC-H query types 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 2: Run time, t_j, of different TPC-H query types on a 10GB database.

interactions in the different runs. We show later that query interactions affect not only end-to-end workload completion time but also resource consumption. Therefore, we argue that it is important to consider query interactions when answering questions about different aspects of a database system such as testing, tuning, capacity planning, and performance prediction.

Surprisingly, very little work in the database literature deals with studying query interactions in the general sense. There are specific research works like work on multi-query optimization (e.g., [12]), and work on sharing scans in the buffer pool (e.g., [10]). Some works deal with transaction mixes but define a transaction mix as the set of transactions observed during a monitoring interval, without considering the concurrent execution of these transactions and the interactions that this concurrent execution induces. Such interaction-oblivious mix models have been used for performance prediction, capacity planning, and anomaly detection [14, 18]. In this paper we show that the interactions ignored by these works can have a significant impact on performance.

Our model of a database workload is that it consists of a *sequence of query mixes*. Some previous work models a workload as a sequence of queries, but ignores the effect of concurrent query execution [1]. To study the effect of query interactions on an entire workload, we start by studying the query interactions within individual query mixes. We assume that the query types that appear in the mixes are known a-priori. A query mix consists of a number of different instances of each query type, where different instances of a query type may have different parameter values. It is relatively simple to determine the query types a-priori if the query workload is generated by a fixed set of applications (e.g., report generation applications in a business intelligence setting). Having a fixed set of applications is a common mode of operation for many database systems, so the discussion in this paper is widely applicable. However, there are cases where a query workload is purely ad-hoc. In these cases, an extra step is required to classify the ad-hoc queries into a fixed set of query types. This extra step is beyond the scope of this paper and is an interesting direction for future work.

In this paper, we present some results from an experimental study of the impact of query interactions on performance and resource consumption. We present examples of interactions and their effect in Section 2, and we discuss the implications of these interactions in Section 3.

2. EXAMPLES OF QUERY INTERACTION

To illustrate the effect of query interactions in different query mixes on run time and resource consumption, we conducted an experimental study using queries from the TPC-H benchmark with two database sizes, 1GB and 10GB. The database system we use is DB2 version 8.1, and we ran our experiments on machines with dual 3.4GHz Intel Xeon CPUs and 4.0GB of RAM running Windows Server 2003. The buffer pool size of the database was set to 400MB for the 1GB database, and 2.4GB for the 10GB database. We used the DB2 Design Advisor to recommend a set of indexes for our workload, and we ran the DB2 Configuration Advisor to ensure that the configuration parameters are well tuned.

Let $Q_1, Q_2, \ldots, Q_{22}$ be the 22 TPC-H query types. We use query mixes consisting of different numbers of instances of different TPC-H query types, where the instances have different parameter values as required by the TPC-H specification (the queries are generated using the TPC-H QGEN program). Table 1 shows the run time of the 6 longest running TPC-H queries on a 1GB database when they run alone in the system, which we denote by t_j. Table 2 shows the run time of these queries on a 10GB database. Each run time represents the average run time of 10 instances of the particular query type. There is little variance in run time for a specific query type since TPC-H uses uniform distributions for data and query parameters. We also experimented with skewed (Zipfian) data distributions and observed significant effects of query interactions in this setting. Our approach to dealing with skewed data distributions is to sub-divide each query type into sub-types according to the range of parameter values to minimize the variance in run time within a query type. We omit a detailed discussion of skewed distributions from this paper.

The multi-programming level (MPL) of the database is M. The MPL represents the number of queries that execute concurrently in the system at any time. A set of queries that execute concurrently in the system is referred to as a *query mix*. 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 $\sum_{j=1}^{T} N_{ij} = M$. We denote the average run time of queries of type Q_j in mix m_i by A_{ij}.

We start with a simple example demonstrating the impact of interactions in a query mix on the completion time of a given query type in this mix. Table 3 shows three mixes consisting of the 6 long-running query types on the 10GB database. The high variability in A_{ij} illustrates the effect of query interactions. Consider the average run time of Q_1 and Q_7 in the first two mixes. Both mixes have $M = 5$, and both have one instance each of Q_1 and Q_7, but there is an increase in A_{ij} in m_2 for all query types. In particular, the run time of Q_7 is more than twice its time in m_1. One may be tempted to think that this is just because of the characteristics of Q_{13} which was introduced in m_2. The next mix m_3 shows that this is not true. In this mix, both Q_1 and Q_7 actually improve their performance from m_2, even when we increase the number of instances of Q_{13}. The effect of query interactions in a 1GB database can be seen in Table 4. Consider the average run time of Q_{21} in the

Mix	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
	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	1	1093.14	1	578.36	3	1190.15	0	0.0	0	0.0	0	0.0
m_2	1	1794.97	1	1261.39	2	2638.62	1	432.12	0	0.0	0	0.0
m_3	1	1186.74	1	663.97	0	0.0	3	311.53	0	0.0	0	0.0

Table 3: A_{ij} for different query types in query mixes on a 10GB database.

Mix	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
	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_4	8	114.4	2	45.76	1	193.16	4	71.12	4	111.87	1	55.38
m_5	2	109.88	6	88.13	3	191.48	5	61.23	3	114.21	1	159.95

Table 4: A_{ij} for different query types in query mixes on a 1GB database.

Mix	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
	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_6	1	1897.4	2	72.7	5	2919.3	0	0.0	2	1904.1	0	0.0
m_7	4	538.0	0	0.0	0	0.0	0	0.0	1	539.3	0	0.0
m_{7a}	4	537.98	0	0.0	0	0.0	0	0.0	1	541.39	0	0.0
m_{7b}	4	542.94	0	0.0	0	0.0	0	0.0	1	538.26	0	0.0

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

two mixes m_4 and m_5. Both these mixes have $M = 20$, yet A_{ij} for Q_{21} in m_5 is almost three times that for m_4. The performance of these mixes and all other mixes used in this paper is repeatable and consistent across different runs of the experiment.

Next, we present interesting cases of "positive interactions." Table 5 shows some query mixes for this setting. Mix m_6 in this table presents an example of positive interaction for Q_7. The average run time of Q_7 in this mix, A_{ij}, is 72.7 seconds, while the run time of Q_7 when it is run alone in the system is 102.06 seconds (Table 2). Thus, Q_7 *benefits* from being run in this mix, taking less time on average than if it were run alone. Mix m_7 presents another example of positive interaction, this time for Q_{18}. The average run time of Q_{18} in this mix is 539.3 seconds, compared to a run time of 554.56 seconds when it is run alone. Thus, Q_{18} benefits from being run with 4 instances of Q_1. Mixes m_{7a} and m_{7b} show repeated runs of mix m_7 with different instances of the same query types. The results for all variants of m_7 are similar, illustrating the repeatability of our results. The above examples show that query interactions can be negative (where $A_{ij} > t_j$) or positive (where $A_{ij} < t_j$). Interestingly, mix m_6 exhibits both positive and negative interactions: Q_7 benefits from running in this mix, but the performance of the other three query types is severely degraded.

Next, we demonstrate that query interactions can be fairly complex, with small changes in the query mix sometimes having a huge impact on performance that may be very difficult to predict. In Table 6, we focus on three-way interactions for mixes with one instance of Q_{21} on the 10GB database. In all examples we have $M = 5$. The completion time first increases with the introduction of an instance of Q_9, then it decreases and increases alternatively as we keep increasing the number of instance of Q_9 and decreasing the number of instances of Q_{13}. The same behavior for three-way interaction can be seen on the 1GB database in Figure 2. Here we fix $N_{ij} = 3$ for Q_{21} and vary the number of instances of Q_7 and Q_9 such that M is always fixed to be 30. Once again we can see that A_{ij} for Q_{21} varies significantly with no easily predictable pattern.

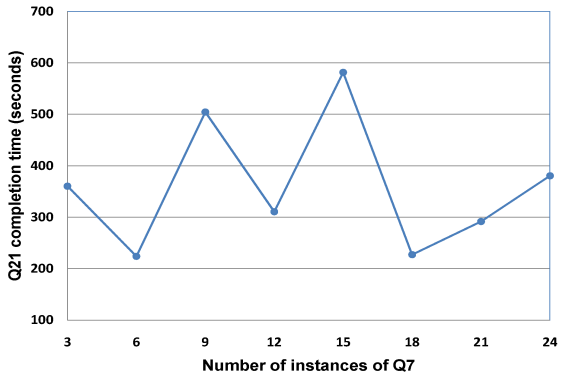

Figure 2: Three-way interaction: effect on Q_{21} of different mixes.

mix	$Q9$ (N_{ij})	$Q13$ (N_{ij})	$Q21$ (N_{ij})	$Q21$ (A_{ij})
m_8	0	4	1	4188.20
m_9	1	3	1	5463.80
m_{10}	2	2	1	3476.1
m_{11}	4	1	1	3581.7
m_{12}	4	0	1	2782.4

Table 6: A_{ij} for Q_{21}.

These examples demonstrate that we cannot accurately predict the performance of queries unless we are able to model the effect of other queries running concurrently with them in the query mix. Focusing on individual query types and ignoring interactions can lead to inaccurate conclusions about performance. Thus, it is important to develop mix-based characterization of query workloads to better understand the performance of database systems.

Next, we present experiments measuring resource consumption in different query mixes. Here again we will see that traditional approaches that profile the resource consumption of individual queries and workloads while ignoring interactions may not be useful. When queries run concurrently, resource utilization and performance bottlenecks can

mix	$Q13$ (N_{ij})	$Q9$ (N_{ij})	$Q13$ (A_{ij})	$Q9$ (A_{ij})	CPU Utilization	Sec / Disk Transfer
m_{13}	0	5	0	919	24.11	0.0253
m_{14}	1	4	356	1547.16	27.43	0.025
m_{15}	2	3	422.59	2079.72	19.68	0.0228
m_{16}	3	2	388.74	2508.33	4.97	0.0098
m_{17}	4	1	289.1	3762.55	53.145	0.026
m_{18}	5	0	224.42	0	86.055	0.01655

Table 7: Resource consumption for different mixes of Q_{13} and Q_9 on a 10GB database.

mix	$Q13$ (N_{ij})	$Q21$ (N_{ij})	$Q13$ (A_{ij})	$Q21$ (A_{ij})	CPU Utilization	Sec / Disk Transfer
m_{19}	0	5	0	1300.735	5.785	0.0072
m_{20}	1	4	372.05	2196.81	9.3	0.0114
m_{21}	2	3	436.62	2283.41	14.38	0.0128
m_{22}	3	2	322.71	2576.06	30.68	0.0177
m_{23}	4	1	206.88	4188.2	59.36	0.0215
m_{24}	5	0	224.42	0	86.055	0.01655

Table 8: Resource consumption for different mixes of Q_{13} and Q_{21} on a 10GB database.

change considerably from one mix to another. Tables 7 and 8 show the resource consumption of different mixes with two-way query interaction and $M = 5$ on the 10GB database. The tables report average CPU utilization (in %) and average seconds per disk transfer for the different mixes. Seconds per disk transfer is a direct measure of disk response time including the queueing time (so it captures the effect of varying load). In both tables we run Q_{13} with one other query type and observe the resource consumption of the mixes. As expected, resource consumption varies as we vary the query mix. However, what is interesting is that even when we have only two query types, just replacing an instance of one query type with an instance of the other can significantly change resource consumption, further demonstrating the significance of query interactions. Consider mixes m_{13} to m_{18} which all consist of instances of query types Q_{13} and Q_9. In m_{16}, the CPU utilization and disk transfer time are significantly lower than m_{15} and m_{17}. The CPU utilization for m_{18} is considerably higher than m_{17}. All these changes are the result of changing just one query instance from one mix to the next. The pattern of resource consumption is complex and rapidly changing due to the nature of query interactions. In many cases the bottleneck resource is neither CPU nor disk, but some other resource not being monitored such as locks, memory, database or operating system latches, etc. It is clear from these tables that considering query mixes is important for answering questions not only about query run time but also about resource consumption. Another interesting observation from these tables is that there is little correlation between resource consumption and query run time. This is further illustrated by Figures 3 and 4. The figures plot CPU utilization and seconds per disk transfer in different query mixes (on the 10GB database) against the average completion time of Q_9 in these mixes. There is no clear correlation between mix resource consumption and query completion time.

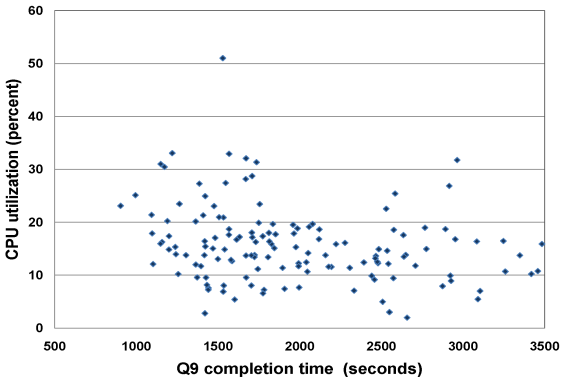

Figure 3: CPU utilization vs. Q_9 completion time.

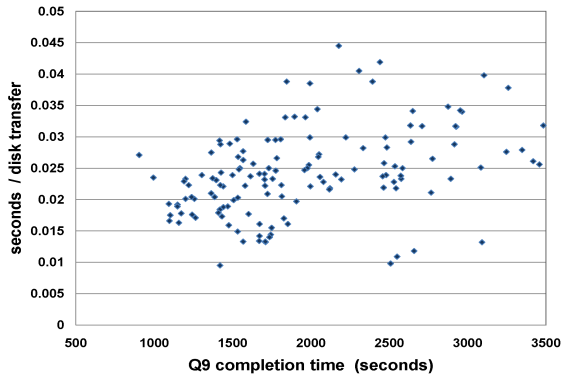

Figure 4: Disk performance vs. Q_9 completion time.

3. IMPLICATIONS OF QUERY INTERACTION

After presenting examples of query interactions and demonstrating the significant impact that they have on performance and resource consumption, we now ask the question: How do these interactions affect database testing and tuning? In this section, we present some implications of query interactions on database testing, and some challenges that arise due to these interactions.

Sampling and Test Case Design: An important consequence of query interactions is that the design of test cases and workloads for testing and evaluating database systems should take these interactions into account. For example, consider the two workloads in Figure 1. These workloads use the same queries, and the only difference between them is in query interactions. If database testing is not interaction aware, two test runs may use these queries in the two orders shown in Figure 1, resulting in a 2.1 hour difference in run time that is not due to any feature of the database system being tested, but rather due to query interactions.

To make database testing and tuning interaction aware,

we need to view the problem of designing test cases and test workloads as a problem of *sampling from the space of possible query interactions*. An important challenge is how to design the test workloads to maximize coverage of the space of possible query interactions while minimizing the sampling budget. Principled sampling approaches such as Latin Hypercube Sampling (LHS) [8] can help address this challenge, and it is important for database testers to employ such approaches and design interaction-aware test cases.

In addition to the "active sampling" approach proposed above, it may be possible to obtain useful information about query interactions through "passive sampling" from the workloads in a production system. If we (passively) monitor the execution of production workloads, we could determine which query mixes were actually encountered in these workloads, how long each mix ran, what effect each mix had on resource consumption, etc. This passive sampling cannot guarantee the same comprehensive coverage of the space of possible query mixes as active sampling, since it is restricted to mixes that have actually been observed and does not provide information about potential mixes that have not been seen yet. However, passive sampling can provide a database tester, DBA, or automatic tuning tool with a compact and useful characterization of the workload on the system.

Performance Modeling: Sampling the space of possible query interactions is a first step towards understanding the effect of these interactions on performance. To build performance models that reflect the effect of query interactions, we advocate the use of "black-box" modeling techniques that fit statistical models to the observed sample data.

It may be possible to build "white-box" performance models that capture the effect of query interactions, but as the systems and the query workloads become more complex, it becomes increasingly difficult to build such models from the ground up. Query interactions can happen at many different levels and their effect on performance can be subtle, so an interaction-aware white-box model would need to consider hardware resources (CPU, memory disks, and disk controllers), database internals (query plan, access methods, buffer pools, working memory, locks, latches), operating system internals (scheduling, latches, queuing delays), and possibly other factors. All these factors would need to be monitored and would need to be reflected in the model. To avoid having such complex (and likely brittle) models with high monitoring requirements, we advocate moving to an experimental modeling approach. Note that such an approach has long been used in the Internet Measurement community (e.g., [4]) as a way to model the performance of a very complex system. Work that uses a black box modeling approach are also starting to appear in the database literature [7].

To use experimental modeling techniques, we need to fit a statistical model to the observed performance in our samples. Our goal is to obtain a set of functions of the form

$$y = f(x_1, x_2, \ldots, x_n)$$

where y is the performance characteristic that we are modeling for a specific mix (e.g., CPU utilization, average completion time of a specific query type, etc.), x_i is the number of queries of type i in the mix (for the purpose of modeling, a mix is defined by a vector of x_i's), and $f(.)$ is a function representing the statistical model. The form of $f(.)$ depends on the type of model that we are using (the *model structure*). There are many well-known model structures, such as linear

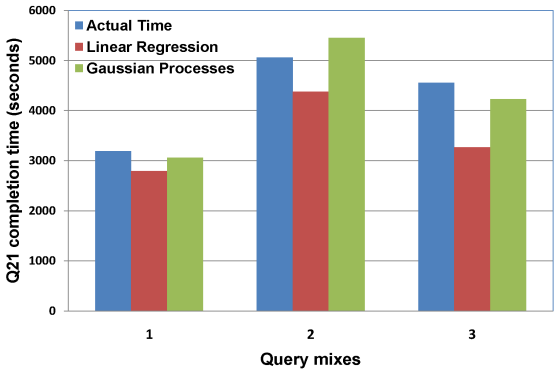

Figure 5: Accuracy of modeling.

regression, regression trees, locally weighted linear regression [17], Gaussian processes [13], and others. All of these model structures can be fit to the observed sample data (the model training data) using well-known techniques. Choice of the model structure impacts model accuracy, but if the training data is representative, then a good model can be typically found easily. In our work, we have found linear regression and Gaussian processes to be good model structures that are accurate for a broad spectrum of query mixes.

To illustrate the practicality and accuracy of our proposed approach, we present an example of its applicability to predicting query completion times in different mixes. For this example, we use LHS to collect 24 sample query mixes on the 10GB database, and we use the Weka data mining toolkit [16] to build models for the completion time of the different query types in different mixes. Figure 5 shows the average completion time of Q_{21} in three different mixes on this database. The figure shows the actual measured completion time, and the predicted completion time using two model structures in Weka: linear regression and the more advanced Gaussian processes [13]. From the figure we can see that, as we saw before, the average completion time of Q_{21} varies significantly from mix to mix. The figure also shows that linear regression can predict the completion time with some accuracy, and that Gaussian processes are more accurate than linear regression. Thus, we can see that (1) the sampling and experimental modeling approach works even with a small number of samples, (2) simple models such as linear regression can be reasonably accurate, and (3) with careful choice of more advanced modeling techniques we can achieve very high accuracy. We have observed similar accuracy results in other experiments. In one experiment, we measured the accuracy of the model on a test workload of 20 random mixes. We found the mean relative error to be 31% for linear regression and 20% for Gaussian processes.

Accurate performance models are a very useful tool for characterizing and understanding query interactions. For example, they can inform a tester or DBA about parts of the space of possible interactions that place a high load on resources. These models can also indicate which queries interact negatively with each other and which interact positively. Thus, they can serve as a basis for focusing testing and tuning efforts.

Interaction-aware Tuning: Understanding query interactions through sampling and performance modeling can be used to improve the solutions to many administration and performance tuning problems in database systems. For ex-

ample, in our prior work we have developed an interaction-aware query scheduler for long-running queries that was able to improve performance by up to 4x [2]. The scheduler schedules batches of long running queries in a report generation setting. It relies on sampling the space of possible mixes and fitting linear regression models to predict query completion times in different mixes. The models are used in a linear programming formulation of the scheduling problem, and the solution of the linear program is the basis of the interaction-aware schedule.

Since query interaction has such a significant effect on run time, it would be useful to develop interaction-aware techniques for predicting the completion time of complex query workloads, or the progress of workloads or specific queries. It may also be beneficial to incorporate interaction awareness into other database system components such as query optimizers or physical design advisors. All of these possibilities represent interesting avenues for future research.

The notion of a database workload as a mix of queries has been taken into account by prior work in database system tuning as discussed in Section 1. While such works are definitely relevant, in this paper we show that query interactions can have complex and subtle effects, and we advocate an end-to-end view of query interactions and using experimental modeling to capture their effects.

Interactions in Other Contexts: While the focus of this paper is on interaction awareness in traditional database systems, we point out that interactions can arise in other database contexts, and it should be possible to extend our approach of sampling and experimental modeling to these contexts. For example, a particularly interesting area for extension is parallel dataflow execution frameworks such as Map-Reduce [6] or Dryad [9]. In a typical usage scenario for these frameworks (and for systems that build on top of them such as Pig [11] or SCOPE [5]), there is a set of related, long-running, resource-intensive tasks that execute concurrently (e.g., Map-Reduce tasks). We believe that it may be very useful to develop scheduling and tuning algorithms that take into account the interactions between these tasks. For us, this is still an open area of exploration.

Leveraging Computing Clouds for Experimentation: Our approach to modeling query interactions requires running experiments to sample the space of possible query interactions. An important question when adopting this approach is where to run these sampling experiments? The production system may not be available for extensive experimentation, and there may not be an available test system. Testing and tuning always requires experimentation, so whether the approach is interaction-aware or not, some capacity for experimentation has to be provided. The emergence of computing clouds such as Amazon's EC2 [3] makes experimentation much easier by providing virtually unlimited computing capacity for running experiments. So if the experiments required to sample the space of possible interactions cannot be run on the production or a test system, it should be possible to implement a full sampling and testing infrastructure that can easily be run on Amazon's EC2. This is especially relevant if the production system is itself running on Amazon's EC2.

4. CONCLUSIONS

In this paper, we show that query interactions have a significant effect on database system performance. Thus, it is important to take these interactions into account in database testing and tuning. We advocate using an experimental sampling and modeling approach for capturing query interactions, and we discuss some potential benefits of interaction awareness. The paper outlines several opportunities and research directions for database testing and tuning.

5. REFERENCES

[1] S. Agrawal, E. Chu, and V. R. Narasayya. Automatic physical design tuning: Workload as a sequence. In *SIGMOD*, 2006.

[2] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Modeling and exploiting query interactions in database systems. In *CIKM*, 2008.

[3] Amazon Elastic Computing Cloud. http://aws.amazon.com/ec2/.

[4] N. Brownlee and K. C. Claffy. Internet measurement. *IEEE Internet Computing*, 8(5), 2004.

[5] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and efficient parallel processing of massive data sets. In *VLDB*, 2008.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[7] 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 *ICDE*, 2009.

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

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

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

[11] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A not-so-foreign language for data processing. In *SIGMOD*, 2008.

[12] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD*, 2008.

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

[14] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *EuroSys*, 2007.

[15] Transaction processing performance council (TPC). http://www.tpc.org/.

[16] Weka 3: Data mining software in Java. http://www.cs.waikato.ac.nz/ml/weka/.

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

[18] Q. Zhang, L. Cherkasova, and E. Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *ICAC*, 2007.