26th IEEE International Conference on Data Engineering (ICDE 2010).

Q-Cop: Avoiding Bad Query Mixes to Minimize Client Timeouts Under Heavy Loads

Sean Tozer, Tim Brecht*, Ashraf Aboulnaga

David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada
{srtozer,brecht,ashraf}@cs.uwaterloo.ca

***Abstract*— In three-tiered web applications, some form of admission control is required to ensure that throughput and response times are not significantly harmed during periods of heavy load. We propose Q-Cop, a prototype system for improving admission control decisions that considers a combination of the load on the system, the number of simultaneous queries being executed, the actual mix of queries being executed, and the expected time a user may wait for a reply before they or their browser give up (i.e., time out). Using TPC-W queries, we show that the response times of different types of queries can vary significantly depending not just on the number of queries being processed but on the mix of other queries that are running simultaneously. We develop a model of expected query execution times that accounts for the mix of queries being executed and integrate this model into a three-tiered system to make admission control decisions. Our results show that this approach makes more informed decisions about which queries to reject and as a result significantly reduces the number of requests that time out. Across the range of workloads examined an average of 47% fewer requests are unsuccessful than the next best approach.**

I. Introduction

Web applications are typically structured as three-tier systems, with a web server, an application server, and a database system. In these applications, the database tier is typically the performance bottleneck, so managing the performance of the database system is important for the overall performance of the web application.

Managing database system performance is especially important under conditions of heavy load. As the system approaches overload, throughput will decrease and response time will increase. Since clients have a timeout beyond which they stop waiting for the server to respond (either due to limits on human patience or time limits set in browsers), the increase in response time will result in more timeouts for client requests. In the limit, response time will increase to the point where very few requests are served within their timeout period.

Under loads that exceed the capacity of the server, the system must intelligently decide which requests to process. That is, it must perform *admission control* or suffer from significant performance degradation when compared with its peak performance. The system should only process a request it will be able to finish within the timeout period. For requests that will not finish within that time, the database system should produce some type of feedback to higher tiers, perhaps returning a "system overloaded" error message to the application server. Without such load regulation, the database system can spend a large amount of time working on requests that ultimately time out at the client. This wastes server resources that could have been better allocated to other requests. From the perspective of the client, requests will time out and the sever will appear unresponsive or the server will drop connections and appear unavailable. A common reaction in this case is for the client to immediately retry the failed request, which exacerbates the overload problem at the server. On the other hand, if the database system returns an overload error to the application server, the web application can return a meaningful error or a partial response to the client.

In this paper, we present *Q-Cop*, a prototype system for deciding which requests a database system should process and which it should reject, with the goal of minimizing client timeouts. Unlike prior work on admission control, Q-Cop makes its decisions not based on resource utilization or the number of requests in the system, but instead based on the *mix of queries* being executed by the system. The notion of query mixes is important because admission control decisions require an estimate of how long each request will take. To accurately predict how long a request will take, we need to have an estimate of how busy or loaded the system is, since execution time is affected by load. However, the notion of load is itself somewhat difficult to quantify. In a series of experiments, we found very low correlation between CPU usage, which might be expected to be a very strong indicator of load, and response times. Conversely, we found that the number of currently running queries *is* correlated to response times. We also found that the response time of a particular query depends on the mix of queries that is running concurrently with this query. It has previously been shown that the specific mix of queries has a strong effect on system performance [1], [2], [3]. Q-Cop therefore uses the query mix, *how many queries of each type are currently running in the system*, as the basis for measuring current system load and predicting query response times. To build the required performance models, Q-Cop adopts an experiment-driven modeling approach: experiments are conducted to sample the space of possible query mixes, and regression models are built based on these samples.

The contributions of this work are as follows:

*Some of this work was done while Brecht was on sabbatical at EPFL.

- We develop a query mix model (QMM) that considers the current mix of executing queries to estimate the expected run time of a newly arriving query.
- We implement a prototype (Q-Cop) that uses the query mix model and the notion of how long a user will wait for a response to make more informed admission control decisions than previously possible.
- An experimental evaluation conducted using TPC-W demonstrates that Q-Cop is able to make better decisions about *which* requests to reject than other methods. This improvement significantly reduces the number of requests not being handled by the server under high loads, precisely the time when it is most important to handle as many requests as possible before users time out.

II. Related Work

A. Admission Control

Admission control in web servers and database systems is an established and well-studied research area. Q-Cop distinguishes itself from prior work in this area by explicitly taking query mixes into account when making admission control decisions. We present some examples of prior work in this area, which we classify into three categories.

The first category of admission control approaches is based on controlling the *multi-programming limit (MPL)*, which is the maximum number of requests that can be served concurrently. Many admission control approaches are essentially methods for tuning the MPL. Mönkeberg and Weikum [4] monitor a data contention metric called the conflict ratio to dynamically control the MPL. Liu et al. [5] propose approaches for tuning the MPL of an Apache web server using numerical optimization, fuzzy control, and heuristics based on resource utilization. Schroeder et al. [6] determine the lowest MPL that corresponds to maximum system performance by using queuing analysis to determine an initial MPL value and a feedback controller to dynamically adjust this value. We compare Q-Cop to the best possible MPL (obtained by extensive search), and we show that MPL approaches are of marginal effectiveness at reducing timeouts because they are oblivious to query types and query mixes.

The second category of admission control approaches is based on *average query response time*. These approaches build models of query response time and use these models to reject queries with the goal of keeping the average query response time below a desired threshold. These approaches do not distinguish between the different query types and are oblivious to query mixes. For example, Yaksha [7] probabilistically rejects requests to keep the average response time at a desired value. The probability of rejecting requests is adjusted dynamically by a proportional integral controller based on the difference between the observed response times and the desired response times. Bartolini et al. [8] distinguish between normal load and overload (or "flash crowds"). A performance model is learned dynamically and used to determine the probability of rejecting requests under normal load and overload. SERT [9] avoids overload by preempting requests if their execution time exceeds a timeout threshold. The threshold is set dynamically, but it does not vary by query type. Controlling MPL and controlling average query response time are two fundamentally different approaches to admission control. However, for the purpose of minimizing unsuccessful client requests, these two approaches are equivalent. The system can concurrently process a certain number of queries while keeping the average response time below the required timeout threshold, and it estimates the average response time with every arriving query. If the system estimates that a new query would drive the average response time above the timeout threshold, the system would reject this query – regardless of its type – to avoid this overload. Essentially, the system is controlling MPL by estimating query response times. Note that an approach that does not distinguishing between query types will reject more queries than necessary because it cannot be discerning about which queries to reject.

The third category of admission control approaches is based on *query types*. These approaches build models of response time for different query types and use these models to make admission control decisions. For example, Gatekeeper [10] maintains moving averages of the response time of different request types and uses these averages as predictions of the response times of future requests of these types. The predicted response time of a request type is used as a measure of the load that this request type places on the system, and Gatekeeper tries to keep the total load below a threshold that is determined experimentally using off-line execution of the expected workload. Quorum [11] tracks the average service times of different request types and uses these times to decide if the waiting plus service time of a specific request will exceed an administrator-specified timeout. These approaches distinguish between query types, so they come close to taking query mixes into account. However, Quroum builds a model for a specific workload, and hence a specific set of mixes, while Gatekeeper determines thresholds for a specific workload. If the workload and mixes change, the Quroum model has to be re-learned and Gatekeeper must re-determine the threshold.

Recent work [12] has proposed a way of modeling multiple workloads by using multiple Bayesian networks, one per workload. However, this requires specifying in advance the workloads of potential interest. In contrast, our approach builds a performance model that works for arbitrary query mixes, so it can deal with changes in the distribution or intensity of the workload without any re-learning. Moreover, we show experimentally that using a model based on query types without taking query interactions into account is not as accurate as using query mix models. We show that approaches based on query types and those based on query mixes may reject a similar number of queries, but the approaches based on query types cannot accurately decide *which* queries to reject, so they end up resulting in more client timeouts than query mix approaches.

B. Query Mixes

The interactions between queries running concurrently in a query mix can have a significant impact on performance. This has been shown in prior work [3], and mix-aware techniques for query scheduling [1], [2] and for predicting workload completion times [13] have been proposed. These papers have shown that: (1) query mixes have a significant impact on performance, (2) the interactions within query mixes are complex and difficult to model analytically, and (3) experiment-driven modeling can effectively capture these interactions and produce accurate performance models. These three points are part of our motivation in this paper. However, the problem we focus on (admission control) is fundamentally different from those addressed in the previous work (scheduling and completion time prediction for long running queries), and required developing completely new techniques.

C. Experiment-Driven Performance Modeling

A key feature of Q-Cop is using an experiment-driven approach for modeling query interactions, similar to the approach used in [1], [2], [13]. The experiment-driven approach to performance modeling is gaining wide acceptance as a way to build robust performance models for software systems, especially as these systems are becoming increasingly complex. Ganapathi et al. [14] use an experiment-driven approach combined with machine learning models to predict performance metrics for queries. That work demonstrates the feasibility and effectiveness of the experiment-driven approach for database performance modeling. The experiment-driven approach has more recently been used for tuning database configuration parameters [15], [16]. An infrastructure for running experiments in a data center has also been proposed [17] and Oracle 11g uses an experiment-driven approach to test the recommendations of its SQL Tuning Advisor before implementing these recommendations [18]. As has been done in this previous work, we use experiment-driven performance modeling for its simplicity and robustness.

III. Query Mixes

In this section, we motivate the need for modeling response time based on query mixes when making admission control decisions. We start by demonstrating the significant effect of query mixes on response times using experiments with a TPC-W Browsing Mix (details of our experimental environment are given in Section VI). We execute each query repeatedly within a separate thread on the client machine. We vary the MPL and mix of queries by controlling the number of executing threads and the query performed by each thread. For a particular mix of queries, there is a fixed number of threads repeatedly performing the same query and recording the response time. In this case, each thread waits for as long as required for a response (i.e., there are no timeouts). Each experiment runs one particular query mix, and we calculate an average response time for every query type in this mix. In post processing, we group the data by query type and MPL in order to compare loads that would appear the same from an admission control perspective if the query mix is not considered. For each query type and MPL we identify the experiment (query mix) with the minimum and maximum average response times for this query type, and we calculate the overall average response time for this query type at this MPL. Figure 1 shows these response times and their 95% confidence intervals for the TPC-W Best Sellers query type.

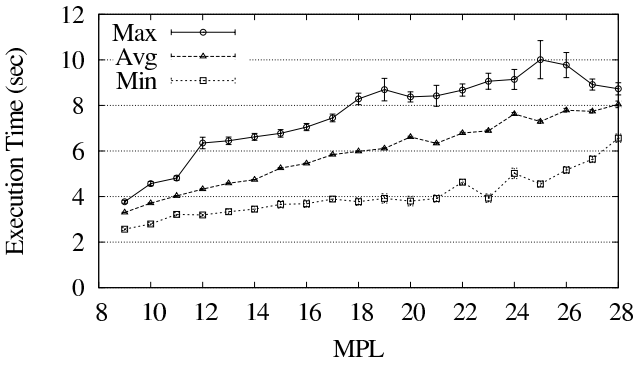

Fig. 1. Variation of average query response times with different query mixes for the TPC-W Best Sellers query.

As expected, the response time of the queries increases as the MPL increases because the system becomes more loaded. The interesting observation in this figure is the difference between the minimum, maximum, and average response times at the same MPL. The graphs show that this difference in response time is large and statistically significant (the confidence intervals do not overlap), which demonstrates the considerable effect of query mixes on performance. We have examined graphs for each of the different query types [19] and found similar behaviour in each case. Therefore, looking solely at the MPL is not sufficient to accurately predict the performance of the system.

Figure 1 also show that the best MPL at any time depends on the mix of queries being processed at that time. For example, consider Figure 1 and the average amount of processing that can be carried out in six seconds. The MPL that can be supported without having the Best Sellers query time out depends on the query mix. If the query mix happens to be the one that causes the maximum response time for the Best Sellers query, we can only support an MPL of 12 (since the maximum response time line reaches 6 seconds at MPL 12). If the query mix is one that causes an average response time, we can support an MPL of 19 requests. If the query mix is the one in which we get the minimum response time, we can support 27 requests. An approach that sets MPL without considering query mixes would not be able to distinguish between these cases and would have to be conservative. Such an approach would therefore perform poorly, as we show in our experiments.

Next, we show that CPU utilization is not correlated to the response time of a given query type. Figure 2 shows a scatter plot of the CPU utilization observed while running different query mixes against the response time of the Best

Sellers query in these mixes. The correlation coefficient is 0.025, indicating almost no correlation between CPU utilization and the response time of the query. For many mixes with widely varying response times for the Best Sellers query, CPU utilization is between 80% and 100%. We also see a wide range of CPU utilizations (between 20% and 100%) for a wide range of query response times. Thus, CPU utilization cannot be used to predict query response time. We cannot monitor CPU utilization and use it to predict query response times and instead we directly model query response times using query mixes. We have seen a similar lack of correlation between CPU utilization and response time for other query types [19].

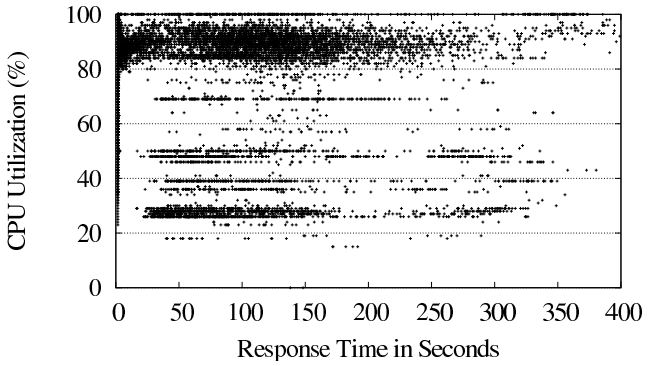

Fig. 2. CPU utilization for different response times of the Best Sellers query, cropped to 400 seconds. There are some points out to 600 seconds, and a few as high as 800 seconds.

IV. Admission Control and Implementation

In this section, we provide some motivation for requiring admission control, even when clients have limits on how long they will wait for a response to a request. In addition, we describe how we have implemented different admission control techniques that we use as a basis for comparison.

A. The Perils of Uncontrolled Overload

Existing admission control studies include graphs depicting how in a three-tiered system throughput can plummet and response time can skyrocket if admission control is not performed [7], [10]. Typically these studies are performed with a closed-loop workload generator in which clients wait for as long as needed for a response before issuing their next request.

Figure 3 shows the average throughput (left y-axis) and response time (right y-axis) of a server which is driven from manageable load into overload conditions. While the profile of these curves is similar to those seen in previous work, the drop in throughput and rise in response times are not as severe in Figure 3 as seen in previous work. This is because Figure 3 is generated using clients which will not wait arbitrarily long for a response, but instead will eventually time out (in this case after 30 seconds), and because responses that time out are not included when computing the average response time.

Nevertheless, this graph shows that the reply rate falls off its peak level of nearly 14 replies per second to around 11 replies per second. Perhaps even more concerning from a client perspective is that average response time grows from less than half a second to nearly eight seconds. It is important to remember that the average response time shown does not include requests that timed out after 30 seconds, which at the highest load level accounts for more than 50% of the requests.

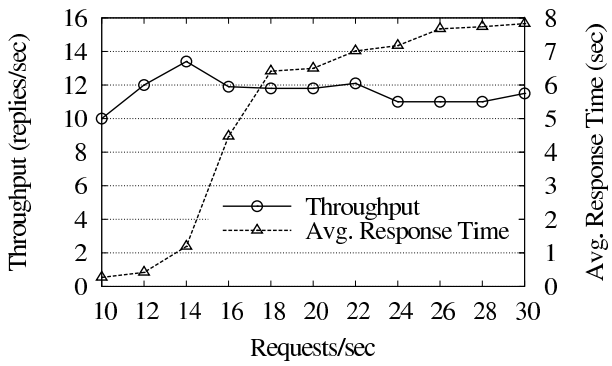

Fig. 3. Mean throughput and response time without admission control.

Clearly, a server must be capable of coping with overload conditions in some way. A popular method for controlling overload is to allow the server to perform some type of admission control. That is, to give the server the ability to reject some requests in order to help it improve some performance metric of interest (e.g., replies per second, mean response time, or dollars per minute).

B. Implementation Framework

In order to compare different approaches to admission control, we have implemented a common framework within which each of the different schemes can be implemented and studied. We use this approach to ensure that differences in performance are due to differences in the admission control scheme rather than differences in implementation details. Specifically, the following factors are held constant across all implementations:

- All admission control decisions are performed within the database system, since that is where most of the work is being done.
- All approaches use the same method for counting the current number of running requests.
- All decisions to reject a request are made only when the request is first received by the database system, using information available at that time. Requests are not removed once they have been admitted.

C. MPL-based Approach

One popular and simple technique for performing admission control is to limit the MPL. Limiting the MPL is commonly used because implementing it is as simple as setting a web server or database system configuration parameter. Admission control techniques that limit MPL use various levels of complexity to determine and adjust the best MPL [4], [5], [6]. A common drawback of these approaches is that while they may dynamically adjust their MPL to account for changes in load, they do not take into account the differences in queries being

considered for rejection nor how those queries might interact with the mix of currently executing queries.

We did not implement any specific MPL-based approach because we did not want to choose only one approach to compare Q-Cop against. Instead, in Section VII we conduct a series of experiments to determine the best MPL value for our environment, which is the best that any MPL-based approach would be able to achieve. We compare Q-Cop against this best MPL.

D. Query Type Approach

Simple approaches that are based solely on the MPL or on the mean response time of queries over all requests do not consider the fact that different query types may have different response times. For example, recent work [10] clearly shows that in a TPC-W workload the query type has a big impact on the query's response time. That work develops a system which makes admission control decisions based on the type of query being considered for admission and the current system load.

We implement a variant of this scheme that we call TYPE. When making an admission control decision it uses knowledge of the type of query being considered along with an estimate of the expected run time of this query. Information about query run times is obtained from a regression model that is built by executing the full workload in an off-line learning phase.

In this learning phase, we run the full workload without admission control and collect the following information for all queries: the query type, the number of other queries executing when the query arrives, and the response time of the query. We then build a linear regression model for each query type, which can be done using many analysis tools (e.g., Excel). This model predicts the execution time of this query type given the number of currently executing queries.

The model for query type i has a coefficient c_i representing the amount of time each query in the system adds to the execution time of a query of type i. If M is the number of queries currently executing, we estimate the execution time of query type i as $Est_i = c_i * M + C_i$, where C_i estimates how long the query would take with no load. A different linear regression model is derived for each query type. For example, in our experiments the execution time of the TPC-W Best Sellers query in milliseconds is estimated by: $Est_{BestSellers} = 302.2 * M + 1893.0$. Details for all query types can be found in [19].

As each query arrives at the database system, the system computes an expected execution time using the model for this type of query and the current number of executing queries. If the expected execution time exceeds the timeout threshold, the query is rejected and a message is returned to the client indicating that the server is overloaded. Otherwise, the query is executed.

V. The Query Mix Model and Q-Cop

The experiments in Section III motivate the importance of considering the mix of queries executing when making admission control decisions. Q-Cop explicitly takes query mixes into account, and in this section we describe the details of the Q-Cop prototype system. We describe how we construct a query mix model using an experiment-driven approach, and how this model is utilized within Q-Cop. The process of constructing the query mix model involves two phases: (1) conducting experiments to sample the space of possible query mixes and collect the necessary data, and (2) constructing a regression model that best fits the collected data. These phases are performed off-line before deploying the system.

A. Sampling the Space of Query Mixes

To gather data about how the execution time of each of the N different query types is affected by the mix of other simultaneously executing queries, we need to sample the space of possible query mixes. The number of different possible query mixes is exponential, so having an effective sampling approach is important. We sample the N-dimensional space (where every query type is a dimension) using a Latin Hypercube Sampling (LHS) protocol [20]. This protocol significantly reduces the number of experiments necessary while providing good coverage of the space of possible mixes. The LHS protocol has been successfully used in other work on database systems [13], [16].

When collecting samples and making admission control decisions, we consider only the query types that place a measurable load on the system. Query types that do not place a measurable load on the system are excluded from Q-Cop's decisions, which means that they are always admitted to the system and their effect on performance is not modeled. This approximation enables us to restrict the dimensionality of the space from which we sample and thereby get good coverage with the LHS protocol. The loss in accuracy is minimal because the excluded queries have minimal impact on performance. For example, the workload we use in our experiments (described in Section VI) has six types of requests that a URL can refer to. These requests are listed in Table I. In these requests, there are several queries that place little or no load on the database system (executing in 1 ms or less even under extremely high request rates). These queries are the ones in the Search Request, Product Detail, and Home Interaction requests, plus the Subject Search query in the Search Results request. We exclude these queries from sampling and from admission control decisions, which leaves six query types considered by Q-Cop for this workload. These six query types are shown in the last column of Table I.

The LHS protocol is designed to draw a specified number of samples from a multi-dimensional space in which the lower and upper bounds of each dimension are known. The protocol draws the required number of samples in such a way that these samples uniformly cover the given space. In Q-Cop, the query types define the dimensions of the space. The minimum number of instances of a query type in a sample mix (i.e., the lower bound of the dimensions) is 0. However, we need to specify the maximum number of instances of a query type in a sample mix (i.e., the upper bound of the dimensions) in order to fully specify the space to be covered by LHS. We

require an administrator using Q-Cop to specify the maximum number of concurrently executing queries to be considered in Q-Cop's model, M_{max}. In our TPC-W experiments, we use $M_{max} = 40$. A query mix can potentially contain up to M_{max} instances of one query type (if it contains 0 instances of all other query types). Therefore, we set the upper bound of the dimensions of the space covered by LHS (which represent the different query types) to M_{max}. We use the LHS protocol to generate S samples from this multi-dimensional space. Some of these samples will have a total number of queries greater than M_{max} (for example, the samples from the region of the space where every query type occurs close to M_{max} times). We remove these samples from our sample set and we are left with S' samples, which we use for our experiment-driven data collection. We try different values of S until we obtain a sample set whose size is close to the required number of samples S'. In our TPC-W experiments, we use $S' = 1000$.

B. Experiment-Driven Data Collection

We conduct experiments to collect information about the sample query mixes that are chosen by LHS. We create N request URLs, each corresponding to one of the query types. For a sample with M concurrent queries, we use M client programs, with each client program requesting the URL for a specific query type. Each client program requests its URL from the server, reads the reply, logs the response time in memory, and then re-requests that URL. We use HTTP 1.1, so connections are not closed between URL requests. This approach maintains a constant load on the database system and keeps the query mix constant. Parameters are provided randomly from a set of valid entries to those URLs that require them. For example, the TPC-W queries include a Best Sellers request which takes as a parameter the category of book, and an Author Search request which takes a string to search for. The parameters are chosen such that a non-empty result set will be returned from the server. Each query mix experiment is run for two minutes, at which point all the clients terminate. When the experiment is finished, statistics are logged to disk, and the server is allowed to cool down for 30 seconds to finish all requests. This experiment is then repeated for each query mix that is chosen by the LHS protocol.

In our TPC-W experiments, we use 1,000 sample query mixes, which results in approximately 70,000 query executions requiring about 143 hours of execution time (about 6 days). Longer queries account for a smaller fraction of those executions than shorter queries because fewer would run in the time allotted for each experiment. However, the smallest number of queries executed for any query type is 395.

Experiment-driven data collection is a one-time off-line process, so its cost can likely be tolerated by administrators using Q-Cop. However, it is still useful to try to reduce this cost as much as possible. Using LHS is one way of reducing this cost because the LHS protocol makes good use of the available number of samples so it allows Q-Cop to work with a smaller number of samples. Further reducing this cost without compromising quality is an interesting area for future work.

C. Constructing the Query Mix Model

To derive a predictive model from the raw data, we use the Waikato Environment for Knowledge Analysis (WEKA) toolkit [21]. We built and tested models using several different learning algorithms available in WEKA, including linear regression, locally-weighted linear regression, Gaussian processes, and multilayer perceptron. When compared with the linear regression model, some of the more advanced algorithms showed improvements in accuracy of approximately 5-10%, as measured by the relative mean squared error. However, we made a design decision to use a simple linear regression model, even though we are aware that it probably over-simplifies the complex nature of query interactions. We made this design decision because the linear regression model was sufficient to give us very good results, so there was no strong justification for including the additional complexity of the more advanced models. An interesting direction for future work is to see if improvements in model accuracy improve the performance of our system.

When the model building process finishes, we have a set of linear regression coefficients learned from the training data. For each query type i, we have a set of coefficients $c_{i1}, c_{i2}, \ldots c_{iN}$, where c_{ij} represents the amount of time a query of type j will add to the execution time of a query of type i. Additionally, there is an estimate C_i for how long the query would take given no load. With N different types of queries, if n_j is the number of queries of type j in a mix then the estimated query time for a query of type i in this mix is calculated as: $Est_i = (c_{i1} * n_1) + (c_{i2} * n_2) + \ldots + (c_{iN} * n_N) + C_i$. For example, the model for the Best Sellers query type in our experiments is: $Est_{BestSellers} = (31.5 * n_1) + (30.2 * n_2) + (586.5 * n_3) + (675.6 * n_4) + (102.5 * n_5) + (18.5 * n_6) + 3063.2$. Details for the other query types can be found in [19].

The decision process used by Q-Cop is very simple: When a query of type i arrives, it is added to the current query mix and the regression model for this query type (Est_i) is used to estimate its execution time. If this estimate is less than the timeout value for the query type, the query is admitted and allowed to run to completion. Otherwise, an error message is returned to the client to indicate that the server is overloaded (e.g., HTTP 503 Service Unavailable).

We investigated the effect of sample size on model accuracy using WEKA's built-in accuracy measures and percentage split functionality. The resulting data shows no indication that increasing the sample size would yield a more accurate model.

D. Strengths and Weaknesses

Important aspects of Q-Cop's modeling approach are that it does not require:

- a priori information about the expected system load,
- a priori information about the expected mix of queries, or
- a complex mathematical model of how the system works.

As a result, after running the initial set of experiments determined by the LHS protocol and then constructing the

model, no changes are required for different load intensities (e.g., higher request rates) or different distributions of the query types. Since the model is generated from the underlying query types but not from any specific usage log, it should perform well across a variety of query distributions. This is in contrast to models that require the query distribution to be known in advance to calculate average response time curves.

The experiment-driven model building approach does, however, require a priori knowledge of the different types of queries in the workload. This is a reasonable assumption particularly in web-application environments because queries are initiated through web interfaces that are implemented and known by the application designer. Such web-applications generate a fixed set of query types that can easily be determined by an administrator, either from the application source code or by logging queries sent to the database. The administrator would also need to identify the query types to be excluded from Q-Cop's analysis and admission control decisions, which is not a difficult task since these are query types that do not place any load on the system under all conditions. For example, these could be query types that always finish in under 1 ms.

Adding or changing query types or the hardware configuration could change the relative impact of queries on each other and would necessitate re-running the initial sampling and model-building phases.

VI. Experimental Environment

We now briefly describe the hardware and software used in our experiments, as well as the workload that we use.

A. Hardware

All of the experiments described in this paper were conducted on an IBM Blade Center with a Model H chassis. We use one blade for the server processes and one blade for the client processes. Because nearly all work for our workload is done in one tier, namely the database tier, we conducted all experiments with the web server, application server, and database system running on the same server machine. The server and client machines both have two 2.0 GHz AMD dual-core 2212 HE CPUs, 10 GB of RAM, and one 67 GB 10,000 RPM Fujitsu MBB2073RC disk. The two machines are connected through an internal switch in the Blade Center chassis with 1 Gbps of bandwidth. This is sufficient bandwidth to prevent the network from being a bottleneck.

B. Software

The client and server machines run the OpenSUSE Linux distribution with version 2.6.22.18-0.2 (x64 SMP) of the kernel. The web server is Apache version 2.2.9. The application server is version 4.1.37 of Apache Tomcat. The database system is version 10.4.2.0 of Derby. We use a 5.2 GB TPC-W data set which fits in memory, so no paging or I/O are required once the cache has been warmed. The partially-open loop workload [22] is generated using httperf [23] by generating session log files that produce a TPC-W-like workload. The httperf workload generator is used because it provides mechanisms for placing a time limit on requests and because it is designed for producing overload conditions. This is in contrast to closed-loop workload generators which have been shown to be unable to generate overload conditions because their rate of requests can be throttled by the speed of the server [24]. In our TPC-W experiments, the client-side timeout is set to 30 seconds for all requests. Versions 6.x of Internet Explorer use a 60 second timeout while versions 7 and 8 use 30 seconds [25]. Internet Explorer versions 7 and 8 are reported to have the highest market share of all browsers, followed by Firefox [26]. We were unable to find definitive sources for the timeout limit used in Firefox. We assume that 1 second of the timeout limit is required outside the database system, for processing in the application server and web server and for communication with the client. Therefore, we set the timeout threshold used by Q-Cop to 29 seconds. While the query type approach and Q-Cop allow for the ability to set different timeout limits for different query types, we currently use the same limit for all query types.

In order to obtain detailed and accurate information about which queries result in timeouts while still being able to use httperf's ability to generate high loads, we avoid the use of persistent connections (sessions) in which multiple requests are sent via one TCP/IP connection. With httperf, if one request within a session times out, the entire session times out and information is not provided about which request within a session was being processed when the timeout occurred. The impact of this decision on our results is negligible because this only results in the web server (which is not the bottleneck) handling slightly more TCP/IP connection requests than it would if multiple requests used the same connection.

C. Workload

The series of requests issued by httperf to the web server is generated to adhere to the Browsing Mix distribution of requests specified by TPC-W [27]. The precise details of the distribution are provided in Table I and described below. Exactly the same logs are used for each experiment to ensure that the load in all experiments is identical.

Our workload is TPC-W-"like" because it does not include all request types included in the Browsing Mix distribution. We use the TPC-W servlets implemented by the University of Wisconsin [28] and use a subset of the TPC-W servlets in our workload. The Wisconsin implementation is written for a DB2 DBMS, while we use the Derby DBMS. Using Derby required modifying the SQL used by each servlet for compatibility. Instead of modifying all servlets, we decided to remove the Ordering requests from our workload mix, since these requests represent a very small part of the Browsing Mix distribution. Seven of the Ordering requests each issue less than 1% of the requests and one issues only 2% of the requests. In total, this accounts for only 5% of the requests. Removing these requests results in slightly increasing the percentage share of the remaining requests.

Table I provides the name of each of the TPC-W servlets, a database query number (Type Num, which we assign for

easy reference), what percentage of requests each contributes to the Browsing Mix of the TPC-W workload (Orig Mix), what percentage each contributes to our workload (Our Mix), and whether or not it is included in our query mix model (column "In QMM"). Recall that the query mix model is constructed for only the "measurable" queries. The Search Results request actually results in one of three different types of database queries, two of which are included in the query mix model.

Servlet ⇒ Database Queries	Type Num	Orig Mix	Our Mix	In QMM
Home Interaction	–	29.00	31.30	No
Product Detail	–	21.00	21.83	No
Search Request	–	12.00	12.07	No
Search Results		11.00	11.03	
⇒ Author Search	1			Yes
⇒ Title Search	2			Yes
⇒ Subject Search	–			No
Best Sellers		11.00	11.98	
⇒ Setup (Part I)	3			Yes
⇒ Main (Part II)	4			Yes
New Products	5	11.00	11.79	Yes
(Multiple Servlets)				
⇒ Related Products	6			Yes
Total Browsing		95.00	100.00	
Shopping Cart	–	2.00	–	–
Customer Register	–	0.82	–	–
Buy Request	–	0.75	–	–
Buy Confirm	–	0.69	–	–
Order Inquiry	–	0.30	–	–
Order Display	–	0.25	–	–
Admin Request	–	0.10	–	–
Admin Confirm	–	0.09	–	–
Total Order		5.00	0.00	

TABLE I
LIST OF TPC-W SERVLETS, QUERY TYPES, AND WHERE THEY ARE USED.

Because Derby does not support the LIMIT SQL keyword, we changed the Best Sellers query to avoid using this keyword. Instead of computing the 50 best selling books of the most recent 50,000 books purchased as the TPC-W specification requires, we loosely approximate this query by asking for the 50 best selling books of the most recent 20,000 orders. We use 20,000 orders because the number of books in each order can be greater than one. The Best Sellers request is now implemented using two database queries, labeled Setup (Part I) and Main (Part II) in Table I. When we refer to the Best Sellers query we are referring to the Main (Part II) query which does the bulk of the work.

VII. EXPERIMENTAL EVALUATION

We now evaluate how well each of the different methods we have described performs at minimizing the number of requests that are not handled. All experiments in this section are conducted using the TPC-W workload described in Section VI. However, before we can compare the different approaches we need to determine the best MPL against which to compare.

A. Finding the Best MPL

As noted previously, a popular technique for performing admission control is to limit the maximum number of concurrent requests (i.e., the MPL). The difficulty with this approach lies in correctly choosing the best MPL for high loads. If the chosen MPL is too low, some requests will be denied that could have actually been processed. If the chosen MPL is too high, some requests will not be handled before the client times out. Therefore, we begin by conducting a series of experiments to determine the best MPL in our environment. The performance metric used is the *percentage of queries not serviced* as the load on the system is increased. This percentage includes both the queries that are rejected by the admission control mechanism and the queries that time out at the client (i.e., the server's response does not arrive at the client within 30 seconds of the client sending its request).

We run our TPC-W workload with MPLs in the range 5 – 70, and we vary the request rate at each MPL. Figure 4 shows the percentage of queries not serviced as the number of requests per second increases. In looking at our data, we see that MPL = 30 and MPL = 40 have the lowest total number of unsuccessful requests. We present those results, along with MPL = 5 and MPL = 70 for comparison. This graph shows that MPL = 30 and MPL = 40 provide the best overall performance, with their performance being roughly equal. MPL = 70 only does well for light loads while MPL = 5 does not do well for any loads. For clarity, we have excluded MPLs 10, 20, 50, and 60 from the graph, as their not-serviced percentages are generally worse than MPL = 30 and MPL = 40.

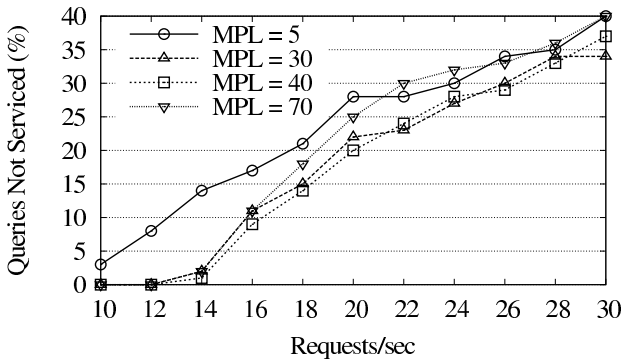

Fig. 4. Percentage of queries not serviced for different MPL values.

When examining the raw data, we find that MPL = 40 is slightly less aggressive about rejecting requests than MPL = 30, resulting in more timeouts. The performance of these two MPL settings is similar and we break the tie by observing that across the range of loads examined, MPL = 40 had a slightly lower total number of unsuccessful requests and overall average of unsuccessful requests. Since the total number of unsuccessful requests is comparable for MPL = 30 and 40, and we found that MPL = 20 and 50 each have significantly more unsuccessful requests, we do not further examine other MPL values in the 20 – 50 range.

We have performed a fairly extensive search across a variety of MPL values in order to find the best-performing MPL, which in our case turned out to be MPL = 40. While this is

not practical in production environments, it gives us a strong basis against which other techniques can be compared.

B. Comparing Different Methods

The graph in Figure 5 compares the percentage of queries not serviced using different admission control techniques. Table II enumerates these methods, the label used in the graphs in this section to refer to these methods, and which section of the paper provides a detailed description of the method. Figure 5 shows that the performance of Q-Cop is quite good across the full range of loads examined, and that it outperforms the other admission control methods. The figure also shows that performance is extremely bad when no admission control (NoAC) is used. Under the highest loads used in this experiment, more than 50% of the requests are not serviced before the client times out (i.e., within 30 seconds).

Graph Label	Admission Control Method	Paper Section
NoAC	No Admission Control	Section IV-A
TYPE	Query Type Approach	Section IV-D
MPL = 40	Best MPL for this Environment	Section IV-C
Q-Cop	Query Mix Model Approach	Section V

TABLE II
DIFFERENT ADMISSION CONTROL TECHNIQUES BEING COMPARED.

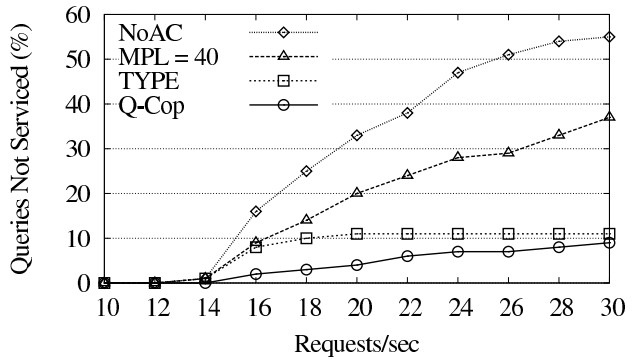

Fig. 5. Percentage of queries not serviced using different approaches.

Table III shows the average percentage of queries not serviced by each method across the full range of loads examined (i.e., 10 – 30 requests per second). The second row of this table shows the percentage by which Q-Cop reduces the total number of queries not serviced for each of the methods explored. The table shows that Q-Cop significantly outperforms both the MPL-based and TYPE approaches in this comparison. Q-Cop yields an overall advantage of 76.8% fewer unserviced requests as compared to MPL = 40 and 46.9% as compared to TYPE.

Approach	NoAC	MPL = 40	TYPE	Q-Cop
Avg % Not Serviced	31.9	19.6	8.6	4.6
Q-Cop Reduction (%)	85.8	76.8	46.9	–

TABLE III
AVERAGE PERCENTAGE OF QUERIES NOT SERVICED ACROSS ALL LOADS.

Using MPL = 40 for admission control results in substantially more queries not serviced than TYPE and Q-Cop. A key problem with an MPL-based approach is that it does not distinguish admissions by the type of query. It will just as likely reject a small request that may actually finish before the timeout as a large request that might not. Under higher loads, TYPE has significantly fewer unsuccessful requests than MPL = 40 because it considers the type of request when making admission decisions and rejects only Best Sellers requests, which are the largest requests in our TPC-W workload.

Figure 6 provides a more detailed view of the two best techniques, TYPE and Q-Cop. It plots the total number of queries not serviced as a function of the number of requests per second. With low request rates (12 requests per second or less), the system is sufficiently provisioned to be able to handle all queries. As the request rate increases, the system is unable to service all of the queries. Some queries get rejected by the admission control method and some queries time out. The difference between the total and rejected lines in Figure 6 shows the number of queries that time out.

Figure 6 shows that the TYPE and Q-Cop approaches reject about the same number of queries for the different request rates. However, Q-Cop has significantly fewer queries not serviced because it avoids significantly more timeouts than TYPE. Q-Cop avoids these timeouts by using information about the query mix to make better decisions than TYPE about when to admit the large Best Sellers requests and when to reject them. Q-Cop rejects about the same number of queries as TYPE, and they both reject only Best Sellers queries. However, Q-Cop rejects queries at different, more appropriate times than TYPE. Q-Cop's better-informed decisions help the server to process more requests and as a result the clients experience much fewer timeouts.

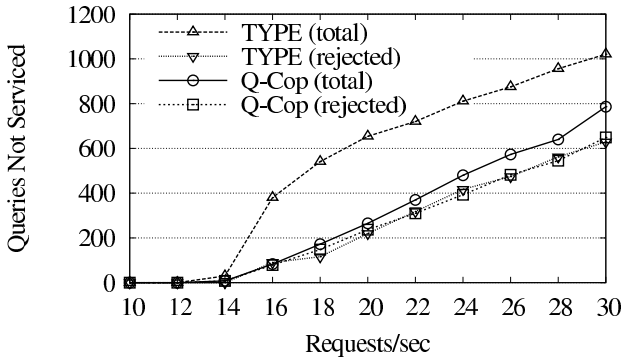

Fig. 6. Number of rejected and unsuccessful queries for TYPE and Q-Cop.

C. Average Response Times

To examine the effect that the different admission control methods have on the average response time, Figure 7 plots the average response time as measured by the clients versus the request rate. This graph must be carefully interpreted because response times for rejected queries *are* included (and are typically very low) and queries that time out are *not*

included. More importantly, the goal of the techniques we have implemented is *not* to minimize average response time, but instead to minimize the number of requests that cannot be serviced. However, the graph does show that the average response time is now more controlled when compared with the average response times observed without admission control (Figure 3). These low response times are also obtained while processing significantly more requests. The key is that Q-Cop judiciously rejects requests that will combine with existing requests in bad ways and would in any case be unlikely to be completed before the timeout limit.

Figure 7 also shows that, as expected, the approach that uses information about the type of query but not the query mix (TYPE) has the lowest average response time. This is because it is servicing significantly fewer requests and in particular fewer of the large Best Sellers requests than the other two approaches.

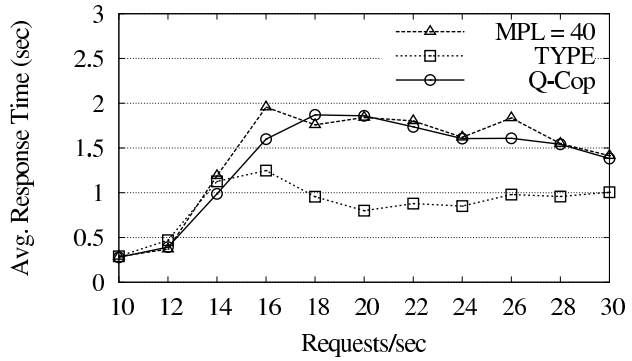

Fig. 7. Average response times using different admission control methods.

D. Probabilistic Approaches

Q-Cop also provides significant advantages when compared with probabilistic approaches to admission control, which are utilized in systems like Self-* [8] and Yaksha [7]. These approaches reject requests with some probability, and tune the probability of rejecting requests to achieve the desired performance objective. If probabilistic admission control is used in a database system without considering query types, it will suffer from the same problem as MPL-based approaches, namely rejecting many small queries that may have completed before the client timed out (see Table III). If probabilistic admission control does consider query types and only rejects large queries, it will suffer from the same problem as the TYPE approach, namely not knowing which large queries to reject and therefore rejecting too many queries or causing too many timeouts (see Figure 6). A probabilistic approach may be able to determine the correct percentage of large queries to reject, perhaps using control-theoretic techniques, but it will not be able to make informed decision about which large queries to reject and when to reject them.

To demonstrate the advantage of Q-Cop over probabilistic approaches, we conduct a simple experiment in which we compute the percentage of Best Sellers queries rejected by Q-Cop at a certain load. For this experiment, we consider a load of 28 requests per second, and observe that Q-Cop rejects 545 queries in total at this load. The rejected queries are all Best Sellers queries, and they represent 59.0% of the Best Sellers queries. Of the admitted queries, 95 queries timed out, all of them Best Sellers queries. This represents 10.3% of the total number of Best Sellers queries and 1.1% of the total number of queries. We implement a simple probabilistic approach and configure it to reject the same percentage of Best Sellers queries as were rejected by Q-Cop. We reject only Best Sellers queries since Q-Cop only rejected Best Sellers queries. This approach results in a comparable number of Best Sellers queries being rejected – 565 (61.1% of the Best Sellers queries). However, the probabilistic approach results in 156 timeouts, 1.6 times as many timeouts as Q-Cop. Note that the probabilistic approach actually rejected *more* queries than Q-Cop, which should give it an advantage in reducing timeouts. However, since Q-Cop bases its decisions on execution time estimates that include the mix of queries being executed, it rejects different queries from those rejected by the probabilistic approach. These rejections result in fewer timeouts because queries are rejected only if they are projected to interact poorly with the currently-executing mix of queries.

E. Why Does Q-Cop Have Any Timeouts?

Figure 6 shows that when Q-Cop is used for admission control, clients still experience some timeouts. So although considering the query mix significantly reduces the number of timeouts, it does not completely eliminate them. This may be due to the inaccuracy of the model learned from the LHS experiments, the simplicity of the decision process used to decide whether or not to admit a request, or because of variations in execution times of the same query type due to different parameters used for a particular query.

Despite these inaccuracies, Q-Cop performs quite well when compared with the alternatives. We discuss the accuracy of our query estimation algorithm in the next section and possible improvements and future work in Section IX.

VIII. Estimation Accuracy

Because the Best Sellers queries execute for significantly longer than the other query types in the TPC-W workload, they are the only queries that are rejected or time out when using Q-Cop. As a result, this is the most critical query type and we now examine this query type more closely. If Q-Cop was completely accurate, it would admit a query if, and only if, the query would complete before timing out. While we are not able to determine if any rejected queries would have completed had they not been rejected (false negatives), we can examine those admitted Best Sellers queries that were accepted but did time out (false positives).

We start by studying the accuracy of the execution time estimates for the Best Sellers queries. Figure 8 shows a scatter plot of the estimated versus actual execution time of all Best Sellers queries. This data was obtained while using Q-Cop, so there are no data points for which estimates were

above 29 (all such queries were rejected).[1]

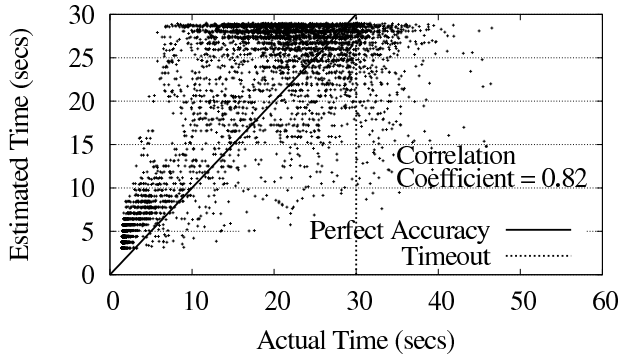

Fig. 8. Actual versus estimated query execution times.

The diagonal line joining the points (0,0) and (30,30) shows where estimates that are perfectly accurate should lie. The actual timeout value of 30 is also shown. Any data points to the right of this vertical line indicate requests that have timed out. This graph shows that there is a fairly good correlation between the estimated and actual query execution times. The correlation coefficient is 0.82 (strongly correlated), which shows that Q-Cop's estimates are quite accurate. However, we can see from the number of points to the right of the timeout line that there may be room for improving accuracy. These points are the false positive cases, where Q-Cop has estimated they would complete before timing out but they did not. These points are especially important because the database expends significant resources computing a result when in the end the client is no longer there or no longer interested in receiving the result. Having sufficient accuracy to avoid accepting these queries would improve the effectiveness of Q-Cop.

Next, we examine model accuracy more closely for these points. Figure 9 shows several views of these false positive decisions. The bottom line in the graph (% of all requests) shows the percentage of all requests that timed out. The second line from the bottom (% of all requests not rejected) shows the percentage of all admitted requests that timed out. These two lines show that across all query types and loads, Q-Cop was quite accurate in admitting requests that would not time out. Across the different loads, a maximum of 2.0% or less of all requests timed out and 2.2% or less of all admitted requests timed out. However, since the execution time of most of the query types is actually quite small relative to the Best Sellers queries, it is not difficult to make a correct decision for the small requests. The second line from the top (% of all Best Sellers queries), shows the percentage of all queries of type Best Sellers that timed out. This includes all queries of type Best Sellers, even those that have been rejected. Finally the top line (% of Best Sellers not rejected) shows the percentage of all admitted queries of type Best Sellers that timed out. This top line shows that of those queries of type Best Sellers that Q-Cop decided to run, a significant number of them timed out (20 – 40% for request rates above 24).

We expect that we could have easily reduced this number by tweaking Q-Cop to add some sort of adjustment (i.e., hack) to artificially lower the rate of false positives by setting a lower timeout threshold for rejecting queries. This could, of course, introduce more false negatives. Instead, we point out that even with the potential for improvement in the accuracy of our query mix model and Q-Cop, they make significantly better admission control decisions than the approaches that do not consider the mix of executing queries.

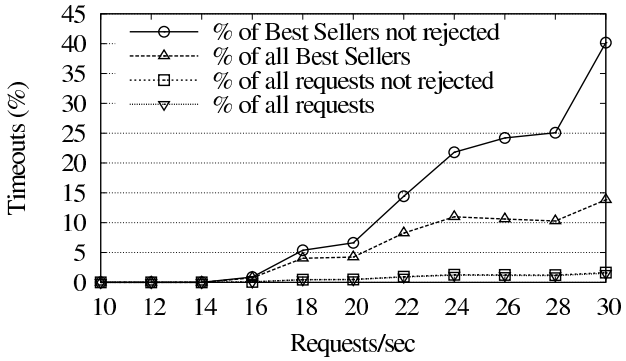

Fig. 9. Percentage of requests that timeout using Q-Cop.

In the next section we describe the several possible sources of inaccuracy in our estimates and decision process and discuss possibilities for future work in these areas.

IX. Discussion and Future Work

There are several potential sources for the inaccurate decisions made in our prototype implementation of Q-Cop. In future work, we plan to further investigate these issues.

A. Query Type Assumption

We assume, as previous work has shown [10], that the query type has a much larger impact on the execution time of queries than the actual parameters to the query. Our model is constructed and our decisions are made based on the assumption that all queries of the same type will exhibit similar performance under the same system load. It would be interesting to see if the parameters to the queries have any impact on the accuracy of our decisions. The LHS experiments used to construct our model include a variety of parameters chosen randomly from a set of valid parameters for each query type. However, TPC-W data is uniform, obviating any potential differences arising from data skew that would cause parameter values to have more impact.

B. Model Inaccuracy

A simple linear regression model makes strong simplifying assumptions about the interactions between different query types, and likely does not accurately capture the full complexity of these interactions. Nevertheless, we chose to

[1]We could have conducted an experiment without admission control and examined the accuracy of those estimates but our model was constructed with admission control in mind (i.e., assuming that we would not encounter the large numbers of big queries that accumulate without admission control).

use this simple model over other more complex but slightly more accurate models available in WEKA such as Gaussian processes because we obtained good results with the simple linear regression model. It would be interesting to see if using the more complex models would improve performance.

Currently, our model does not account for the variation that is seen during the experimental data collection phase of the model building process. We expect that having some estimate of the possible variation or the confidence of the estimates may help to reduce the number of queries that are admitted by Q-Cop, only to have the client time out.

C. Decision Process

Our approach to deciding whether or not to admit a query is intentionally simple. This is done to demonstrate the importance of using information about the mix of queries being executed when making admission control decisions. We currently look only at the impact of the query mix on a newly arriving query to decide whether or not to admit it. We do not consider the impact the new query has on the already executing queries. While this simple approach provides significant improvements over existing approaches, it would be interesting to see if further improvements to this decision would yield further benefits.

X. Conclusions

We propose Q-Cop, a system for performing admission control with the goal of minimizing unserviced requests. A unique and defining feature of Q-Cop is that it makes its admission control decisions based on the mix of queries currently running in the system. To model the performance of different query mixes, Q-Cop uses an off-line, experiment-driven approach that samples the space of possible query mixes and learns statistical models that can be used to predict the performance of any query mix observed in the workload. Q-Cop then uses these models in its on-line operation to decide which queries to reject. Using queries from the TPC-W benchmark, we experimentally demonstrate that the mix of queries can indeed have a significant impact on query performance. We also demonstrate that Q-Cop outperforms mix-oblivious techniques for admission control, reducing the number of queries not serviced by up to 47%.

XI. Acknowledgments

We thank Ryan Stedman for helping us to modify Derby and Mumtaz Ahmad for his guidance with the Latin Hypercube Sampling protocol and for sharing his WEKA and Matlab expertise. Funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada and by the Ontario Centres of Excellence.

References

[1] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "QShuffler: Getting the query mix right," in *Proc. Int. Conf. on Data Engineering (ICDE)*, 2008.

[2] ——, "Modeling and exploiting query interactions in database systems," in *Proc. ACM Conf. on Information and Knowledge Management (CIKM)*, 2008.

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

[4] A. Mönkeberg and G. Weikum, "Performance evaluation of an adaptive and robust load control method for the avoidance of data contention thrashing," in *Proc. Int. Conf. on Very Large Databases (VLDB)*, 1992.

[5] X. Liu, L. Sha, Y. Diao, S. Froehlich, J. L. Hellerstein, and S. S. Parekh, "Online response time optimization of Apache web server," in *Proc. IEEE Int. Workshop on Quality of Service (IWQoS)*, 2003.

[6] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. M. Nahum, and A. Wierman, "How to determine a good multi-programming level for external scheduling," in *Proc. Int. Conf. on Data Engineering (ICDE)*, 2006.

[7] A. Kamra, V. Misra, and E. M. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites," in *Proc. IEEE Int. Workshop on Quality of Service (IWQoS)*, 2004.

[8] N. Bartolini, G. C. Bongiovanni, and S. Silvestri, "Self-* through self-learning: Overload control for distributed web systems," *Computer Networks*, vol. 53, no. 5, 2009.

[9] J. Zhou and T. Yang, "Selective early request termination for busy internet services," in *Proc. Int. Conf. on World Wide Web (WWW)*, 2006.

[10] S. Elnikety, E. M. Nahum, J. M. Tracey, and W. Zwaenepoel, "A method for transparent admission control and request scheduling in e-commerce web sites," in *Proc. Int. Conf. on World Wide Web (WWW)*, 2004.

[11] J. M. Blanquer, A. Batchelli, K. E. Schauser, and R. Wolski, "Quorum: Flexible quality of service for internet services," in *Proc. Symp. on Networked Systems Design and Implementation (NSDI)*, 2005.

[12] J. Rao and C.-Z. Xu, "CoSL: A coordinated statistical learning approach to measuring the capacity of multi-tier websites," in *Proc. IEEE Int. Symp. on Parallel and Distributed Processing (IPDPS)*, 2008.

[13] 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.

[14] 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.

[15] 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.

[16] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with iTuned," in *Proc. Int. Conf. on Very Large Databases (VLDB)*, 2009.

[17] 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.

[18] 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.

[19] S. Tozer, "Avoiding Bad Query Mixes to Minimize Unsuccessful Client Requests," M.Math Thesis, University of Waterloo, 2009.

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

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

[22] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Closed versus open system models: a cautionary tale," in *Proc. Symp. on Networked Systems Design and Implementation (NSDI)*, 2006.

[23] D. Mosberger and T. Jin, "httperf: A tool for measuring web server performance," in *Proc. Workshop on Internet Server Performance*, 1998.

[24] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *Proc. USENIX Symp. on Internet Technologies and Systems (USITS)*, 1997.

[25] "Microsoft Support Web Page," http://support.microsoft-.com/kb/181050i.

[26] "Browser Version Market Share," http://marketshare.hitslink.com/-browser-market-share.aspx?qprid=2.

[27] Transaction Processing Performance Council, "TPC-W Benchmark," http://www.tpc.org/tpcw/default.asp.

[28] T. Bezenek, T. Cain, R. Dickson, T. Heil, M. Martin, C. McCurdy, R. Rajwar, E. Weglarz, C. Zilles, and M. Lipasti, "Characterizing a Java implementation of TPC-W," in *Proc. Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2000.