Avoiding Bad Query Mixes to Minimize Unsuccessful Client Requests Under Heavy Loads

by

Sean Tozer

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2009

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 computes measures of load on the system based on the actual mix of queries being executed. This measure of load is used to estimate execution times for incoming queries, which allows Q-Cop to make control decisions with the goal of minimizing the number of requests that are not serviced before the client, or their browser, times out.

Using TPC-W queries, we show that the response times of different types of queries can vary significantly, in excess of 50% in our experiments, depending not just on the number of queries being processed but on the mix of other queries that are running simultaneously. The variation implies that admission control can benefit from taking into account not just the number of queries being processed, but also the mix of queries. We develop a model of expected query execution times that accounts for the mix of queries being executed and integrate that model into a three-tiered system to make admission control decisions. This approach makes more informed decisions about which queries to reject, and our results show that it significantly reduces the number of unsuccessful client requests. Our results show that this approach makes more informed decisions about which queries to reject and as a result significantly reduces the number of unsuccessful client requests.

For comparison, we develop several other models which represent related work in the field, including an MPL-based approach and an approach that considers the type of query but not the mix of queries. We show that Q-Cop does not need to re-compute any modelling information in order to perform well, a strong advantage over most other approaches. Across the range of workloads examined, an average of 47% fewer requests are denied than the next best approach.

Acknowledgments

I would first like to thank Tim Brecht, my advisor, for supporting me through this process from starting grad school to finishing this work.

I would also like to thank Ashraf Aboulnaga, my co-supervisor, for giving another perspective and a number of good ideas and guidance through the formative stages of my thesis. My thesis readers, Ken Salem and Johnny Wong, were both very helpful for giving their time and feedback.

Finally, I would like to say thanks to Karly for being there for me throughout. Thank you for your encouragement and support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The goal of this thesis is to reduce the number of unserved requests encountered by an overloaded web application. We hypothesize that the execution time of individual queries is strongly influenced by the mix of queries running at the same time. We develop an admission control algorithm that uses information about the mix of queries currently executing to make decisions about whether or not to admit new queries. We evaluate this approach by experimentally comparing its prformance against that of existing methods.

1.2 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 [5], so managing the performance of the database system is important for the overall performance of the web application.

In this thesis, 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 the number of queries that are not serviced before timing out. Unlike prior work on admission control, Q-Cop makes its decisions not based simply on resource utilization or the number of requests in the system, but instead based on the *mix of queries* currently 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, shown in Section 4.3, we found very low correlation between CPU utilization (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, as shown in Section 4.2.

We also found that the response time of a particular query depends on the mix of other queries that are simultaneously executing. It has previously been shown that the specific mix of queries has a strong effect on system performance [2, 3, 4]. 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 modelling approach: experiments are conducted to sample the space of possible query mixes, and regression models are built based on these samples.

1.3 Motivation

Web application performance is a pervasive concern for individuals and businesses. Indeed, nearly any deployment of a web application would benefit from increased throughput or lower response times. More importantly, performance is of particular concern when the server is experiencing higher than usual loads.

Companies typically have a limited set of resources and cannot afford to simply over-provision to meet potential demand. A server handling a relatively small number of requests can be very quickly overwhelmed by a flash crowd that increases the incoming load by an order of magnitude or more [36]. This can be devastating to a company that relies on their web presence to conduct business. In companies with a large number of servers, increasing the amount of load that each server can handle, and decreasing the amount of over-provisioning required for potential burst demand, can lead to significant savings by reducing the number of total machines required. Each machine is able to run closer to capacity, without fear of sudden service degradation.

In short, public-facing servers must be prepared in some way for overload conditions. While massive over-provisioning is effective, it is prohibitively expensive. It is therefore critical to limit the negative effects of overload while simultaneously increasing the capacity of the server to handle requests at near saturation. Otherwise, even if the server is able to serve a large number of clients under normal usage, it can become ineffective when it is needed the most.

Examples of commercial systems prone to periods of high loads include news services and online retailers. In the case of news services, high loads likely indicate some kind of real-world event which is causing a large number of people to suddenly become interested simultaneously. Examples of this can include the "Slashdot Effect" [1], where a link can drive huge numbers of users to request a news article in a very short period of time. If the service is unable to handle that load effectively, users will likely look elsewhere for their news, often to a competitor. This loss of business affects not just the user's behaviour in this instance, but could lead to a long-term loss in business as users find other sources of information which are more reliable. For online retailers, high loads could be the result of, for instance, an unexpectedly popular sale, or large seasonal increases, as with much higher traffic just before Christmas. Increased traffic can also depend on external factors, such as being linked from some large, popular aggregation site. Examples of this include the “Oprah Effect” [17], where a favourable review of a product can result in an extreme volume of traffic on an unsuspecting web retailer. New customers are unlikely to be persuaded by an unresponsive website, and in fact are likely to have a very negative impression of the retailer. This results in a net loss of business for the site, larger than the immediate loss from specific requests that did not finish.

In situations with unexpectedly high load, it can look better for the system to explicitly return a simple “overloaded” error message to the customer, rather than appearing completely unresponsive. With an error message, the customer can see that the server is encountering a specific transient issue. The customer might even link that issue to the underlying cause of the increased load, if they came from the same source, further mitigating any frustration at not being able to access the server.

1.3.1 Server Overload from a User’s Perspective

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. The system is extremely loaded, doing a large amount of work, but generally failing at meeting the deadlines for any client requests.

This overload behaviour can lead to a system that is almost completely unresponsive. Users will tend to exacerbate the problem by continually re-requesting resources from an already overloaded system. This behaviour is even worse considering the timeliness of the problem. For most commercial servers, periods of heightened load are exactly the time when serving requests is the most crucial.

When loads exceed the capacity of the server, 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 drop connections and appear unavailable. A common reaction in this case is for the client to retry the failed request, which exacerbates the overload problem at the server. Alternately, the client could take their business to another provider, perhaps permanently. 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.

1.4 Contributions

The contributions of this thesis are as follows:

- **Query Mix Model** We propose and 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. This is distinct from existing models which consider the type of query, but fail to consider the impact that the specific mix of queries currently running in the system can have on execution time.

- **Prototype Controller** We implement a prototype (Q-Cop) that uses the query mix model and the idea that clients will eventually time out to make more informed admission control decisions than previously possible.

  By using a more accurate predictive model for execution time, the system is able to more-accurately deduce how many requests it is able to handle concurrently within a given time-frame. This leads to less wasted effort and an increase in the total productivity of the server.

- **Experimental Evaluation** An experimental evaluation conducted using TPC-W-like benchmarks 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.

  This evaluation shows that, in a real-world application like the one modelled by TPC-W, Q-Cop yields a decrease in requests not served under overload, compared to alternate admission control techniques.

1.5 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents an overview of related work. Chapter 3 presents the methodology used in the thesis. Chapter 4 outlines the problem being investigated. Chapter 5 describes the implementation of our system. Chapter 6 presents our experimental results. Chapter 7 discusses the strengths and weakness of our approach. Chapter 8 provides conclusions and future work.

Chapter 2

Related Work

2.1 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 prior work in this area, which we classify into three categories.

2.1.1 Controlling MPL

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 processed concurrently by the server. Many admission control approaches are essentially methods for tuning the MPL.

Mönkeberg and Weikum [28] monitor a data contention metric called the conflict ratio to dynamically control the MPL. This is very early work in the area, outlining the effects of thrashing and contention as they relate to a highly loaded system. The goal of their paper is to reduce the need for the server operator to manually tune the server by setting limits on the number of connections accepted. The workload being considered is one with dynamic changes in load characteristics, making approaches that statically set the MPL setting inadequate. The proposed system self-adjusts its admission control to account for these changes. It also has the ability to cancel requests which are blocking due to database locks when the contention metric, which is a measure of how often database locks are blocked, reaches a critical level. Unlike other research on admission control, their paper uses a real-world trace of web server accesses. Due to the nature of the metric being used for control, which is a measure of the amount of data-contention in the system, relatively little information or processing is required before applying the control mechanism, allowing the admission control algorithm to be applied to a system without a large setup cost.

Liu et al. [25] propose approaches for tuning the MPL of an Apache web server using numerical optimization, fuzzy control, and heuristics based on resource utilization. Their work focuses on using the built-in capability of Apache to limit the number of client connections it accepts. They use hill-climbing methods to find the best setting for the number of connections concurrently processing within the web server with the goal of minimizing average response time. The resulting setting, however, is highly specific to the workload being considered and can show inconsistent or unstable results given high variability of response times. It is, however, an online system, dynamically adjusting to workload conditions. Certainly, limiting the number of connections provides an improvement over a system that allows itself to be thrown completely into resource contention.

Schroeder et al. [33] investigate how to effectively schedule requests, rather than how to choose an optimal MPL. Finding a good MPL is simply a requirement for being able to perform good scheduling. The system does not perform admission control, but rather queues and re-orders requests. They start with queueing analysis based on a study of the various factors affecting overload, including I/O- and lock-contention, to find an initial MPL value. They then use a feedback controller to dynamically adjust this value. The paper mentions that one of the goals of the investigation is to set the MPL low enough that there is room for the schedule to show differentiation in effectiveness, since there can be a range of MPLs which give similar throughput. This is a different goal from papers which aim to use MPL as their mechanism for optimizing performance. The mechanism used is a relatively simple feedback controller based on mean response time.

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 the number of requests not served because they are oblivious to query types and query mixes. While many MPL-based approaches attempt to dynamically adjust to varying workloads, or flash crowds, Q-Cop handles workload changes directly, by reacting to the mix of queries currently in the system. This is a strong advantage over MPL-based approaches, which must constantly adjust the MPL to account for changes in the moving average load, reacting indirectly to the change by monitoring the effect rather than the cause, effectively lagging their estimates of current load.

2.1.2 Monitoring and Controlling Response Time

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 [24] monitors load and probabilistically rejects requests when the load gets high 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 time and the desired response time.

This metric is effective as long as requests all tend to be similar in imposed load and average response time. Even in workloads with diverse request types, the Yaksha self-tuning mechanism will still provide much improved average response times under heavy loads, compared to no admission control, by simply ramping up the number of requests rejected. The indiscriminate nature of request rejection will, however, likely lead to many more requests being rejected than necessary.

Bartolini et al. [10] distinguish, using statistical measures of expected load deviations, between normal load and overload (or "flash crowds"). A performance model is learned dynamically and used to decide the rejection probability of requests under normal load and overload. The focus in their work is on meeting service-level agreements (SLAs) even under sudden extremely bursty loads. They recognizes that over-provisioning is a solution to average-to-high loads and is ineffective at mitigating overload. The system limits the incoming rate of admissions based on a monitoring module's measurement of average response time and session arrival rate. In order to deal with flash crowds, the system has the ability to operate in two modes, with the overload module essentially giving a much more reactive controller. The discard mechanism is similar to the probabilistic-rejection used in Yaksha.

SERT [40] 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. This is an approach that can be used in addition to admission control, by removing requests that have already gone over their time allotment. This can be beneficial in some cases, as in the system for which SERT was developed. In their a web search engine, requests for search results which have not been seen before or recently can take several orders of magnitude longer than cached results. However, as the SERT paper shows, terminating requests before they finish naturally can incur a significant cost on a server. Running those searches can place significant load on the system, reducing the number of cached requests that can be served. However, by performing the longer requests, the results become cached, leading to much faster processing for that request in the future. It is therefore in the system's best interest to attempt to perform long requests when possible in order to cache the results, even if the client has already timed out or otherwise cancelled the request, but to prevent long-running requests from negatively affecting the overall performance of the server when it is already heavily loaded. Since the processing time for the query is not known in advance, the system attempts to start all requests and then preempts any request that exceeds a timeout threshold. However, while the system investigated shows a benefit for future requests from caching, it is not necessarily the case in the majority of web applications. In general, early termination of requests can require a significant amount of processing time and infrastructure in internet applications. In applications that are handling a large volume of requests on a short time schedule, especially where most requests do not provide any real benefit for later requests, the benefits of preemption must be carefully weighted against the drawbacks. Therefore, Q-Cop is designed to reject requests rather than pre-empt already admitted requests.

Monitoring response time has the advantage, over pure MPL-based approaches, that it gives the operator of the system more direct control over a metric that is of concern to them. While the number of concurrent clients might be a useful metric to be aware of, it is usually the performance of client requests that is the driving concern. Both MPL-based and probabilistic approaches suffer, however, from a large degree of reliance on average queries. If the workload requests are sufficiently skewed, so that some queries have a larger effect on average execution time than others, probabilistic controller models will tend to reject many more requests than necessary in order to maintain their desired load level.

2.1.3 Approaches That Distinguish Between Query Types

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 [19] maintains moving averages of the response times 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 predetermined threshold which is determined experimentally using off-line execution of the expected workload.

Gatekeeper has the advantage that it is implemented as a proxy server, rather than a modification of existing software, so it is relatively easy to apply to any underlying server technology. The proxy uses preferential scheduling to admit smaller requests before bigger requests when overload conditions exist. Care is taken to avoid starving large requests. However, depending on the degree of overload, there are many situations where not all requests can be processed and some requests must be rejected. Gatekeeper functions best as a method for simply smoothing transient overload conditions in a server that is largely performing at the edge of its capacity. It functions less well at preventing extreme overload conditions because it is designed for ordering requests rather than rejecting requests.

Quorum [13] tracks the average service times of different request types and uses these times to decide if the sum of the waiting and service times of a specific request will exceed an administrator-specified timeout. The system's focus is on maintaining quality of service guarantees on systems comprised of multiple load-balanced server clusters. Particular care is taken to function on systems where source code is not available for modification. While Quorum could be deployed on a smaller scale, it is simply not targeted at that area of research, showing no experimental results for a cluster of fewer than four machines. It requires more complex classification and service level agreement targets, and makes some assumptions about the nature of the cluster of machines that it is protecting, such as the ability to reassign excess capacity. Additionally, the results as presented make use of traffic shaping and load balancing, making a comparison to a strictly admission control-based system difficult and possibly misleading. Quorum's main contribution is using algorithms similar to Yaksha and Gatekeeper, but on a much larger scale.

Both Quorum and Gatekeeper use more information than previous approaches, but they still do not account for query mixes. They build models for a specific workload and hence a specific distribution of requests. If the workload and mixes change, the Quorum model has to be re-learned and if the workload mix changes, Gatekeeper must determine its new thresholds. These approaches consider query types only in terms of how queries tend to have different average response times. While response time is certainly an important metric, it is also important that queries affect other running queries in different ways. Gatekeeper and Quorum consider query type explicitly, but their approach of averaging over recent requests captures some query mix information only incidently. The lack of explicit consideration for query mixes is the reason they both need both to recompute their models under changing loads and also for their moving average calculations. These drawbacks lead to less flexibility and make these approaches less practical because of the difficulties in deploying them than Q-Cop.

2.2 Unsuccessful Requests as a Performance Metric

Much work from a web server perspective focuses on throughput [8]. Since single-tiered servers can handle extremely high loads with minimal slowdown, simply pushing as much data as possible is a viable goal. In the realm of dynamic content, however, an issue much more likely to arise is the amount of time individual requests take before the user sees their results [30]. Under heavily loaded conditions, many individual queries can timeout on the client side. This is more prevalent in dynamic systems as individual queries tend to place much higher demand on system resources than a static request would. It is not unusual in very highly loaded systems for a majority of requests to not be returned to the user at all, even driving the system into live-lock [16]. This timeout effect is not captured by average response time. In fact, a client timeout might not even be counted [29] as a response at all, meaning that it could be in the algorithm's best interests to let some requests timeout. When timeouts have the potential of occurring, average response time can be a misleading and incomplete picture of the performance of a system, especially as it is easy to reduce average response time by rejecting all larger queries.

Q-Cop is designed to maximize the number of successful replies returned to the client by minimizing the number of requests that timeout or are rejected. By minimizing the number of requests that are not serviced, Q-Cop can allow a better user experience in interacting with an overloaded system.

2.3 Query Mixes

The interactions between queries running concurrently as part of a query mix can have a significant impact on execution time. This has been shown in prior work [2], and mix-aware query schedulers have been proposed [3, 4]. Their work focuses primarily on large batches of non-interactive queries, with the goal of reducing the total processing time. These schedulers focus on scheduling long running business intelligence queries, and they do not perform any admission control. In contrast, we focus on admission control, particularly for three-tier systems. In a web-based interactive system, with relatively short requests, re-ordering certain requests is of limited benefit. Since response time is generally quite important, significantly delaying requests is potentially worse than quickly returning an error message indicating overload.

2.4 Experiment-Driven Performance Modelling

A key feature of Q-Cop is using an experiment-driven approach for modelling query interactions, similar to the approach used in Q-Shuffler [3, 4]. Q-Cop improves on previous work by using Latin Hypercube Sampling (LHS) [22] instead of random sampling, and by constructing a performance model based on one set of experiments that works for different workloads, including different request rates and mixes.

The experiment-driven approach to performance modelling 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. [20] use an experiment-driven approach combined with machine learning models to predict performance metrics for database queries. Their work demonstrates the feasibility and effectiveness of the experiment-driven approach for database performance modelling. The experiment-driven approach has recently been used for tuning database configuration parameters [7, 18]. An infrastructure for running experiments in a data center has been proposed [39], which allows an experiment-driven approach to be conducted with less manual intervention. Oracle 11g uses an experiment-driven approach to test the recommendations of their SQL Tuning Advisor before implementing these recommendations [11]. As in previous work, we adopt an experiment-driven performance modelling approach for its simplicity and robustness.

Chapter 3

Methodology

3.1 Experimental Environment

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

3.1.1 Hardware

All of the experiments described in this thesis have been 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 the workload we use is being done in the same tier, namely the database tier, we conduct 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, for a total of 4 CPU cores in each machine, 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.

3.1.2 Software

The client and server machines run the OpenSUSE Linux distribution. The kernel version is 2.6.22.18-0.2 (x64 SMP). The web server is Apache version 2.2.9. The application server is version 4.1.37 of Apache Tomcat. The database 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 [32] is generated using httperf [29] 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 [8]. 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 [27]. Internet Explorer versions 7 and 8 are reported to be used by the majority of users [26] (about 48% compared to 20% for Firefox). We were unable to find definitive sources for the timeout limit used in Firefox. We assume that 1 second of this 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 Q-Cop and other admission control algorithms that distinguish between query types provide for the ability to set a per query time time limit, 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 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 TCP connection.

3.2 Workload

We use the TPC-W benchmark [34] as the basis for generating our workload. The series of requests issued by httperf to the web server is generated to adhere to the Browsing Mix distribution of requests based on the client decision process described by the TPC-W benchmark specification. The details of the request distribution are provided in Table 3.1 and described below. Exactly the same logs are used for each experiment to ensure that the load in all experiments is identical.

We describe our workload as 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 [12] and use a subset of these TPC-W servlets in our workload. The Wisconsin implementation is written for the DB2 DMBS, while we are using the Derby DBMS. Using Derby required modifying the SQL statements used by each servlet because of differences between the SQL dialects supported by DB2 and Derby. 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 the other issues only 2% of the requests. In total, this accounts for only 5% of the requests. In our workload, these requests are replaced by slightly increasing the percentage of remaining requests. Note that we do not have a completely uniform redistribution of requests because of randomness intrinsic to the log generation process.

Table 3.1 provides the name of each of the TPC-W servlets, a database query type number (which we assign for easy reference), the percentage of requests each contributes to the Browsing Mix of the TPC-W workload, the percentage each contributes to our TPC-W-like workload (Our Mix) and the adjusted workload (New Mix) discussed in Section 6.2, and whether or not it was included in experiments conducted to generate our Query Mix Model (QMM), as described in Section 5.3. Additionally, we point out that the Search Results request actually generates one of three different types of database queries, two of which are included in the query mix model while the third has a negligible execution time and is not included.

Servlet ⇒ Database Queries	Query type #	Browsing Mix (%)	Our Mix (%)	New Mix (%)	Query in QMM
Home Interaction	–	29.00	31.30	31.26	No
Product Detail	–	21.00	21.83	16.38	No
Search Request	–	12.00	12.07	13.31	No
Search Results		11.00	11.03	12.22	
⇒ Author Search	1				Yes
⇒ Title Search	2				Yes
⇒ Subject Search	–				No
Best Sellers		11.00	11.98	**21.22**	
⇒ Setup (Part I)	3				Yes
⇒ Main (Part II)	4				Yes
New Products	5	11.00	11.79	5.61	Yes
(Multiple Servlets)					
⇒ Related Products	6				Yes
Total Browsing		95.00	100.00	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	0.00	–

Table 3.1: List of TPC-W servlets, query types, and where they are used.

Because Derby does not support the LIMIT SQL keyword, we also changed the Best Sellers query. Instead of computing the 50 best selling books of the most recent 50,000 books purchased, 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, labelled Setup (Part I) and Main (Part II) in Table 3.1. When we refer to the Best Sellers query we are referring to the Main (Part II) query which does the bulk of the work.

3.3 Timeouts and Performance Metrics

In this section, we explain the importance of choosing a good performance metric and describe the system of evaluation used in this thesis.

Requests may be not served either because they are rejected or because they encountered a client timeout. Client timeouts happen when either the browser software does not receive a response from the server within a browser-specified period of time or when the user themself navigates away from the page before a response is received. Timeouts become an issue only under overload and are problematic for users as well as operators. Users do not like timeouts because they are a waste of their time with no actual response at the end. Operators do not like timeouts because they are a waste of their resources with no benefit to the processing time spent. Reducing the occurrence of timeouts is generally an indication of better overall quality of service. However, in an admission control system, it is relatively easy to completely eliminate timeouts, if that is the only metric being considered. The system could simply reject queries very aggressively and be sure that none would ever time out.

Our metric is, instead, the total number of requests not serviced, which is the sum of requests rejected and requests which time out. An unsuccessful request represents a client who was attempting to get a response from your server and failed. In a business context, this potentially represents a lost sale or customer. As such, reducing the occurence of unsuccessful requests is highly desirable.

Chapter 4

Overload and Query Mixes

4.1 The Perils of Uncontrolled Overload

Existing admission control studies include graphs depicting how in a three-tiered system throughput can plummet and the response time can skyrocket, if admission control is not performed [19, 24]. 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. Of course, this is an unrealistic model of client performance, since client browsers and users will not wait indefinitely for a response.

Figure 4.1 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 4.1 as seen in previous work. This is because Figure 4.1 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 mean response time.

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

Clearly, a server must be capable of coping with overload conditions in some way. A popular method for controlling overload conditions 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). Figure 4.2 shows the steady increase in the total number of queries not served as load increases. The goal of this work is to turn these unsuccessful requests into successful replies.

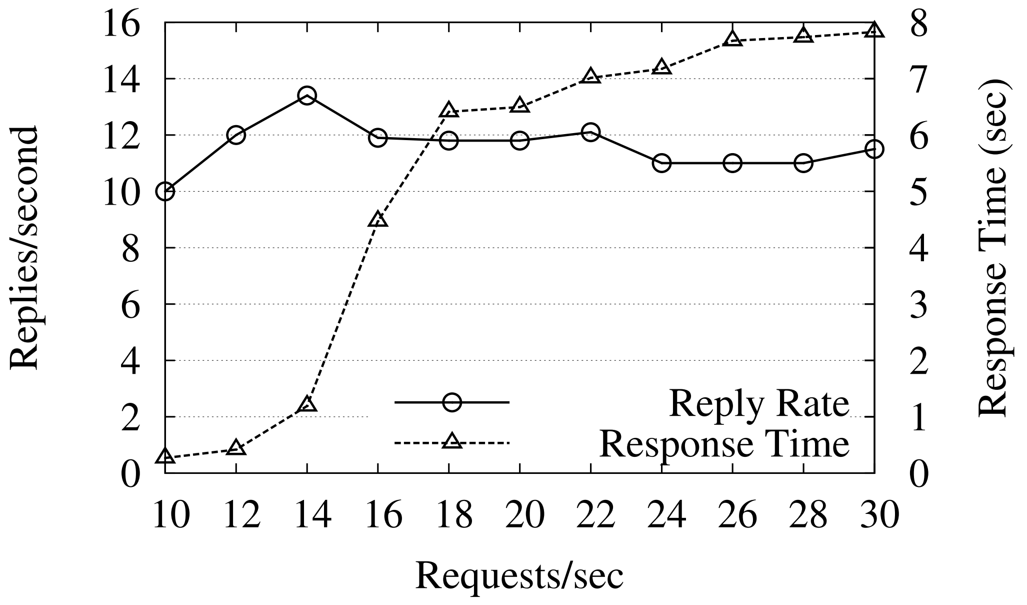


Figure 4.1: Mean throughput and mean response time of a three-tiered system without admission control.

4.2 Query Mixes

In this section we motivate the need for modelling 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 the four requests sampled for QMM, as described in Section 3.1 and specifically as enumerated in Table 3.1. We use a sampling methodology similar to that described in detail in Section 5.3.1, 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 is required for a response (i.e., there are no timeouts). For each experiment of one particular mix, we calculate an average response time. In post processing, we split the response data into groups by query type and then compare executions occurring at the same MPL in order to investigate requests that would appear the same from an admission control perspective if mix is not considered. For each MPL, we identify the experiment (query mix) with the minimum and maximum average response times, and we also calculate the overall average response time at that MPL. Figure 4.3 shows the response time for four TPC-W request types: Best Sellers (4.3(a)), Title Search (4.3(b)), Author Search (4.3(c)), and New Products (4.3(d)). The figures also show 95% confidence intervals for the average response times.

As expected, the response time of the queries increases as the MPL increases because the system becomes more loaded. The interesting observation about these graphs is the difference between the minimum, maximum, and average response times at the same

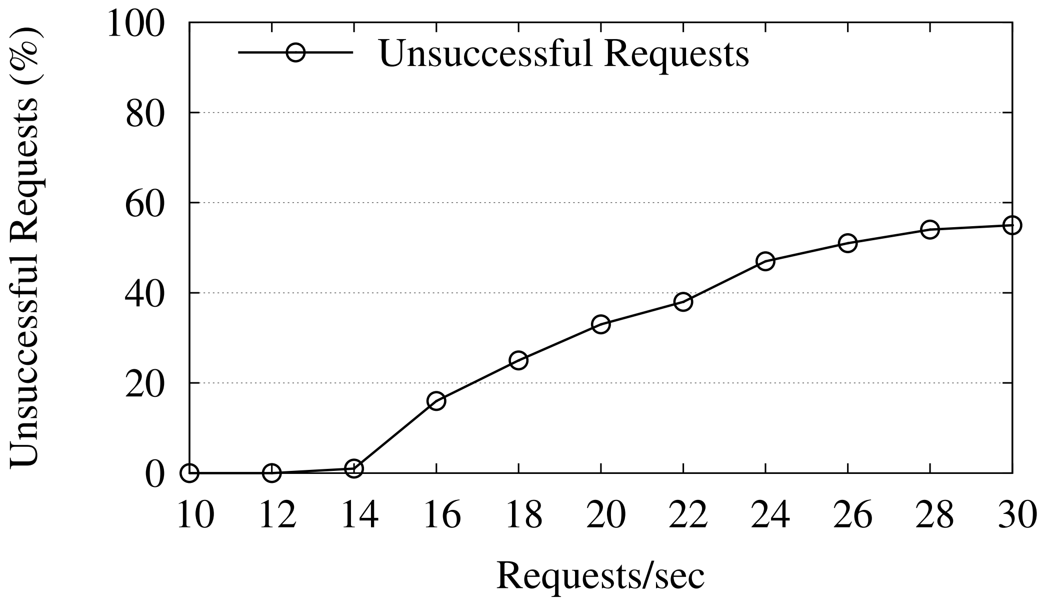


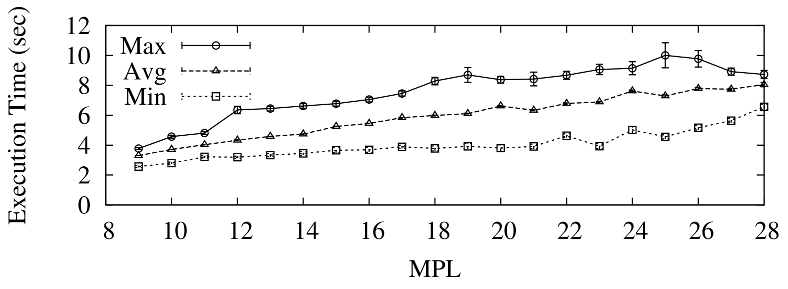
Figure 4.2: Unsuccessful requests in a three-tiered system without admission control.

MPL. These 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 query execution time. Therefore, looking solely at the MPL is insufficient information to accurately predict the average query execution time in the system, and so the admission control algorithm should take into account the current query mix in the system.

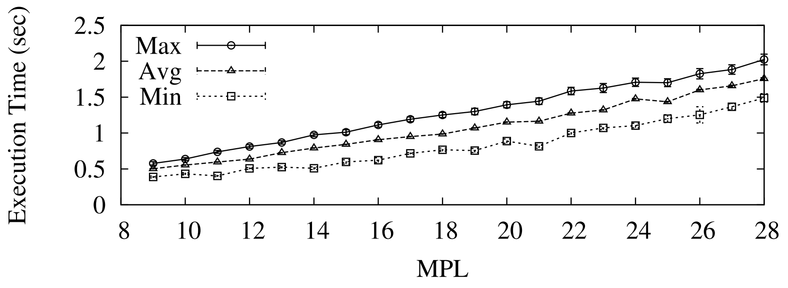
These figures also show that the best MPL at any time depends on the mix of queries being processed at that time. For example, consider Figure 4.3(a) and the average amount of processing that can be carried out in six seconds. The MPL that can be supported without having the Best Sellers request timeout depends on the query mix. If the query mix happens to be the one that causes the maximum response time for the Best Sellers request, 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 20 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 in Chapter 6.

4.3 CPU Correlation

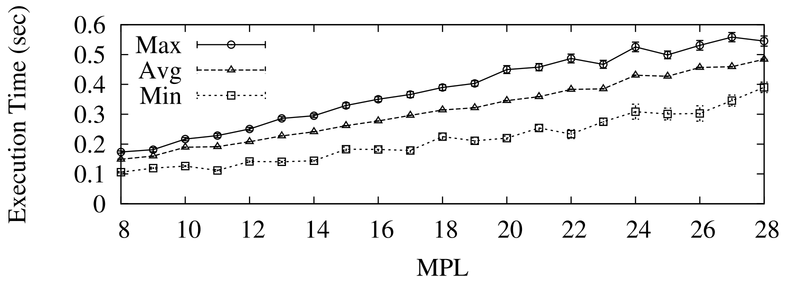
Next, we show that CPU utilization is not correlated to the execution time of a given query type. These experiments were performed by running a range of requests rates from



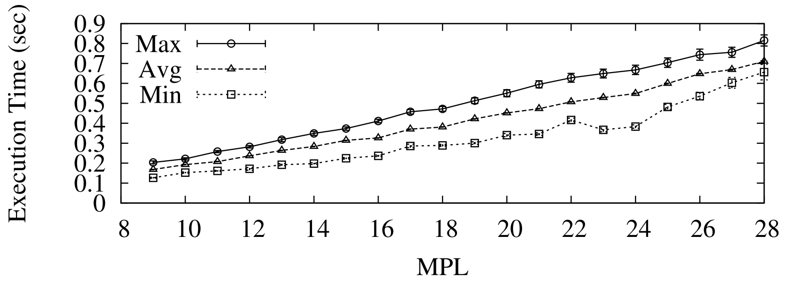
(a) TPC-W Best Sellers Request



(b) TPC-W Title Search Request



(c) TPC-W Author Search Request

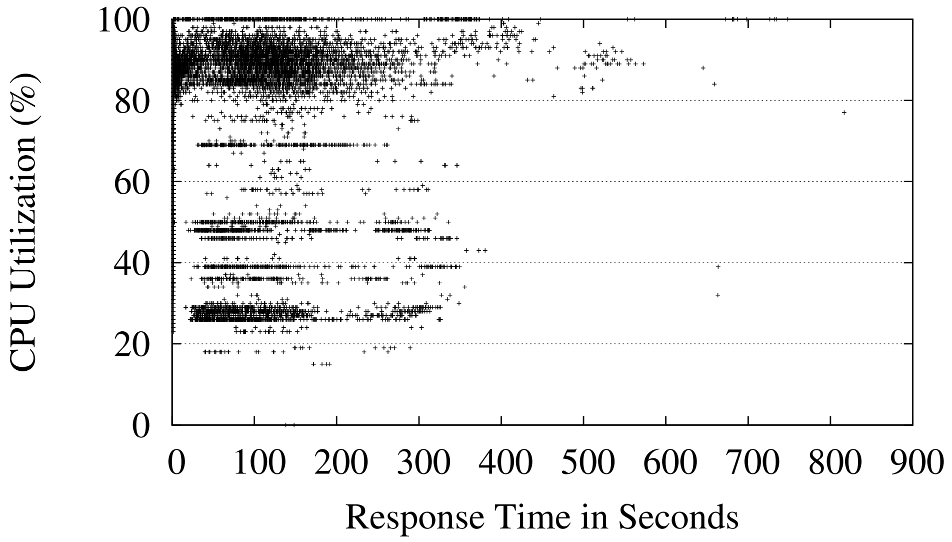


(d) TPC-W New Products Request

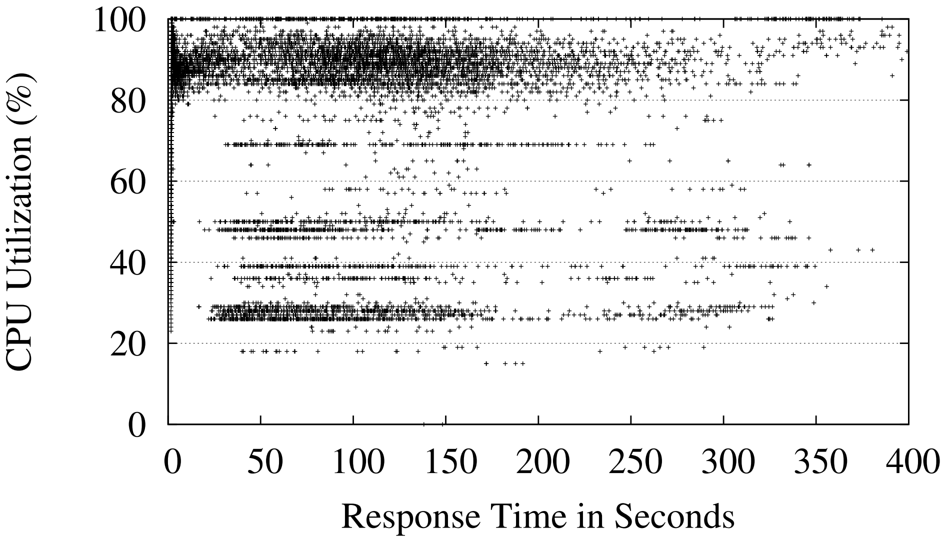
Figure 4.3: Variation of average query response times with different query mixes.

ten through thirty requests per second with no admission control. The execution time for each query was logged, as was the CPU utilization averaged over the five seconds prior to the query starting execution. We first look in detail at the Best Seller query, shown in Figure 4.4, to investigate correlation. In the range of CPU utilizations between 80%

and 100%, we see a very wide range of execution times. Similarly in the range of CPU utilizations between 0% and 80%, we see a slightly smaller, but still quite varied, range of execution times. There is not any CPU utilization level at which there is a response time that could be expected. The correlation coefficient for the data shown is 0.025, almost no correlation between CPU utilization and the response time of the query. In the complete data set, there are some points out to 600 seconds, with a very few as high as 800 seconds, as seen in Figure 4.4(a). The range of Figure 4.4(b) has been cropped on the x-axis to 400 seconds. Along the x-axis, we show the response time for each execution of the query, while the y-axis shows the CPU utilization at that time.



(a) Uncropped

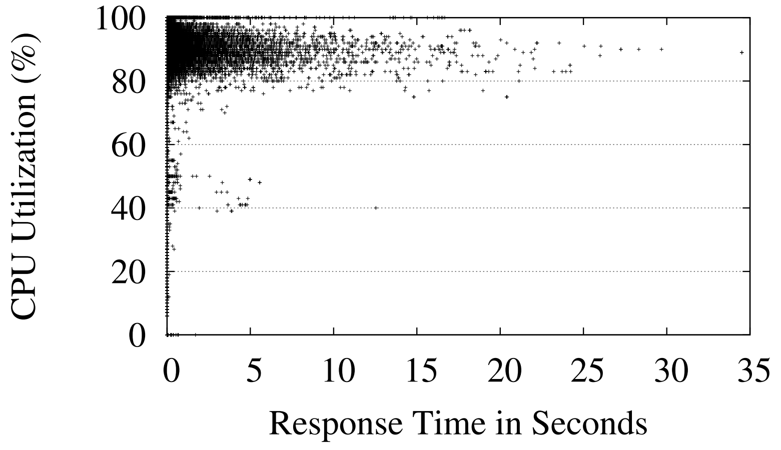


(b) Cropped to 400 seconds

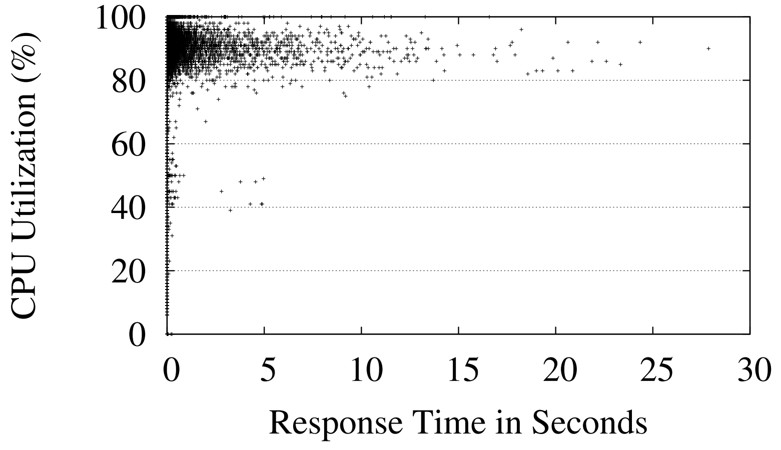
Figure 4.4: CPU utilization for different response times of the Best Sellers query

Figure 4.5 shows scatter plots of the CPU utilization observed while running different query mixes against the response time of the Title Search (4.5(a)), Author Search (4.5(b)), and New Products (4.5(c)) queries in these mixes. In these graphs, if there was a correlation to be found, we would expect the points to largely align on the bottom-left to top-right diagonal. Instead, we see a large cloud of points at high utilization extending through a wide range of execution times.

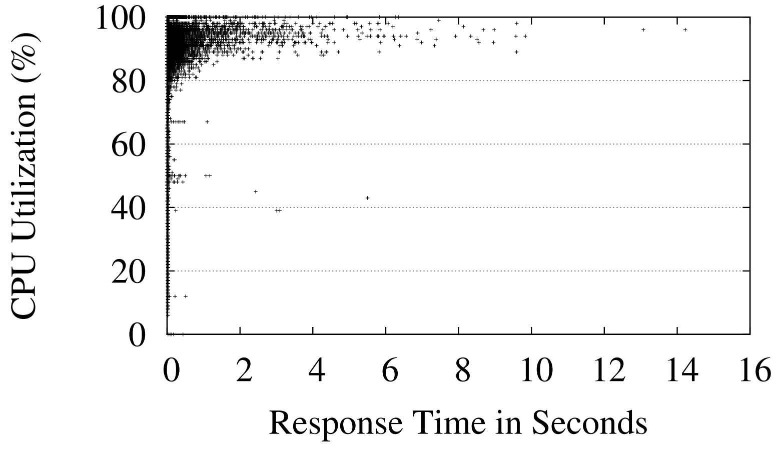
This lack of correlation is counter-intuitive. It would seem from casual consideration that response time should depend on CPU utilization, but that is not the case. Indeed, since the database fits in memory and the network is not a limiting factor, CPU is the only hardware resource being taxed, and therefore a resource-utilization approach for execution time estimation in dynamic web applications is not viable in this environment. The bottleneck in the system could be, for instance, database locks or some other software resource. We proceed in an attempt to find another method for estimating execution time for use in admission control.



(a) Title Search Query



(b) Author Search Query



(c) New Products Query

Figure 4.5: CPU utilization for different response times

Chapter 5

Implementation of Q-Cop

5.1 Implementation Framework

To compare different admission control schemes, 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, as that is where most of the work is being done. [1]
- 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 using information available at that time. We do not pre-empt requests after they have been accepted.

In this thesis, we refer to *requests* at the web server level and *queries* at the database level. In TPC-W, requests may cause multiple queries, but no more than one concurrent query. That is, the number of queries is never more than the number of requests in the system at any time. However, the total number of queries handled in the system is generally significantly higher than the total number of requests handled.

[1] This could have also been done elsewhere in the system, for example via a proxy [19], but we used the database for ease of implementation.

5.2 Implementation of Prior Admission Control Approaches

5.2.1 MPL-based Approach

One popular and simple technique for performing admission control is to limit the Multi-Programming Level (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 [25, 28, 33]. 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.

Instead of implementing a specific MPL-based approach, we have decided to emulate the performance of a good MPL-based approach. In Section 6.1, we conduct a series of experiments to determine the best MPL value for our environment. This value provides comparable performance to what an MPL-based approached might achieve without dynamic adjustment. We compare the performance of Q-Cop against this best MPL.

5.2.2 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 [19] clearly shows that in a TPC-W workload the query type has a big impact on the query's response time. The authors develop a system that makes admission control decisions based on the type of query being considered for admission and the current system load, as measured by a moving average response time metric.

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. Load is measured more directly than MPL-based approaches by monitoring the total number of running queries of all query types within the system.

In this learning phase, we run the full workload without admission control and collect the following information: the query type, the number of other queries executing when the query arrives (which in some sense captures the load on the system when the query begins processing), 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. Table 5.2.2 shows the coefficients for our experiments.

Query Type	c_i	C_i
Author Search	41.3	102.5
Title Search	6.1	41.7
Best Sellers I	0.4	-0.4
Best Sellers II	302.2	1893.0
New Products	24.2	-20.1
Related Products	0.3	-0.4

We note that some of the query types show negative coefficients, which is unexpected since the coefficients represent the amount of time that other running queries add to execution time. This is likely an indication of some inaccuracy, especially as it occurs mainly for the smaller queries which are generally less predictable than the Best Sellers query. It is also possible that there are certain queries that benefit from the presence of others, for instance by taking advantage of caching, as seen in previous work [2].

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

5.3 The Query Mix Model and Q-Cop

The experiments in Section 4.2 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 mode 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 gather 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. As seen in Section 6.2, these estimates allow the system to predict execution times even with different mixes of those queries.

5.3.1 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 conduct experiments and collect performance data. However, the number of different possible query mixes is exponential, meaning that exhaustively running all possible query mixes would be infeasible. and we therefore need to sample the space of possible query mixes. Having an effective sampling approach is important, since the eventual model will rely on having sufficient input data to predict behaviour over a variety of different loads. We sample the N-dimensional space (where every query type is a dimension) using a Latin Hypercube Sampling (LHS) [22] protocol. This protocol significantly reduces the number of experiments necessary while providing good coverage of the possible mixes. The LHS protocol has been successfully used in other work on the experimental tuning of database systems [18].

The maximum number of queries in each dimensions is the same, although this is chosen for simplicity and is not important in the running of the experiments. Some accuracy could be gained by restricting each query type to have no more active requests than could be handled in a moderately-overloaded system. However, this would require either prior knowledge of the performance of the system or further experimentation to discover the limits for each query.

When collecting samples and making admission control decisions, we consider only the query types that place a significant load on the system. Query types that do not place a significant load on the system are excluded from Q-Cop 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 better coverage with the LHS protocol for the amount of experimental time spent. 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 3.2) has six types of requests that a URL can refer to. These requests are listed in Table 3.1, which shows the descriptive name of the query, the distribution of queries in our workload, and whether the query is included in our admission control decisions. 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 those resulting from the Search Request, Product Detail, and Home Interaction requests, and the Subject Search query from the Search Results request. We exclude these queries from our sampling and from our admission control decisions, which leaves six query types considered by Q-Cop for our workload. These six query types, as shown in the last column of Table 3.1, are Author Search, Title Search, Best Sellers (Setup), Best Sellers (Main), New Products, and Related Products.

The LHS protocol draws a specified number of samples from the full space of possible query mixes such that there is uniform coverage across the full space. The full space of query mixes we used in our TPC-W experiments can be described as follows. If M is the number of queries executing simultaneously, n_i is the number of queries of type i, and N is the number of different types of queries, we sample from the space where $1 \leq M \leq 40$ and $0 \leq n_i \leq 40$ such that $\sum_{i=1}^{N} n_i = M$.

5.3.2 Experiment-Driven Data Gathering

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 is done in order to maintain a constant load on the database system and keep 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, such as "ARTS", and an Author Search which takes a string to search for, such as "Tolkien". 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 data-gathering experiments, we use LHS to select 1,000 query mixes. These runs result in approximately 70,000 query executions requiring about forty-two hours of execution time). While this is a significant amount of experimentation, it is a one-time offline process Longer requests account for a smaller fraction of those executions than smaller request because fewer would run within the time allotted for each experiment. However, the smallest number of queries executed for any query type was 395.

5.3.3 Constructing the Query Mix Model

To derive a predictive model from the raw data, we use the Waikato Environment for Knowledge Analysis (WEKA) toolkit [38]. We built and tested models using several different learning algorithms available in WEKA, including linear regression, locally-weighted linear regression, Gaussian Processes, and multi-layer perceptron. When compared with the linear regression model, some of the more sophisticated models showed improvements of approximately 5-10%, as measured by relative mean squared error. However, we use a simple linear regression model due to the additional complexity required by the more advanced models. An interesting direction for future work (discussed in Section 7.6) 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 coefficients learned from the training data for each query type. 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 constant factor, C_i, which is 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: $Est_i = (c_{i1} * n_1) + (c_{i2} * n_2) + \ldots + (c_{iN} * n_N) + C_i$. Table 5.1 shows the coefficients obtained for each of the query types. As in Section 5.2.2, we note that some query types show negative coefficients for similar reasons.

Query	c_{1i}	c_{2i}	c_{3i}	c_{4i}	c_{5i}	c_{6i}	C_i
1. Author Search	-2.4	11.4	-22.9	2.0	3.2	1.9	121.0
2. Title Search	-6.6	14.7	-135.7	11.6	14.3	48.5	486.9
3. Best Sellers I (Setup)	0.0	0.3	3.4	0.0	1.92	0.2	-3.0
4. Best Sellers II (Main)	31.5	30.2	586.5	675.6	102.5	18.5	3063.2
5. New Products	34.7	-6.5	-25.8	21.0	0.0	-6.3	132.0
6. Related Products	-1.0	-0.4	6.4	1.6	8.0	-0.5	3.5

Table 5.1: QMM Coefficients

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). In our implementation, we have chosen to return an HTTP 500 error as that allows us to send additional debugging information to the client. We have done experiments using HTTP 503 errors as well, which is an extremely simple change in the implementation.

We investigated the effect of sample size on model accuracy using WEKA's built-in accuracy measures and ability to split the input data into training and testing sets of various sizes. The resulting data shows no indication that increasing the sample size would yield a more accurate model, meaning that our sample size is large enough.

Chapter 6

Experimentation

6.1 Experimental Evaluation

We now evaluate how well each of the different methods we have described in Chapter 5 does at minimizing the number of requests that are not handled. All experiments in this section are conducted using the TPC-W-like workload described in Section 3.1. However, before we can compare the different approaches we need to first determine the best MPL against which to compare.

6.1.1 Finding the Best MPL

As noted previously in Section 5.2.1, a popular technique for performing admission control is to limit the maximum number of concurrent requests (i.e., setting the MPL). The difficulty with this admission control 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. With a high MPL, the system is accepting a large number of requests for processing simultaneously, each of which tends to increase the system's average execution time. If enough requests are accepted, the requests which require longer processing times can take longer to finish than the client is willing to wait.

Therefore, we begin by conducting a series of experiments to determine the best MPL for our environment and workload. The metric of interest is the percentage of requests that *are not serviced* as the load on the system is increased. This includes both the number of requests that are rejected by the admission control mechanism and the number of requests that time out at the client (a 30 second timeout is used).

Figure 6.1 shows the percentage of requests not serviced as the number of requests per second increases. In looking at our raw data, we can see that MPL = 30 and MPL = 40 have the lowest number of total unsuccessful requests. We present those results and 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 very light loads while MPL = 5 does not do very 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 slightly worse than MPL = 30 and MPL = 40.

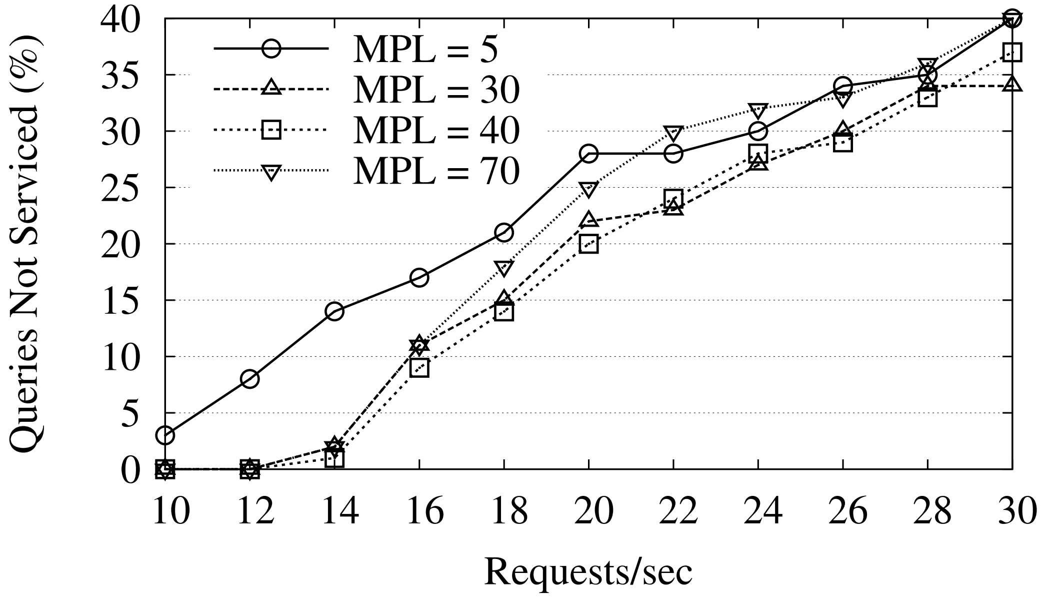


Figure 6.1: Percentage of queries not serviced for different MPL values.

To better understand the behaviour of the different MPL values, we examine both the total number of requests that are not serviced (total) and the number of requests that were rejected (rejected). Figure 6.2 plots these values for MPL = 5 and MPL = 30 while Figure 6.3 plots MPL = 40 and MPL = 70.

Figure 6.2 shows the total unsuccessful requests and the total rejected requests. The difference between the values is the total requests that encountered timeouts. These requests were rejected because adding the request would have exceeded the current maximum multi-programming level. Recall that requests may be unsuccessful either because they are rejected by the admission control policy or because they timeout. This graph shows that for MPL = 5 and MPL = 30 the unsuccessful requests are predominantly due to rejections rather than timeouts, accounting for more than 99.9% of the total unfinished requests.

Figure 6.3 shows that for MPL = 40 nearly all unsuccessful requests are due to rejections, accounting for more than 98% of the total unfinished requests, while MPL = 70 has only 82.5% due to rejections. For request rates of 22 – 30 requests per second, MPL = 70 rejects roughly the same number of requests as MPL = 40. However, MPL = 70 suffers from significantly more timeouts at all request rates. Note that MPL = 50 could be considered a viable candidate as well, if it were to show less unsuccessful requests than MPL = 40 by admitting somewhat more requests. However, after examining the data, we find that it does not perform as well as MPL = 30 or MPL = 40, as its increase in

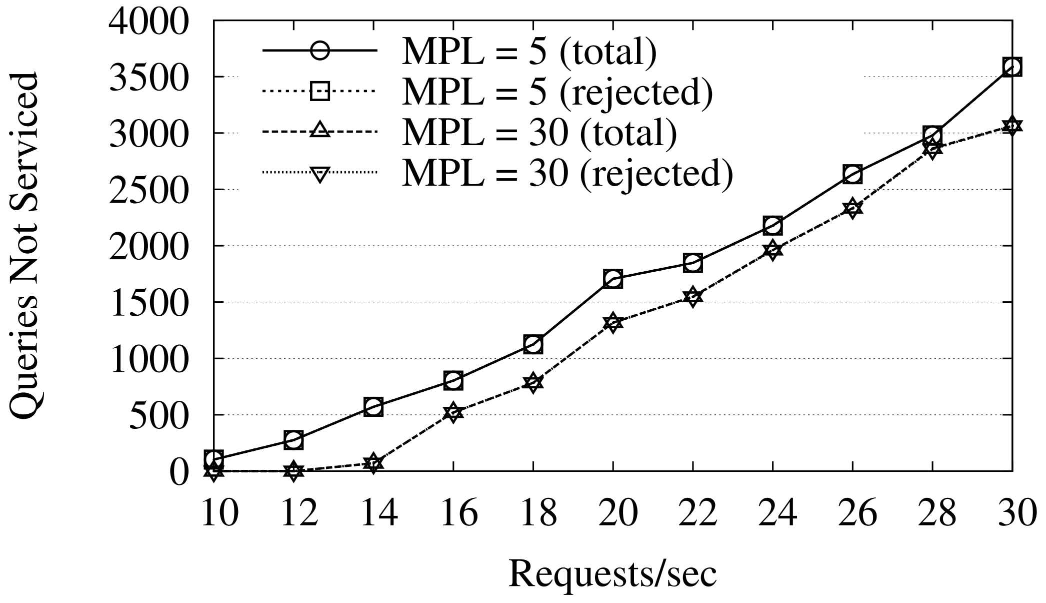


Figure 6.2: Total unsuccessful and rejected queries for different MPL values.

timeouts outweighs its decrease in rejections, and we therefore omit it to reduce clutter. This indicates that although the server is willing to permit more simultaneous queries, they combine to create a load that the server is unable to handle within a reasonable amount of time and as a result client requests start to timeout. Upon closer inspection of the raw data, we do see that for some loads there are a small number of timeouts for MPL = 30 and MPL = 40, with slightly more timeouts for MPL = 40. There are no timeouts for any MPL lower than 30.

At 22 and 30 requests per second, MPL = 70 rejects noticeably more requests than MPL = 40. It might seem strange that MPL = 70 rejects more requests than a lower MPL value, because one might think that a higher MPL should always reject fewer requests. However, this is not necessarily true. If too many queries are permitted into the system, long-running queries may not complete before new long-running queries arrive. These long-running queries accumulate over time, eventually occupying enough of the 70 available spots that more requests have to be rejected. To make matters worse, MPL = 70 not only rejects more requests under higher loads, but it also suffers from more timeouts. From the empirical evidence, we can see that MPL = 70 is simply not restrictive enough for this load.

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 mechanisms is very similar and we break the tie by observing that across the range of loads examined, MPL = 40 had a slightly lower number of total unsuccessful requests and overall average of unsuccessful requests. Since the total number of unsuccessful requests is very comparable for MPL = 30,40 and MPL = 20,50 show significantly more unsuccessful requests, we do not further examine other MPL values in the 30 – 50 range.

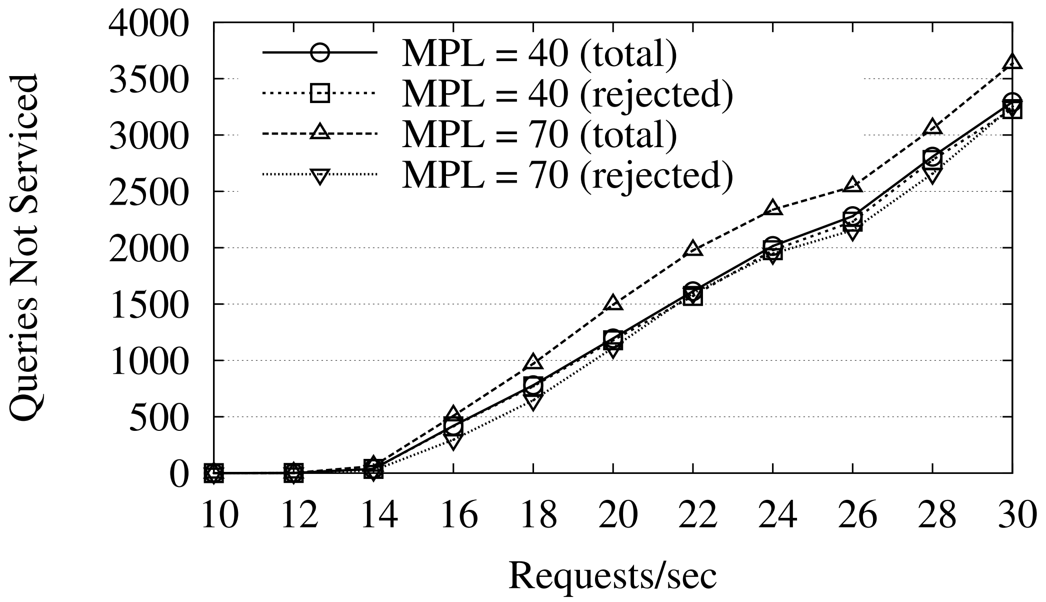


Figure 6.3: Total unsuccessful and rejected queries for different MPL values.

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.

6.1.2 Comparing Different Methods

The graph in Figure 6.4 compares the percentage of queries not serviced using different admission control techniques. Table 6.1 enumerates these methods, the label used in the graphs in this section to refer to these techniques, and which section of the thesis provides a detailed description of the method.

Graph Label	Admission Control Method	Thesis Section
NoAC	No Admission Control	Section 4.1
MPL = 40	The best MPL for this environment	Section 2.1.1
TYPE	Query Type Approach	Section 5.2.2
Q-Cop	Query Mix Model Approach	Section 5.3

Table 6.1: Different admission control techniques being compared.

As can be seen in Figure 6.4, the performance of Q-Cop is quite good across the full range of loads examined. Figure 6.5 (discussed in more detail later) provides a view of only the two best methods. This graph shows the significant difference between these approaches more clearly than in Figure 6.4. Figure 6.4 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 request are not serviced before the client times out (i.e., within 30 seconds).

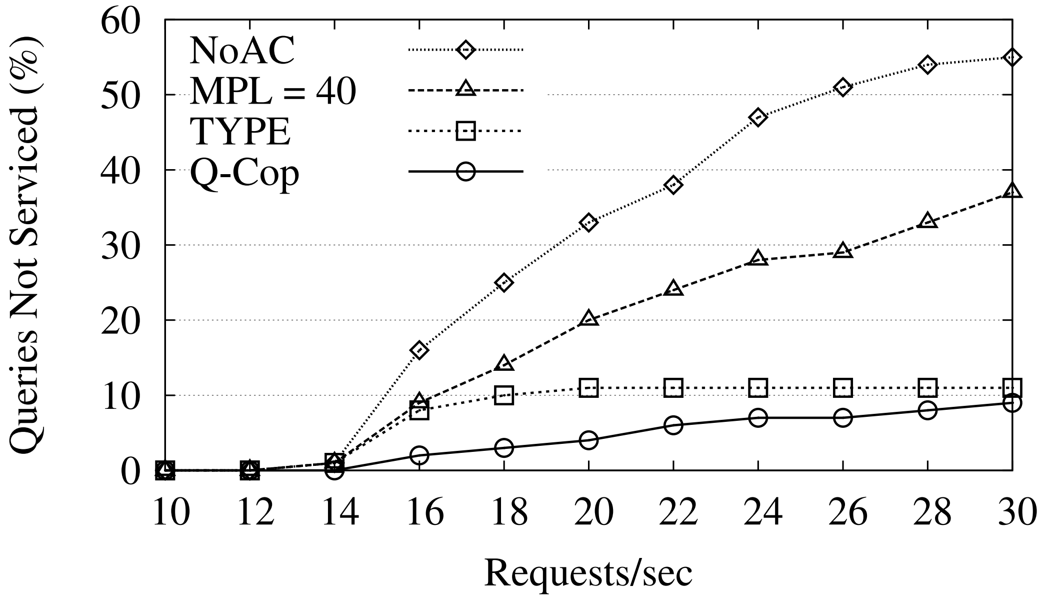


Figure 6.4: Percentage of queries not serviced as a function of load for different admission control policies.

Table 6.2 shows the average percentage of requests 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 has reduced the total number of requests not serviced for each of the methods explored. This 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 6.2: Average percentage of queries not processed across all loads.

As a result of the very large number of rejections caused by MPL = 40's admission control, it has substantially more queries that are not serviced than Q-Cop and TYPE. A key problem with an MPL-based approach is that it does not distinguish admissions by the type of query. It will therefore just as happily 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 the large Best Sellers requests.

Figure 6.5 provides a more detailed view of the two best techniques. It plots the total number of requests not serviced as a function of the number of requests per second. With low request rates (no more than 12 requests per second), the system is sufficiently provisioned to be able to handle all of the requests. As the request rate increases, the system is unable to service all of the requests. The difference between the total and rejected lines shows the number of queries which timeout.

Note that although the TYPE and Q-Cop approaches reject about the same number of queries for the different request rates, Q-Cop's approach (which uses information about the query mix) has significantly fewer queries not serviced. This is because Q-Cop avoids significantly more timeouts, as seen in Figure 6.5. These timeouts are avoided because it makes better decisions about when to admit the Best Sellers requests. Q-Cop rejects queries at different times than TYPE by using information about the expected run time of an arriving query given the mix of other queries currently executing. Q-Cop's more-informed decisions help the server to process more requests and as a result the clients experience significantly fewer timeouts.

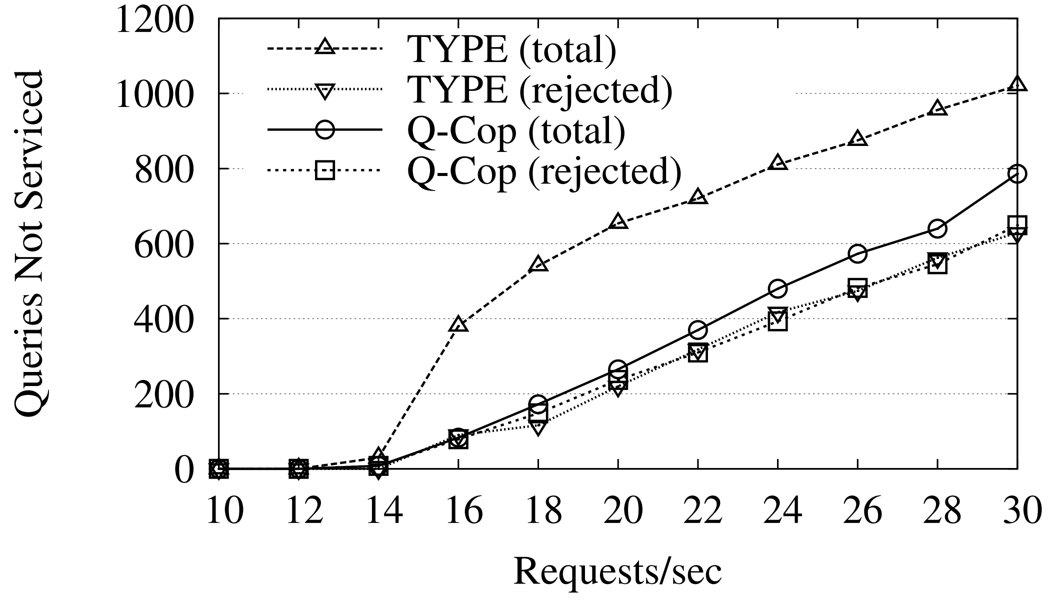


Figure 6.5: Number of rejected and unsuccessful queries versus the request rate.

6.1.3 Average Response Times

To examine the effect that these admission control schemes have on the average response time, Figure 6.6 plots the mean average response time versus the request rate as measured by the clients. This graph must be carefully interpreted because response times for rejected requests *are* included (and are typically very low) and requests that time out are *not* included. More importantly, the goal of the techniques we have implemented is *not* to minimize mean response time, but instead to minimize the number of requests that cannot be serviced. However, the graph, in comparison with Figure 4.1, does show that the mean response time is now more controlled when compared with the mean response times observed without admission control. 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.

Figure 6.6 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 requests than the other two approaches.

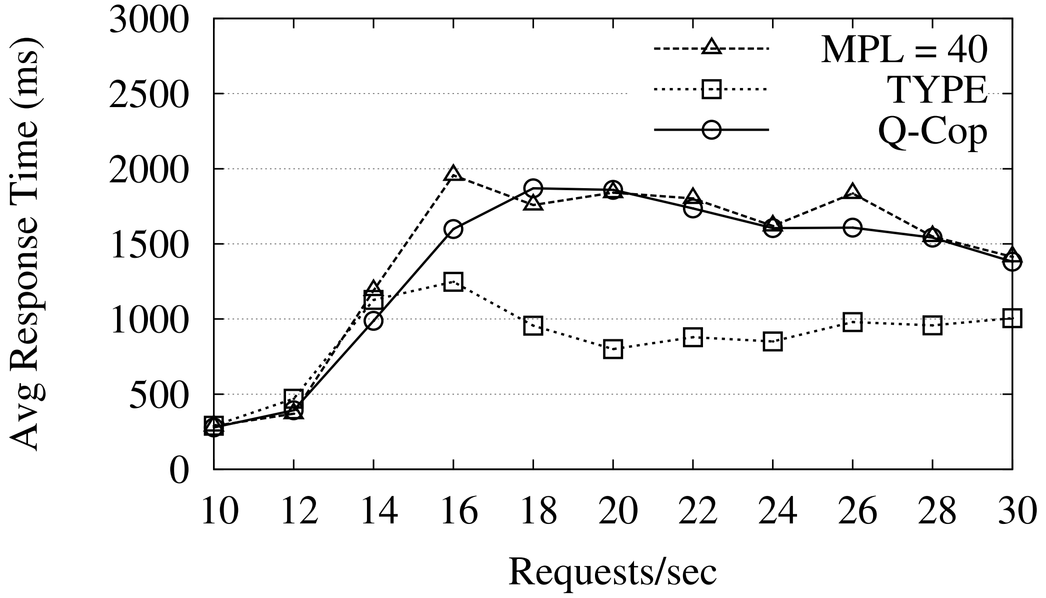


Figure 6.6: Response times

6.1.4 Probabilistic Approaches

Q-Cop also provides significant advantages when compared with probabilistic-based approaches to admission control, which to some extent are utilized in systems like Self* [10] and Yaksha [24]. If probabilistic-based admission control is done without considering the type of request, it will suffer from the problem of rejecting lots of small queries that may have easily completed within the time limit. If it does consider the type of query and only rejects large queries, it will suffer for the same reasons that the type-based approach fails. Namely, it does not have sufficient information to correctly determine *which* requests to reject and will therefore either reject too many requests or cause too many timeouts. Figure 6.5 shows that although a method may be able to reject the same number (or percentage) of requests as Q-Cop, it may not make the correct decision regarding which queries to reject, resulting in considerably more timeouts than Q-Cop.

Even though a probabilistic approach may be able to determine the correct percentage of large requests to admit, perhaps using control-theoretic approaches, it will not be able to make the informed decision about precisely which of those large requests to reject and when to reject them. To further make this point, we conduct a simple experiment where we compute the percentage of Best Sellers requests rejected by Q-Cop at a certain load. At a rate of 28 requests per second Q-Cop, rejects 545 requests in total (59.0% of the Best Sellers requests). 95 of the admitted requests time out which is 10.3% of the total number of Best Sellers requests and 1.1% of the total number of requests. We implement a simple probabilistic-based scheme and configure it to reject the same percentage of Best Sellers requests as were rejected by Q-Cop. This approach results in a comparable number of Best Sellers queries being rejected – 565 (61.1% of Best Sellers requests). We reject only Best Sellers, since Q-Cop only rejected Best Sellers. However, the probabilistic approach results in 156 timeouts, 1.6 times as many timeouts as resulted from using Q-Cop. Note that the probabilistic approach actually rejected *more* requests 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-based scheme. These rejections result in fewer timeouts because queries are prevented from running only if they are projected to interact poorly with the currently-executing mix.

6.2 Changes in Workload

One of the big advantages of our approach is that once the initial Latin Hyper-squares experiments have been conducted and the model has been constructed, it should work well for any changes to the workload that do not change the underlying queries. In web-based applications the query types are typically all known in advance, so this is a completely reasonable assumption. Previous work has shown that query times are quite similar across query types, with parameters to the query having significantly less influence on the query times than the actual query type [19].

To demonstrate the resilience of our method to changes in the workload, we now change the distribution of the mix of query types and compare the performance of the best approaches in each category. Figure 6.7 shows the result of these experiments.

Note that for this mix, we now use MPL = 30. In experiments similar to those conducted in Section 6.1.1, we found that MPL = 30 now slightly outperforms MPL = 40, likely due to the higher proportion of Best Sellers requests. We also have lowered the request rates under investigation, as the number of requests that the server can handle on average is lower with this distribution. Additionally, we have generated model analagous to the TYPE model with estimation parameters specific to the new distribution. This new model is referred to as TYPE-NEW. TYPE-NEW differs only slightly from TYPE both

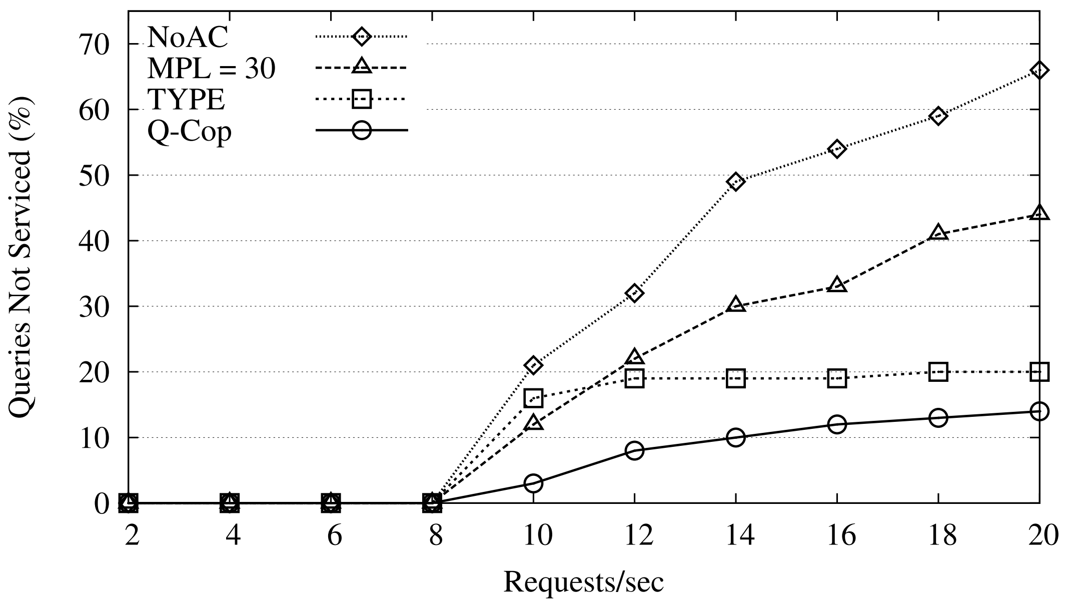


Figure 6.7: Performance using a different distribution of query types.

in model parameters and performance, but we present the results here for completeness. Despite not generating a new model for the new workload mix, Q-Cop still performs better than the new model, TYPE-NEW.

Table 6.3 shows the total number of requests not serviced by each method across the full range of loads examined (i.e., 2 – 20 requests per second). The second row of this table shows the percentage by which Q-Cop has reduced the total number of requests not serviced for each of the methods explored. This table shows that Q-Cop significantly outperforms the MPL-based approach in this comparison and shows a convincing advantage over TYPE at all request rates. Q-Cop yields an average overall advantage of 67.4% fewer unserviced requests when compared to MPL = 30 and 47.4% when compared to TYPE and TYPE-NEW.

Approach	NoAC	MPL = 30	TYPE	TYPE-NEW	Q-Cop
Avg % Not Serviced	46.7	22.7	18.8	18.8	9.9
Q-Cop Reduction (%)	79.0	67.4	47.4	47.4	–

Table 6.3: Average percentage of queries not processed across all loads for the new workload.

Figure 6.7 shows that Q-Cop has consistently fewer unsuccessful requests either through timeouts or rejections. MPL-based approaches suffer from a large number of rejections, across a variety of level of which only the best performing (MPL = 30) is shown here.

Figure 6.8 shows the total unsuccessful requests and the requests rejected for the TYPE, TYPE-NEW, and Q-Cop methods. The difference between the total number of

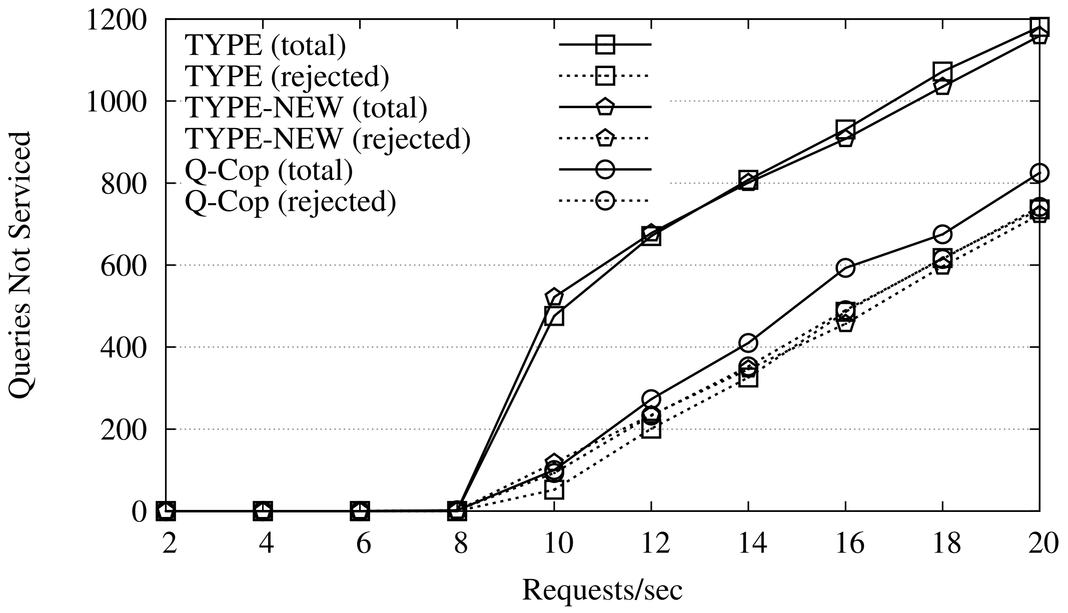


Figure 6.8: Total unsuccessful and rejected queries for the new mix.

unsuccessful requests and the number of rejected requests is the number of requests that time out. Once again, note that TYPE rejects roughly the same number of requests as Q-Cop, but suffers from significantly more total unsuccessful requests.

Not only is our query mix model-based approach to admission control robust across different distributions of query types but we also expect it to be robust in the face of dynamically changing workloads. For example, if the workload were to change to use significantly fewer or no Best Sellers requests, no changes are required to our model. In future work we hope to conduct experiments that include a workload that changes over time (discussed in more detail in Section 8.2.3).

Chapter 7

Strengths and Weaknesses

7.1 Reflecting on Q-Cop

Having presented experimental results, we now evaluate the approach taken in this thesis.We highlight some strengths and weaknesses and discuss some practical sources of error.

7.2 The Power of Query Mix Modelling

Important aspects of the way in which the QMM model is constructed are that it does not require:

- a priori information about the expected system load,
- a priori information about the expected mix of queries,
- 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 used to construct the model. Since the model is generated from the underlying queries but not from any specific usage log, it should perform well across a variety of distributions of requests. This is in contrast to models which require the query distribution to be known in advance to calculate average response time curves, models which must be tuned, and models constructed using the actual workload being served.

7.3 Drawbacks of Experiment-Driven Approach

Using an experiment-driven methodology allows us to account for a number of potential issues, such as specific hardware concerns or peculiarities of queueing specific to the software being used. Compared to other experiment-driven techniques [13, 40], our methodology requires less prior knowledge of the system being controlled. For instance, we do not need any information about the distribution of requests. Our experiment-driven model building approach does, however, require a priori knowledge of the different types of expected queries. This requirement 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 provided by a developer using Q-Cop.

The developer 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 place very little load on the system. For example, these could be query types that always finish in under 1 ms. Allowing additional queries in the set under consideration serves to increase the amount of time required for experimentation without increasing the accuracy of the model. Setting up Q-Cop also requires some a priori knowledge of what request rates are reasonable for the system in question in order to choose a sampling space, also not usually a difficult question for a system administrator.

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. Running the sampling protocol and building the model can take a significant amount of time depending on the accuracy desired and the number of query types in the system. For an application with a large number of query types, the exponential growth of the search space might lead to impractical time requirements.

7.4 Changing Workloads

Public web applications face changing workloads on a number of dimensions [6]. Sudden increases in the incoming request rate, of an order of magnitude or more, can very quickly put a server into severe overload conditions [10]. A change in the distribution of requests, perhaps a transient skew in favour of more-demanding requests, can put a server into overload without changing the total number of incoming requests at all. By explicitly capturing the effect that requests have on other requests, Q-Cop is able to adapt to changing workloads without the use of any kind of moving window average, such as that used by Gatekeeper [19], attempting to implicitly capture this effect. However, Q-Cop is not able to handle the addition of a new query type without re-running the experiment-driven phase. Note that, although additional experiments would need to be run to build the model as it relates to the new query type, the experimental data from the original system would still be valid for a subset of the new sampling space.

7.5 Estimate Accuracy

In our experiments in Section 6.1, with the TPC-W-like distribution of queries as described in Section 3.2, the Best Sellers queries execute for significantly longer than the other query types in the TPC-W workload. Therefore, they are the only queries that are rejected or timeout 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 requests would have completed had they not been rejected (false negatives), we can examine those admitted queries that were accepted but did timeout (false positives).

We therefore examine the accuracy with which admission control decisions are made within Q-Cop for the Best Sellers queries. We start by studying the accuracy of the execution time estimates for those queries. Figure 7.1 shows a scatter plot of the estimated versus actual execution time of all Best Sellers queries. This data was obtained while using Q-Cop. As a result, there are no data points for which estimates were above 29 (since they were rejected, we do not know the actual execution time).[1]

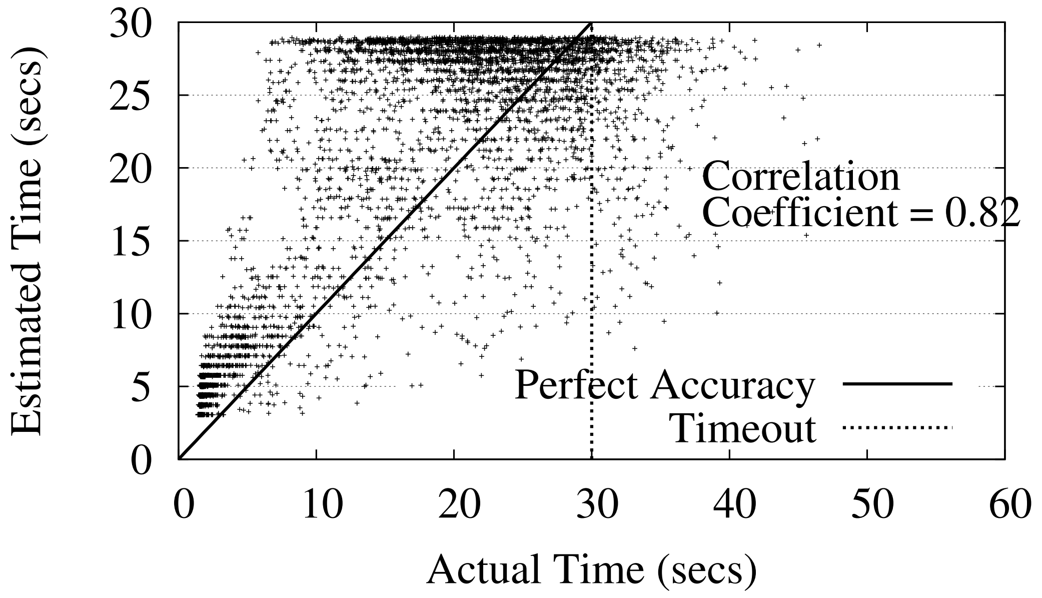


Figure 7.1: Actual versus estimated query execution times.

The 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

[1] We could have conducted an experiment without admission control for which we examined the accuracy of the 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 quickly accumulate without admission control).

there is a fairly good correlation between the estimated and actual query execution times. The correlation coefficient is 0.82 (strongly correlated).

However, there are two issues with respect to the accuracy of these estimates. The first issue is their degree of accuracy, which Figure 7.1 shows is quite good. The second issue is having sufficient accuracy to avoid accepting requests whose clients timeout before the request is completed. We can see from the number of points to the right of the timeout line that there may be room for improvement. 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 interested in receiving the result.

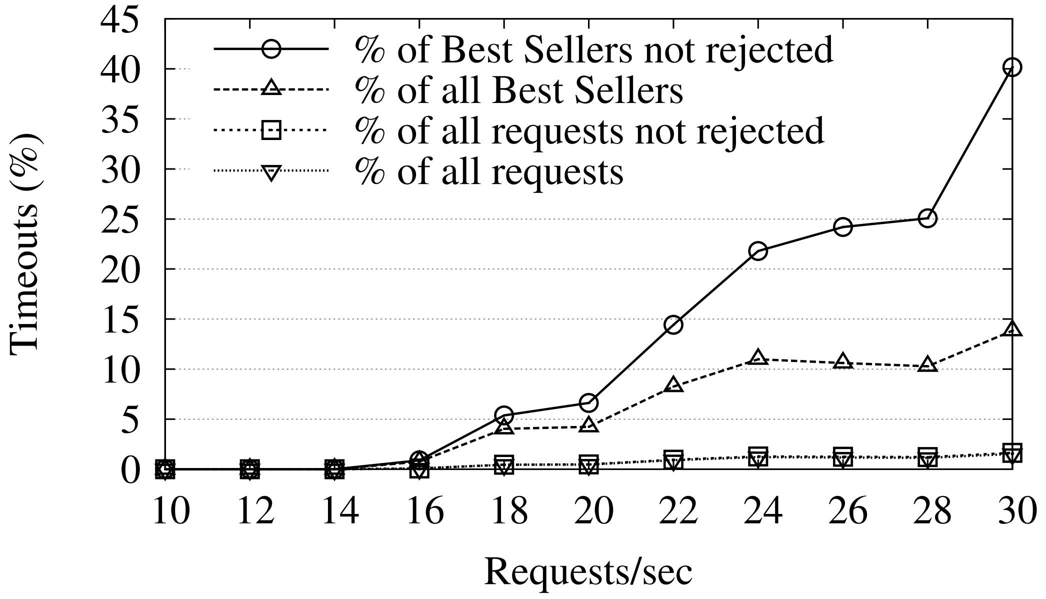


Figure 7.2: Percentage of requests that timeout using Q-Cop.

Next, we examine model accuracy more closely for these points. Figure 7.2 shows several views of these false positive decisions. The bottom line in the graph (% of all requests) shows the percentage of all requests that did timeout. The second line from the bottom (% of all requests not rejected) shows the percentage of all admitted requests that did time 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 middle line (% 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).

Note that 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 we have examined that do not consider the mix of executing queries.

The ability for Q-Cop to function under inaccuracy is in some ways a strength and in others a weakness. In beating other approaches even with a model that we have demonstrated to be of somewhat limited accuracy, we show that our approach is not overly sensitive to building a perfect model. Indeed, since the underlying execution data is highly variable in nature, it is necessary for the controller to be insensitive to a potentially large amount of error in modelling. In some ways, the model simply has to provide data that is "good enough" to perform admission control successfully, not necessarily predict the exact execution time for each incoming query. That is, it is sufficient in this case to predict with a large degree of accuracy which queries would timeout and which queries would finish successfully if admitted. QMM does provide that quality, as can be seen by the reduced number of timeouts.

However, the degree of inaccuracy does raise some issues with respect to the Query Mix Model itself. While the inflation of execution time experienced by requests seems to be much more correlated to the query mix than to CPU utilization, it is possible that there are metrics which provide even better estimates of execution time. Although there is not a large amount of room for improvement in the number of total requests not serviced, our approach is not perfect. Each request not serviced is potentially lost income, and increasing the accuracy of the model would further decrease potentially both the number of timeouts and the number of rejected requests. Some methods for improving model accuracy and compensating for incorrect choices are discussed in Sections 7.6 and 8.2.

7.6 Sources of Inaccuracy

There are several potential sources for the inaccurate decisions made in our prototype implementation of Q-Cop. Investigating these issues is an interesting direction for future work.

7.6.1 Why Does Q-Cop Have Any Timeouts?

There are a number of timeouts from Best Sellers queries even under Q-Cop admission control as seen in Figure 7.2, 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.

7.6.2 Query Type Assumption

We assume, as previous work has shown [19], 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 a significant impact on the accuracy of our decisions. The set of experiments used to construct our model (chosen using the LHS method) use 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. Real-world workloads, conversely, are not necessarily uniform, and could potentially exhibit significantly different behaviour which would need to be accounted for in the estimate model.

7.6.3 Model Inaccuracy

We chose to use a simple linear regression model over other significantly more complex but slightly more accurate models available in WEKA such as Gaussian Processes. It would be interesting to see if using these more complex models would improve the accuracy of the model sufficiently to further reduce the number of unsuccessful requests.

Currently, our model does not account for the variation that is seen during the experimental data gathering phase of the model building procedure. 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.

7.6.4 Estimation Inaccuracy

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 the admitted query may complete before the timeout, its execution may cause one or more existing queries to time out. Although the current simple approach provides significant improvements over existing approaches, it would be interesting to see if further improvements to this decision would yield further benefits.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Three-tiered web applications need some form of admission control in order to ensure that performance does not dramatically suffer under high loads. Without admission control, high load would result in reduced throughput and greatly increased response times. Additionally, real-world clients do not wait indefinitely for a response, so uncontrolled web applications see an increase in the number of client timeouts. We demonstrate the need for overload control when clients time out by experimentally showing a 28-fold increase in average response time, an 18% drop from peak reply rate, and a 55% client timeout rate when the system encounters overload conditions. One previous approach to handling overload is over-provisioning, but this can be cost prohibitive. Other approaches use various admission control algorithms, which we compare against experimentally. We show that query mix is important by running experiments with constant MPL but varying query mix. The 50% spread in average execution time implies that MPL alone is an insufficient predictor of execution time.

We propose Q-Cop, a system for performing admission control with the goal of minimizing the total number of client requests that are unsuccessful. 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 execution time of different query mixes, Q-Cop uses an off-line, experiment-driven approach to model-building that samples the space of possible query mixes and learns statistical models that can be used to predict the execution time of any query mix observed in any live workload comprised of the query types used to create the model. It does not require prior knowledge of the actual workload distribution. Q-Cop then uses these models in its on-line operation to decide which queries to reject, using the current number of running queries of each type. Any incoming query which Q-Cop estimates will take longer than the specified timeout value is rejected.

Using queries from the TPC-W benchmark, we experimentally demonstrate that Q-Cop outperforms mix-oblivious techniques for admission control, reducing the number of queries not serviced by up to 47% as compared to the next best admission control scheme (TYPE), up to 77% as compared to MPL, and up to 86% as compared to an uncontrolled system. Q-Cop also has the advantage that it can deal with changing workload distributions using the same performance models, whereas other techniques may require relearning the models. We demonstrate this experimentally, showing a reduction in the number of queries not serviced by up to 46% over TYPE in a workload using TPC-W queries in a different distribution. Although our study of the accuracy of Q-Cop shows that there it is room for improvement, we believe that the performance benefits combined with the benefits explained above provide strong incentive to conduct future work into techniques for improving the accuracy of the performance model and of Q-Cop's decision process.

8.2 Future Work

8.2.1 Modelling Accuracy and Complexity

All experimental results presented in this thesis used a straight-forward linear regression model to produce estimates of the execution time for queries. The linear model provides, in our model-building experiments, a roughly 56% relative mean-squared error. Another model, using Gaussian Processes, showed approximately a 42% relative mean-squared error. However, implementation of that model requires more infrastructure work for the integration of the WEKA module into the admission control system, since WEKA does not return the function parameters, but rather a Java object which performs the estimation. By contrast, since a linear model is quite simple in its definition, it is relatively easy to produce code that computes the linear model directly within the database system.

However, it is clear that the effect of queries on each other is not linear. It is possible that using a more-accurate model could yield significantly more accurate results for query execution estimation, which would then lead to better performance of the server. An investigation into the nature of the interaction between query types, specifically what that relationship looks like, might yield insight into what type of model would be most accurate.

8.2.2 Multi-Tiered Admission Control

All admission control decisions in this thesis are made in the database system. In the TPC-W benchmark, the vast majority of processing time is spent in the database. In alternate applications, however, it is possible for the database to show lower utilization [5], with either the application server or web server being the bottleneck. To add flexibility to the Q-Cop prototype, the admission control algorithm could be implemented at each level of the three-tiered stack. As the request progresses through the system, the algorithm is exposed to different levels of information about the environment as well as the request itself. To make accurate admission control decisions, control should be implemented at each level, especially when considering the diverse nature of systems that the scheme might be applied to. By making admission control decisions at each tier, the system as a whole also avoids a greater amount of work on requests that will timeout. A request that has reached the database has already had some processing done and has already used some resources. Since one of the benefits of this system is reducing wasted work, doing multi-tiered admission control could be a beneficial avenue for future work.

8.2.3 Load Complexity and Changes

The experimentation in this thesis has been carried out using largely similar workloads based on the TPC-W benchmark. In order to more thoroughly investigate the capabilities of Q-Cop, especially with respect to flexibility, the use of different experimental environments, especially different workloads, would be beneficial. Moving to a different workload entirely, replacing TPC-W, could also provide some insight. The TPC-W benchmark tends to be bottle-necked primarily in the database tier, but that is not necessarily true of other benchmarks or loads.

While we have performed some initial experimentation on the sensitivity to timeout values, there is certainly more room for investigation in this area. One specific area of interest lies in assigning different timeouts to different request types, or even client types. Q-Cop should be able to deal with changes in timeout values easily, while other approaches would need to be re-calibrated or trained.

Some previous work in the area [10] has looked specifically at the issue of extremely bursty loads, or flash crowds. Transient periods of heightened overload should be handled by Q-Cop with no changes to the mechanism or control apparatus, but we have not shown that experimentally.

8.2.4 Load Estimation

One of the key issues to moving forward with a more sophisticated approach is updating the estimated remaining execution time of each query. One approach would be to, for each query, update the time spent executing that query and the expected remaining time. This would be required on the arrival and completion of every query. While it might be tempting to include some notion of expected future arrivals, this would require information about the expected workload, which we have been trying to avoid. The benefit to gathering this information could be a more accurate estimate of the execution time for a newly arriving query.

One issue with the current admission control algorithm is that it makes decisions based only on the instantaneous information available when it encounters the request. Q-Cop is aware of the current query mix, but that is valid only at the moment the query is admitted and will quickly be inaccurate as more queries arrive. This inaccuracy of query mix data yields a potentially inaccurate execution estimate, which can lead to incorrect admission decisions. While some inaccuracy is self-corrected by later decisions when load gets too high, it can still lead to both false positives and negatives.

8.2.5 Other Value Metrics

One potentially interesting line of research is the combination of admission control algorithms with business level objectives. While it seems natural to attempt to maximize total throughput or minimize requests not served, in some cases the most important factor to the business using the server might be something less directly related to performance. In the end, the purpose of the server, in most commercial situations, is to maximize the *monetary* throughput. Businesses may have specific request types that are much more important to their business than others, out of proportion to the load placed on the server.

Q-Cop could be adapted to use the user-specified importance values rejecting requests that represent the lowest value-to-load ratios whenever it finds that some requests are going to timeout. It could do this by rejecting proportional to the importance value whenever it detects overload conditions, or rejecting requests that have a lower value than the request it estimates will timeout, or dividing requests into categories and rejecting all requests from lower categories until the system is no longer overloaded.

References

[1] S. Adler. The Slashdot effect: an analysis of three Internet publications. *Linux Gazette*, 38, 1999. 2

[2] Mumtaz Ahmad, Ashraf Aboulnaga, and Shivnath Babu. Query Interactions in Database Workloads. In *Proceedings of the International Workshop on Testing Database Systems (DBTest)*, 2009. 2, 9, 24

[3] Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. Modeling and Exploiting Query Interactions in Database Systems. In *Proceedings of the ACM Confefernce on Information and Knowledge Management (CIKM)*, 2008. 2, 9, 10

[4] Mumtaz Ahmad, Ashraf Aboulnaga, Shivnath Babu, and Kamesh Munagala. QShuffler: Getting the Query Mix Right. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2008. 2, 9, 10

[5] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck Characterization of Dynamic Web Site Benchmarks. In *Third IBM CAS Conference*, 2002. 1, 45

[6] M.F. Arlitt and C.L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM New York, NY, USA, 1996. 39

[7] Shivnath Babu, Nedyalko Borisov, Songyun Duan, Herodotos Herodotou, and Vamsidhar Thummala. Automated Experiment-Driven Management of (Database) Systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, 2009. 10

[8] Gaurav Banga and Peter Druschel. Measuring the Capacity of a Web Server. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey CA, December 1997. 9, 12

[9] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 1998.

[10] Novella Bartolini, Gian Carlo Bongiovanni, and Simone Silvestri. Self-* Through Self-Learning: Overload Control for Distributed Web Systems. *Computer Networks*, 53(5), 2009. 7, 34, 39, 46

[11] Peter Belknap, Benoît Dageville, Karl Dias, and Khaled Yagoub. Self-Tuning for SQL Performance in Oracle Database 11g. In *Proceedings of the International Workshop on Self Managing Database Systems (SMDB)*, 2009. 10

[12] 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 *Third Workshop on Computer Architecture Evaluation Using Commercial Workloads*, 2000. 12

[13] Josep M. Blanquer, Antoni Batchelli, Klaus E. Schauser, and Richard Wolski. Quorum: Flexible Quality of Service for Internet Services. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, 2005. 8, 39

[14] D. Breitgand, E. Henis, and O. Shehory. Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management. In *Proceedings of the Second International Conference on Autonomic Computing*, 2005.

[15] J. Carlstrom and R. Rom. Application-Aware Admission Control and Scheduling in Web Servers. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002.

[16] R. Carter and L. Cherkasova. Detecting Timed-Out Client Requests for Avoiding Livelock and Improving Web Server Performance. In *Proceedings of the Fifth IEEE Symposium on Computers and Communications*, 2000. 9

[17] CNBC. The Oprah Effect: The Million-Dollar Touch, 2009. Available at http://www.cnbc.com/id/29961298. 3

[18] Songyun Duan, Vamsidhar Thummala, and Shivnath Babu. Tuning Database Configuration Parameters with iTuned. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2009. 10, 25

[19] Sameh Elnikety, Erich M. Nahum, John M. Tracey, and Willy Zwaenepoel. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. In *Proceedings of the International Conference on World Wide Web (WWW)*, 2004. 8, 15, 22, 23, 35, 39, 43

[20] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet Wiener, Armando Fox, Michael Jordan, and David Patterson. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2009. 10

[21] J.L. Hellerstein. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[22] Charles R. Hicks and Kenneth V. Turner. *Fundamental Concepts in the Design of Experiments*. Oxford University Press, 1999. 10, 25

[23] H. Jamjoom and K.G. Shin. Persistent Dropping: An Efficient Control of Traffic Aggregates. *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003.

[24] Abhinav Kamra, Vishal Misra, and Erich M. Nahum. Yaksha: a Self-Tuning Controller for Managing the Performance of 3-tiered Web Sites. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, 2004. 6, 15, 34

[25] Xue Liu, Lui Sha, Yixin Diao, Steve Froehlich, Joseph L. Hellerstein, and Sujay S. Parekh. Online Response Time Optimization of Apache Web Server. In *Proceedings of the IEEE International Workshop on Quality of Service (IWQoS)*, 2003. 6, 23

[26] Market Share. Browser Version Market Share, 2009. Available at http://marketshare.hitslink.com/browser-market-share.aspx?qprid=2. 12

[27] Microsoft. Microsoft Support Web Page, 2009. Available at http://support-.microsoft.com/kb/181050i. 12

[28] A. Mönkeberg and G. Weikum. Performance Evaluation of an Adaptive and Robust Load Control Method for the Avoidance of Data Contention Thrashing. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1992. 5, 23

[29] D. Mosberger and T. Jin. httperf: A Tool for Measuring Web Server Performance. In *The First Workshop on Internet Server Performance*, Madison, WI, June 1998. 9, 11

[30] D.P. Olshefski, J. Nieh, and D. Agrawal. Inferring Client Response Time at the Web Server. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2002. 9

[31] Jia Rao and Cheng-Zhong Xu. CoSL: A Coordinated Statistical Learning Approach to Measuring the Capacity of Multi-Tier Websites. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008.

[32] B. Schroeder, A. Wierman, and M. Harchol-Balter. Closed Versus Open System Models: a Cautionary Tale. In *Symposium on Networked Systems Design and Implementation*, 2006. 11

[33] Bianca Schroeder, Mor Harchol-Balter, Arun Iyengar, Erich M. Nahum, and Adam Wierman. How to Determine a Good Multi-Programming Level for External Scheduling. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2006. 6, 23

[34] Transaction Processing Performance Council. TPC-W Benchmark, 2009. Available at http://www.tpc.org/tpcw/default.asp. 12

[35] T. Voigt and P. Gunningberg. Handling Multiple Bottlenecks in Web Servers Using Adaptive Inbound Controls. *Proceedings of the International Workshop on Protocols for High Speed Networks*, 2002.

[36] M. Welsh and D. Culler. Adaptive Overload Control for Busy Internet Servers. *Proceedings of the 4th USENIX Conference on Internet Technologies and Systems*, 2003. 2

[37] A. Williams, M. Arlitt, C. Williamson, and K. Barker. Web Workload Characterization: Ten Years Later. *Web Content Delivery. Springer*, 2005.

[38] Ian H. Witten and Eibe Frank. *Data Mining Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005. 26

[39] Wei Zheng, Ricardo Bianchini, G. John Janakiraman, Jose Renato Santos, and Yoshio Turner. JustRunIt: Experiment-Based Management of Virtualized Data Centers. In *Proceedings of the USENIX Annual Technical Conference*, 2009. 10

[40] Jingyu Zhou and Tao Yang. Selective Early Request Termination for Busy Internet Services. In *Proceedings of the International Conference on World Wide Web (WWW)*, 2006. 7, 39