SCISPACE
formerly Typeset

Open access • Proceedings Article • DOI:10.1109/IWQOS.2000.847959

Multi-class latency-bounded Web services — Source link

V. Kanodia, Edward W. Knightly

Institutions: Rice University

Published on: 05 Jun 2000 - International Workshop on Quality of Service

Topics: Server, Web service, Client–server model, Web server and Service abstraction

Related papers:

- A measurement-based admission-controlled Web server
- Web server support for tiered services
- Cluster reserves: a mechanism for resource management in cluster-based network servers
- An admission control scheme for predictable server response time for web accesses
- Kernel Mechanisms for Service Differentiation in Overloaded Web Servers

Share this paper:

View more about this paper here: https://typeset.io/papers/multi-class-latency-bounded-web-services-1u6g9ree3d

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

RICE UNIVERSITY

Multi - Class Latency Bounded Web Services

by

Vikram Kanodia

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

[illegible]

Dr. Edward Knightly, Director
Assistant Professor
Electrical and Computer Engineering

Willy Zwaenepoel

Dr. Willy Zwaenepoel
Professor
Computer Science

[illegible]

Dr. Richard Baraniuk
Associate Professor
Electrical and Computer Engineering

Houston, Texas

April, 2000

UMI Number: 1399277

UMI®

UMI Microform 1399277

Copyright 2000 by Bell & Howell Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Multi - Class Latency Bounded Web Services

Vikram Kanodia

Abstract

Two recent advances have resulted in significant improvements in web server quality-of-service. First, both centralized and distributed web servers can provide isolation among service classes by fairly distributing system resources. Second, session admission control can protect classes from performance degradation due to overload. The goal of this thesis is to design a general "front-end" algorithm that uses these two building blocks to support a new web service model, namely, multi-class services which control response latencies to within pre-specified targets. Our key technique is to devise a general service abstraction to adaptively control not only the latency of a particular class, but also to assess the inter-class relationships. In this way, we capture the extent to which classes are isolated or share system resources (as determined by the server architecture and system internals) and hence their effects on each other's QoS. We validate the scheme with trace driven simulations.

Acknowledgments

This Thesis emerged from discussions and collaboration with a wide range of people.

First and foremost i would like to thank my advisor Dr Edward Knightly for encouraging me to push the bounds (without breaking them), and for pushing *me* when I needed it. Your guidance and more importantly patient help played a major role in the actual shaping of this thesis.

I would also like to thank Willy Zwaenepoel, Peter Druschel, Mohit Aron and Vivek Pai from the CS department at Rice University for contributing their precious time in providing suggestions, comments, discussions and also for helping with important infrastructural support.

I thank my office mate Coskun Cetinkaya for several enlightening discussions which have made this past year so memorable.

Last but not the least i thank my parents and my brother for their support and for just being there. This thesis is a result of your continuous support and love.

Contents

Tables

Illustrations

Chapter 1

Introduction

In this thesis we introduce a new model for web services, namely multi-class web services with the request latency controlled to within pre specified targets.

1.1 Motivation

The World-Wide Web (WWW) is a client-server system that integrates various types of information on the global Internet and on enterprise Internet Protocol (IP) networks. WWW technology involves a combination of Web browsers and Web servers. The former provides an easy-to-use graphical interface for browsing information resources on the Internet. A Web server is a system on a network that can process incoming HTTP requests.

The perceived web latency i.e., the request response time, is affected by the performance of three components: the client, the server and the network that connects clients and servers. The client delay is the time required by the browser to show documents, considering the different types of media, such as, audio, video, text, and graphics. The network latency represents the time required to provide the remote access plus the data transmission time. Server delay is the time required by the server to service the request.

An increasingly dominant factor in poor end-to-end performance of web traffic is excessive latencies due to overloaded web servers. Recent studies have shown that the processing delay at the server is a significant part of end to end delay experienced by an HTTP request. Consequently, reducing and controlling server latencies is a key challenge for delivering end-to-end quality-of-service.

Towards this end, two key mechanisms have been introduced to improve web QoS. First, *admission control* has been proposed as a mechanism to prevent web servers from entering overload situations [1, 2, 3]. Specifically, by admitting new sessions only if the measured load is below a pre-specified threshold, admission control can prevent the server from entering a regime in which latencies are excessive, or session throughput collapses due to dropped requests and aborted sessions.

Second, web servers can now provide *performance isolation and differentiation* among the different service classes hosted by the site. In particular, a server may support a number of service classes which may represent different classes of users or different applications (news, email, static documents, dynamic content, etc.). Whether such classes are supported in a single-node server or a distributed cluster, mechanisms devised in [4] and [5] respectively can ensure that each service class receives a certain share of system resources (disk, CPU, memory, etc.). Moreover, by appropriately weighting the share of system resources, *differentiation* among service classes is achieved. The Eclipse/BSD operating system [6] provides mechanisms to ensure that each class receives its share of system resources. Similarly, as delays are also incurred in the system's request queues, prioritization of incoming requests can further differentiate the performance among classes [1, 7].

Thus, differentiation and isolation can be achieved by prioritized scheduling of system resources, and protection from overload can be achieved by admission control. However, even if taken together, these two mechanisms cannot ensure that a request's targeted delay will be satisfied. Consequently, because end-to-end latency is a key component of user-perceived quality-of-service, new mechanisms are needed to ensure that the service class' request delays are limited to within the targeted value.

1.2 Contributions

In this thesis, we introduce a new framework for multi-class web server control which can satisfy per-class latency constraints, and devise an algorithm termed Latency-targeted Multiclass Admission Control (LMAC). Our key technique is to design a scheme within a general framework of request and service envelopes. Such envelopes statistically describe the server's request load and service capacity as a function of interval length, resulting in a high-level service abstraction which circumvents the need to model or measure the components of a request's delay. For example, a request incurs delays in the request queue, CPU processing, memory, disk in the case of cache misses, and so on: individually controlling the latency in each subsystem is simply an intractable and impractical task in a modern server. Instead, we utilize the envelopes as a simple tool for controlling class quality-of-service while maximizing utilization of system resources.

Our approach has three key distinctions. First, it enables web servers to support a strong service model with class latencies bounded to a pre-specified target, i.e., a minimum fraction of accepted requests will be serviced within the class delay target.

Second, it provides a mechanism to characterize and control the inter-class relationships. For example, suppose server resources are allocated to classes in a weighted-fair manner so that classes have performance isolation, yet a class is able to utilize unused resources of other classes. In general, the extent to which an increased load in one class affects the performance of another class is a complex function of the total system load, the particular resource scheduling algorithm, and the low-level interactions among the server's resources. Building on the results of [8], we use the envelopes as a way to characterize the high-level isolation/sharing relationships among classes and design a general multi-class algorithm to exploit these effects.

Finally, by decoupling access control and resource allocation from the internals of the server, we obtain a general solution that applies to a broad class of servers, including single-node and distributed servers, and servers with varying levels of quality-of-service support. Consequently, as the server is enhanced with functionalities such as weighted-fair resource allocation, the algorithm naturally exploits these features to more highly utilize the available resources and support an increased number of sessions per service class.

To evaluate our scheme, we perform a broad set of trace-driven-simulation experiments. We first compare our scheme with an uncontrolled system and illustrate that the algorithm is able to prevent performance degradation due to overload. Next, comparing the delays obtained in simulations with the class QoS objectives, we find that in many cases, latencies can be controlled to within several percent of the targeted value. Moreover, in the single-class case, we compare with a simple queuing theoretic approach, and find that envelopes control the system to a significantly higher degree of accuracy. Finally, in the multi-class case, simulations indicate that substantial inter-class resource sharing gains are available. Here, we find that the approach is able to extract these gains, and efficiently utilize system resources while satisfying each class' delay targets.

1.3 Organization

The remainder of this thesis is organized as follows.In chapter 2, we describe the server architecture and the system abstraction used for QoS management. Next, in chapter 3, we describe a simple single-class queuing theoretic approach to serve as a benchmark for performance analysis, and illustration of the key problems in meeting delay targets. In chapter 4, we introduce the request and service envelopes and develop an access control algorithm based on the properties of these envelopes. We next describe the simulation scenario and present experimental results in chapter 5. Finally, in chapter 6, we conclude and also discuss some possible avenues for future work.

Chapter 2

System Architecture

In this chapter we describe the basic system architecture of a scalable QoS web server. We are not proposing a new architecture as all of the mechanisms described below have been introduced previously. Rather, our goal is to consider a general system model for admission control which can exploit various QoS server mechanisms to efficiently satisfy targeted class latency objectives.

2.1 System Model

Figure 2.1 depicts the general system model that we consider. The system consists of a state-of-the-art web server augmented with admission control capabilities as in [1, 2, 3]. All incoming requests, which can be sessions (as in [2]) or individual "page" requests, are classified into different quality-of-service classes. There are a number of possible classification criteria including the address of the server (in case of web hosting applications), the identity of the user issuing the request, or the particular application or data type. The goal of our admission control algorithm is to determine whether admission of a new request in a particular service class can be supported while meeting the latency targets of *all* classes. If it is not possible, the request should be rejected outright, or redirected to a lower priority class or a different server.

As shown in the figure, incoming requests are first queued onto the listen queue or dropped if the listen queue is full. The admission controller dequeues requests from the listen queue and determines if they will admitted or rejected. Notice that

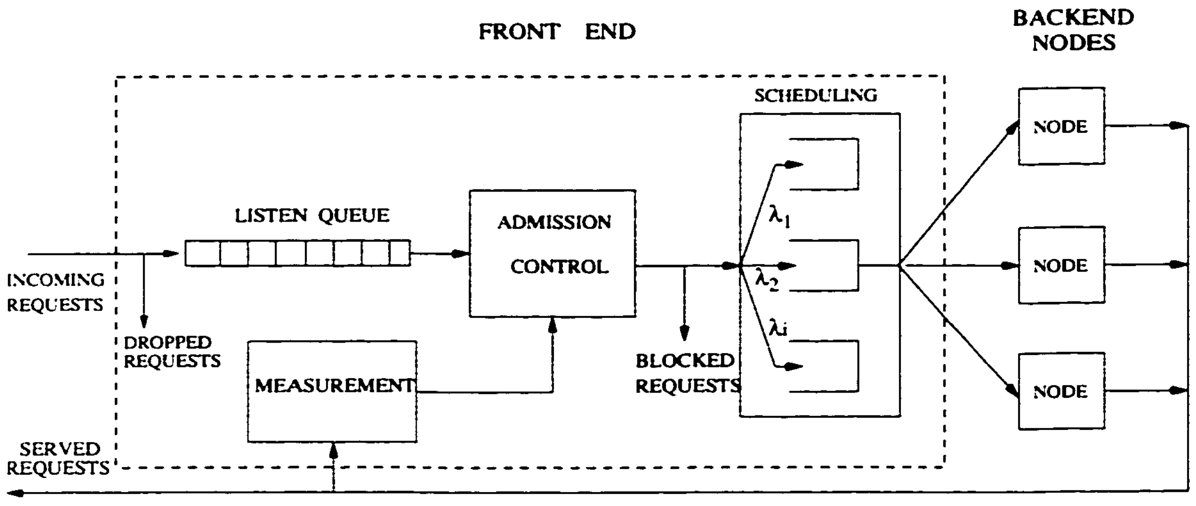
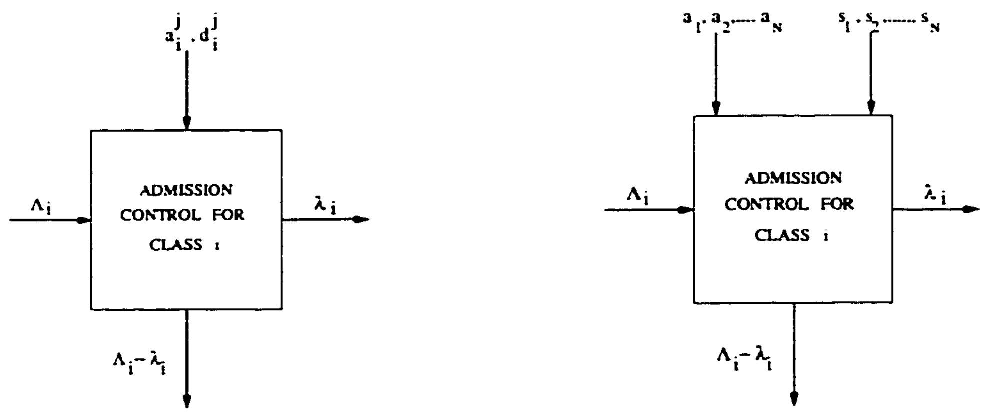


Figure 2.1 System Model

the admission control unit is part of the front-end and monitors all of the server's arrivals and departures. As depicted in Figure 2.2, the admission control unit performs observation-based control of the server using measured request and service rates of each class. Further, notice from Figure 2.2(b) that our class-based admission control will incorporate effects of inter-class resource sharing by also considering the effects of a new admission on *other* service classes.

If admitted, requests are then scheduled according to the server's request scheduling algorithm which can be first-come-first-serve or class based [1, 7]. Finally, requests are submitted to the back-end nodes in a distributed server [5], or in the case of a single node server are simply processed by the node itself.

A key point is that the admission controller applies to a general system model including single-node and distributed servers, FCFS and class-based scheduling, and standard as well as QoS-enhanced operating systems. When QoS mechanisms are present in the server (such as class-based rather than FCFS request scheduling), the admission controller will measure the corresponding performance improvements and



(a) Baseline Algorithm (b) LMAC

Figure 2.2 Admission Control Model

exploit the QoS functionality by admitting more requests per class, thereby increasing the overall system efficiency. For example, consider a server farm where the front-end does sophisticated load balancing to achieve better overall throughput by exploiting locality information at the back-end [9]. In this case, the admission controller will measure the decreased serviced latencies and be able to admit an increased number of sessions into various classes, thereby exploiting the efficiency gain of load balancing.

Finally, notice from Figure 2.2 that the admission controller does not measure or model resources at the operating system level, such as disk, memory, or CPU. Instead, we abstract all low-level resources into a virtual server which allows us to design an admission controller that is applicable to a broad class of web server architectures and applications.

a_i^j	arrival time of request j in class i
s_i^j	service time of request j in class i
$\bar{d}_i$	mean delay of class i requests
λ_i	mean arrival rate of admitted class i requests
μ_i	mean service rate of class i requests
d_i^*	class i target delay
ϵ_i^*	class i delay-violation probability
$\sigma_i^2(\tau)$	variance of class i requests over durations τ
$\bar{d}_i(k)$	mean class i latency for k concurrent requests
$\gamma_i^2(k)$	variance of class i latency for k concurrent requests
T	measurement window

Table 2.1 Notation

Chapter 3

Baseline Scheme

In this chapter, we sketch a simple queuing theoretic algorithm devised to satisfy a delay target. The goal here is threefold. First, we illustrate an abstraction of the server resources into a simple queuing model. We present a very simple and high level view of the web server modeled as an open single queue server. Second, we highlight key issues for managing multi-class web services. Finally, we use the approach as a baseline for experimental comparisons and, by highlighting its limitations, we further motivate the LMAC scheme.

3.1 Queuing Theory

As is often the case in computer systems, Web servers typically process many simultaneous jobs (i.e., file requests), each of which contends for various shared resources: processor time, file access, and network bandwidth. Since only one job may use a resource at any time, all other jobs must wait in a queue for their turn at the resource. As jobs receive service at the resource, they are removed from the queue; all the while new jobs arrive and join the queue. Queuing theory is a tool that helps to compute the size of the queues and also the time that a job (request) spends in the queues. In this thesis we are mainly concerned with the *simultaneous* number of HTTP requests handled by a server, and the total time required to service a request.

In the next subsection, we present a very simple review of important concepts from queuing theory. More information can be found in [10].

3.1.1 Queuing Theory Results

Queuing theory views every service or resource as an abstract system consisting of a single queue feeding one or more servers. Associated with every queue is an arrival process, a service process and the size of the queue (if bounded).

Consider a single-channel (server) queue with Poisson arrivals at rate $\lambda$and exponential service rate μ: the M/M/1 [1] queue. From Queuing theory [10], waiting time in the system (service time plus the delay in the queue), W, is an exponentially distributed random variable with parameter $(\mu - \lambda)$. That is :

$$W \sim exp(\mu - \lambda) \tag{3.1}$$

The mean of the distribution is equal to the average waiting time in the system, w, given by :

$$w = \frac{1}{(\mu - \lambda)} \tag{3.2}$$

It is to be noted here that an M/M/1 queue is called a stable queue if the arrival rate λ is less that the service rate μ. In case of a stable queue all jobs will eventually be serviced, and the average queue size is bounded. If λ is greater that or equal to μ, the queue is labeled as unstable and the queue length grows without bound.

In the next section we will actually use the above queuing theory results to model the behavior of a busy web server.

[1]The notation A/B/c is used to classify queuing models, where c is the number of channels, A denotes the inter-arrival time distribution, and B denotes the service time distribution. The symbol M denotes Poisson (exponential interarrival times).

3.2 Problem Formulation

Typically a single queue is insufficient for modeling a complex system such as a web server. In many such cases a system is modeled as a graph or a network of queues in which each queue represents one node. However modeling a web server as a system of queues leads to a complex model since the different queues (representing the different server resources) interact with each other in an unpredictable and complex manner. We wanted to keep the model of a server simple and also independent of the actual resource level details of the server architecture. Jackson's theorem ([10]) states that under specific conditions (that are beyond the scope of this thesis), the complex interactions between nodes in the network may be ignored. Even if the arrival distribution of the queue is no longer exponential - because it is influenced by the rest of the network - it is approximately exponential.

Consider a single class with quality-of-service targets given by a delay bound of 0.5 seconds to be met by 99% of requests. Further consider a stationary and homogeneous arrival of sessions and requests within sessions, so that there exists some maximum number of requests per second which can be serviced so that this QoS requirement is met. If the overall arrival rate of requests to the server is greater than this maximum, the difference should be blocked (or redirected) by the access controller to prevent an overload situation.

The key question is, how to determine which load level is the maximum one that can support the service. Specifically, if the current load is below this maximum, then the current 99 percentile delay will be below the target. However, when a new session requests access to the server, the new 99 percentile delay of this class and others is in general a complex function of the system workload, and the low-level interactions among the many resources consumed such as disk, bandwidth, memory, and CPU.

In the next section we sketch a baseline approach for assessing the impact of new requests and sessions on the delay target via a simple queuing theoretic abstraction.

3.3 Sketch Algorithm

In this section we describe a sketch baseline algorithm which uses results from queuing theory to model the request latency while modeling the server as a simple M/M/1 queue.

We approximate class j's service by an M/M/1 queue with an unknown service rate. In particular, as described earlier, a request's service latency includes delays at multiple system resources like CPU, disk, etc. Each of these system level resources have their own separate queues and the net latency of a request is composed of processing time at each of the system resources coupled with the time spent waiting in the resource queue. The M/M/1 model abstracts these resources and their queues into a single virtual server with independent and exponential requests and services as follows.

Over the last T seconds from the current time t, the mean arrival rate is [2]

$$\lambda_i = \frac{\sum_j 1(t - T \le a_i^j \le t)}{T} \tag{3.3}$$

where $1(\cdot)$ is an indicator function, and the mean delay is

$$\bar{d}_i = \frac{\sum_j (s_i^j - a_i^j) 1(t - T \le s_i^j \le t)}{\sum_j 1(t - T \le s_i^j \le t)}. \tag{3.4}$$

Under the assumptions of the M/M/1 model and using equation 3.2, the unknown mean service rate is simply

[2]The notation used is enumerated in table 2.1.

$$\mu_i = \frac{1}{\bar{d}_i} + \lambda_i \tag{3.5}$$

Using the above result and the fact that the waiting time in the system is an exponentially distributed random variable (equation 3.1), the delay violation probability under an increased load $\lambda_i' > \lambda_i$ will be

$$P(D_i > d_i^*) = \exp\left(-d_i^*(\mu_i - \lambda_i')\right). \tag{3.6}$$

Thus, the increased load due to the new session should be admitted if the estimated $P(D_i > d_i^*)$ is less than the class' target ϵ_i^*. Consequently, under the particular assumptions of the M/M/1 model, the above scheme limits the class' latency to within the target d_i^* for the specified fraction of requests ϵ_i^*.

3.4 Limitations of the Baseline Scheme

While server access control based on Equations (3.3) - (3.6) does provides ability to meet a class' latency objectives with a high level abstraction of system resources, it encounters several key problems which preclude its practicality to realistic web servers.

First, it offers no support for multiple services classes. That is, by treating each class independently, the impact of a new session on *other* classes is ignored. Second, the assumption that inter-request times are independent and exponentially distributed conflicts with measurement studies [11]. Third, the assumption of independent and exponentially distributed *service* times cannot account for the highly variable service times of requests and ignores the strong effects of caching, namely, that consecutive requests for the same document can result in highly correlated, as well as highly variable, service times. The reasons behind the shortcomings of the baseline scheme arise primarily from the fact that we model the server as a single queue when it is in fact a network of queues, with each node in the network (representing a resource queue) interacts with each other in a complex manner.

In Chapter 5, we experimentally quantify the impact of these limitations in a realistic scenario.

Chapter 4

Multi-Class Admission Control

In this chapter, we build on the previous baseline admission control model and introduce the Latency-targeted Multi-class Admission Control (LMAC) algorithm. The goal is to provide a strong service model for web classes that controls statistical latency targets of multiple service classes. The LMAC algorithm has two key distinctions from the baseline scheme. First, we introduce use of envelopes as a general yet accurate way of describing a class' service and demand. As for the baseline scheme, this is a high-level workload and service characterization, yet, unlike the baseline scheme, envelopes capture effects of temporal correlation and high variability in requests and service latencies. Second, exploiting the inter-class theory of [8], we show how the performance effects of one class on another can be incorporated into admission control decisions.

4.1 Envelopes: A General Service and Demand Abstraction

Deterministic [12] and statistical [8, 13] traffic envelopes have been developed to manage network QoS. Moreover, deterministic [12] and statistical [8] *service* envelopes have provided a foundation for multi-class network QoS. Inter class resource sharing has also been incorporated in [8]. Below, we extend these techniques to the scenario of measurement-based web services exploiting two key properties of envelopes: characterization of temporal correlation and variance and simple on-line measurement via jumping windows.

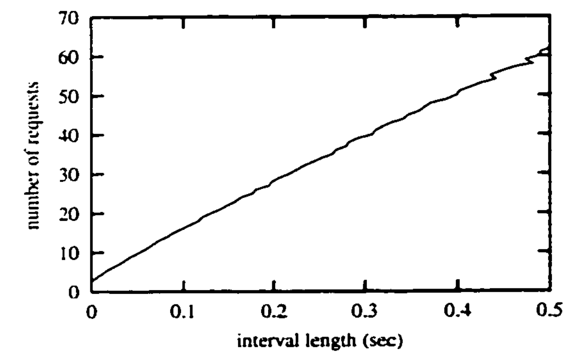
Denoting the number of class i requests in the interval $[t_1, t_2]$ by

$$N_i(t_1, t_2) = \sum_j 1\left(t_1 < a_i^j < t_2\right), \tag{4.1}$$

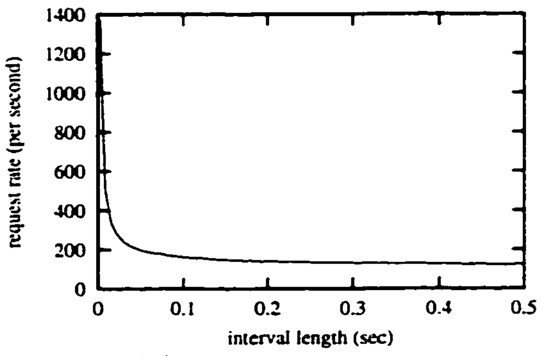
the mean number of requests in an interval of length τ is $\lambda_i\tau$, and the variance is given by

$$\sigma_i^2(\tau) = \frac{1}{\lfloor T/\tau\rfloor - 2}\sum_{m=0}^{\lfloor T/\tau\rfloor-1}\left(N_i^2(t-(m+1)\tau, t-m\tau) - (\tau\lambda_i)^2\right) \tag{4.2}$$

As an example envelope, Figure 4.1 shows the request envelope for the trace described in Chapter 5. Specifically, the figure depicts $\lambda_i + 1.645\sigma_i(\tau)/\tau$ vs. τ, where $1.645\sigma_i(\tau)$ yields the 95% tail of a Gaussian distribution. In other words, under a Gaussian distribution of total requests with empirical mean and variance as above, the figure shows the value of r such that $P(N_i(t-\tau, t)/\tau > r) = 0.95$. Figure 4.1(b) shows the envelope normalized to the interval length so that the y-axis is a rate.



(a) Cumulative Envelope



(b) Rate Envelope

Figure 4.1 95 Percentile Request Envelopes

For example, in Figure 4.1(a) the point (100 msec, 17) on the curve indicates that 17 consecutive requests arrive within 100 msec 95% of the time. This corresponds to a rate of 170 requests per second over the same interval length which is depicted in

Figure 4.1(b). Thus, the figure shows that that over short interval lengths, significantly more requests than the mean 100 per second can arrive. Such characteristics of the request workload are a key input to admission control.

4.1.1 Measurement-Based Service Envelopes

Here, we define and show how to adaptively measure a class' service envelope. Analogous to the above request envelope, it describes the service latencies of consecutive requests which simultaneously compete for system resources, characterizing the variance and temporal correlation of services. In particular, we measure this envelope by monitoring the service latencies of requests as a function of the number of concurrent requests. For example, let k be the number of consecutive requests in consideration. For $k = 1$, the service envelope consists of the mean and variance of the time required to service a single request. For $k = 2$, the envelope characterizes the mean and variance of the time required to service two requests that are concurrently competing for system resources. That is, if the j'th request enters the system before the $(j-1)^{th}$ request is serviced, then $(s_i^j - a_i^{j-1})$ represents the total time required to service the two requests.[3]

Thus, in general, we describe the service of class i by $d_i(k)$ and $\gamma_i^2(k)$ which are the mean and variance of the time to service k overlapping requests. Denoting $\beta_i^j(k)$ as indicator of whether request j overlapped with the previous k requests such that

$$\beta_i^j(k) = \begin{cases} 1 & s_i^{j+m} > a_i^{j+m+1},\ 0 \le m \le k-2s \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

[3]Note that if the j'th request enters the system after the $(j-1)^{th}$ request is serviced, then this duration reflects the two request's inter-arrival times rather than the time to service two requests.

then the mean latency to service k concurrent requests is given by

$$\bar{d}_i(k) = \frac{\sum_j (s_i^j - a_i^{j-k+1})\beta_i^j(k)1(t - T \leq s_i^j \leq t)}{\sum_j \beta_i^j(k)1(t - T \leq s_i^j \leq t)} \quad (4.4)$$

and likewise for the service variance $\gamma_i^2(k)$. Notice that $d_i(1) = \bar{d}_i$ as in Equation (3.4).

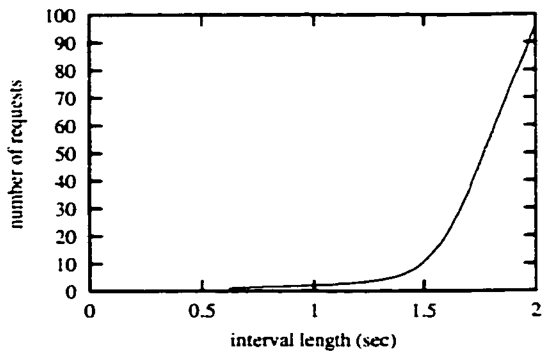


Figure 4.2 Service Envelope

Figure 4.2 depicts an example service envelope from the simulation experiments of Chapter 5. Analogous to Figure 4.1, it depicts the number of concurrent requests serviced as a function of the the latency incurred. The key property of the figure is its convexity so that, for example, the time required to service n requests is far less than n times the time required to service 1 request. This is due to the effects of caching (requests for the same document within close temporal proximity experience significantly smaller latencies) and more generally, the servers ability to efficiently service concurrent requests.

4.2 Sketch LMAC Algorithm

The LMAC test is invoked upon arrival of a new session or request in class i which will increase the request rate from its current value λ_i to $\lambda_i' > \lambda_i$. The LMAC test consists of two parts: the first ensures class i's delay target is satisfied and the second ensures that other classes will not suffer QoS violations due to the increased workload of class i. We illustrate the test pictorially using Figures 4.1 and 4.2.

For class i itself, with a statistical characterization of both requests and service, maintaining a maximum horizontal distance of d_i^* between the two curves ensures that the delay target is satisfied with probability ϵ_i (see [8, 14] for further details). With an increase in λ_i, class i itself increases its request rate yet retains its previous service level. Hence, the latency target is satisfied if the two curves remain d_i^* apart after an increase of $(\lambda_i' - \lambda_i)\tau$ in the request envelope.

For classes $l \neq i$, we must also consider the performance effects of class i on class l's delay targets. Our approach is to bound the effects of the incremental load. Specifically, by an upper bound on class l's new latency, we ensure that class i does not force class l into QoS violations. Moreover, by applying this bound only to the *incremental* load the performance penalty for this worst case approach is mitigated. In other words, class l's current service measurements incorporate the effects of class i's load λ_i, so that only $\lambda_i' - \lambda_i$ is included in the bound. The bound is obtained by considering that the incremental requests $\lambda_i' - \lambda_i$ have strict priority over class l sessions. Under this worst-case scenario, class l's workload remains the same yet its service over intervals of length τ is decreased by $(\tau + D)(\lambda_i' - \lambda_i)$. Hence, the new request can be admitted if each class $l \neq i$ can satisfy its d_l^*, even under a reduction in service by $(\tau + D)(\lambda_i' - \lambda_i)$.

We make three observations about the LMAC test. First, each class' service envelope captures the gains from inter-class resource sharing. For example, if class i can exploit unused capacity from an idle or lower priority class l, class i measures a correspondingly larger service envelope. Similarly, if the server has complete isolation of classes (e.g., via separate back-end nodes in the extreme case), then no such gains will be available and the algorithm will correctly limit admissions to reflect this. Second, the algorithm also ensures class performance isolation, i.e., that admissions in one class do not cause violations in another class, by incorporating the effects of inter-class interference. Finally, we note that while LMAC attempts to maximize the utilization of the web server subject to the QoS constraints independent of the server architecture, QoS functionality in the server itself remains critical. For example, if a web server provides no QoS support and no class differentiation, LMAC infers that only a single service can truly be provided, and restricts admissions to satisfy the most stringent class requirements. Alternatively, when QoS mechanisms are deployed in the web server [4, 5], the resulting efficiency gains are in turn exploited by the LMAC algorithm, which increases the number of admitted sessions in each class and hence the overall system utilization.

Chapter 5

Experimental Investigations

In this chapter, we evaluate the performance of the LMAC algorithm. We perform a set of trace driven simulations. First we describe the simulator used and also the experimental methodology. Next we discuss the experiments performed and also present our results.

5.1 Simulation Scenario

Our simulation scenario consists of a prototype implementation of the LMAC algorithm built into the simulator described in [9], which was developed to approximate the behavior of OS management for CPU, memory and caching/disk storage. The front node has a listen queue in which all incoming requests are queued before being serviced. Each back-end node consists of a CPU and locally attached disk(s) with separate queues for each. In addition, each node maintains its own main memory cache of configurable size and replacement policy.

Upon arrival, each request is queued onto the listen queue or dropped if the listen queue is full. Processing a request requires the following steps: dequeuing from the listen queue, connection establishment, disk reads (if needed), data transmission and finally connection tear down. The processing occurs as a sequence of CPU and I/O bursts. The CPU and I/O bursts of different requests can be overlapped but the individual processing steps for each request must be performed in sequence. Also, data transmission immediately follows disk read for each block.

We have used the same costs for basic request processing as in [9]. The numbers were derived by performing measurements on a 300 MHz Pentium II machine running freeBSD 2.2.5 and an aggressive experimental server.

Connection establishment and tear down costs are set to 145 μs of CPU time each. Transmit processing costs are $40\mu s$ of CPU time per 512 bytes. Reading a file from the disk requires a latency of 28 ms for 2 seeks + rotational latency followed by a transfer time of 410 μs per 4 kByte (resulting in a peak transfer rate of 10 MBytes/sec). A file larger than 44 kBytes is charged an additional latency of 14ms (1 seek + rotational latency) for every 44 kByte block length in excess of 44 kBytes. The cache replacement policy is Greedy-Dual-Size [15]. To incorporate cache behavior, we deliberately set the cache size in our simulation to be 32 MB. The small cache size effectively compensates for the relatively small data set of our traces.

The input to the simulator are streams of tokenized requests, one stream for each user class. Requests within a user class arrive with a user defined mean rate. Each request represents a file (and the corresponding file size in bytes). We generate the arrival stream from logs collected from real web servers.

One of the traces used in our simulations is generated from the CS departmental server log at Rice University. Although we do have the request arrival times embedded in the logs, they do not represent the workload of an overloaded web server. For simplicity we simulate inter-arrival times as exponential. (As this particular assumption is unrealistic, we plan to collect additional traces which also include access times.) The latency experienced by a request is the delay from the time the request arrives at the listen queue until the time when that connection is torn down. The time taken to make admission control decisions are assumed to be negligible.

5.2 Overload Protection

The first experiment was performed to demonstrate LMAC's capability to actually protect the server from overload. In particular, without admission control, as the load offered to a server is increased beyond the server's capacity, the request latencies become excessive. Admission control [2] provides protection from overload by monitoring the utilization of the server and blocking requests which will yield unacceptable performance.

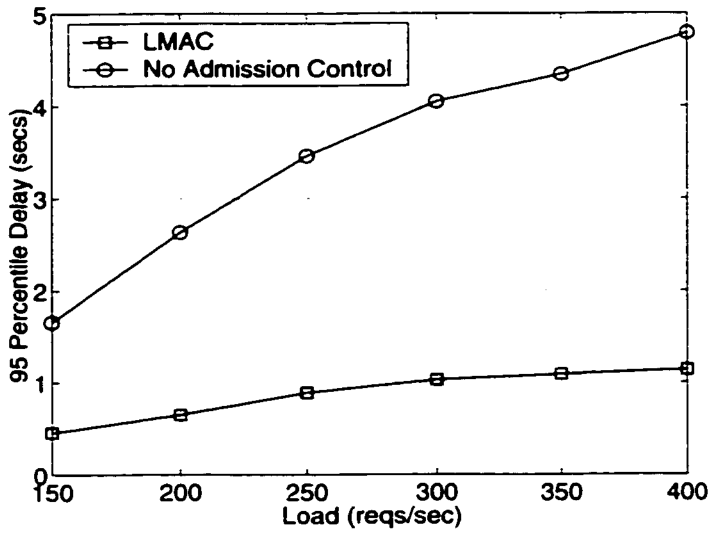


Figure 5.1 95 Percentile Latency vs. Load

To demonstrate the overload protection capabilities of the LMAC algorithm, we simulate various offered loads to the web server, keeping the targeted request latencies to be the same. We compare the performance of LMAC with a web server without admission control capabilities and measure the 95 percentile delay in both cases. Figure 5.1 shows the results for a targeted delay of 1 second. As depicted in the figure, latencies in the unmodified server increase without bound as the load is increased. On the other hand, the LMAC algorithm blocks requests to meet the delay constraint and thus protects the web server from overload. In addition to overload protection, the figure indicates that LMAC controls latencies to within a small error of the target.

5.3 Comparison with the Baseline Scheme

For the second experiment, we compare the performance of LMAC with the queuing theoretic baseline approach of Chapter 3. (We do not compare with admission control schemes from the literature as none have latency targets.) Figure 5.2 shows the measured utilization versus 95 percentile delay for an offered load of 200 requests/sec. The server is a stand alone server with all incoming requests belonging to a single user class. The simulation curve depicts the 95 percentile delay obtained in simulations for a given throughput, while the LMAC and baseline curves depict the throughput obtained when targeting a given 95 percentile delay value.

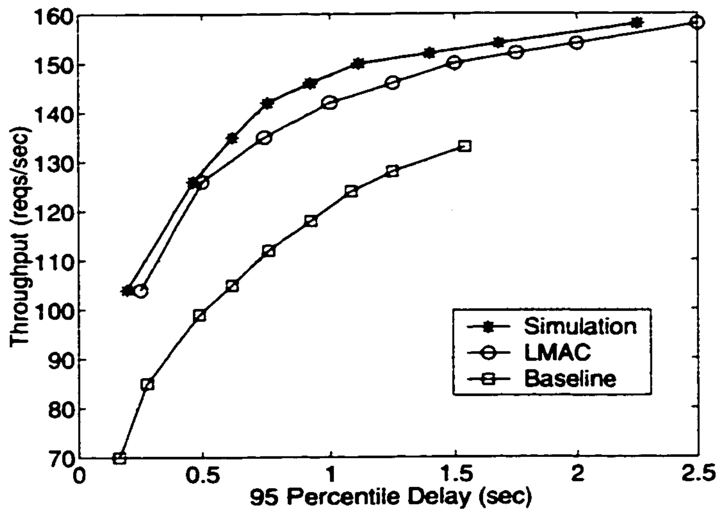


Figure 5.2 Throughput vs. 95 Percentile Latency

Both the LMAC algorithm and the baseline approach meet the latency targets, yet the baseline scheme blocks an excessive number of requests thereby unnecessarily restricting throughput. The low utilization level of the baseline approach can be explained by the fact that the assumption of independent and exponentially distributed service and arrival times does not take into account the inherent variability of traces and web server. For example, back-to-back requests for the same document result in lower delays for subsequent requests (since the document will reside in cache), yet the baseline approach does not exploit this correlation when performing admission control. On the other hand, the LMAC algorithm incorporates temporal correlation and variance properties of requests and services and achieves a correspondingly higher throughput. Regardless, LMAC is still somewhat conservative, for example, for a target 95 percentile latency of 1 second, a 95 percentile latency of .76 seconds is measured at a throughput of 150 reqs/sec, whereas the maximum throughput achievable while still meeting this latency is 157 reqs/sec.

5.4 Multi-Class Performance

To investigate the performance of LMAC in a multi-class environment, we simulate a two-class scenario by randomly classifying incoming requests as belonging to one of the two classes, with each class having a different arrival rate and latency target.

An important point to note here is that a web server without QoS capabilities would only be able to provide a single level of service. Hence, if two differentiated classes are targeted by admission control, the resulting request latencies will be those of the class with the minimum targeted latency: indeed, this behavior was confirmed by our experiments with the simulator.

In order to explore a true multi-class scenario, we devise an artificial resource isolation policy. We consider a server with two back-end nodes and a front-end policy in which the scheme of [9] is modified so that class A requests can be directed to either back-end nodes but all class B requests are directed only to one particular back end node (figure 5.3). Thus, class A receives a minimum of one node's resources yet is able to exploit unused resources of node 2, whereas class B receives a *maximum* of one node's resources. While further class differentiation can be provided by additional QoS server mechanisms described in Chapter 1, this scenario allows a basic exploration of multi-class issues.

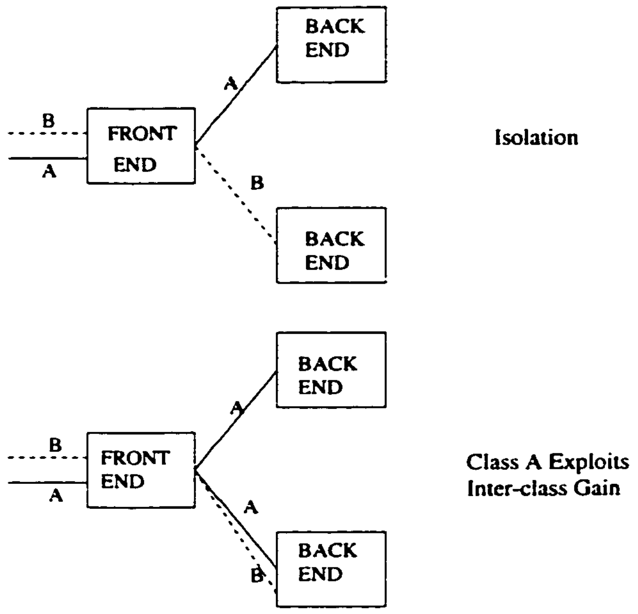


Figure 5.3 Experimental Setup for Investigating LMAC's interclass performance

We perform two experiments. First, we perform simulations with complete isolation of the two classes (all class A jobs are directed to node 1 and all class B jobs are directed to node 2) so that there is no inter-class resource sharing. Next, we perform the experiment as described above so that class A exploits inter-class resource sharing. The results of the experiments are shown in Table 5.1.

Class	Isolation		Multi-class with Sharing	
	Throughput	Delay	Throughput	Delay
A	92	.467	120	.501
B	147	.912	136	.935

Table 5.1 Multi Class Performance of LMAC

The request rate for class A is 300 reqs/sec with a delay target of 0.5 sec and for class B the request rate is 200 reqs/sec with a delay target of 1 sec. Observe from Table 5.3 that when isolated, both classes meet their delay targets at different throughputs as obtained by the LMAC algorithm. More importantly, when the scheduling policy among the backend facilitates inter class resource sharing, the system itself is providing inter-class resource sharing and the LMAC algorithm exploits these gains. Specifically, with the above load balancing scheme, class A increases significantly while both classes' delay targets remain satisfied. Also Class B does not suffer a significant drop in throughput, though its throughput does go down a bit. Also note that the overall throughput of the system goes up, while the different user classes still are able to meet their targeted delay.

Thus, as described in Chapter 4, LMAC can satisfy an arbitrary set of class QoS targets, yet its efficiency in doing so relies in the QoS functionality of the server itself. Regardless, the goal of LMAC is to maximally utilize system resources, given whatever QoS targets are required, and whatever server QoS mechanisms are present.

Chapter 6

Conclusions and Future Work

In this chapter we conclude with a brief summary of the contribution of this thesis and also discuss possible extensions in section 6.2

6.1 Conclusions

In this thesis, we developed a scheme termed Latency-targeted Multiclass Admission Control (LMAC). The algorithm uses measurements of requests and service latencies to control each class' quality-of-service. By abstracting system resources into a high-level virtual server rather than modeling the intricate interactions of low-level system resources, our approach can be applied to off-the-shelf servers enhanced with monitoring and admission control. Moreover, as QoS and performance functionalities are added to servers, e.g., class-based request scheduling, quality-of-service operating systems, or locality aware load balancing, we have shown that the LMAC algorithm exploits these features and realizes a corresponding increase in utilization to various service classes.

6.2 Future Work

In this section we discuss some of the possible future directions that can be taken to extend the work presented in this thesis.

One important area where we would like to extend our work is in the direction of heterogenous content. The work in this thesis has focused upon static content pages only. But recent studies have shown that a significant percentage of web traffic consists of dynamically generated pages. Keeping this in mind we intend to generalize the LMAC framework to encompass the scenario where the typical ,load consists of a mix of static and dynamic requests.

Further, we would like to perform further experiments with a additional traces and also explore the possible prototype implementation of our framework in a commercial web server.

Bibliography

[1] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, September 1999.

[2] L. Cherkasova and P. Phaal. Session-based admission control - a mechanism for improving performance of commercial web servers. In *Proceedings of IEEE/IFIP IWQoS '99*, London, UK, June 1999.

[3] K. Li and S. Jamin. A measurement-based admission controlled web server. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[4] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[5] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: A mechanism for resource management in cluster-based network servers. In *Proceedings of ACM SIGMETRICS 2000*, June 2000.

[6] J. Bruno, E. Gabber, B. Ozden, and A. Silberschatz. The eclipse operating system: Providing quality of service via reservation domains. In *Proceedings of the 1998 USENIX Annual Technical Conference*, New Orleans, Louisiana, 1998.

[7] M. Crovella, R. Frangioso, and M. Harchol-Balte. Connection scheduling in web servers. In *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems (USITS '99)*, October 1999.

[8] J. Qiu and E. Knightly. Inter-class resource sharing using statistical service envelopes. In *Proceedings of IEEE INFOCOM '99*, New York, NY, March 1999.

[9] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the 8th Conference on Architectural Support for Programming Languages and Operating systems*, San Jose, CA, October 1998.

[10] R. Wolff. *Stochastic Modelling and The Theory Of Queues.* Prentice - hall, 1989.

[11] M. Crovella and A. Bestavros. Self-similarity in world wide web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997.

[12] R. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[13] R. Boorstyn, A. Burchard, J. Liebeherr, and C. Oottamakorn. Effective envelopes: Statistical bounds on multiplexed traffic in packet networks. In *Proceedings of IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[14] E. Knightly and N. Shroff. Admission control for statistical QoS: Theory and practice. *IEEE Network*, 13(2):20–29, March 1999.

[15] P. Cao and S. Irani. Cost aware WWW proxy caching algorithms. In *Proceedings of the USENIX symposium on Internet technologies and Systems (USITS)*, December 1997.