# An Admission Control Scheme for Predictable Server Response Time for Web Accesses*

Xiangping Chen
Network Storage Division
EMC Corp.
Hopkinton, MA 01545, USA
xchen@emc.com

Prasant Mohapatra
3115 Engineering Building
Department of Computer
Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
prasant@cse.msu.edu

Huamin Chen
3115 Engineering Building
Department of Computer
Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
chenhuam@cse.msu.edu

## ABSTRACT

The diversity in web object types and their resource requirements contributes to the unpredictability of web service provisioning. In this paper, an efficient admission control algorithm, PACERS, is proposed to provide different levels of services based on the server workload characteristics. Service quality is ensured by periodical allocation of system resources based on the estimation of request rate and service requirements of prioritized tasks. Admission of lower priority tasks is restricted during high load periods to prevent denial-of-services to high priority tasks. A double-queue structure is implemented to reduce the effects of estimation inaccuracy and to utilize the spare capacity of the server, thus increasing the system throughput. Response delays of the high priority tasks are bounded by the length of the prediction period. Theoretical analysis and experimental study show that the PACERS algorithm provides desirable throughput and bounded response delay to the prioritized tasks, without any significant impact on the aggregate throughput of the system under various workload.

## Keywords

Admission Control, Bounded Response Time, Internet, PACERS, QoS, Service Differentiating Internet Servers

## 1. INTRODUCTION

The Internet and its services, especially the use of World Wide Web (WWW) in commercial activities, are increasing explosively. A widely existing problem in contemporary web servers, however, is the unpredictability of response time, which is caused by the *first-come-first-serve* (FCFS) service model and the "bursty" workload behaviors. Usually, *one second* response time is desired from web sites, which is appropriate to the human response speed [4]. Long response delay frustrates user interest in interaction with servers, thus devalues the web service quality. Although current web servers are able to serve thousands of requests per second, the average response delay of a popular server can be several orders of magnitudes higher than that expected during high load periods, causing the *de facto* "denial-of-service" effects. It was estimated that in 1998 about $10-25\%$ of e-commerce transactions were aborted owing to long response delay, which translated to about 1.9 billion dollars loss of revenue [27].

The response delay of Internet service is determined by two factors: the quality (delay, delay jitter, and loss rate, etc.) of network transmission, and the processing capacity of the server. Study of quality of service (QoS) in network transmission is active in recent years, including efforts of building *Integrated Services* (IntServ) architecture [3] with end-to-end QoS guarantee, and *Differentiated Service* (DiffServ) [2] architecture with alternative levels of services provisioning. However, network layer QoS is not sufficient in providing user perceivable performance if the server does not support any service differentiation. A premium class of flow with end-to-end QoS guarantee may still experience service rejection when the server is overloaded. With the increase of resource requirements of web based applications, value added services will emerge with competitive differentiation of service offerings based on profits instead of the *best-effort* service discipline.

Recent studies on QoS support in web servers have addressed the issues of prioritized processing in web servers [9, 6, 12, 15]. We call these kinds of servers as service differentiating Internet servers (SDIS), which complements the network-level QoS efforts. The basic attributes of SDIS include classification of client requests into groups with different service requirements, resource allocations based on the task groups, and prioritized scheduling and task assignments schemes. A detailed study on the concept and performance evaluation of SDIS is reported in [13].

Prioritized scheduling in web servers has been proven effective in providing significantly better delay performance to high priority tasks at relatively low cost to lower priority tasks. However, it is still possible that a significant amount of high priority tasks are dropped under extremely high load situations and user acceptable response delay cannot

*This research was supported in part by the National Science Foundation through the grant CCR-09988179.


*WWW10,* May 1-5, 2001, Hong Kong.
ACM 1-58113-348-0/01/0005.

be ensured. Effective admission control (AC) mechanisms are needed to assure the drop rate and the delay bounds of tasks.

Most contemporary web servers use a rather naive AC scheme, namely, *tail-dropping* AC, in which incoming requests are dropped when the number of tasks awaiting exceeds a predefined threshold. The *tail-dropping* AC scheme requires careful system capacity planning and works well only in steady workload situations. In a highly variable workload environment, more flexible AC schemes are needed to adapt to the dynamics of traffic. For example, secure transactions on the Web, which is popular in E-commerce, consume much more CPU resources than regular transactions due to encryption/decryption and multiple handshaking overheads. The AC algorithm should be designed to consider the characteristics of a wide variety of requests.

In this study we propose a simple and effective AC algorithm, PACERS (*Periodical Admission Control based on Estimation of Request rate and Service time*), which provide bounds on response delay for incoming requests under highly variant workload environments. The PACERS algorithm dynamically allocates system resources for each priority group by estimating the request rate of tasks. Admission of a request is decided by comparing the available computation power for a duration equivalent to the predetermined delay bound with the estimated service time of the request. The service time estimation is done on the basis of the request types. A *double-queueing* organization is used to handle the inaccuracy in estimation while exploiting the spare capacity of the system. Theoretical analysis and experimental study show that the PACERS algorithm bounds the response delay for tasks in different priority groups, assures the service availability to high priority tasks even under high load, and preserves system throughput under various workload.

The rest of the paper is organized as follows. Section 2 answers how to estimate request rate and service times. Section 3 discusses the PACERS algorithm in detail and its implementation issues. Section 4 provides a theoretical proof of the bounded delay provided by the PACERS algorithm. Simulation results and performance evaluation of the algorithm are reported in Sections 5 and 6, respectively. Related works are discussed in Section 7. Section 8 concludes the study.

# 2. WORKLOAD CHARACTERIZATION

Admission control policies are used extensively to provide congestion control and to enforce desirable performance in computer and communication systems. The workload on web servers is different from that of the network traffic; thus existing network-level AC algorithms might not suit well in web server environments. Apparent self-similarity and long-term dependency are prevalent in the WWW traffic pattern [14, 21, 7]. Processing of web requests is much more complicated than the handling of network packets. To explore a simple and effective admission control algorithm which fits to the web server environments, we analyze the traces of a busy web server to track trends and characteristics of resource requirements of the requests. The trace data is collected for a week from the web server of the Department of Computer Science and Engineering at Michigan State University, which encountered about a million requests during the period.

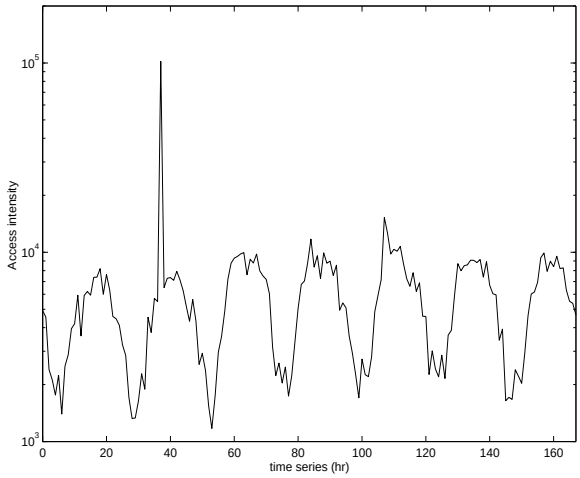


**Figure 1: Access trends in a week.**

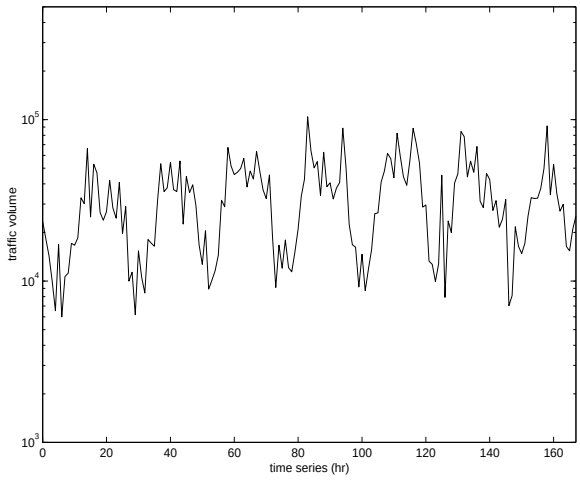


**Figure 2: Traffic trends in a week.**

## 2.1 Access Distribution

Figures 1 and 2 show the request intensity and traffic volume per hour in a week. Distinct seasonal/periodic behaviors according to the time series can be observed from the figures. For example, the access intensity continues to increase from early morning at 10AM, reaches the high intensity area, and stays high till late afternoon, then decreases slowly to the lowest point in a day, which is about 3AM in the next day. Similar traffic patterns repeat around the same time each day, which suggests that both short and long term history should be used as predictors of future workloads.

The fluctuation in traffic intensity can be several orders of magnitudes, which causes the high variances in access rate and bandwidth requirements, thus resulting in high average waiting time of tasks in a system. Care should be taken to eliminate the effects of variances. An interesting phenomenon is that the access pattern of the web server tends to be consistent in a short time window. For example, the coefficient of variation (CoV) of access rates decrease to around 1 when they are measured on an hourly basis. The decrease for CoV is even more obvious during busy hours. Simi-

lar observations have been reported in [24, 19, 23], which suggest that a multi-state Modulated Markov Poisson Process (MMPP) can be used to approximate or predict the burstiness of the aggregate input of the web server, which is discussed in more detail in Section 3.3.

Note that during a week's observation period, there is one overload point which causes request access rate of about 10 times higher than the common peak load of the day. We examined the logs and found that the abnormal overload was caused by a group of careless CGI based requests. Those requests directly caused the web server failing to respond to any other request for 21 minutes. This is one of the examples which justify the need for appropriate AC to prevent system resources being wasted in careless or malicious request attempts.

## 2.2 Object type distribution

Table 1 lists the requested web object types and corresponding traffic distribution of the traces. Web object types are categorized on the same basis as reported in [11]. It can be observed that requests for small and static web objects, i.e., HTML and image files, still dominate the incoming requests and web traffic. The data show that the CoV of web traffic is as low as 1 to 2 for each type of web objects. However, the CoV of the aggregated web traffic can be as high as 14.41.

**Table 1: Traffic distribution vs. object type.**

| Item | Req. (%) | Traffic (%) | Mean (KB) | CoV |
|---|---|---|---|---|
| HTML | 19.2 | 15.0 | 5.76 | 1.90 |
| Image | 68.8 | 49.2 | 4.98 | 2.46 |
| Audio | 0.2 | 2.5 | 579.9 | 1.76 |
| Video | 0.1 | 6.7 | 2503.9 | 1.56 |
| Dynamic | 4.9 | 4.4 | 6.84 | 1.33 |
| Other | 6.8 | 20.2 | 19.0 | 7.90 |
| Total | 100 | 100 | 7.39 | 14.4 |

There are non-negligible percentage of dynamic objects. Previous studies [18] have estimated that a dynamic object requires 10 to 100 times of service time than a typical static object. The processing time difference was confirmed by recent benchmark tests of popular commercial web servers [8] and the following service time characteristic study. The throughput of serving dynamic tasks is usually lower than serving static tasks by a factor of 10. Continuous media (CM) objects (audio and video clips) are now being used more extensively in the web environments (compared to the previous results in [11]). These data types require high storage capacity (thus uncacheable), high density I/O processing, and sustained bandwidth with bounded delay constraints. The average CPU time for serving a CM object is also much higher than the average CPU time for serving small static objects. Significant differences in size of different type of web objects, high volume of CM objects, and computation-intensive dynamic requests suggest that the service time variance of the requested web objects should be considered in estimating the server capacity and response time.

## 2.3 Service Time Distribution

Previous studies [5, 17, 22] suggested that the service time of retrieving static web objects such as HTML and

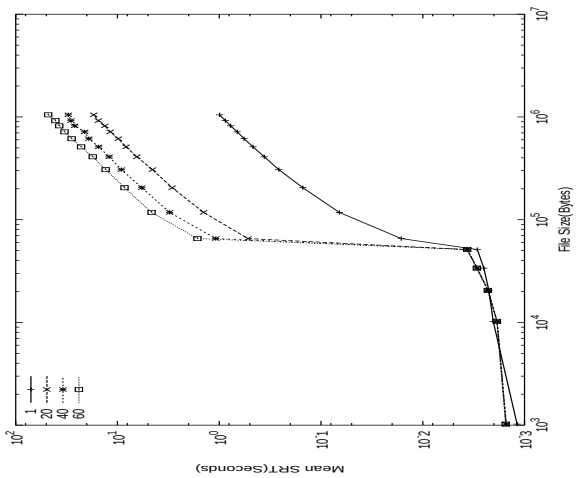


**Figure 3: Mean service time of web objects.**

image files can be divided into two parts: an almost fixed URL processing time, and the content transfer time which increases linearly with the file sizes. To collect the service time attributes, we set up an experimental environment consisting of one Apache (version 1.3.12) web server and three WebStone (version 2.5) clients, connected through a 10 Mb Ethernet. The web server hardware is based on a Pentium II 233MHZ CPU with 128 MB memory. The operating system used in the server was Linux 2.2.9. The clients were installed in Sun UltraSparc 10 with 128 MB memory, running Solaris 2.6. The CPU cycles are used as time scale to obtain a precise time resolution. The *performance monitoring counters* provided by the Intel P6 family processors are used to record the elapsed busy CPU cycles of the current process.

The service time is defined as the CPU cycles consumed between the time that the server accepts a client request, and the time that the server finishes sending the response. Figure 3 depicts the mean service time of static objects versus requested file size under different *maximum number of processes* (MNP) in the server. It is well known that the MNP has direct influence in service time distribution, which is discussed in detail in following paragraphs. CPU cycles have been converted into seconds in the figure.

It can be observed from the figure that the curves have two phases. If the files are smaller than 64 Kbytes, the service times with the same MNP are almost constant. If the file sizes exceed 64 Kbytes, the mean service times increase linearly with the file sizes and the MNP value. We call this phenomenon the *64 KB leap*. The *64 KB leap* is mainly due to the underlying operating system kernel. In Linux socket implementation, a 64KB send buffer limit is imposed for every transmission. Any send request of more than 64KB is fragmented and the first 64KB is processed while the remaining parts wait for the buffer space. The kernel can thus avoid crash from any imprudent code asking for more memory than what the kernel could provide. This limit is tunable via ioctl() function. However, a large send buffer will not necessarily improve the overall performance since smaller transmission may spend more time in the queue waiting for the larger requests to be transmitted. Some other factors also contribute to the *64 KB leap*. For

example, asynchronized disk I/O is widely used in current UNIX operating systems, and the default size for *read-ahead* operation is 64 KB. At most 64 KB can be loaded from the hard disk in one I/O operation.

The slopes of service time increase linearly with the number of maximum processes, because the increase of process number caused higher probability of page faults, higher context switching, and synchronization overhead. Based on the observation from Figure 3, the service time of a task $T(s,n)$ can be estimated by the file size $s$ KB and MNP value of $n$.

$$T(s,n) = a + [s/64] * (b + c * n), \quad (1)$$

where $a$ is the service time for small static web objects, i.e., requests for files smaller than 64 KB. $b$ is the data transferring time factor, and $c$ is the context switching overhead factor. Using linear regression, we get the relative value for $a, b$, and $c$ as: $a : b : c = 1 : 0.06 : 0.18$.

Rechecking the file size distributions, we find that less than 1% of HTML and image files are larger than 64 KB. Thus the service time of most HTML and image files are rather uniform. Service times of dynamic objects, mainly CGI requests, depend on the computation complexity of the URL processing instead of response size. Using the CGI scripts test set provided by WebStone 2.5 test set, the average service time of a CGI request is around one order of magnitude higher than the service times of static objects with a file size less than 64 KB. The experimental results indicate that the object type is a good indicator of the requested CPU time, which can be derived from the requested URL. Besides, classification of object types introduces less overhead than retrieving file size information, which requires one *fstat()* system call in UNIX operating systems.

# 3. PACERS ADMISSION CONTROL ALGORITHM

This section presents how the web server workload characteristics impacts the design of our AC algorithm. The goal of the PACERS algorithm is to provide response delay bounds to incoming requests while preserving the system throughput of busy web servers.

## 3.1 Overview of the algorithm

Usually a queue of incoming requests is maintained in a web server awaiting to be processed. Using the *tail-dropping* AC scheme, incoming requests are put in the queue until the queue is full. Queue length is not always a good indicator of system load, especially when the variance of processing time is high. Without effective AC, the server response time and throughput deteriorate drastically when the aggregate request rate exceeds server capacity, indiscriminately affecting all clients. Abdelzaher and Bhatti [5] reported that as much as half of the web system's processing capacity is wasted on eventually aborted/rejected requests when the load is high.

To accommodate the high variance in request rate and service time of web servers, we propose a simple and adaptive AC algorithm, *Periodic AC based on Estimation of Request rate and Service time* (PACERS), to ensure the availability and delay performance of prioritized tasks. The predictive strategy estimates the periodic request rate of each priority group, and guarantees the performance of higher priority tasks by restricting admission of lower priority tasks. The request rate estimation is based on the history of access pattern. Service time of each task is estimated based on the request types which is more or less delineated by the size of the responses. Inaccurate estimations are dynamically adjusted by a *double-queue* architecture as described later.

## 3.2 Admission Control and Delay Bounds

We first examine a non-prioritized system to have a systematic understanding of the proposed algorithm. Assume that the time is divided into discrete slots $(0,T)$, $(T,2T)$, ..., $(kT,(k+1)T)$, ... . For simplicity, we use the beginning point of each time period, $kT$, to represent the duration $(kT,(k+1)T)$. Let $c$ be the unit processing capacity of the system, $C(kT)$ be the predicted system processing capacity in period $kT$, $S(i,kT)$ be the service time needed by the $i$th task at period $kT$, and $n(kT)$ be the number of admitted tasks in period $kT$. If the server has a precise knowledge of service time needed by each task, the admission decision can be made based on the following expression,

$$C(kT) = c * T >= \sum_{i=1}^{n(kT)} S(i,kT). \quad (2)$$

If expression 2 is true, then the request is admitted, otherwise it is rejected. The maximum response delay of each task is bounded by the value of $T$ if the service time of each task is known prior to the admission decision phase.

In a prioritized system, the periodic system capacity seen by different priority groups changes with the prediction of resource requirements of each priority group. By adjusting the assigned system capacity to each priority group, the PACERS algorithm provides service quality assurance to prioritized tasks. There are two kinds of service disciplines that can be provided by the PACERS algorithm: *prioritized resource allocation* (PRA) and *weighted fair allocation* (WFA). PRA is implemented by assigning resources equal to the whole system capacity to the highest priority tasks (or premium tasks). Lower priority tasks get the remaining resources. The WFA is realized by setting shares of system resources in each priority group, where each priority group get at most/least their shares. In this study we only discuss the PRA control scheme. The WFA scheme can be easily extended from the PRA scheme by setting a minimum or maximum resource ratio for each priority group.

The objective of the server is to ensure QoS to high priority tasks whenever their arrival rate is lower than the system capacity. Thus, for high priority tasks, the available resources are equal to the system period capacity. For lower priority tasks, the available resources are the system capacity minus the predicted resource requirements of higher priority requests during a time period. Since the priority ratio and distribution of the types of incoming tasks vary over time, dynamic assignment of system capacity is needed to preserve the system throughput.

Assume all the requests are classified and are assigned a priority $p, (p = 1, .., P)$, wherein $P$ denotes the highest priority. Let $\lambda_i^{predicted}, (i = 1, .., P)$ be the predicted inter-arrival rates of tasks for each priority group. The system capacity available to priority group $p$ at time $kT \leq t \leq (k+1)T$, denoted as $C_p(kT,t)$, is:

$$C_p(kT,t) = C(kT) - \sum_{i=p+1}^{P} \lambda_i^{predicted}(kT) * T - \lambda_p(kT) * t. \quad (3)$$

A task with priority $p$ is admitted if the service time is equal or less than available capacity $C_p(kT, t)$.

## 3.3 Estimation of Request Rate

The resources allocated to each priority group is based on the prediction of the request rate of incoming prioritized tasks. Apparent seasonal workload patterns corresponding to daily cycles discussed in the previous section can be used to predict current traffic intensity based on the workload history. On the other hand, reports in [19, 23] suggested that the aggregate web traffic tends to smooth out as Poisson traffic in short observation time windows. This fact was further proved by Morris and Lin in [24]. Based on the above published results, we decided to use Markov-Modulated Poisson Process (MMPP) described in [16] to capture the seasonal behavior of the aggregate workload of web servers, while preserving the tractability of modulated Poisson process. The request arrival process is described as a Markov process $M(i)$ with state space $1, 2, ..., i, ..., N$. State $i$ has arrivals with Poisson process at rate $\lambda_i$. To follow the seasonal behavior, especially the day/night cyclic behavior of the web server load, the observed traffic data is chopped into subsets for each hour on a daily basis. The Markov transition matrix $Q(n) = [Q_{ij}(n)], (n = 1, ..., 24)$ for each hour can be easily calculated by quantizing the inter-arrival rate in each observation period, and calculating the frequency at which $M(n)$ is switched from state $i$ to state $j$. The predicted access rate $\lambda^{predicted}(kT)$ can be expressed by the following equation:

$$\lambda^{predicted}(kT) = [\lambda((k-1)T)]. * Q(kT), \qquad (4)$$

where $[\lambda((k-1)T)]$ is the state vector of measured inter-arrival rate in the previous period. $Q(kT)$ can be further adjusted by comparing the differences between predicted data and measured data, to catch up with the long term trends of a web server load. In the experiment, we use three state $(p_{inc}, p_{same}, p_{dec})$ transition matrices to capture the traffic trends in each observation period. $p_{inc}$ is the probability of increment request rate, $p_{same}$ the probability of the same request rate, and $p_{dec}$ the probability of decrement request rate. The $\delta$ value for increment and decrement is 10% of measured value. The experiment shows that there is not much difference in the capability of capturing the traffic trends in each observation period between a three state transition matrix and more complicated state transition matrices.

## 3.4 Estimation of Service Time

While deciding to accept a request, the system should ensure that the sum of service time of accepted tasks do not exceed the system capacity. In reality, it is not possible to know the service time $S(i, kT)$ in advance. High variance in resource requirement is a widely recognized characteristic of web server workload. As indicated in the previous section, however, the service time and bandwidth requirement of the same type of requests are more or less consistent. Service time of web objects can be thus estimated based on the distribution of the types of requests to the server. We approximate the service time of each task by using weighted *computation quantums* (CQ) matching the CPU processing time of different type of tasks.

When a new request arrives, the system checks if there are enough CQ available to grant to the request. Only requests that are granted enough CQ can be enqueued and served eventually. The number of CQ issued to each task is determined by the resource requirement of the request. Here we denote $T_n$ as the number of types of requests to a web server, which can be derived from the workload characterization of a web server. Let $N_i(kT)$ be the number of requests of type $i$ in period $kT$, $CQ_i$ be the weighted CQ matching the CPU processing time for type $i$ tasks. Then Equations (2) and (3) can be approximated as:

$$C(kT) \geq \sum_{i=1}^{Tn} CQ_i * N_i(kT) \qquad (5)$$

$$C_p(kT) \geq \sum_{i=1}^{Tn} CQ_i * N_{i,p}(kT). \qquad (6)$$

As in the case of Equations (2) and (3), a request is admitted if Equations (5) and (6) hold true, otherwise is rejected.

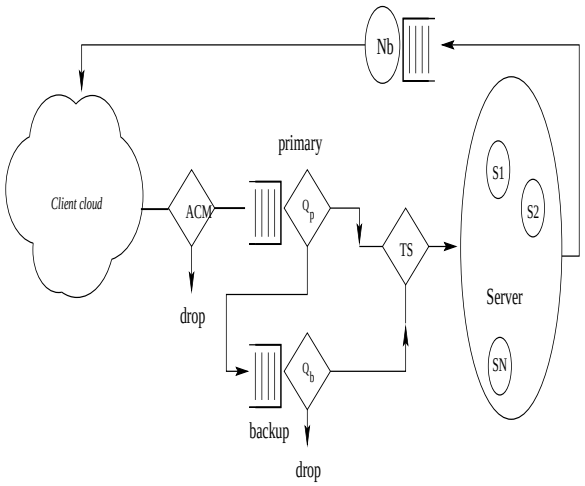


**Figure 4: Web server system structure.**

## 3.5 The Double Queue Structure

Since Equations (5) and (6) are approximations of Equations (2) and (3), care needs to be taken to amortize the accumulated delay influence of over-admission during a time period. Assume that a restricted prioritized processing is enforced inside a queue, i.e., no lower priority tasks get served if there is a higher priority task waiting, incoming requests are queued in the order from high priorities to low priorities. Requests in the same priority are queued in FCFS order. When over-admission happens, it is possible that low priority tasks stay in the queue for a long time awaiting services while high priority tasks get dropped due to lack of queue space. On the other hand, under-admission wastes system capacity. A *double-queue* structure is used to handle the over/under admission problems. A primary queue is used as the incoming task queue, and a secondary queue is added for the backed up requests (we call this as the backup queue). The system structure is shown in Figure 4.

An incoming request is first sent to the AC manager ACM. The ACM classifies the request priority and decides if the request can be enqueued. Enqueued requests wait to be served in the primary queue Qp. At the beginning of the next period, unfinished requests are sent to the backup queue Qb.

When Qb becomes full, it is cleared up and queued tasks are dropped. Other methods for expunging tasks from Qb can be explored. The task scheduler TS picks up requests in the queues and sends to the server pool. No request in the Qb can be picked up unless the Qp is empty. Replies are sent back through the server network interface Nb. By using a double queue structure, newly accepted requests need not wait for a long time for service, thus bounded delay is achieved for most of the requests.

# 4. RESPONSE DELAY ANALYSIS

The PACERS algorithm provides delay bounds and average waiting time assurance as well as throughput regulation. In this section, we analyze the delay bounds and the waiting time for the requests.

## 4.1 Ideal Case Delay Bounds

When the short term arrival rate is faster than the processing rate, newly arrived requests have to wait in the queue for service. If the variations of job arrival rate and processing rate are high, more queue space is needed, and the average queueing time and response delay tend to be high. If enough buffer space is not available, denial of service occurs when the queue is full.

One way to increase the response time predictability of a system is to limit the number of requests in a given "short time" period $T$ to no more than what the system can handle in that period of time. Borowsky et al. [10] provided a theoretical proof that the service time required by pending jobs is no more than $T$ in a FCFS work-conserving system, if the sum of service time required by the arriving jobs at any duration $T$ is bounded by time $T$. Thus the response time (or delay) of all requests can be bounded by $T$ by restricting the workload arriving in every interval of length $T$ time.

In a prioritized environment where each request is assigned a specific priority, the processing order and resource allocation are based on the priority of each request. The FCFS processing assumption in the above theorem is no longer valid. In [12], we proved that the mean response time of a request is bounded by the arrival rate with equal or higher priority and the service rate of the system, if requests with different priorities have the same service requirement distribution and a strict priority scheduling is used.

Let $p(1 \leq p \leq P)$ denote the priority of the requests and $P$ be the total number of priorities. Let $s_p(i)$ be the service time requirement of task $i$ belonging to priority $p$, and $N_p(t-T,t)$ be the number of requests of priority $p$ arriving during the period $T$.

**Lemma :** In a prioritized preemptive work-conserving environment with no pending work, if $N(t-T,t)$ requests arrive belonging to a single priority level $p$, then their response time is bounded by $T_p(T_p < T)$ if $\sum_{i=1}^{N(t-T,t)} s_p(i) \leq T_p$ (proved in [10]).

**Theorem :** If the priority levels are in the increasing order of $p$, the response time of a task with priority $p$ is bounded by $\sum_{q=p}^{P} T_q$. If $\sum_{q=1}^{P} T_q \leq T$, the response time of any task is bounded by $T$.

**Proof :** Let $R_{p,i}(n)$ be the response time of the $i$th task in the priority class $p$ during period $n$, and $busy\_time(n)$ be the amount of time in $(n)$ that the system is busy during period $n$. Let the request time series be partitioned into periods of $n$ with length $T$.

For $n = 1, i.e., t = (0,T)$, without loss of generality, let all requests arriving during (0, T) be reshuffled to be served in the decreasing order of $p$ (for $p = 1,..,P$), then $R_{P,i}(1) <= T_P, for i = 1,...,N_P(1)$, and $R_{p,i}(1) <= \sum_{q=p+1}^{P} T_q + T_p <= \sum_{q=p}^{P} T_q, for\ p = 1,..,P$.

Assume the above expressions hold true in the period $k-1$, reshuffle the requests in queue in the decreasing order of $p$ during period $k$. Let $Q_p(k)$ be the sum of the service times for the $p$th priority tasks, then

$$Q_p(k) = Q_p(k-1) + \sum_{q=p}^{P} \sum_{i=1}^{N_q(k-1)} s_q(i) - busy\_time(k), \quad (7)$$

Clearly, $Q_p(k-1) <= busy\_time(k)$. By assumption, $\sum_{q=p}^{P} \sum_{i=1}^{N_q(k-1)} s_q(i) <= \sum_{q=p}^{P} T_q$, so

$$Q_p(k) <= \sum_{q=p}^{P} \sum_{i=1}^{N_q(k-1)} s_q(i) <= \sum_{q=p}^{P} T_q. \quad (8)$$

Hence, by induction from Equation (8), we get $R_{p,i}(k) <= \sum_{q=p}^{P} T_q (i = 1,...,N_p(k)$, for $p = 1,...,P$, and $k = 1,2,...,$ i.e., the response delay boundary of each prioritized task can be achieved by controlling the requested service time in each period of length $T$. As long as the sum of service times requested by high priority tasks does not exceed $T$, the system response time can be assured by restricting the number of admitted low priority tasks.

The granularity of $T$ effects the system performance. Increasing the value of $T$ smoothes out the access variance between adjacent periods and allows more requests to be admitted. However, larger value of $T$ also increases the variance of response time and degrades the system responsiveness. Both user perceivable delay and system throughput should be considered in determining an appropriate value of $T$.

## 4.2 Waiting Time Estimation

The service time of a web object can be estimated from statistics of the same or similar type of web object with acceptable variance. The number of requests allowed by the system can be derived from the expected waiting time and the job arrival rate. Let $W_p(t)$ denote the expected waiting time of tasks with priority $p$ at time $t$, $A_p(t)$ be the expected arrival rate of tasks with priority $p$ at time $t$. Let $W(t)$ denote the expected waiting time of a task in the system at time $t$, $A(t)$ be the expected arrival rate of tasks of the system at time $t$. In the period $k$, the expected number of tasks in the system equals:

$$N(k) = W(k) * A(k) = \sum_{p=1}^{P} A_p(k) W_p(k). \quad (9)$$

According to Little's Law [20], the waiting times of each priority group equals:

$$W_p(k) = \frac{N(k) - \sum_{i=p+1}^{P} A_i(k) W_i(k)}{A_p(k)}. \quad (10)$$

On the other hand, the waiting time of a task in the $p$th priority group equals the residual life of the executing task $W_0$ plus the sum of service times of equal or higher priority tasks in the queue, and the sum of service times of higher priority tasks arriving while it waits in the queue. The mean

service time of group $i$ in period $k$ is represented as $S_i(k)$. The waiting time is thus equal to :

$$W_p(k) = W_0 + \sum_{i=p}^{P} N_i(k) + \sum_{i=p+1}^{P} W_p(k)A_i(k)S_i(k). \quad (11)$$

Let $\rho_i(k) = A_i(k)S_i(k)$. By combining results of Equations (10) and (11) we can get,

$$W_p(k) = \frac{W_0}{(1-\sum_{i=p}^{P}\rho_i(k))(1-\sum_{i=p+1}^{P}\rho_i(k))} \leq \frac{1}{c(1-\sum_{i=p}^{P}\rho_i(k))(1-\sum_{i=p+1}^{P}\rho_i(k))}. \quad (12)$$

$c$ is the unit processing capacity of the system, its inverse value is the worst case expected residual life of an executing task, which happens when the utilization factor approaches to 1. Equation (12) shows the mean waiting time of prioritized tasks by using the estimation of the inter-arrival rate of equal or higher priority tasks.

Similarly, we can get the expected task inter-arrival rate and acceptable number of tasks in each period based on the expected waiting time. The result is shown as:

$$N_p(k) = A_p(k) * T = \frac{W_0 - W_p(k)(1-\sum_{i=p+1}^{P}\rho_i(k))^2}{1 - W_p(k)S_p(k)\sum_{i=p+1}^{P}\rho_i(k)}T. \quad (13)$$

Note that the above analyses are based on the periodic data collection assumption. In fact, all the data used in the above equations can be easily obtained from a few counters which are reset periodically. The workload of each priority level is estimated at the end of each period. Expected number of tasks is determined based on the expected inter-arrival rate. During one period, if the number of incoming tasks of one priority level exceeds the expected number, then there is no need to accept new requests in the same priority level until the beginning of next period.

# 5. SIMULATION MODEL AND PARAMETERS

In this study, we simulate an event driven server model using empirical workload from real trace files. The reference locality, size and object type distribution are extracted from the logs of the Computer Science departmental web server at Michigan State University and are used for the workload generation. We only consider two priority levels to simplify the study since the primary concern is to examine the effectiveness of the PACERS algorithm. The system configuration is shown in Table 2.

Three performance metrics are used in the simulation: server throughput, mean response time, and response delay bound miss rate of each priority group. System throughput indicates the server capability. Mean response time and delay bound miss rate quantify the service qualities in each priority group. The delay bound miss rate presents the proportion of tasks whose response time exceed the bounded delay.

**Table 2: Simulation Configuration.**

| Parameter | value |
|---|---|
| Priority Level | 2 |
| Scheduling Period | 1 sec. |
| Static Obj. Throughput | 1000 req./sec |
| Dynamic Obj. Throughput | 100 req./sec |
| Network Bandwidth | 100 Mbps |
| Disk Bandwidth | 100 Mbps |
| Caching hit ratio | 0.7 |
| Maximum open connections | 1000 |
| Maximum queue length | 1000 |
| Maximum server process number | 30 |
| Response delay bound | 1 sec. |

Based on the data reported in Section 2, web objects are classified into 4 types: static, dynamic, audio and video objects. The CQs consumed by each object types and their input percentage are listed in Table 3. The CQs allocated to each object type is based on the results in Section 2.3.

**Table 3: CQs requested by each object types.**

| Object Types | Static | Dynamic | Audio | Video |
|---|---|---|---|---|
| CQs | 1 | 10 | 20 | 100 |
| Request Freq. | 94.7 | 5 | 0.2 | 0.1 |

# 6. PERFORMANCE EVALUATION

Our experiments compare the performance of the PACERS algorithm to that of a simple admission control (SAC) algorithm, which is analogous to the leaky bucket [26] algorithm used in traffic engineering in network transmissions. The SAC algorithm produces CQ at a constant rate matching the server throughput; each admitted task consumes one CQ. Incoming tasks are dropped if there is no CQ available. However, estimation of the request rates is not implemented in the SAC algorithm. The SAC scheme outperforms the *tail-drop* scheme, since it smoothes out the web server traffic and also provides preliminary overload protection.

In the experiment, the ratio of high priority to low priority tasks is 1 to 1, and both types of tasks are randomly distributed in the whole arrival sequence. The web server trace is used to generate request at various rate and the performance is recorded from the simulated server environment.

## 6.1 Throughput Performance

Figures 5 and 6 plot the throughout of stress test of two priority groups. Normalized load in the x-axis is the ratio of real load to the system capacity, and the normalized throughput is ratio of the throughput to the system capacity. In the stress test, the aggregate request rate increases continuously till 2.5 times of the system capacity. The objective of this experiment is to explore the capability of the algorithm in preserving the system throughput and delay performance under extremely high load by the stress test.

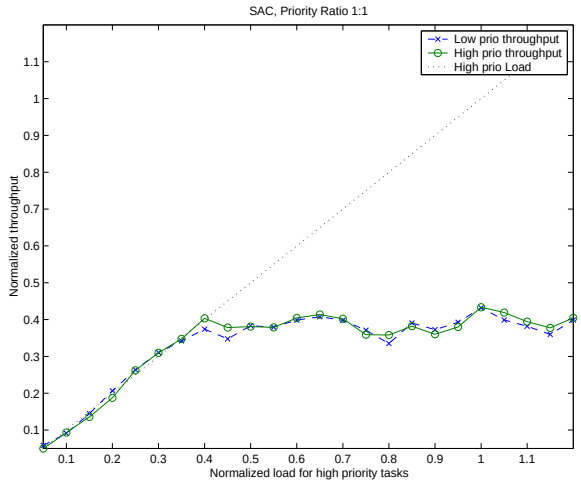


**Figure 5: Throughput using SAC.**

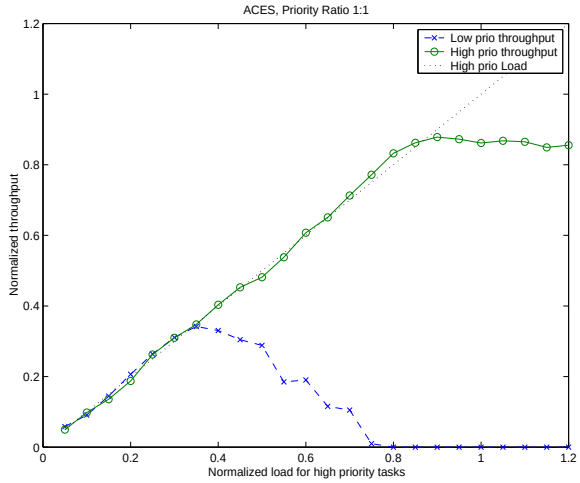


**Figure 6: Throughput using PACERS.**

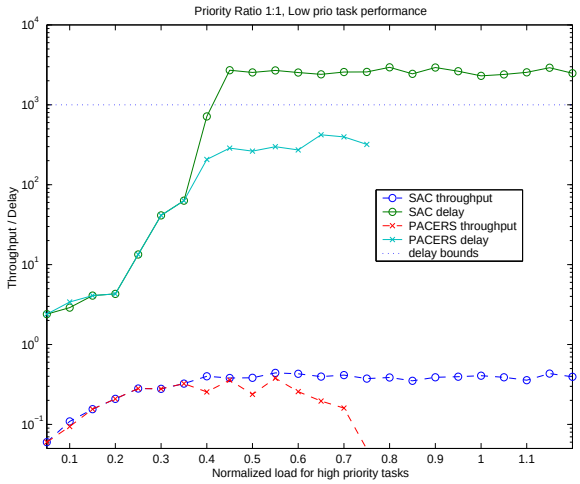


**Figure 7: Performance of low priority tasks.**

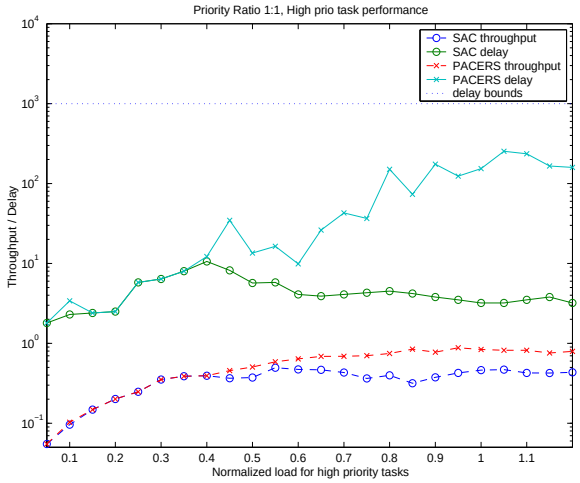


**Figure 8: Performance of high priority tasks.**

Figure 5 shows the throughput variation using the SAC algorithm. It can be observed that the throughput of each priority group is proportional to the traffic percentage of each group. High priority tasks suffer from the same low throughput as low priority tasks when the aggregate workload is high. Predefined QoS profile strictly prioritized admission and scheduling, are not ensured.

Figure 6 shows the throughput of each priority group when the PACERS scheme is deployed. The system stops serving low priority tasks when the high priority load approaches about 80% of system capacity. On the other hand, the throughput of high priority tasks equals the request rate of high priority group until the system utilization reaches 0.8, which is the highest throughput of the system using the traces as input workload. The throughput of high priority tasks remains at the 80% level when the high priority tasks overload the system. It can also be observed that the aggregate throughput of the two AC schemes remains at the same level, which confirms that the PACERS scheme does not degrade the system aggregate throughput.

## 6.2 Delay Performance

The average delay for each priority group is also well controlled in the PACERS scheme. As shown in Figures 7 and 8, the average delay of each priority group is much lower than the predefined delay bounds of 1 second.

Figure 7 shows the delay and throughput variation of low priority tasks using the two AC algorithms. Using SAC algorithm, the throughput of low priority tasks is proportional to its traffic ratio. However, extremely long delays are introduced when prioritized processing is used. On the contrary, the PACERS algorithm blocks admission of low priority tasks during high load periods in exchange of low delay bounds miss ratio. The prediction and CQ assignment behavior of the PACERS scheme avoids unnecessary waiting of low priority tasks, thus decrease the system overload during high load periods.

Figure 8 shows the high priority task delay and throughput performance of the two AC algorithms. Using SAC algorithm, high priority task throughput is proportional to the ratio of high priority traffic to the total traffic volume, although the high priority task traffic is low. Delays of high

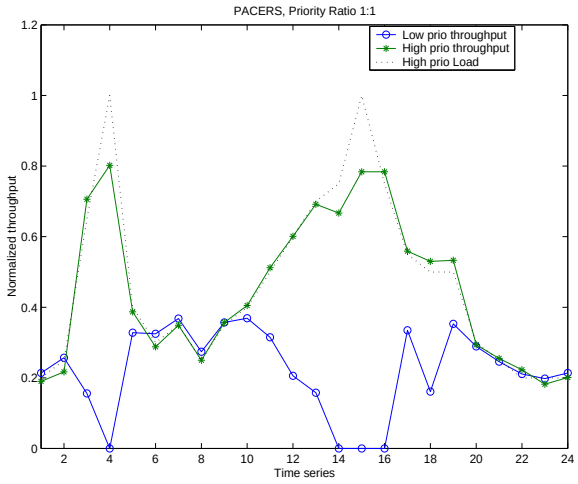


**Figure 9: Throughput under fluctuating load.**

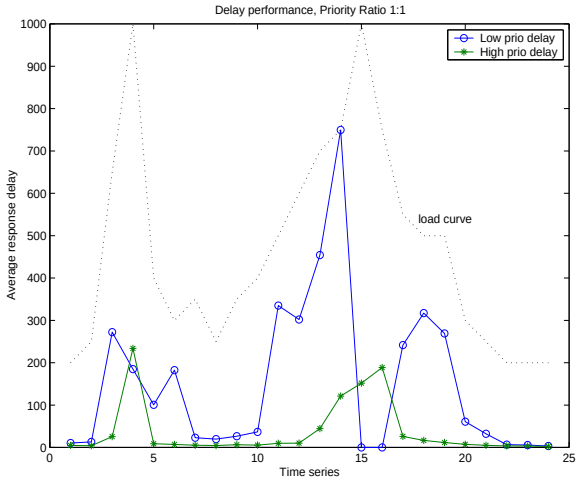


**Figure 10: Delay under fluctuating load.**

priority tasks are kept low because prioritized processing is used. On the contrary, the PACERS algorithm preserves throughput performance of high priority tasks during high load periods at the cost of increased average response delay, but still within the delay bounds.

## 6.3 Sensitivity Test

We test the performance of the PACERS algorithm under fluctuating load situation. The result shows that the PACERS algorithm is very sensitive in detecting variation of system loads, thus preserving throughput and delay performance in different priority groups under varying loads. Figures 9, 10 and 11 plot throughput and delay variation under fluctuating workload. The experiment is aimed at examining the sensitivity of the PACERS algorithm to the variation in the workload. The workload can be classified as sustained lightload or overload, and occasional lightload or overload. The occasional overload duration is 2 time units, and the sustained overload duration is 10 time units. Each time unit is 1000 times of the observation period.

The maximum resource requirements of high priority requests equal the system capacity. It can be observed from

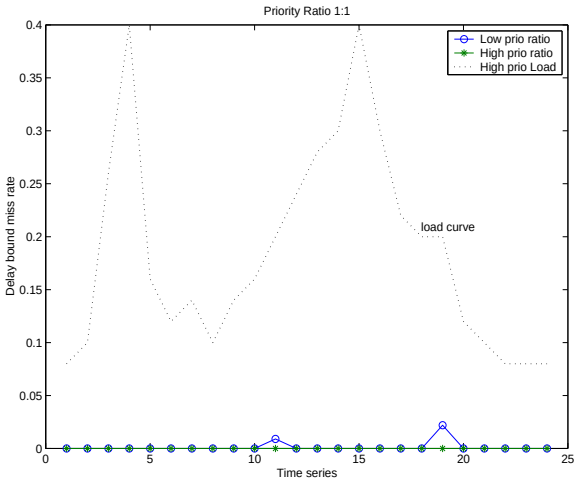


**Figure 11: Delay bound miss ratio under fluctuating load.**

Figure 9 that the PACERS scheme is very sensitive to the load fluctuation, which blocks the low priority tasks under high load situation to preserve the throughput of high priority tasks, and resume service of low priority tasks during medium to low load situation. Figure 10 shows the delay variation of the two priority groups. The delay of high priority tasks follow the trends of incoming high priority workload, but the maximum value is only about one-fifth of the delay bounds. The delay of low priority tasks is also well controlled under the delay bound. Figure 11 shows the delay bounds miss ratio of the two priority groups. High priority tasks experience zero delay bounds miss ratio under various load situations; delay bound miss ratio of low priority tasks occasionally exceeds zero.

# 7. RELATED WORK

Several studies on QoS support in web servers have addressed the technology of prioritized task processing and admission control issues. Bhatti and Friedrich [9] addressed the importance of server QoS mechanisms to support tiered service levels and overload management. A Web-QoS architecture prototype was developed by adding connection manager module to the Apache [1] Web server. Admission control is implemented by blocking low priority tasks when the number of high priority waiting tasks exceeds the threshold. Eggert and Heidemann [15] evaluated application level mechanisms to provide two different levels of web services. The admission control is implemented by limiting process pool size and response transmission rate to different priority groups. Pandey et al. [25] described a distributed HTTP server which enables QoS by prioritizing pages on a web site. The admission control is realized by assigning communication channel to prioritized pages. Most of these admission control mechanisms are based on a predefined "threshold". Performance of high priority tasks is guaranteed by emulation of a fixed bandwidth "leased line". However, it is expensive to satisfy the requirements of bursty workload using "leased line" scheme, since peak loads are several orders of magnitude higher than the average load.

## 8. CONCLUSION

When a server is unable to provide satisfactory service to all requests, selective resource allocation is a promising technique to assure service to requests which are more important to clients or servers. In this study, we present a novel and effective admission control algorithm, PACERS, to adapt to the highly variant access patterns and processing of web server environments. Tasks are admitted based on the estimation of periodical behavior of prioritized task groups and service times. Theoretical proof and simulation results demonstrate that the PACERS algorithm is able to provide assurance of response time and throughput performance for each priority group under various workload situations. The algorithm can be expanded for web server session control if the session/connection establishment requests are assigned lower priority to protect existing sessions in the server. Estimation of the average response time of the server system can be used to determine the appropriate queue space, thus can be used for capacity planning.

## 9. REFERENCES


[1] Apache server project. http://www.apache.org.

[2] Differentiated services (diffserv). http://www.ietf.org/html.charters/diffserv-charter.html.

[3] *Integrated Services (intserv).* http://www.ietf.org/html.charters/intserv-charter.html.

[4] *Usability Engineering.* Academic Press, 1993.

[5] T. F. Abdelzaher and N. Bhatti. Web server qos management by adaptive content delivery. In *IEEE Infocom*, 2000. http://www.ieee-infocom.org/2000/papers.

[6] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated quality-of-service in web hosting services. In *1998 Workshop on Internet Server Performance*, June 1998. http://www.cs.wisc.edu/~cao/publications.html.

[7] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the www. In *In Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, December 1996. IEEE.

[8] G. Alwang. Web servers benchmark tests. *PC Magazine*, May 1998.

[9] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, pages 64–71, September/October 1999.

[10] E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes. Capacity planning with phased workloads. In *Proc. WOSP'98*, Santa Fe, NM, October 1998. ACM. www.hpl.hp.com/research/itc/csl/ssp/papers/index.html.

[11] X. Chen and P. Mohapatra. Lifetime behavior and its impact on web caching. In *Proceedings of the IEEE Workshop on Internet Applications (WIAPP'99)*, San Jose, CA, July 1999. dlib.computer.org/conferen/wiapp/0197/pdf/01970054.pdf.

[12] X. Chen and P. Mohapatra. Providing differentiated service from an internet server. In *Proceedings of IEEE Internet Conference on Computer Communications and Networks (ICCCN'99)*, Boston, MA, October 1999.

[13] X. Chen and P. Mohapatra. Service differentiating internet servers. submitted to IEEE Transaction on Computers, 2000.

[14] M. E. 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.

[15] L. Eggert and J. Heidemann. Application-level differentiated services for web servers. *In World Wide Web Journal*, 3(2):133–142, 1999.

[16] V. S. Frost and B. Melamed. Traffic modeling for telecommunications networks. *IEEE Communications Magazine*, 32(3):70–81, March 1994.

[17] M. Harchol-Balter, M. E. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. In *Proceedings of Performance Tools '98, Lecture Notes in Computer Science*, volume 1469, pages 231–242, 1998.

[18] A. K. Iyengar, E. MacNair, and T. Nguyen. An analysis web server performance. In *Proceedings of the IEEE 1997 Global Telecommunications Conference (GLOBECOM '97)*, Phoeniz, AZ, November 1997.

[19] A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, pages 85–100, 1999.

[20] L. Kleinrock. *Queueing Systems.* John Wiley & Sons, 1976.

[21] T. T. Kwan, R. E. McGrath, and D. A. Reed. User access patterns to ncsa's world wide web server. CS Tech Report UIUCDCS-R-95-1934, University of Illinois at Urbana-Champaign, February 1995. ftp://ftp.cs.uiuc.edu/pub/dept/tech_reports/1995/.

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

[23] J. C. Mogul. Network behavior of a busy web server and its clients. Technical Report Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, October 1995.

[24] R. Morris and D. Lin. Variance of aggregated web traffic. In *IEEE Infocom*, 2000. http://www.ieee-infocom.org/2000/papers.

[25] R. Pandey, J. F. Barnes, and R. Olsson. Supporting Quality Of Service in HTTP Servers. In *Proceedings of the Seventeenth Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 247–256, Puerto Vallarta, Mexico, June 1998. ACM.

[26] K. Sohraby and M. Sidi. On the performance of bursty and correlated sources subject to leaky bucket rate-based access control schemes. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, pages 426–434, Bal Harbour, Florida, April 1991.

[27] T. Wilson. E-biz bucks lost under ssl strain. *Internet Week Online*, May 20 1999. http://www.internetwk.com/lead/lead052099.htm.