Optimal Time-variant Resource Allocation for Internet Servers With Delay Constraints

Xiliang Zhong, Cheng-Zhong Xu, Minghua Xu
Department of Electrical & Computer Engg.
Wayne State University, Detroit, Michigan 48202
{xlzhong, czxu, mxu}@wayne.edu

Abstract

The increasing popularity of high-volume performance-critical Internet applications calls for a scalable server design that allows meeting individual response-time guarantees. Considering the fact that most Internet applications can tolerate a small percentage of deadline misses, we define delay constraint as a statistical guarantee so as to relax server resource requirements. A recent decay function model characterizes the relationship between the request delay constraint, deadline miss rate, and server capacity in a transfer function based filter system. This paper extends the model and develops a time-variant scheduling policy that minimizes system load variances and capacity requirement. The scheduler assumes no priori knowledge about the input request distribution and correlation structure. The resultant server capacity bound is further tightened by utilizing the information of request arrival distribution. Simulation results validate the extended decay function model and show the superiority of the scheduler in comparison with other scheduling algorithms.

I. Introduction

Resource configuration and allocation for an Internet server has long been an active research area. It is important to design an Internet server with appropriate capacity that can allocate sufficient resources to different applications for a certain level of Quality of Service (QoS). Plentiful research work has been conducted using queueing network models. The classic queue theory is based on an assumption of input renewal or Markovian processes. However, recent measurements of traffic all point out the presence of long-range dependence (LRD) and self-similarity: packets on networks, requests on Web servers, and frames from VBR video. They indicate that a Poisson process is insufficient for traffic modeling.

Various models have been suggested to capture these effects, including effective bandwidth based queueing analysis [4] [12] [11], hierarchical renewal processes like Markov Modulated Poisson Process (MMPP) [3] [15], Fractional Gaussian Noise (FGN) and M/G/∞ (see [25] and references therein). However, most queueing analysis involving LRD and heavy-tailed distributions only provides asymptotic results. Albeit insightful, they only give partial information on system performance when buffer sizes or service time are sufficiently large.

In this work, we take a different prospective motivated from the fact that Internet applications or requests are often delay-sensitive. It means their service time cannot be arbitrarily long. For example, in audio and video conference, the response would be meaningless if it could not be produced within a sufficiently small amount of time. Servers that support multiple classes of clients may specify multiple levels of delay guarantees so that a high-paying customer may experience a short delay. We assume each request is associated with a delay constraint determined once the request is admitted according to a predefined service-level agreement (SLA). While the timeliness requirement is crucial to delay-sensitive applications, in many cases a small percentage of deadline misses can be tolerated without a significantly adverse effect on service quality. Instead of providing hard guarantee, we relax the requirement and provide a statistical guarantee by a deadline miss probability. We refer to the percentage of request deadline misses as overload probability in the context of server capacity planning. A small overload probability can significantly reduce the requirement for server capacity without over-compromising the quality of service [5].

Scheduling tasks according to their delay constraints has been extensively investigated in the real-time community. Abdelzaher and Lu, et al. developed feedback control approaches to deal with the impact of high order moments of Internet traffic [1] [17]. They treated requests as aperiodic real-time tasks with time constraints and pro-
vided absolute delay guarantee. Although the approaches ensure the robustness of resource allocation, the impact of requests scheduling on server performance under various input processes was left to be investigated.

Instead of providing hard delay guarantee, research in real-time systems often assumes deadline misses can be tolerated with a particular constraint. For example, Bernat, et al. proposed weakly hard real-time systems in which the deadlines missed during a window of time were precisely bounded [5]. Similar models were studied in [10] [23] [27] [18]. However, their approaches are not readily applicable to requests scheduling in Internet servers because they focus more on periodic or repeating tasks and are most effective for packet streams scheduling.

In [26], we investigated the relationship between resource allocation and delay constraints by constructing a decay function model for requests scheduling. It assumes a general request arrival process combined with a general request size distribution. Each request is scheduled by a decay function which determines how much resource is allocated at each scheduling epoch. The server load changes with the resource allocation. The model reveals a relationship between delay constraints, server capacity, and server overload probability and facilitates the derivation of an optimal time-invariant scheduling strategy that minimizes server load variance. The impact of high order moments of the request processes on the resource requirement is minimized for an important class of delay-constrained time-invariant scheduling.

In this paper, we extend the decay function model in three aspects. First, we extend the scheduler to be time-variant. By allowing the scheduler to be adaptive according to input traffic dynamics, we can further reduce the server load variance and hence the requirements for server resource. Furthermore, we propose an optimal time-variant scheme that minimizes load variance with guaranteed delay constraints.

Second, we provide a robust resource allocation function that assumes no knowledge of input correlation structure and distribution. In practice, such knowledge is not always available or accurate enough. A scheduler based on such knowledge would behave poorly in the presence of correlation structure mismatch. The proposed resource allocation scheme performs based on the current and past request arrivals.

Third, we show that the decay function model can be extended to support requests with different delay constraints instead of restricting all requests with the same constraint. In addition, we derive an optimal time-invariant scheduling policy in a closed form expression.

The rest of the paper is organized as follows. In Section II, we introduce the decay function model and its extension to support requests with different delay constraints. In Section III, we provide a time-variant scheduler that minimizes server capacity requirement under statistical delay guarantee. Special cases to the model are discussed in Section IV. Section V shows simulation results of the schemes. Section VI reviews related work. Section VII concludes the article with remarks on future work.

II. Extended Decay Function Model

In this section, we show that the decay function model is readily to support inputs with different delay constraints. Furthermore, we derive optimal time-invariant scheduler for each class of traffic in closed-form expressions.

A. The Decay Function model

Consider an Internet server that takes requests as an input process, passes them through scheduling and generates a system load output process. Each incoming request needs to consume a certain amount of resource, which is defined as the request size. An Internet request often needs to consume more than one type of resource, for example CPU time, memory space, and network-IO bandwidth. Since the server scalability is often limited by a bottleneck resource class, we focus on the management of the single critical resource. The request size is a generic index and can be interpreted as different measures with respect to different types of resources. It is different from the size of its target file. In certain cases, such as FTP and static contents Web servers, the request size is proportional to the file size. In a QoS-aware system, the resource requirements of a request can be determined from the quality indices in different QoS dimensions [14].

As shown in Figure 1, the decay function model consists of three key components: request incoming process, scheduling, and system load process. The system is a discrete time model, with \( t \) as server scheduling epoch index. The request arrivals are modeled as a general fluid typed process \( \{n(1), n(2), n(3), ..., n(t), ...\} \), where \( n(t) \) is the number of requests arrived between time \( t-1 \) and \( t \). Taking the request size into the model, the input process can be written as: \( \{w_i(t)\}_{i=1,2,...,n(t)} \) \( t=0,1,... \), where \( w_i(t) \) is the size of the \( i \)th request that arrives from time \( t-1 \) to \( t \). The compound input process \( \bar{w}(t) \) is the sum of \( n(t) \) input requests at time \( t \).

Assume each request has the same delay constraint \( t_d \). The system is modeled as a linear time-invariant low pass filter for delay bounded requests scheduling. The scheduling function \( h(t) \) is formulated as the transfer function of the system. Let \( l(t) \) denote the system load. The decay function model formulates the system load as...
When the input process is not independent, it is characterized by a correlation structure. The coefficients are no longer uniformly distributed and determined by solving the optimization problem.

After the load variance is determined, server capacity can be estimated without assuming input distributions. Generally speaking, a smaller load variance leads to a smaller upper bound of the capacity requirement. Due to the burstiness of Internet traffic, the goal of resource allocation is to find a scheme that yields the minimum load variance and at the same time the minimum amount of resource requirement.

Let \( v = \text{prob}(l > c) \) be a pre-defined probability that system load \( l \) exceeds the server capacity \( c \). Denote \( \mu_l \) as the mean server load, \( \sigma_l^2 \) as the variance. An estimate of capacity is [26]

\[
c = \sqrt{\frac{1 - v}{v}} \cdot \sigma_l + \mu_l.
\]

Note that the bound is derived without any assumptions regarding input arrival distributions. Thus it is only a loose upper capacity bound. More accurate estimates are possible depending on different underlying distributions. Consider the fact the input requests are unpredictable and may not necessarily follow a certain distribution, this bound is useful. It provides a uniform solution that applies to all types of input with finite first and second order moments.

B. Extension for requests with different delay constraints

Let \( t_d \) denote the supreme of the constraints of all the requests. We make \( t_d \) separate queues which contain requests with remaining delays 1, 2, ..., \( t_d \). All requests with delay constraints \( j \) are put into queue \( j \). Scheduling is conducted in all the \( t_d \) running queues. Requests in each queue are given a fractional-capacity on a full-time basis, which is similar to the processor-sharing concept in CPU scheduling. After all the scheduled requests have been served at the current scheduling slot, all requests with remaining delay \( j \) now have delay \( j-1 \). Thus queue 1 becomes empty and queues 2, 3, ..., \( t_d \) become queue 1, 2, ..., \( t_d-1 \). Newly arrived inputs are dispatched into the queues according to their delay constraints. A request entering queue \( j \) at time \( t \) enters queue 1 at time \( t + j - 1 \) and out of the system at the beginning of \( t + j \).

If an input request could result in system overload, this request should not be dispatched. We can either adopt an admission control policy to decline the request or put it into a waiting queue to wait for the next scheduling slot. The choice can be made depending the type of the server. For example, if it is a media-server providing a stream of video frames at a certain rate. A frame that misses its deadline can be simply dropped because the application is very delay-sensitive. One can still tolerate a few missed deadlines without a significant degradation in video quality. For a Web contents server providing web contents, since customers may tolerate a little bit longer
delays, a request that would miss its deadline can be put into the waiting queue temporarily.

Applying the decay function model to queue $j$, the load $l_j(t)$ becomes the output of aggregated inputs $\hat{w}_j(t)$ passing through a system with resource allocation function $h_j(t)$. Define $h_j' = [h_{j0}, h_{j1}, \ldots, h_{j(j-1)}]$ and $w_j(t) = [\hat{w}_j(t), \hat{w}_j(t-1), \ldots, \hat{w}_j(t - j + 1)]$. The load at queue $j$ can be expressed as

$$l_j(t) = \hat{w}_j(t) * h_j(t) = \sum_{i=0}^{j-1} h_{ji}\hat{w}_j(t-i) = h_j'w_j(t)$$

subject to $\sum_{i=0}^{j-1} h_{ji} = 1$ and $h_{ji} \geq 0$, \hspace{1cm} (5)

where $h_{ji}$ means the proportion of resource allocated in queue $j$ to the request $\hat{w}_j(t)$ at time slot $t+i$.

The system load at time $t$, $l(t)$, is simply the sum of load over all the queues. Assume the load in each queue is independent, the variance of system load at time $t$ over all the queues. Assume the load in each queue can be expressed as $

\begin{align*}
\text{Var}[l(t)] &= \text{Var}[\sum_{j=1}^{t_d} l_j(t)] = \sum_{j=1}^{t_d} \text{Var}[l_j(t)] \\
&= \sum_{j=1}^{t_d} E[l_j^2(t)] - \sum_{j=1}^{t_d} (E[l_j(t)])^2.
\end{align*}$

Assume the input process to queue $j$ is WSS. Then $E[l_j(t)] = E[\hat{w}_j]$ is a constant. According to (5), we have

$$E[l_j^2(t)] = E[h_j'w_j(t)h_j'w_j(t)] = h_j'\Omega_jh_j.$$ \hspace{1cm} (6)

We hence have the following result.

**Theorem 2.2** The optimization of server load variance due to a time-invariant decay function for input with different delay constraints is to

Minimize $\sum_{j=1}^{t_d} h_j'\Omega_jh_j$

subject to $c_jh_j = 1$ and $h_{ji} \geq 0$, \hspace{1cm} (7)

where $c_j$ is a row vector with $j$ components all equal to 1.

This is a typical minimization problem subject to a linear constraint and can be solved using Lagrange multipliers. We first ignore the non-negativity constraint and consider the $j$th item in the Lagrangian,

$$L(h_j, \lambda_j) = h_j'\Omega_jh_j + \lambda_j(c_jh_j - 1).$$

Its gradient is $2\Omega_jh_j + \lambda_jc_j'$ with a solution $h_j^* = -(\frac{\lambda_jc_j'}{2\Omega_j})$. To find the value of the Lagrange multiplier, the solution must satisfy the constraint. Imposing the constraint, the solution can be expressed as

$$h_j^* = \frac{\Omega_j^{-1}c_j'}{c_j\Omega_j^{-1}c_j'}.$$ \hspace{1cm} (8)

A special case is when the input arrivals to the $j$th queue is independent, the covariance matrix becomes diagonal matrix and the resource allocation function is uniform with values $1/j$.

Note that the solution must be positive to satisfy the non-negativity constraint of (7). As $\Omega_j$ is a symmetric positive definite matrix, the denominator of the solution is always positive. However, the numerator is not necessarily positive. Since $c_j$ is a constant for a given queue, the covariance matrix determines the sign of the numerator. There exists some covariance matrix that leads to negative solution. We show by simulation in Section V-A that the solution gives negative values only in extreme cases for synthetic input. The equation (8) provides an extremely simple solution for the resource allocation function for real world traffic. In case it produces a negative value, the solution can still be retrieved by solving the optimization problem in (7) directly.

**III. Optimal Delay-bounded Scheduling**

The resource allocation function discussed in the preceding section is optimal only in the set of feasible functions that do not change during the scheduling process. The load variance can be further reduced if $h(t)$ can be adapted according to input traffic in each scheduling epoch. In this section, we propose a time-variant scheduler and prove it to be optimal in the minimization of server load variance. Define $q_j$ as the backlog of queue $j$. We formulate and solve the optimization problem in the following theorem.

**Theorem 3.1:** The optimal delay bounded scheduling can be achieved by finding a scheduler that

Minimize $L = \sum_{i=0}^{t_d-1} (\sum_{j=i+1}^{t_d} l_j(i))^2$

subject to $l_j(i) > 0, \hspace{1cm} 0 \leq i \leq t_d - 1, \hspace{1cm} 1 \leq j \leq t_d$

$$\sum_{i=0}^{j-1} l_j(i) = q_j, \hspace{1cm} 1 \leq j \leq t_d,$$ \hspace{1cm} (9)

The optimal solution can be achieved by the following process for each queue $j, 1 \leq j \leq t_d$,

$$l_j(i) = (\mu_j - \sum_{k=i+1}^{j-1} l_k(i))^+, \hspace{1cm} 0 \leq i \leq (j-1),$$ \hspace{1cm} (10)

where the notation $(x)^+ = \max(x, 0)$ and $\mu_j$ is determined by evaluating the backlog constraint in (9).

**Proof:** We first show that the optimization solution can be achieved by the optimization of $L$. Recall that for a WSS input process, we have $E[l(t)] = E[\hat{w}]$ and it is a constant. The variance of system load is

$$\text{Var}[l(t)] = E[l^2(t)] - (E[l(t)])^2 = E[l^2(t)] - (E[\hat{w}])^2.$$
Therefore, the optimization problem is to find a scheduler that minimizes the second moment \( E[l^2(t)] \).

At any of the scheduling slot, the scheduler needs to consider the current queue backlog including new arrivals in the last time slot and history arrivals not yet finished. To complete all the requests in the \( t_d \) queues, the second moment it generates can be estimated as

\[
E[l^2(t)] = \frac{1}{t_d} \sum_{i=0}^{t_d-1} l^2(i) = \frac{1}{t_d} \sum_{i=0}^{t_d-1} \left( \sum_{j=i+1}^{t_d} l_j(i) \right)^2. \tag{11}
\]

It is an average of the square of load in the next \( t_d \) time-slots summed over all queues. It means the minimization of the second moment of load can be achieved by minimizing the \( L \) from (9). The solution to the problem gives optimal load for all queues from the current time to the next \((t_d-1)\) time slots. The sum of load from all queues at the current time, \( \sum_{j=1}^{t_d} l_j(0) \), forms the optimal solution for current load. However, for all other time, the solutions are not necessarily optimal if there are new input arrivals after the current time. It is because the arrival may change the queue backlog. In this case, the optimization process needs to be performed again to get the optimal load.

The minimization objective in (9) is a standard constrained optimization problem. It can be solved by using Lagrange Multipliers. For all \( i \) and \( j \), we form the Lagrangian

\[
L(l_j(i), \lambda_j) = \sum_{i=0}^{t_d-1} \left( \sum_{j=i+1}^{t_d} l_j(i) \right)^2 - \lambda_j \left( \sum_{j=0}^{t_d-1} l_j(i) - q_j \right)
\]

and differentiate it with respect to \( l_j(i). \) Applying the Kuhn-Tucker conditions [19], we have

\[
\sum_{j=i+1}^{t_d} l_j(i) - \frac{1}{2} \lambda_j = 0
\]

\[
\sum_{k=i+1}^{t_d} l_k(i) + l_j(i) + \sum_{k=j+1}^{t_d} l_k(i) - \frac{1}{2} \lambda_j = 0
\]

\[
l_j(i) = \frac{1}{2} \lambda_j - \sum_{k=j+1}^{t_d} l_k(i) - \sum_{k=j+1}^{t_d} l_k(i). \tag{12}
\]

The condition in (12) gives the solution for queue \( j \) at time \( i \), depending on values of all other queues. This dependence makes the condition hard to solve. Define the variable \( \mu_j = \sum_{k=j+1}^{t_d} l_k(i) \) for each queue \( j \). The condition (12) becomes

\[
l_j(i) = \mu_j - \sum_{k=j+1}^{t_d} l_k(i).
\]

Consider the fact that the load \( l_j(i) \) is nonnegative. This leads to the condition in (12) with an unknown value \( \mu_j \). The value can be determined by the constraint in (9) for queue backlog \( q_j \). Therefore, we can get the values of \( \mu_j \) and \( l_j(i) \) that satisfy the condition in (12). Note that (12) only gives necessary conditions for the existence and solution of a global minimum. Since the optimization problem we address is a convex function, the conditions are also necessary.

The optimal solution of (9) can be illustrated graphically in Figure 2. For queue \( j \) at time \( i \), starting from \( i = 0 \) for simplicity, the vertical white area indicates the current accumulated load from queue \( i+1 \) to \( j-1 \). As input size increases from zero, the requests are first distributed to the time slot with the lowest accumulated load, denoted by the shaded areas. The first time slot to be distributed is always \( j - 1 \). As the input increases further, parts of the requests will be put into busier slots. Once the load in a queue reaches the level denoted by \( \mu_j \), no more requests will be distributed to that queue in the current scheduling slot. This process in which requests are distributed among different time slots is similar to the way in which water is distributed in a vessel. It was initially referred to as a “water-filling” process when distributing power among a set of parallel Gaussian channels for maximum information capacity [6]. The water-filling process has been applied to various fields. A recent application is the minimization of average transmit power over Gaussian channels where the water-filling process is interpreted as a time-variant low-pass filter [13].

The optimal solution can be fit into the decay function model as time-variant coefficients in (11). The coefficients from current to the next \((j-1)\) time slots for queue \( j \) can be written as:

\[
h_j(t + i) = \frac{(\mu_j - \sum_{k=j+1}^{t_d} l_k(t + i))}{\tilde{\omega}_j(t)}. \tag{13}
\]

When all requests have the same delay constraints \( t_d \), the process only needs to be performed at the queue \( t_d \). Load in other queues remains the same. Since the solution depends on accumulated values of all queues with smaller delays, the complexity of the solution is \( O(t_d) \).

When requests have different delay constraints, the requests may enter different queues instead of only queue
By aggregating input requests with identical delay constraints into to the same queue, the input process becomes less bursty and more controllable. All queues with new arrivals during the last time slot need to follow the solution process to get the new scheduled load. The worst case is when requests arrived in the last time slot have delays ranges from 1 to \( t_d \). Then the complexity of the process is \( O(t_d^3) \).

IV. Special Cases for Tight Capacity Bounds

In this section, we consider two special cases to tight the bound in (4) when more information about the input distribution is assumed.

A. Input in unimodal distributions

We first show that a better bound can be derived if the distribution of the load is unimodal distributed. In [1], the load is modeled as the convolution between input and the resource allocation function. Assume the input distribution is continuous and its density \( f \) is logconcave (i.e., \( \log(f) \) is concave). The logconcave property can be found in several widely used standard distributions such as Poisson distributions, Gaussian distributions. For example, Gaussian distributions are commonly used as the marginal distributions in modeling self-similar process with FGN and F-ARIMA [16]. The set of logconcave distributions are strongly unimodal [8] in the sense that they are closed under convolutions and hence the resulted load is unimodal distributed.

Let \( X \) be a random variable with a unimodal distribution, meaning that for some mode \( m \), its CDF is concave on \([m, \infty)\) and convex on \((-\infty, m)\). From the Vysochanskiï and Petunin inequality [22], we can derive the probability that \( X \) exceeds \( k \):

\[
P(X \geq k) \leq \begin{cases} \frac{4E[X^2]}{9E[X^2]+k^2} & \text{if } k^2 \geq \frac{2}{3}E[X^2], \\ \frac{4E[X^2]}{3E[X^2]+k^2} - \frac{1}{3} & \text{if } k^2 \leq \frac{2}{3}E[X^2]. \end{cases}
\]

By using (14), the probability that system load \( l \) exceeds capacity \( c \) becomes

\[
P(l \geq c) = P(l - \mu_l \geq c - \mu_l) \leq \begin{cases} \frac{4c^2}{9\sigma_l^2 + (c - \mu_l)^2} & \text{if } c \geq \sqrt{\frac{2}{3}}\sigma_l + \mu_l, \\ \frac{4c^2}{3\sigma_l^2 + (c - \mu_l)^2} - \frac{1}{3} & \text{if } c \leq \sqrt{\frac{2}{3}}\sigma_l + \mu_l. \end{cases}
\]

Recall that the probability \( v \) should be well below 1/6 for an Internet server to provide acceptable performance. The first bound is thus more useful. In both cases, the capacities are much tightened in comparison with the bound from (4).

B. Independent input

We next consider the case when the number of arrival requests at each time and their sizes are i.i.d. random variables. For independent input, the load distributions can be determined according to the input distributions and delay constraints. The implication of deriving the distribution is a sharp capacity bound can be calculated directly by considering the system load tail distribution.

Let the size of each request \( w_1(t), w_2(t), \ldots \) at time \( t \) be mutually independent random variables with a common distribution \( f(w) \). Recall that \( n(t) \) is the number of requests arrived from time \( t-1 \) to \( t \). The aggregated input \( \hat{w}(t) = \sum_{i=1}^{n(t)} w_i(t) \) has the density \( f(w)^{n(t)} \), namely the \( n(t) \)-fold convolution of \( f(w) \) with itself. Denote \( L(t) \), \( N(t) \), \( W(t) \), and \( \hat{w}(t) \) as the random variables of \( l(t), n(t), w(t), \) and \( \hat{w}(t) \). Let \( P(N(t) = n(t)) = p_n \). The PDF of the resulting random sums is the density of the aggregated input \( W(t) \) [9]

\[
f(\hat{w}(t)) = \sum_{n=1}^{\infty} p_n f^{n(t)}(w(t)).
\]

By using (3), the distribution of the load at time \( t \) is

\[
f(l(t)) = \frac{1}{t_d} f^{t_d*}(\hat{w}(t)) = \frac{1}{t_d} \sum_{n=1}^{\infty} p_n f^{n(t)}(\hat{w}(t))^{t_d*}.
\]

A simpler approach to get the output distribution is to use characteristic functions. Let \( \Phi_{W(t)}(\omega) \) be the characteristic function corresponding to the random variable \( W(t) \) and \( G_{N(t)}(z) \) be the generating function of \( N(t) \). For distributions that the functions exist, the characteristic function of the aggregated input \( \hat{W}(t) \) can be retrieved as [9]

\[
\Phi_{\hat{W}(t)}(\omega) = E[z^{N(t)}(t)]_{z = \Phi_{W(t)}(\omega)} = G_{N(t)}(\Phi_{W(t)}(\omega)).
\]

That is, the characteristic function of \( \hat{W}(t) \) can be found by evaluating the generating function of \( N(t) \) at \( z = \Phi_{W(t)}(\omega) \).

From (4), the characteristic function of load \( L(t) \) at time \( t \) can be derived as

\[
\Phi_{L(t)}(\omega) = (\Phi_{\hat{W}(t)}(\frac{\omega}{t_d}))^{t_d} = (G_{N(t)}(\Phi_{W(t)}(\omega)))^{t_d}.
\]

The PDF of the output load at time \( t \) is found by taking the inverse transform of the above expression.
V. Simulation Results

In this section, we verify the previous analysis and evaluate the proposed scheduler through simulation. The first objective was to verify the effectiveness of the proposed scheduler in minimizing server load variance. When using the time-invariant scheduler, we assume the input correlation structure is known. No assumption about the input is made when using the time-variant scheduler. We also investigated the bound tightness due to both general and special cases.

A. Minimization of server load variance

The first experiment was conducted based on traces from the 1998 World Soccer Cup server logs (WorldCupTrace) [2]. To show performance of the schedulers under traffic with different burstiness, we chose two days logs: day 17 and day 27. Their Hurst parameters are 0.9 and 0.72 respectively calculated by using the variance-time plot method [7]. The Hurst parameter $H$ is used to denote the traffic burstiness. The higher is the Hurst parameter, the greater is the burstiness of the traffic. It means the day 17 trace is more bursty. To determine the request size of each request, we first retrieve the file size of each HTTP request and assume a proportional relationship between the file size and actual request size. Each request with a file size of 1K bytes is assumed to require one unit of resource under consideration.

Figure 3 shows the workload variance with different time constraints for day 27 ($H = 0.72$). We assume each request has the same delay constraint for simplicity. We observe that both schedulers are effective in reducing the impact of input burstiness to $1/t_d$ of the original variance. The variance due to the optimal time-variant scheduler is consistently smaller. For example, when $t_d$ is 10 time units, the time-variant scheduler reduces the variance 20% more than the time-invariant scheduler. We can also observe that as the delay constraints increase, the variances given by both methods decrease. This is expected because the schedulers have more time slots to smooth the input requests so the load variance becomes smaller. With the increase of delay constraint, the time-variant process has more time slots to adapt resource allocation according to the inputs so as to achieve smaller load variance. As a result, it can benefit slightly more from the increase of delay constraints.

Figure 3 also shows the server load variance for the trace of day 17 ($H = 0.9$). We observe that as the traffic becomes more bursty, the benefit by time-variant decay function become less obvious. Intuitively, the time-variant decay function achieves the optimal bound by adapting resource allocation according to the input requests to minimize load variance. When the requests get more bursty, the requests may cause sudden resource demands on the server. In this case an adaptation of resource allocation becomes less effective in keeping the load in small variance. We also observe from the figure that for the trace of day 17, the system load variance decreases slower with the increase of delay constraint than that of day 27. This can be verified from the fact that the presence of self-similarity makes it less effective to smooth the input with longer delay.

We also examined the system load based on a synthetic inputs from FGN processes with different Hurst parameters settings. The load variance and relative performance of the two schedulers show the same trend. Please refer to [28] for more details.

It is interesting to compare the resource allocation functions of the two types schedulers. In the time-variant scheduler, the coefficients of the resource allocation function are adaptive according to the input. For comparison, we record the coefficients by all input requests and take the average to see the overall trend of resource allocation. The solution for the time-invariant function is attained by (8). We tried the same two traces as previous experiment and fixed the delay constraints to be 10. Figure 4 shows the results. The resource allocation denotes the percentage of resource in each time slot. It can be observed that the two functions are very different. In the time-invariant function, $h(t)$ is relatively stable with each value being around 1/10. While in the time-variant scheduling, the $h(t)$ increases slowly in the beginning and abruptly in the last time slot. This shows the scheduler attempts to postpone the resource allocation as long as possible subject to the delay constraint. An intuitive explanation for this is as the deadline approaches, a request should have a higher priority to be processed so as to improve its chance of meeting deadlines. This can also be verified by (13) which shows that with a certain input, the coefficients are proportional to the predicted load at queue $j$ in the next $i$ time slots. With the increase of $i$, system load from less number of queues is accumulated, from queue $i$ to $j - 1$. Therefore, the average coefficients show a trend of increase with time slot $i$. As time slot $i$ becomes $j - 1$, system load becomes the maximum $\mu_j$ and the coefficient also reaches its maximum $\mu_j/\hat{w}(t)$.

We also observe that as the traffic burstiness varies, there is only subtle change in the optimal time-variant function. In the time-invariant case, the minimum value of the function becomes smaller. We tried traces from other days in the WorldCupTrace and another real scene-based MPEG-I video trace from the popular Simpson cartoon clip [20]. We found for all real traces the solution was always positive. We only got negative values for FGN process with Hurst parameter larger than 0.999, when the input is extremely correlated.
B. Estimation of server capacity

In this experiment, we estimate the capacity required by the two types of scheduling in order to guarantee a specified delay constraint according to \( a \). Figure \( 5 \) shows the capacity bounds with different server overload probability. We fixed the delay constraint as 10 time units and used the trace at day 27. It is expected that the capacity requirement becomes higher with lower deadline miss rate, characterized by a lower overload probability. A close look at the data also tells that with a lower overload probability, the time-variant decay function provides a little bit tighter capacity bound. This can be explained by the fact that the time-variant scheduling is best at minimizing load variance. With the decrease of \( v \), more resource is required. According to \( 4 \) and the fact the load mean is a constant, the variance plays a more important role in determining the capacity bound. As a result, the time-variant decay function provides a better bound. However, the difference is only about 3\% as \( v \) varies from 0.1 to 0.01 and it is hard to be observed from the figure.

We show and compare the effectiveness of the bounds for general input \( 4 \), tight bounds for unimodal input \( 15 \) in Figure \( 6 \). We assumed the number of request arrivals at each time conforms lognormal distribution \( ln(1, 1.5) \), the size of request fits normal distribution \( n(5000, 1000) \). We get consistent results using the time-variant and the time-invariant decay functions with previous experiments. We therefore only present the results of time-invariant decay function. The normal distribution is strongly unimodal and the marginal distribution of the aggregated input process is strongly unimodal, as well. The input is slightly correlated due to the default random number generator in use and the correlation structure is used to derive the coefficients in the decay function which are not exactly uniformly distributed. The system load is therefore a convolution between the aggregated inputs; the load distribution is guaranteed to be unimodal distributed. We can then apply the bound in \( 15 \).

Figure \( 6 \) also includes bounds retrieved by approximating the tail distribution of the system load through simulation. From the figure, it is expected that the general bounds are loose. By use of the unimodal density, we reduce the bounds to 72\% of the general bounds.

C. Comparison with other scheduling strategies

Using the same set of parameters as of last experiment, we compared both the optimal time-invariant and time-variant decay function scheduling with other schedulers. We set overload probability as 1\% and delay constraint as 10. To compare the scheduling by a tight capacity configuration, we set the capacity to 171000 as shown from Figure \( 6 \).

The server was simulated with an infinite queue so that we can find exact distribution for the completion time of each request. The simulated server treats all requests with the same priority. When a request arrives at the server, it is first buffered in the queue. If the server has enough resource, the request will be scheduled for processing immediately without any queueing delay. Otherwise, the request will be blocked in the queue. We implemented two other different schedulers on the server module. One
TABLE I. Statistics of server with various scheduling algorithms

<table>
<thead>
<tr>
<th>Scheduling</th>
<th>GPS</th>
<th>Random</th>
<th>Time-invar.</th>
<th>Time-var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Mean</td>
<td>37534</td>
<td>37534</td>
<td>37534</td>
<td>37534</td>
</tr>
<tr>
<td>Load Var. (10^6)</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>Capacity Satisf.</td>
<td>99.6%</td>
<td>99.6%</td>
<td>99.6%</td>
<td>99.6%</td>
</tr>
<tr>
<td>Time Mean</td>
<td>10.04</td>
<td>10.04</td>
<td>10.04</td>
<td>10.04</td>
</tr>
<tr>
<td>Time Var.</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
<td>0.19</td>
</tr>
</tbody>
</table>

is GPS scheduling which ensures fair-sharing of resource between requests in processing at any scheduling epoch. In addition, we also implemented a heuristic time-variant randomized allocation algorithm. It takes $t_d$ random numbers from a uniform distribution and then scales them to standard decay functions $h(t)$ so that $\sum_{i=1}^{t_d} h(i) = 1$. Table I shows the comparative results in terms of load mean and variance, completion time, and server capacity satisfactory rate. Completion time of a request includes waiting time in the backlogged queue, plus its processing time. Capacity satisfactory rate is the percentage of server workload less than the pre-defined capacity.

From the table, it can be observed that the optimal time-variant scheduling outperforms the others in two aspects. First, it dramatically reduces the variance of server load by up to 42% and 30% in comparison with GPS and random scheduling, respectively. Second, it provides better guaranteed completion time. GPS scheduling leads to a lower mean completion time and a high variance. Both the optimal schedulers guarantee completion time and yield much smaller time variance comparison with the other schedulers. In agreement with the findings in previous evaluation, the table also shows that the optimal time-variant scheduler consistently outperforms the time-invariant one.

VI. Related Work

Many models have been proposed to capture the impact of LRD and heavy-tailed distributions. Effective bandwidth is one of the most influential framework for service provisioning [4] [12] [11]. It is based on an assumption that the Internet traffic process can be bounded by an envelope process in a much simpler mathematical representation. The envelope process is then studied under resource reservation based scheduling policies. Effective bandwidth expressions have been derived for many traffic types including those with self-similarity [12]. However, the precision of the bounds heavily depends on the burstiness of underlying traffic. The MMPP [3] [15] based queueing models can include correlation statistics and other second order moments into Markov chain transitive matrices. But the exact solution of the advanced model remains open. Fractional Gaussian Noise (FGN) and $M/G/\infty$ (see [25] and references therein) are widely used to model Internet traffic. Most of the work focused on the dependence structure and assumes deterministic service times. There were recent researches on the impact of LRD arrival process on waiting time in $M/G/\infty$ and FGN typed queues [25]. The problem common to most queueing results involving LRD and heavy-tailed distributions is they only provide asymptotic results, which may not hold when buffer sizes or service time are small.

Unlike queueing model approach, our approach is based on a decay function model. We associate each request with a soft service deadline and model the scheduling function as a filter. Optimal scheduling algorithms are derived by optimality analysis of the set of feasible scheduling algorithms. The decay function can also be adapted to meet changing resource availability and the adaptation does not affect the solvability of the model.

Scheduling tasks according to their delay constraints has been extensively investigated in the real-time community. Abdelzaher, et al. treated Internet requests as aperiodic real-time tasks with arbitrary arrival times, computation times, and relative deadlines [1]. It was proven that such a group of tasks scheduled by a deadline-monotonic policy will always meet their deadline constraints as long as the server utilization is less than 0.58. The authors applied linear feedback control theories to admit an appropriate number of requests so as to maintain system utilization at the upper bound. This approach assumes a fixed-priority scheduling policy and considers no information about input request processes. When the residual errors are too big, the linear controller would lead to a poor controllability for a non-linear system. For a robust control in request admission, Lu, et al. recently proposed to integrate a queueing model with feedback control for absolute and relative request delay guarantees in Web servers [17]. The server load mean was first estimated according to the first moment statistics of input request processes. Then a feedback controller was used to deal with the impact of high order moments of Internet traffic, in particular high variability and auto-correlations in an algorithmic manner. Although the approaches ensure the robustness of resource allocation, the impact of requests scheduling on server performance under various input request processes is yet to be studied.

Instead of providing hard guarantee to requests with deadline, researches in the real-time systems generally assume deadlines misses can be tolerated with a particular constraint. An example is weakly hard real-time systems in which the distribution of missed deadlines during a window of time were precisely bounded [5]. Hamdaoui et al. introduced the notion of $(m,k)$-firm deadlines for applications to provide statistical guarantee for $m$ deadlines in any $k$ consecutive invocations [10]. They applied this concept in the context of scheduling packets streams
or periodic tasks. They presented a distance based priority (DBP) algorithm to raise the priority of a class of messages if it is likely that m messages of the class out of k consecutive messages fail to meet their deadlines. Later, the model was relaxed and extended in [23] [27] [18]. For example, West and Zhang, et al. discussed a Windows-Constrained model in which the windows did not overlap [23] [27]. In spite of their fruitful research, their results cannot be directly applied to the resource allocation problem in a general Internet server. It is because most of their work targets at periodic CPU tasks while service requests on an Internet server, in general, arrive in a completely aperiodic fashion. Furthermore, their scheduling algorithms work most effectively for packet streams, where package size is relatively stable or can be bounded by a constant. West, et al. chose the scheduling time slot in a way such that the largest packet in any stream could be finished [23]. Mok, et al. assumed all jobs in a task had unit-size execution time [18]. In a general Internet server, the request sizes can be modeled as heavy-tailed distributions [7] and cannot be assumed to finish in a time slot.

VII. Conclusion

This paper studied the resource configuration and allocation problem with emphasis on statistical delay constraints. We extended a time-invariant decay function model with support for different delay constraints. Our main results are in the extension of the model to support time-variant scheduling so as to minimize server load variance with the delay constraints. The load variance can be used to estimate the capacity bound together with a server overload probability. The scheduler makes decision only depending on the past arrivals of input requests and current queue status without assumption of any prior knowledge of input request size distribution and correlation structure. It shows a time-variant structure that is adaptive to input requests. The solution is proved to be optimal and generate smaller load variance than the time-invariant decay function. Simulation results with various input process verified our analytical results.

We restricted this paper to only single bottle-neck resource configuration and allocation with single QoS dimension, the delay constraint. We will explore the opportunity to extending the model to the case of multiple QoS dimensions.

References


