An Integrated Approach with Feedback Control for Robust Proportional Responsiveness Differentiation on Web Servers

Xiaobo Zhou Yu Cai
Department of Computer Science
University of Colorado at Colorado Springs
1420 Austin Bluffs Parkway
Colorado Spring, CO 80933-7150
{zbo, ycai}@cs.uccs.edu

Abstract

There is a growing demand for provisioning of proportional responsiveness differentiation to various clients on scalable Web servers to meet changing resource availability and to satisfy different client requirements. We presented a queueing-theoretical processing rate allocation scheme. It provides different processing rates to requests of different client classes so as to achieve the differentiation objective. At application level, process is used as the resource allocation principal for achieving processing rates on Apache Web servers. However, an application-level implementation shows weak proportionality with large variance because it does not have fine-grained control over the consumption of resources that the kernel consumes and hence the processing rate is not strictly proportional to the number of processes allocated. We design a feedback controller and integrate it with the queueing-theoretical approach. The integrated approach allocates a certain number of processes to handle requests of different client classes according to the queueing-theoretical process allocation approach. The process allocations are then adjusted according to the difference between the target response time and the achieved response time by using the proportional integral derivative (PID) control. Experimental results demonstrate that this integrated application-level approach can enable Web servers to provide robust proportional response time differentiation. This practical approach can also be easily deployed on Apache Web servers.

Keywords: Quality of Service (QoS), Proportional Differentiation, Process Allocation, Feedback Control, Apache Web Server

1 Introduction

Due to the open and dynamics nature of Internet applications, the last decade has witnessed an increasing demand for provisioning of different levels of quality of service (QoS) to meet changing system configuration and resource availability and satisfy different client requirements. This differentiated QoS provisioning problem was first formulated by the Internet Engineering Task Force in the network core. Differentiated Services (DiffServ) [7] is a major architecture, where the network traffic is divided into a number of classes. It aims to define configurable types of packet forwarding in network core routers, which can provide per-hop differentiated services to per-class aggregates of network traffic. The proportional differentiation model [11] states that certain class QoS metrics should be proportional to their pre-specified differentiation weights, independent of the class loads. Due to its inherent differentiation predictability and proportionality fairness, the model has been accepted as an important DiffServ model and been applied in the proportional queueing-delay differentiation (PDD) in packet scheduling [11, 12, 20, 25, 24] and proportional loss differentiation in packet dropping [16].

There are recent efforts on DiffServ provisioning on end servers [1, 2, 6, 9, 10, 14, 19, 23, 27, 29]. On the server side, response time is a fundamental performance metric. Existing response time differentiation strategies are mostly based on priority scheduling in combination with admission control and content adaptation [1, 2, 6, 9, 10]. The authors in [10] adopted strict priority scheduling strategies to achieve response time differentiation on Internet servers. The results showed that the differentiation can be achieved with requests of higher priority classes receiving lower response time than requests of lower priority classes. However, this kind of strategies cannot quantitatively control the quality spacings among different classes. Time-dependent priority scheduling algorithms developed for PDD provisioning in packet networks can be tailored for PDD provisioning on Web servers [19]. However, they are not applicable for response time differentiation because the response time is not only dependent on a job's queueing delay but also on its service time, which varies significantly depending on the requested services. Therefore, providing proportional response time differentiation on Web servers is not only important, but also challenging.

In [27], we proposed queueing-theoretical processing rate allocation strategies for server-side DiffServ provisioning with respect to slowdown, the ratio of a request's queueing delay to its service time. While the simulation results meet expectations, a challenging implementation issue is, how to practically achieve the processing rate for various traffic classes on servers. In [26], we presented a processing rate allocation scheme based on queueing theory for proportional response time differentiation. We then designed and implemented an adaptive process allocation strategy on an Apache Web server to achieve the processing rates allocated to the request classes. Figure 1 shows the experimental results. The upper line is the 95th percentile; the bar

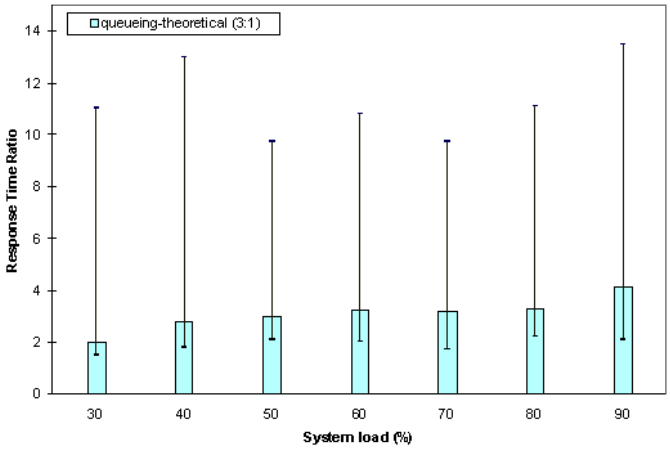


Figure 1: Achieved response time ratio due to the queueing theoretical process allocation approach.

is the mean; and the lower line is the 5th percentile. We can observe that the proportionality is weak and in particular the variance is huge. For example, when the system load is 40%, the achieved average response time ratio is about 2.75 while the target ratio is 3.0. Furthermore, the 5th percentile is 1.8 while the 95th percentile is 12.9. When the workload is close to the system capacity, say 90%, the achieved average response time ratio 4.13 is far away from the target ratio while the 5th percentile is 2.1 and the 95th percentile is 13.5. These large and frequent changes of response time may frustrate end users.

There are three reasons for the weak proportionality and huge variance. First, the workload of traffic classes is stochastic and bursty. It thus is difficult to accurately predict the load conditions based on history information. Especially, the impact of the variance of the interarrival times is significant when the workload is close to the system capacity. Second, the queueing-theoretical processing rate allocation scheme provides no way to control the variance of the response times simultaneously. The third and the most important reason is that the process abstraction serves both as a protection domain and as a resource principal on Apache Web servers. Process is treated as the scheduling entity for an independent activity. It is also the entity for the allocation of resources, such as CPU cycles and memory space. However, resource allocation and scheduling primitives do not extend to the execution of significant parts of kernel code. An application has no control over the consumption of resources that the kernel consumes on behalf of the application. As the result, resource principals do not always coincide with processes (and/or threads). For example, in a network-intensive application, the process does not encompass all of the associated resource consumption since the kernel generally does not control or properly account for resources consumed during the processing of network traffic. Because of the coincidence between protection domain and resource principal, applications lack sufficient control over resource scheduling and management on the servers. This problem makes it difficult to enforce application-level process allocation strategies for proportional response time differentiation on Apache Web servers.

There are efforts on the design of new resource management mechanisms at kernel level to support DiffServ provisioning efficiently, as exemplified by resource containers [5], cluster reserves [4]. Resource container is a new operating system abstraction. It separates the notion of a protection domain from that of a resource principal. A resource container encompasses all system resources that the server uses to perform an independent activity, such as processing a client HTTP request. All user and kernel level processing for an activity is charged to the appropriate resource container and scheduled at the priority of the container. Resource containers allow accurate accounting and scheduling of resources consumed on behalf of a single client request or a class of client requests. Thus, this new mechanism can help provide fine-grained resource management for DiffServ provisioning when combined with an appropriate resource scheduler. However, while kernel-level mechanisms can provide efficient control over resource management, their weaknesses lie on the portability and deployment issues.

In this paper, we seek a practical application-level approach to providing robust proportional response time differentiation. We design an integrated approach based on queueing theory and feedback control theory to achieve more robust proportionality and reduce variance degree. The structure of the paper is as follows. In Section 2, we review other related resource allocation and scheduling disciplines in the DiffServ areas. Section 3 gives the processing rate allocation scheme for proportional response time differentiation. Section 4 presents the design and implementation of the integrated process allocation approach on an Apache Web server. Section 5 gives the experiment results and performance evaluation. Section 6 concludes the paper.

2 Related Work

Providing differentiated services to Internet applications and clients is popular. The proportional differentiation model was proposed in the network core [11]. It was first applied for DiffServ provisioning in packet scheduling and packet dropping, in which packet queueing delay and loss rate are key QoS factors, respectively. Many algorithms have been designed to achieve proportional delay differentiation (PDD) in the network routers. They can be classified into three categories: rate-based; see BPR [11] for example, time-dependent priority based; see WTP [12] and adaptive WTP [20] for examples, and Little's Law-based; see PAD [12] and LAD [24] for examples. The work in [19] demonstrated that some of the algorithms can be tailored for request scheduling for PDD provisioning on the server side. However, the algorithms are not applicable to proportional response time differentiation because response time is not only dependent on a job's queueing delay but also on its service time, which varies significantly depending on the requested services.

Priority-based request scheduling strategies have been investigated for response time differentiation on Web servers [2, 6, 8, 10, 13]. In [10], the authors addressed strict priority scheduling strategies for controlling CPU utilization on Web servers. Incoming requests were categorized into the appropriate queues with different priority levels for the corresponding services. Requests of lower priority classes were only executed if no requests existed in any higher priority classes. The results showed that response time differentiation can be achieved in the sense that higher classes receive less response time than lower classes. However, the quality spacings among different classes cannot be guaranteed by strict priority scheduling. Therefore, this kind of priority-based scheduling strategies cannot achieve proportional response time differentiation on Web servers. Our integrated approach improves over the previous efforts in the sense that it can quantitatively control quality spacings between different classes and provide robust proportional response time differentiation.

In [27], we proposed a processing rate allocation strategy for server-side DiffServ provisioning in terms of slowdown in E-Commerce applications. We left a challenging implementation issue; that is, how to practically achieve the processing rate for various traffic classes on servers. In [29], the authors adopted an $M/M/1$ queueing model to guide node-based resource allocation for stretch factor (a variant of slowdown) DiffServ provisioning in a server cluster. However, to achieve the processing rate of classes, the node partitioning strategy still needs the support of resource allocation on individual servers. In this paper, we design and implement a practical application-level process allocation approach on an Apache Web server to achieve differentiated processing rates.

In [1], the authors utilized feedback control approaches to achieve overload protection and performance guarantees on Web servers. The strategy was based on real-time scheduling theory which states that response time can be guaranteed if server utilization is maintained below a pre-computed bound. Thus, control-theoretical approaches, in combination with content adaptation strategies, were formulated to keep server utilization at or below the bound. In this paper, we design and integrate a PID feedback controller with the queueing-theoretical rate allocation. Our approach is complementary to the previous work in the sense that our approach integrates the queueing theory and control theory for proportional response time differentiation.

There are efforts in resource management for service differentiation on multimedia servers [9, 28]. Multimedia connections impose very different load characteristics compared to those in traditional Web servers. The quality of multimedia content, instead of responsiveness, is often used as the primary QoS metric in service differentiation provisioning, such as image size and resolution, video streaming bandwidth, etc. Content adaptation techniques, such as image and video transcoding, are the enabling technologies. For instance, in [28], we proposed a transcoding-enabled bandwidth allocation strategy, which allows a streaming server to provide acceptable response time to clients by trading off video quality for streaming bit rate. Note that the work in this paper focuses on proportional responsiveness differentiation on traditional Web servers.

3 Differentiation Architecture and Scheme

The proportional response time differentiation model aims to control the ratios of the average response time of classes based on their pre-specified differentiation parameters $\{\delta_i, i = 1, \ldots, N\}$. Let $T_i(k)$ denote the average response time of requests of class i during sampling period k. Specifically, the model requires that the ratio of average response time between class i and j is fixed to the ratio of the differentiation parameters

$$\frac{T_i(k)}{T_j(k)} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i, j \leq N. \tag{1}$$

The proportional model essentially has the proportionality fairness. According to the requirement of the differentiation predictability, the higher classes should receive better service, i.e., lower response time. Without loss of generality, we assume that class 1 is the ’highest class’ and set $0 < \delta_1 < \delta_2 < \ldots < \delta_N$.

There are three requirements of relative service differentiation provisioning.

1. *predictability*: higher classes should receive better or no worse service quality than lower classes, independent of the class load distributions.
2. *Controllability*: the system should have a number of controllable parameters that are adjustable for the control of quality spacings among classes.
3. *fairness*: requests from lower classes should not be over-compromised for requests from higher classes.

The proportional model essentially has the proportionality fairness. According to the requirement of the differentiation predictability, the higher classes should receive better service, i.e., lower response time. Without loss of generality, we assume that class 1 is the ’highest class’ and set $0 < \delta_1 < \delta_2 < \ldots < \delta_N$.

Like others in [22, 29], we adopt Poisson process arrivals and exponentially distributed service times (an $M/M/1$ FCFS queue) for modeling the traffic. We note that there are other popular heavy-tailed distributions, such as Bounded Pareto, for service time distributions [3]. However, we note that the focus of this paper is on process allocation for achieving different processing rates in support of proportional responsiveness differentiation. The processing rate allocation scheme derived by the M/M/1 queueing model is able to give the key insights about the differentiation problem and the feasibility of the proposed process allocation approaches. It will be our future work to further investigate the problem by using other workload distributions.

We divide the processing capacity of a server into N virtual server, each configured with a process pool. Each virtual server handles requests of one class in a FCFS discipline. Let $\mu_i(k), 1 \leq i \leq N$ denote the normalized request processing rate of the virtual server i during the sampling period k. We have

$$\sum_{i=1}^{N} \mu_i(k) = 1. \tag{2}$$

Assume requests of class i in Poisson process arrive at virtual server i during the sampling period k in a rate $\lambda_i(k)$. It follows that the traffic intensity on the server $\rho_i(k) = \lambda_i(k)/\mu_i(k)$. According to the foundations of queueing theory [17], when $\rho_i(k) < 1$, *i.e.*, $\lambda_i(k) < \mu_i(k)$, we have the expected response time of requests in class i as

$$T_i(k) = \frac{\rho_i(k)}{\lambda_i(k)(1-\rho_i(k))} = \frac{1}{\mu_i(k) - \lambda_i(k)} \qquad 1 \leq i \leq N. \tag{3}$$

For feasible processing rate allocation, we must ensure that the system utilization $\sum_{i=1}^{N} \rho_i(k) \leq 1$. That is, the total processing requirement of the N classes of traffic is less than the server's processing capacity. Otherwise, a request's response time can be infinite and proportional differentiation would be infeasible. Admission control mechanisms can be applied to drop requests from lower classes so that the constraint holds [10].

According to the definition of (3), the set of (1) in combination with (2) lead to

$$\mu_i(k) = \lambda_i(k) + \frac{1 - \sum_{l=1}^{N} \lambda_l(k)}{\delta_i / \sum_{l=1}^{N} \delta_l}. \tag{4}$$

The first term of (4) is a baseline that prevents the virtual server from being overloaded. The second term of (4) denotes the excess capacity of the server, which is fairly allocated to processing different request classes with respect to their differentiation parameters and normalized arrival rates.

It follows that the ratio of the processing rate of two classes during the sampling period k is

$$\frac{\mu_i(k)}{\mu_j(k)} = \frac{\lambda_i(k) + \tilde{\lambda}\tilde{\delta}(k)/\delta_i}{\lambda_j(k) + \tilde{\lambda}\tilde{\delta}(k)/\delta_j}, \tag{5}$$

where $\tilde{\lambda}\tilde{\delta}(k) = (1 - \sum_{l=1}^{N} \lambda_l(k)) \sum_{l=1}^{N} \delta_l$. The expected response time of requests of class i, $T_i(k)$, is calculated as:

$$T_i(k) = \frac{\delta_i}{\tilde{\lambda}\tilde{\delta}(k)}. \tag{6}$$

From (6), we have the following three basic properties regarding the predictability and controllability of the proportional responsiveness differentiation given by the rate allocation strategy:

1. Response time of a request class increases with its request arrival rate.
2. With the increase of the differentiation parameter of a request class, its response time increases but all other request classes have lower response times.
3. Increasing the workload (request arrival rate) of a higher request class causes a larger increase in response time of a request class than increasing the workload of a lower request class.

4 An Integrated Process Allocation Approach

4.1 A Coarse-grained Queueing-theoretical Process Allocation

Apache Web servers are popular because of its open source. On a process-per-request Apache Web server, a process is treated as the scheduling entity for an independent activity. It is also treated as the principal for the allocation of resources, such as CPU cycles and memory space. Process abstraction serves both as a protection domain and as a resource principal. Thus, it is often assumed that the processing rate of a virtual server is proportional to the number of active processes allocated to its process pools.

We propose an adaptive process allocation strategy. Its objective is to dynamically and adaptively change the number of processes allocated to process pools for handling different classes while ensuring the ratios of process allocation specified by the processing rate allocation scheme. The rationale is that to achieve the processing rate ratios among classes, the allocation strategy has to assure that most of the processes allocated to the process pools listening to corresponding ports are active. To utilize the advantage of the Apache pre-forking mechanism, it allows that a small number of processes on a port to be idle. The number is identified by a threshold (T). If more than T processes on a port is idle, the strategy is to decrease the number of processes allocated to the process pools proportionally. Algorithm 1 gives the details.

Algorithm 1 A queueing-theoretical adaptive process allocation approach.

```
for each process allocation period do
    get the number of active processes ($p_i$) currently allocated to port $i$ from Apache scoreboard; let $P = \sum_{i=1}^{N} p_i$;
    // Apache server automatically forks new processes according to the workload condition
    get the normalized process allocations $\mu_1^*, \mu_2^*, \cdots, \mu_N^*$ according to (5);
    search for a process multiplier $m$, so that $m \sum_{i=1}^{N} \mu_i^* \le P < (m+1) \sum_{i=1}^{N} \mu_i^*$;
    // $m\mu_i^*$ is the number of processes that the allocation strategy wants to allocate to port $i$.
    // $p_i$ is the number of active processes on port $i$, which is adjusted in the following.
    for each port number $i$ do
        while $p_i - m\mu_i^* > T$ do    // too many processes forked on port $i$.
            prohibit a process on this port from listening new requests;
      // this process will soon become idle and be killed by Apache itself.
    end for
end for
```

In each process allocation period, a multiplier m is used to keep the ratio of the number of active processes of process pools to the normalized value specified by the allocation scheme (5). At line 3, the normalized process allocation (μ_i^*) is the normalized integer value of the number of processes allocated to the process pool i. For example, in a two-class scenario, if $\mu_1/\mu_2 \approx 3/1$, we have $\mu_1^* = 3$ and $\mu_2^* = 1$. At line 4, a desirable value of m is searched. It is incremented if the total number of active processes of all process pools (P) is greater than the target total number ($m \sum_{i=1}^{N} \mu_i^*$). This scenario is possible due to the pre-forking mechanism of Apache Web servers. For instance, although the allocation strategy initially assigns 3 and 1 processes for listening port 1 (process pool for handling class 1) and port 2 (process pool for handling class 2), respectively, the Apache server may actually have forked 10 and 4 processes for listening the two ports respectively. Line 7 adjusts the allocations to ensure the ratio of process allocations among the classes. It lets Apache itself to prohibit a process from listening new requests.

4.2 An Integrated Process Allocation Approach with Feedback Control

To provide more robust proportional response time differentiation, we propose to design and integrate a feedback controller to the adaptive process allocation approach. The adaptive allocation approach is referred to as the queueing-theoretical allocation approach in the following. Proportional integral derivative (PID) control is one of the most classical control design techniques widely used in industrial control systems [15]. In our system, PID controller is used to adjust the number of processes allocated to a process pool according to the difference between the target average response time and the experienced average response time of a request class. Specifically, the operation of the PID controller is described as follows:

$$p_i(k+1) = p_i(0) + K_P e_i(k) + K_I \sum_{j=0}^{k-1} e_i(j) + K_D \Delta e_i(k). \tag{7}$$

$p_i(0)$ denotes the initial number of processes allocated to process pool i by the queueing-theoretical allocation strategy. The other three terms added to $p_i(0)$ in the equation above denote proportional, integral, and derivative components, respectively. The advantage of proportional controller lies on its simplicity. The number of allocated processes is adjusted in proportion to the error between the target response time and the achieved response time ($e_i(k)$). Setting a large proportional feedback gain (K_P) typically leads to faster response at the cost of increasing system instability. The derivative control takes into account the change of errors ($\Delta e_i(k)$) in adjusting the process allocation of a class and hence responds fast to errors. Increasing the derivative gain (K_D) typically results in higher system stability. The integral controller is able to eliminate the steady-state error and avoid over-reactions to measurement noises.

Figure 2 illustrates the translation of the differentiation model into the feedback control structure. In the integrated approach, one class, say class 1, is selected as the base class, and a control loop is associated with

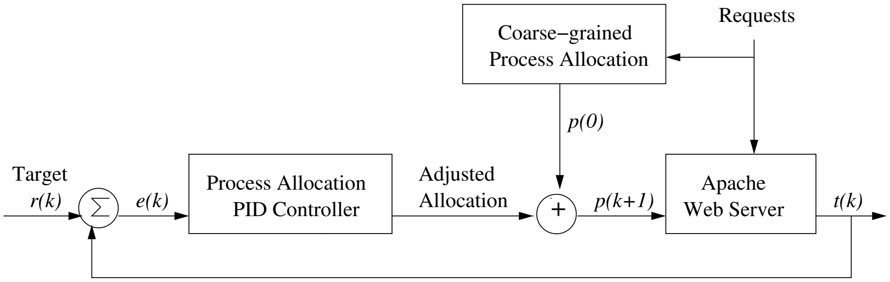


Figure 2: The structure of process allocation control loop.

every class. Thus, there are total N control loops in the system.

Consider the kth sampling period. In the control loop associated with class i, the output of the server is the experienced response time of the class $T_i(k)$. The reference input of the loop is

$$r_i(k) = \frac{\delta_i}{\delta_1} T_1(k). \tag{8}$$

The error associated with class i is calculated as

$$e_i(k) = r_i(k) - T_i(k), \tag{9}$$

and the derivative error with class i is calculated as

$$\Delta e_i(k) = e_i(k) - e_i(k-1). \tag{10}$$

By (7) to (10), the process allocation of class i in the sampling period $k+1$ thus is determined.

5 Performance Evaluation

5.1 Implementation Issues

We implemented the process allocation strategies on an Apache Web server to evaluate the impact of the feedback control on the proportional response time differentiation. Figure 3 depicts the architecture of the integrated process allocation implementation.

Two HP PCs (PIII 1 GHz, 516M RAM) installed with Redhat 9 were used as a router and a Web server, respectively. Four HP PCs (PIII 233 MHz, 96MB RAM) installed with Redhat 9 and Httperf 0.8 [21] were used to generate Http requests of Poisson distribution. The router conducted traffic classifications. We installed Apache 1.3.29 on the Web server. We configured Apache server at application level to make one server listen to different ports. The number of ports was determined by the number of traffic classes to be differentiated.

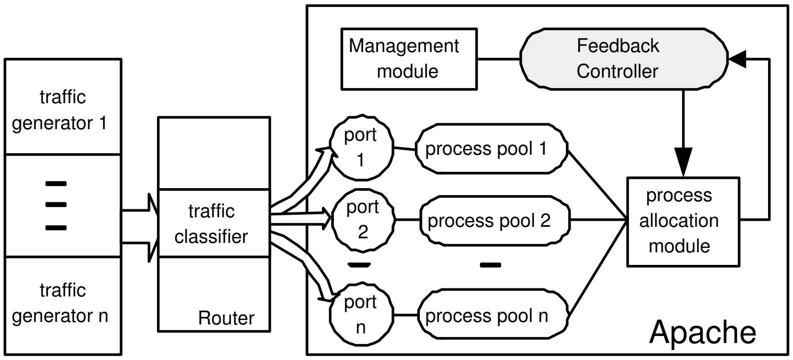


Figure 3: The implementation structure.

For example, we used three ports (80, 8000, 8080) for traffic routing of a three-class workload. The requests of class 1 were routed to port 80 which was handled by the process pool 1, requests of class 2 were routed to port 8000 handled by the process pool 2, and requests of class 3 were routed to port 8080 handled by the process pool 3.

The process allocation module in the Web server calculated the processing rate of each class according to its predicted load condition. The load was predicted for every sampling period, which was the processing time of a thousand average-size requests. We adopted a moving window with exponential averaging for the load prediction. The predicted load was the average of past five sampling periods. We implemented the process allocation approaches by modifying child_main() function in http_main.c file of the Apache server. The process forking and killing mechanisms were not modified and still handled by Apache. This application-level implementation is flexible and portable.

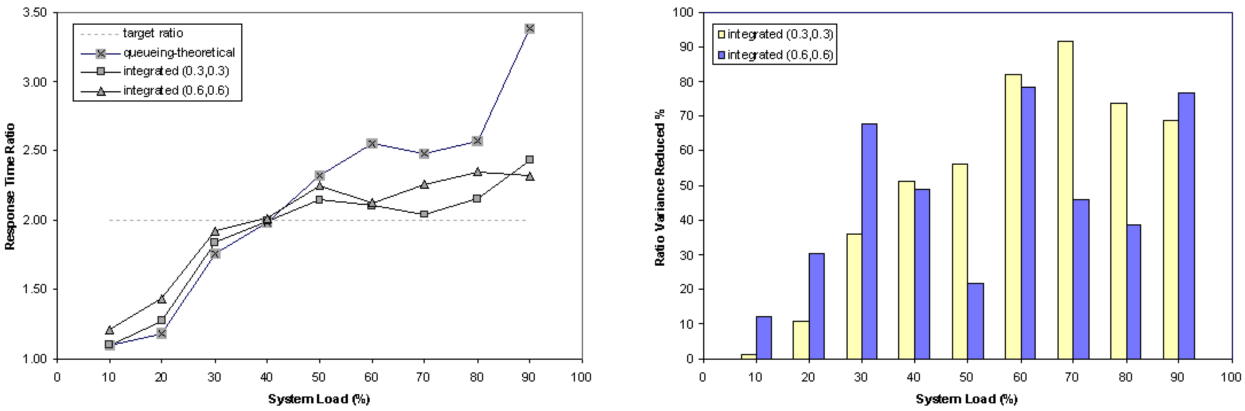
5.1.1 Impact of Integrated Approach on Differentiation Proportionality

The objective of the integrated approach is to reduce the difference between the target response time ratio and the achieved response time ratio of two classes. Therefore, we define the performance improvement metric as the follows

$$\zeta = 1 - \frac{\mid ratio_i - target \mid}{\mid ratio_q - target \mid} \tag{11}$$

where $ratio_i$ and $ratio_q$ are the achieved response time ratio of two classes due to the integrated approach and the queueing-theoretical approach, respectively.

Figure 4(a) depicts the achieved response time ratio due to the two approaches. The arrival rate ratio of two classes ($\lambda_1 : \lambda_2$) is 3:1 and the differentiation weight ratio ($\delta_1 : \delta_2$) is set to be 1:2. In the integrated approach, two different sets of control parameter were adopted. As we observe from the figure, the performance of PID



(a) Achieved response time ratios.

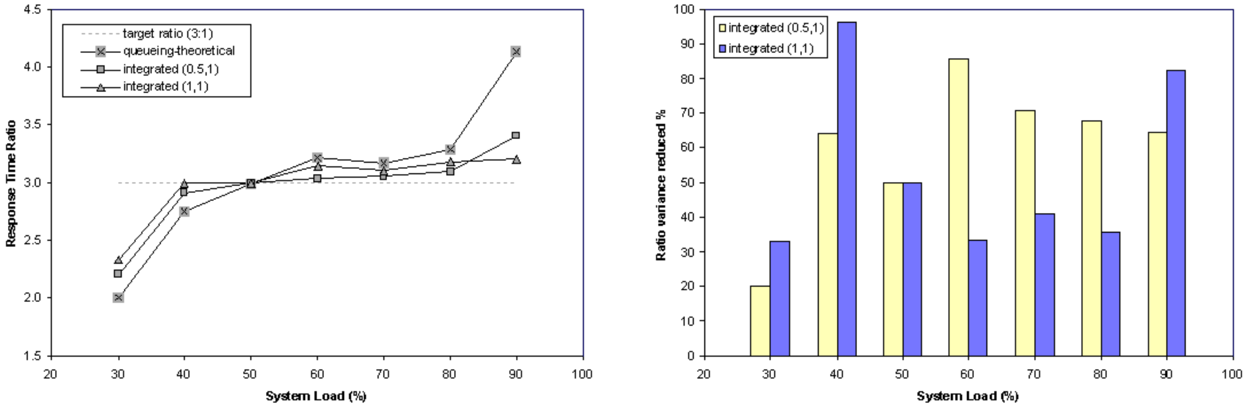
(b) Improvement due to feedback control.

Figure 4: Two-class differentiation due to the process allocation approaches ($\delta_1 : \delta_2 = 1 : 2$).

controller is quite sensitive to parameter settings. Actually, it is a non-trivial task to tune the three parameters to get good performance of proportional differentiation. Like others in [18], we assign the same value to the three PID parameters. Integrated(α, β) gives the results due to the feedback parameter settings: the PID parameters are set as $K_{P1} = K_{I1} = K_{D1} = \alpha$ for class 1 and $K_{P2} = K_{I2} = K_{D2} = \beta$ for class 2. Note that a set of good parameters for one class may not be effective for the other, and vice versa.

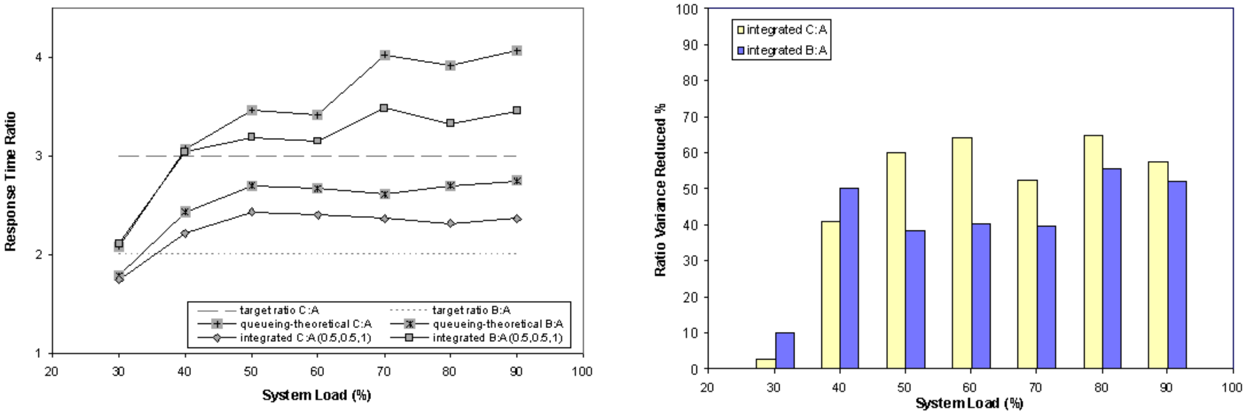
From Figure 4(a), we can observe that the integrated approach with both feedback parameter settings significantly outperforms the queueing-theoretical approach with respect to the response time differentiation proportionality. When the arrival rate is below 30%, the expected response time proportionality cannot be achieved. This is already explained above by the fact that when the workload is light, there is almost no queueing delay observed in all traffic queues. Because the scheduling is work conserving and non-preemptive, DiffServ is not feasible under certain light load conditions [12, 20]. In reality, differentiation may not be necessary during light load conditions since the resources are sufficient. Therefore, in the following diagrams, we will not give the results when the workload is less than 30%. When the system load is close to system capacity, say at 90%, the queueing-theoretical approach generates very poor proportionality. This can be explained by the fact that as the system load is close to its capacity, the impact of the variance of interarrival times on queueing delay dominates and thus queueing delay in all traffic queues increase significantly. This affects the controllability of the queueing-theoretical process allocation approach. On the other hand, the integrated approach with feedback control is able to maintain desirable differentiation proportionality. Figure 4(b) further depicts the improvement of the integrated approach over the queueing-theoretical approach with respect to the metric defined by (11).

Figure 5(a) depicts the achieved response time ratio due to the two approaches, respectively. The arrival



(a) Achieved response time ratios. (b) Improvement due to feedback control.

Figure 5: Two-class differentiation due to the process allocation approaches ($\delta_1 : \delta_2 = 1 : 3$).



(a) Achieved response time ratios. (b) Improvement due to feedback control.

Figure 6: Three-class differentiation due to process allocation approaches ($\delta_1 : \delta_2 : \delta_3 = 1 : 2 : 3$).

rate ratio of two classes ($\lambda_1 : \lambda_2$) is 3:1 and the differentiation weight ratio ($\delta_1 : \delta_2$) is 1:3. It can be observed that the integrated approach significantly outperforms the queueing-theoretical approach. In particular, the integrated approach can maintain desirable differentiation proportionality during heavy load conditions. Figure 5(b) further depicts the improvement of the integrated approach over the queueing-theoretical approach with respect to the metric (11).

We then use a three-class workload to address the sensitivity of the approaches to the number of classes. Figure 6 depicts the differentiation results due to the two approaches. The arrival rate ratio of three classes ($\lambda_1 : \lambda_2 : \lambda_3$) is 3:2:1 and the differentiation weight ratio ($\delta_1 : \delta_2 : \delta_3$) is set to be 1:2:3. Figure 6(a) shows the achieved response time ratios due to the two approaches. The integrated approach adopted setting (0.5, 0.5, 1) for the three PID control parameters for class 1, 2, and 3, respectively. That is, $K_{P1} = K_{I1} = K_{D1} = 0.5$,

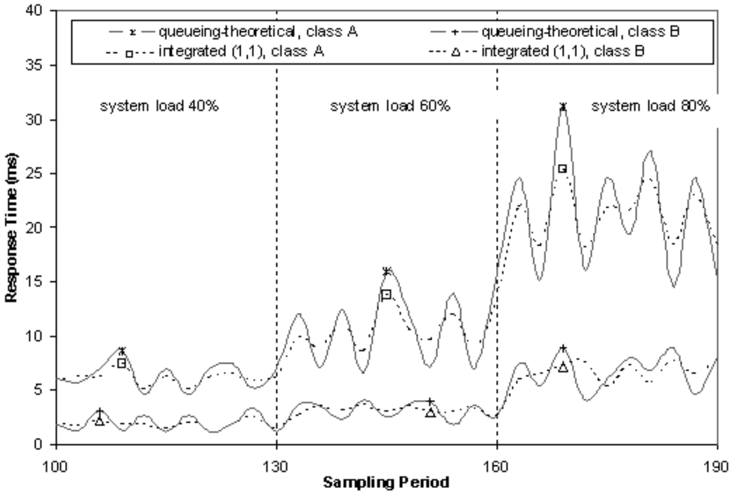


Figure 7: A microscopic view of response times.

$K_{P2} = K_{I2} = K_{D2} = 0.5$, and $K_{P3} = K_{I3} = K_{D3} = 1$. Figure 6(b) further shows the improvement of the integrated approach over the queueing-theoretical approach with respect to the metric (11). It can be observed again that the integrated approach significantly outperforms the queueing-theoretical approach with respect to differentiation proportionality.

We conducted a wide ranger of sensitivity analyses. We varied the number of classes, the arrival rate ratio of the classes, and the differentiation weight ratio of the classes. While we do not have space to present all of the results, it worths note that we did not reach any significantly different conclusion regarding to the robust differentiation proportionality achieved by the integrated approach.

5.1.2 Impact of Integrated Approach on Microscopic Behaviors

In the following, we discuss the effectiveness of the integrated approach by investigating its microscopic behaviors and differentiation sensitivity. Figure 7 shows a microscopic view of the response time of individual requests of the two classes due to the two approaches, when the system workload is 40%, 60%, and 80%, respectively. The target response time ratio of class A to class B is 3:1. The experiments were run for 100 sampling periods for warming up and then the data was collected for 30 sampling periods at each of three workload conditions. Obviously, we can observe that the integrated approach achieves more consistent results during different sampling periods at various workload conditions.

Figure 8 further quantitatively depicts the variance of the proportionality due to the two approaches. At each of the three workload conditions (40%, 60%, 80%), we conducted experiments by using a two-class

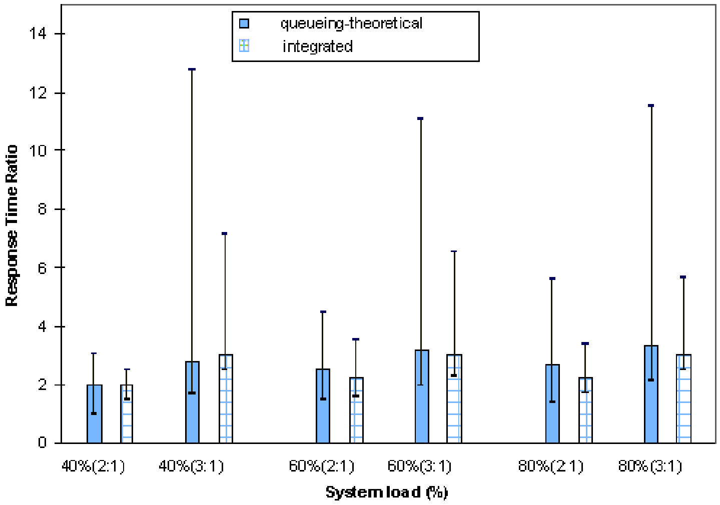


Figure 8: The variance of response time ratios.

workload with the target response time ratio 2:1 and 3:1, respectively. The upper line is the 95th percentile; the bar is the mean; and the lower line is the 5th percentile. We can observe that the integrated approach can significantly reduce the variance. For example, when the workload is 80% and the target proportionality is 2, the difference between the 95th and the 5th percentile is 1.7 and 4.2 due to the integrated approach and the queueing-theoretical approach, respectively. Furthermore, the mean is 2.2 and 2.7, respectively. At 80% workload condition, when the target ratio is 3, the difference between 5th and 95th is 3.1 and 9.3, and the mean is 3 and 3.3, due to the integrated approach and the queueing-theoretical approach, respectively. The better proportionality in terms of the mean of the achieved response time ratios has also been depicted by Figures (4), (5), and (6). It is because the difference between the target response time ratio and the achieved one is fed back into the integrated approach. When an error occurs, the process allocations are adjusted by the feedback control accordingly.

Figure 9 depicts the converge of the response time ratio to the target proportionality (3:1) due to the queueing-theoretical approach and the integrated approach, respectively. It indicates that the integrated approach can converge to the target ratio with less settling time than that of the queueing-theoretical approach. Furthermore, we can observe less oscillation and better differentiation stability due to the integrated approach. Experiments with different class load conditions were also carried out. They yielded similar results as shown in the figure. Hence, our integrated approach is successful in finding a good and practical solution to process allocation for the proportional response time differentiation.

We also note that the results of the integrated approach achieved by the implementation is not as perfect as those that would be achieved by the simulation because of the application-level implementation issues.

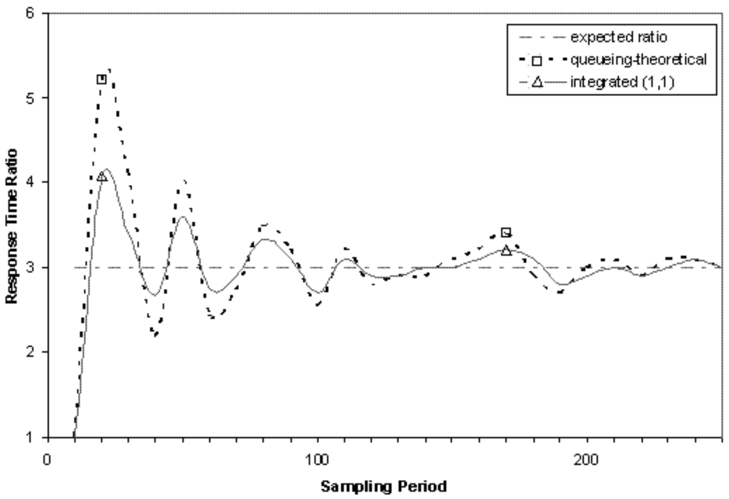


Figure 9: Settling time due to the allocation approaches.

6 Conclusions and Future Work

Providing proportional response time to different client classes is an important and challenging issue. It is important because proportional model is a popular relative DiffServ model and response time is a fundamental QoS metric on Web servers. It is challenging because the conventional application-level process allocation approaches lack fine-grained control of resource allocation and are insensitive to the bursty Internet traffic.

In this paper, we have designed a practical application-level process allocation approach to providing robust and find-grained proportional response time services by integrating feedback control with queueing theory. It first measures the processing rate of classes and allocates the certain number of processes accordingly to handle their requests based on a queueing-theoretical approach. It then adjusts the process allocations according to the error between the target response time ratio and the achieved one by using feedback control. We have implemented the approach on an Apache Web server. Experiment results have demonstrated that the integrated approach can provide more predictable and fine-grained differentiated services than the queueing-theoretical approach. This practical approach can be easily deployed on Apache Web servers.

In this work, we adopted an $M/M/1$ queue for modeling workload consisting of static Web pages. There are other popular heavy-tailed distributions for workload modeling [3]. It will be our future work to further investigate the differentiation problem by using workload consisting of both static and dynamic Web content. Future work will also be on applying other control techniques.

Acknowledgement

This material is based on research sponsored by the Air Force Research Laboratory, under agreement number FA9550-04-1-0239. The authors would also like to thank Network Information and Space Security Center (NISSC) for providing support in the work.

References

[1] T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: a control-theoretical approach. *IEEE Trans. on Parallel and Distributed Systems*, 13(1):80–96, 2002.

[2] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of services in Web content hosting. In *Proc. ACM SIGMETRICS Workshop on Internet Server Performance*, pages 91–102, 1998.

[3] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large Web-based shopping system. *ACM Trans. on Internet Technology*, 1(1):44–69, 2001.

[4] M. Aron, P. Druschel, and W. Zwaenepoel. Cluster reserves: a mechanism for resource management in cluster-based network servers. In *Proc. ACM SIGMETRICS*, pages 90–101, 2000.

[5] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc. USENIX Symposium on Operating System Design and Implementation*, 1999.

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

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *IETF RFC 2475*, 1998.

[8] V. Cardellini, E. Casalicchio, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. *ACM SIGMETRICS Performance Evaluation Review*, 29(2):14–19, 2001.

[9] S. Chandra, C. S. Ellis, and A. Vahdat. Differentiated multimedia Web services using quality aware transcoding. In *Proc. IEEE INFOCOM*, pages 961–968, 2000.

[10] X. Chen and P. Mohapatra. Performance evaluation of service differentiating Internet servers. *IEEE Trans. on Computers*, 51(11):1,368–1,375, 2002.

[11] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *Proc. ACM SIGCOMM*, 1999.

[12] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. *IEEE/ACM Trans. on Networking*, 10(1):12–26, 2002.

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

[14] M. El-Gendy, A. Bose, and K. G. Shin. Evolution of the Internet QoS and support of soft real-time applications. In *Proc. of the IEEE*, July 2003.

[15] G. F. Franklin, J. D. Powell, and A. Emami-naeini. *Feedback control of dynamics systems*. Prentice Hall, 4th edition, 2002.

[16] Y. Huang and R. Gu. A simple fifo-based scheme for differentiated loss guarantees. In *Proc. IWQoS*, 2004.

[17] L. Kleinrock. *Queueing Systems, Volume II*. John Wiley and Sons, 1976.

[18] B. Ko, K. Lee, K. Amiri, and S. Calo. Scalable service differentiation in a shared storage cache. In *Proc. 23rd IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2003.

[19] S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. Admission control and dynamic adaptation for a proportional-delay DiffServ-enabled Web server. In *Proc. ACM SIGMETRICS*, 2002.

[20] M. K. H. Leung, J. C. S. Lui, and D. K. Y. Yau. Adaptive proportional delay differentiated services: Characterization and performance evaluation. *IEEE/ACM Trans. on Networking*, 9(6):908–817, 2001.

[21] D. Mosberger and T. Jin. Httperf: a tool for measuring Web server performance. http://www.hpl.hp.com/personal/David_Mosberger/httperf.html.

[22] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based Internet services. In *Proc. of USENIX OSDI*, pages 225–238, December 2002.

[23] P. J. Shenoy, S. Hasan, P. Kulkarni, and K. Ramamritham. Middleware versus native OS support: architectual considerations for supporting multimedia applications. In *Proc. IEEE Real-Time Technology and Application Symposium*, 2002.

[24] J. Wei, C.-Z. Xu, and X. Zhou. A robust packet scheduling algorithm for proportional delay differentiation services. In *Proc. of IEEE Globecom*, 2004.

[25] B. Yang and P. Mohapatra. Multicasting in differentiated service domains. In *Proc. of IEEE Globecom*, 2002.

[26] X. Zhou, Y. Cai, G. K. Godavari, and C. E. Chow. An adaptive process allocation strategy for proportional responsiveness differentiation on Web servers. In *Proc. IEEE 2nd Int'l Conf. on Web Services (ICWS)*, July 2004.

[27] X. Zhou, J. Wei, and C.-Z. Xu. Modeling and analysis of 2D service differentiation on e-Commerce servers. In *Proc. of IEEE 24th Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 740–747, March 2004.

[28] X. Zhou and C.-Z. Xu. Harmonic proportional bandwidth allocation and scheduling for service differentiation on streaming servers. *IEEE Trans. on Parallel and Distributed Systems*, 15(9):838–551, 2004.

[29] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation for cluster-based network servers. In *Proc. IEEE INFOCOM*, pages 679–688, 2001.