An Adaptive Process Allocation Strategy for Proportional Responsiveness Differentiation on Web Servers [*]

Xiaobo Zhou Yu Cai Ganesh K. Godavari C. Edward Chow
Department of Computer Science
University of Colorado at Colorado Springs, Colorado Springs, CO 80933
{zbo, ycai, gkgodava, chow}@cs.uccs.edu
Paper number:

Abstract

There is a growing demand for provisioning of different levels of quality of service (QoS) on Web servers to meet changing resource availability and satisfy different client requirements. The proportional differentiation model is getting momentum because of its fairness and differentiation predictability. It states that QoS of different traffic classes should be kept proportional to their pre-specified differentiation parameters, independent of the class loads.

In this paper, we present a processing rate allocation scheme for providing proportional response time differentiation on Web servers. A challenging issue is how to achieve processing rates for different request classes in the implementation. We propose a process allocation strategy, which dynamically and adaptively changes the number of processes allocated for handling different request classes while ensuring the ratio of process allocations specified by the processing rate allocation scheme. We implemented the process allocation strategy at application level on Apache Web servers. Experimental results showed that the processing rate can be achieved by the adaptive process allocation strategy and the Web servers can provide predictable and controllable proportional responsive time differentiation.

1 Introduction

Clients of Web applications are different in their access patterns, receiving devices, and service fees. There is a growing demand for Web servers to provide different level of Quality of Service (QoS) to meet changing system resource availability and satisfy different client requirements. For example, in a Web content hosting farm, customers expect that the requests from their potential clients be serviced with a quality proportional to their payments. In an indiscriminate Web site, aggressive clients should be controlled so that other clients are able to use their fair shares of the resources at heavy-load periods. In an e-commerce site, customer checkout requests are more important than catalog browing requests of visitors and therefore should be handled in a more timely manner. Evidently, providing differentiated services at server side is an important issue.

The idea and concept of Differentiated Services (DiffServ) was originally proposed and formulated by the Internet Engineering Task Force (IETF) [6]. Its goal is to define configurable types of packet forwarding in network core routers, which can provide per-hop differentiated services for large aggregates of network traffic. DiffServ has been an active research topic in the arena of packet networks. The proportional differentiation model [9] states that certain class performance metrics should be proportional to their pre-specified differentiation parameters, independent of the class loads. Due to its inherent fairness and differentiation predictability, the model has been accepted as an important relative DiffServ model and been applied in the proportional delay differentiation (PDD) in packet scheduling [9]. Many algorithms have been proposed in achieving PDD provisioning in the networking core; see [10, 11, 16] for representatives.

There are recent efforts on providing DiffServ from server side [1, 2, 5, 7, 8, 15, 17, 18, 19]. In the server side, response time is a fundamental performance metric. Existing responsiveness differentiation strategies are mostly based on priority scheduling in combination with admission control and content adaptation [1, 2, 5, 7, 8]. For instance, the authors in [8] adopted strict priority scheduling strategies to achieve responsive time differentiation on Internet servers. The results showed that the differentiation can be achieved with requests of higher priority classes experiencing lower response time than requests of lower priority classes. However, this kind of strategies cannot control

[*]This research work was supported in part by a NISSC AFOSR Grant award.

the quality spacings proportionally among different classes. They may also lead to starvation for requests in lower priority classes. Time-dependent priority scheduling algorithms developed for PDD provisioning in packet networks can be tailored for PDD provisioning on Web servers [15]. 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.

In [17, 18], we proposed processing rate allocation strategies for server-side DiffServ provisioning in various Web applications. We used slowdown, the ratio of a request's queueing delay to its service time, as the performance metric. We left a challenging implementation issue; that is, how to achieve the processing rate for various traffic classes on servers. In [19], 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 present a processing rate allocation scheme for proportional response time differentiation. We then propose and implement an adaptive process allocation strategy to achieve the processing rate allocated to the request classes on Apache Web servers.

The structure of the paper is as follows. Section 2 gives the processing rate allocation scheme for responsive time differentiation. Section 3 presents the design and implementation of the adaptive process allocation on Apache Web servers. Section 4 focuses on experimental results and performance evaluation. In Section 5, we review other related resource allocation and scheduling disciplines in the DiffServ areas. Section 6 concludes the paper.

2 Processing Rate Allocation for Proportional Responsiveness Differentiation

2.1 Differentiation Architecture

There are two types of DiffServ schemes [6]. One is absolute DiffServ, in which each request class receives an absolute share of resource usages. The other is relative DiffServ, in which a class with a higher desired QoS level (referred to as higher class) will receive better (at least no worse) service quality than a lower class. Although absolute DiffServ is desired to Internet services like audio/video streaming applications that have hard time constraints, relative DiffServ is sufficient for soft real-time Web applications like e-Commerce transactions.

In order for a relative DiffServ scheme to be effective, the scheme must satisfy two basic properties: *predictability* and *controllability*. Predictability requires that higher classes receive better or no worse service quality than lower classes, independent of the class load distributions. Controllability requires that the scheduler contains a number of controllable parameters that are adjustable for the control of quality spacings among classes. An additional requirement on e-Commerce servers is *fairness*. That is, requests from lower classes should not be over-compromised for requests from higher classes. The proportional differentiation model is important because of its inherent proportional fairness property.

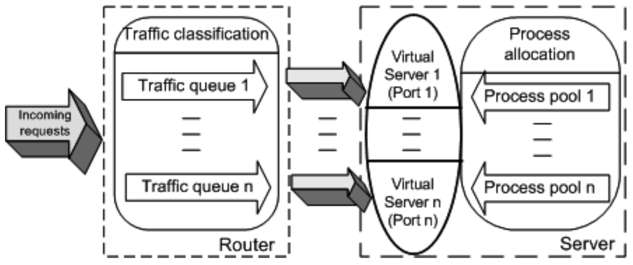


Figure 1. The architecture for proportional responsiveness differentiation.

Figure 1 illustrates the architecture of precessing rate allocation for proportional responsiveness differentiation on Web servers. Incoming requests from different clients are classified into N ($2 \leq N$) classes according to their desired levels of QoS by the terminal router. The request classification can be done based on clients' profile, device, payment, etc. The requests of different classes are then routed to different ports of a Web server. Each port is handled by a virtual server configured with a process pool. The size of the process pool is specified by the process allocation module. Therefore, by running N virtual servers configured with different process pool sizes on the same Web server, we want to achieve different processing rates for different request classes and hence to provide proportional responsiveness differentiation on Web servers.

2.2 A Processing Rate Allocation Scheme

A proportional differentiation model ensures the quality spacing between class i and class j to be proportional to certain pre-specified differentiation parameters δ_i and δ_j [10]; that is,

$$\frac{q_i}{q_j} = \frac{\delta_i}{\delta_j} \qquad 1 \leq i, j \leq N,$$

where q_i and q_j are the QoS factor of class i and class j, respectively. So it is up to applications and clients to select appropriate QoS levels in terms of differentiation parameters that best meets their requirements, cost, and constraints.

The proportional responsiveness differentiation model aims to control the ratios of the average responsive time of classes based on their normalized differentiation parameters $\{\delta_i, i = 1, \ldots, N\}$ ($\delta_i > 0, \sum_{i=1}^{N} \delta_i = 1$). Let T_i denote the average response time of request class i. Specifically, the model requires that the ratio of average responsive time between class i and j is fixed to the ratio of the corresponding differentiation parameters

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

The differentiation predictability property requires that higher classes receive better service, i.e., lower responsive 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 [19], we adopt a $M/M/1$ FCFS queue for modeling the traffic. Recent Internet workload measurements indicate that for some Web applications a heavy-tailed distribution is more accurate for service time distributions [3, 18]. However, we note that the focus of this paper is on adaptive process allocation for achieving different processing rates in support of responsiveness differentiation. The processing rate allocation scheme derived by an M/M/1 queueing model can give the key insights about the differentiation problem and the feasibility of the process allocation strategy.

We partition the request processing rate of a Web server into N virtual servers. Each virtual server handles requests of one class in a FCFS manner. Let $\mu_i, 1 \leq i \leq N$ denote the normalized request processing rate of the virtual server i. We have

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

Assume requests of class i in Poisson process arrive at virtual server i in a rate λ_i. It follows that the traffic intensity on the server $\rho_i = \lambda_i/\mu_i$. Let T_i be the average responsive time of requests in class i. According to the foundations of queueing theory [14], when $\rho_i < 1$ ($\lambda_i < \mu_i$), we have the expected response time as

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

For feasible rate allocation, we must ensure that the system utilization $\sum_{i=1}^{N} \lambda_i \leq 1$. That is, the total processing requirement of the N classes of traffic is less than the Web server's processing capacity.

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

$$\mu_i = \lambda_i + \frac{1 - \sum_{i=1}^{N} \lambda_i}{\delta_i \sum_{i=1}^{N} 1/\delta_i}. \tag{4}$$

From this equation, we can observe that the remaining capacity of the machine is fairly allocated to different request classes with respect to their differentiation parameters.

It follows that the expected response time of requests of class i, T_i, is calculated as:

$$T_i = \frac{\delta_i \sum_{i=1}^{N} 1/\delta_i}{C - \sum_{i=1}^{N} \lambda_i}. \tag{5}$$

From (5), we have the following three basic properties regarding the predictability and controllability of the proportional responsiveness differentiation given by the processing 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.

3 Process Allocation Strategies and Implementation Issues

3.1 A Fixed Process Allocation Strategy

In a process-per-request Web server such as Apache, a 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. Process abstraction serves both as a protection domain and as a resource principal. Thus, we assume that the processing rate of a Web server is proportional to the number of active processes assigned to its classes.

In an Apache Web server, we can impose an upper bound on the number of processes allocated to the process pool for handling incoming requests. This maximum number is usually set to be 32. To achieve the processing rate ratios between classes, a straightforward solution is to split the 32 processes into multiple process pools. Each pool works as a virtual server handling requests of a class in FCFS way. Thus, we want to achieve the processing rates for different classes. We refer to this solution as *fixed process allocation strategy* since the number of total processes allocated to the process pools is fixed.

The problem with the fixed process allocation strategy is that not all allocated processes are active due to the workload dynamics. For example, we consider a two-class response time differentiation scenario. Given the arrival rates, suppose the processing rate ratio of class 1 to class 2 ($\mu_1 : \mu_2$) is 3:1. According to the fixed process allocation strategy, total 32 processes will be partitioned to 24 and 8 and they are allocated to the two process pools of classes 1 and 2, respectively. However, due to the workload dynamics of two classes, it is likely that only 18 of 24 processes allocated to class 1 are active while all 8 processes allocated to class 2 are active. Thus, the real processing rate ratio of class 1 to class 2 is 2:1, instead of 3:1. The fixed process allocation strategy may not be able to achieve proportional response time differentiation. We are going to show its results in Section 4.1.

3.2 An Adaptive Process Allocation Strategy

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 ratio of process allocations specified by the processing rate allocation scheme. The rationale is that to achieve the processing rate ratios among classes, we have to assure that all processes allocated to the process pools are active. If any process is idle, the strategy is to decrease the number of processes allocated to the classes proportionally. Algorithm 1 gives the details.

Algorithm 1 An Adaptive Process Allocation Algorithm.

```
1:  initialize process multiplier $m = 1$;
2:  while true do
3:      get the number of active processes ($p_i$) currently allocated
        to port $i$ from Apache scoreboard; let $P = \sum_{i=1}^{N} p_i$;
4:      get the normalized process allocations $\mu_1^*, \mu_2^*, \cdots, \mu_N^*$;
5:      while $P > m \sum_{i=1}^{N} \mu_i^*$ do
6:          $m = m + 1$;
7:      while $P < m \sum_{i=1}^{N} \mu_i^*$ do
8:          $m = m - 1$;
9:      for each process, get its port number $i$ do
10:         if $p_i > m\mu_i^*$      // too many processes forked on port $i$
11:             prohibit this process from listening new requests;
12:         if $p_i < m\mu_i^*$      // reduce allocations proportionally
13:             m = m - 1;
14:     end for
15: end while
```

At line 1, multiplier m is used to keep the ratio of the number of active processes of classes to the value specified by the allocation scheme (4). At line 4, the normalized process allocations μ_i^*) is the normalized integer value of the number of processes allocated to the virtual server i. For example, in a two-class scenario, if $\mu_1/\mu_2 \approx 3/1$, $\mu_1^* = 3$ and $\mu_2^* = 1$. At line 6, m is increased by 1 if the total number of active processes of all classes is greater than the specified total number. This scenario is possible due to the pre-forking mechanism of Apache Web servers. For example, although the allocator initially assigns 3 and 1 processes for listening port 1 (virtual server for handling class 1) and port 2 (virtual server for handling class 2), respectively, the Apache server may actually have forked 10 and 4 processes for listening the two ports respectively. Lines 6 and 8 aim to get the desirable value of m, which can approximate the real allocations. Lines 9 to 13 modify the allocations to ensure the ratio of process allocations among the classes. Line 11 lets Apache itself kill a process when it is idle. When the number of active processes of a virtual server is less than the desired value, the allocator has to reduce the allocation of all classes proportionally so as to ensure the ratios. This is achieved by reducing m, as shown at line 13. Under the scenario given in 3.1, the algorithm will adaptively change the process allocation of two classes from (24, 8) to (18, 6) so that the processing rate ratio of class 1 to class 2 is still 3:1 and all processes are active. It has $m = 6$.

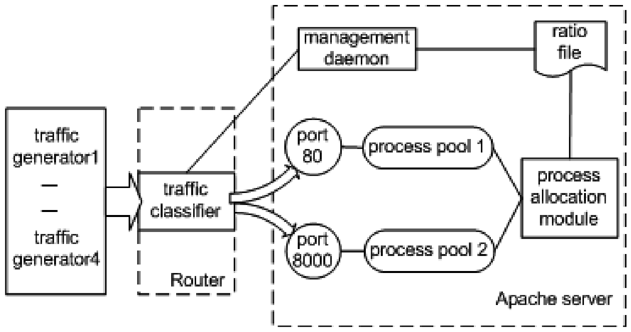


Figure 2. The implementation of process allocation strategies.

3.3 Implementation Issues

We implemented the process allocation strategies on Apache Web servers and used a two-class workload to evaluate the impact of the allocation strategies on the proportional response time differentiation. Figure 2 illustrates the architecture of the implementation.

Two HP PCs (PIII 1 GHz, 516M RAM) installed with Redhat 9 were used as a terminal router and a Web server, respectively. Four HP PCs (PIII 233 MHz, 96M RAM) installed with Redhat 9 were used to generate Http requests. We installed Apache 1.3.29 on the Web server. We modified Apache at application level to make one Apache Web server listen to two different ports (80, 8000). The requests of class 1 were routed to port 80 which was handled by the process pool 1, and requests of class 2 were routed to port 8000 which was handled by the process pool 2.

We implemented the two process allocation strategies in child_main() function in http_main.c file of Apache. The process forking and killing mechanisms were not modified and hence still handled by Apache itself. We expect this application-level implementation is flexible and portable.

4 Performance Evaluation

4.1 Impact of the Fixed Process Allocation Strategy on DiffServ Performance

In this section, we show the response time DiffServ due to the fixed process allocation strategy. Figure 3(b) shows the achieved average response time of class 1 and 2 under various system load conditions. The arrival rate ratio of two classes ($\lambda_1 : \lambda_2$) is 3:1. The differentiation weight ratio ($\delta_1 : \delta_2$) is set to be 1:3. The fixed process allocation strategy dynamically partitions all 32 processes into the two process pools for class 1 and class 2 according to their arrival rates. It can be seen that requests of class 1 always receive lower response time than those of class 2. In this sense, the responsive time DiffServ is achieved by the processing rate allocation scheme.

Figure 3(a) shows the achieved response time ratio of class 2 to class 1. When the system load is less than 60%, the achieved ratio is far less than the expected ratio, due to the reasons we discussed in 3.1. As the system load goes beyond the certain value (70%), the achieved ratio is much greater than the expected ratio. This can be explained by the fact that the variance of interarrival distributions and the variance of service time distributions affect the performance of process allocation and scheduling significantly. The figures have shown that response time DiffServ can be achieved in the sense that the requests from higher priority classes always receive lower response time than requests from lower priority classes. However, the fixed process allocation strategy cannot achieve proportional response time DiffServ because the processing rate of classes cannot be achieved accurately due to the workload dynamics.

4.2 Impact of the Adaptive Process Allocation Strategy on Differentiation Performance

In this section, we show the experimental results to demonstrate that the proportional response time differentiation can be achieved by the adaptive process allocation when the system load is within a certain range. Figure 4(a) depicts the achieved response time of classes 1 and 2 due to the adaptive process allocation strategy under various system load conditions. The arrival rate ratio of two classes ($\lambda_1 : \lambda_2$) is 3:1. The differentiation weight ratio of two classes ($\delta_1 : \delta_2$) is 1:3. It shows the allocation strategy can also achieve response time differentiation. That is, requests of class 1 always receive lower response time than requests of class 2.

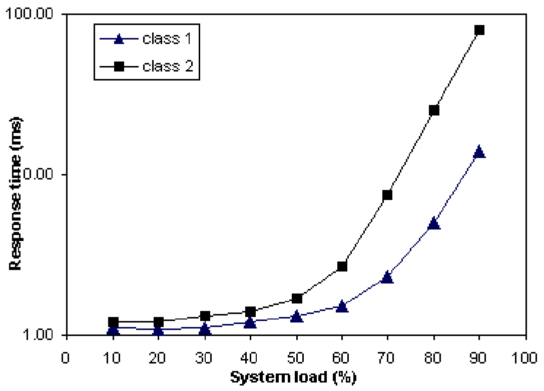
Figure 4(b) further depicts the achieved response time ratio of class 2 to class 1. When the system load is between 40% to 80%, we can see that the proportional response time differentiation can be achieved. The difference between the achieved response time ratio and the expected ratio is trivial. As we know, process abstraction serves both as a protection domain and as a resource principal in current general-purpose operating systems. However, because an application has no control over the consumption of resources that the kernel consumes on behalf of the application, resource principals do not always coincide with processes. We believe that this problem is one of the primary reasons for the difference between the achieved ratio and the expected ratio. There is a demand for new kernel-level resource management mechanisms, such as *resource container*, a new operating system abstraction introduced by [4].

Figure 4(b) also shows that when the arrival rate is below 30%, the expected response time ratio can still not be achieved. This is explained that when the workload is light, there is almost no queueing delay observed in all traffic queues. Note that the request scheduling policy is work conserving. Therefore, DiffServ is not feasible under certain light load conditions, as it was also observed in experiments for PDD provisioning in packet networks [11, 16]. When the system load is higher than 90%, we also find out that the expected ratio is not achieved. This can be explained that as the system load is close to its capacity, the impact of the variance of incoming traffic on queueing delay dominates and thus queueing delay in all traffic queues increase significantly. This affects the controllability of the process allocation strategy significantly.

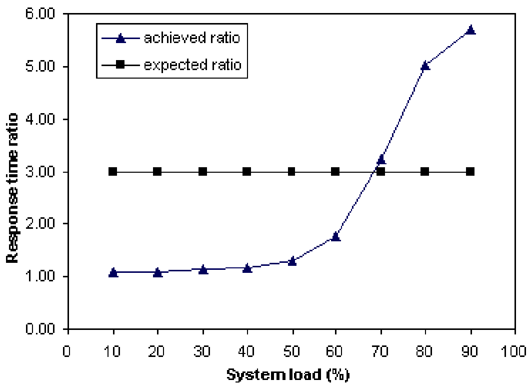
Figure 5(a) depicts the achieved response time of classes 1 and 2 due to the adaptive process allocation strategy under various system load conditions. We change the differentiation weight ratio of two classes ($\delta_1 : \delta_2$) from 1:3 to 1:2. The arrival rate ratio of two classes ($\lambda_1 : \lambda_2$) is kept to be the same at 3:1. It shows the adaptive process allocation strategy can also achieve response time differentiation. As shown by Figure 5(b), the expected response time ratio can be achieved when the system load is between 30% and 80% and hence the proportional response time differentiation is achieved.

To give more sensitivity analyses of the adaptive process allocation strategy, we vary the arrival rate ratio in the following experiment. Class 1 consistently contributes 40% of system load, while the traffic of class 2 increases. Figure 6 illustrates the achieved average response time of classes 1 and 2 and their ratios under various system load conditions.It can be seen that the adaptive process allocation strategy can achieve predictable and controllable response time differentiation at various system load conditions. Furthermore, it can achieve the proportional response time differentiation when the system load is between 50% and 80%. This demonstrates that our proposed process allocation strategy is effective in support of proportional responsiveness differentiation on Web servers.

We performed a wide range of sensitivity analyses. We



(a) Achieved average response time.



(b) Achieved response time ratio.

Figure 3. Two-class response time DiffServ due to the fixed process allocation ($\delta_1 : \delta_2 = 1 : 3$).

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, we note that we did not reach any significantly different conclusions regarding to the predictability and controllability of proportional response time differentiation achieved by the proposed adaptive process allocation strategy.

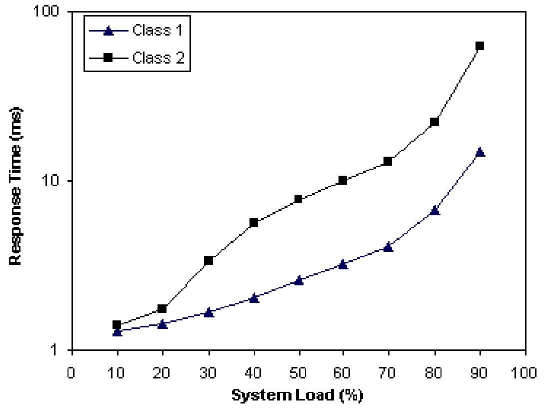
5 Related Work

The DiffServ provisioning problem was first addressed in the network core. The proportional differentiation model [9] is getting momentum because of its inherent fairness and differentiation predictability properties. It been accepted as an important DiffServ model and been applied in the proportional delay differentiation (PDD) in packet scheduling [10]. Many algorithms have been designed to achieve the PDD provisioning in the network routers. They can be classified into two categories: rate-based; see BPR [10] for example, time-dependent priority based; see WTP, PAD, and HPD [11], adaptive WTP [13, 16] for examples. Servers play an important role in end-to-end DiffServ provisioning. Those algorithms can be tailored for request scheduling for PDD provisioning in the server side [15]. However, the algorithms are not applicable to proportional response time differentiation in the server side 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.

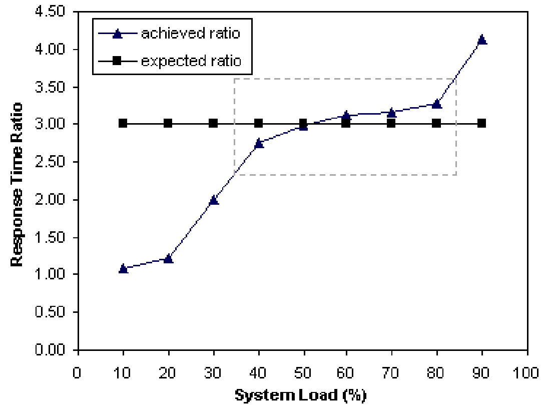
In the server side, there are recent efforts on priority-based request scheduling for responsive time differentiation [2, 5, 8, 12]. For example, in [2, 8], the authors addressed strict priority scheduling strategies for controlling CPU utilization in Web content hosting servers. QoS was introduced by assigning priorities to requests for different contents. Requests of lower priority classes were only executed if no requests existed in any higher priority classes. The results showed that service differentiation can be achieved but the quality spacings among different classes cannot be guaranteed by this kind of strict priority scheduling. Therefore, this kind of priority-based scheduling schemes cannot achieve proportional response time differentiation provisioning. In this paper, we propose a processing rate allocation scheme to achieve the response time differentiation. Our work is complementary to the previous efforts.

Admission control is often used in combination with priority-based scheduling for service differentiation provisioning. For example, in [1], the authors used classical feedback control theory to achieve overload protection, performance guarantees, and service differentiation in 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-theory approaches, in combination with content adaptation strategies, were formulated to keep server utilization at or below the bound. In [15], the authors proposed admission control algorithms in combination with time-dependent priority scheduling for proportional queueing-delay differentiation on a Web server. Therefore, this kind of admission control itself is not sufficient in PDD provisioning and is not applicable to the proportional response time provisioning.

In [17, 18], we proposed processing rate allocation schemes for proportional slowdown differentiation on various Internet application servers. In this paper, we extend our previous work not only by proposing a rate allocation scheme for proportional response time differentiation, but also by designing and implementing an adaptive process allocation strategy for achieving the processing rates.

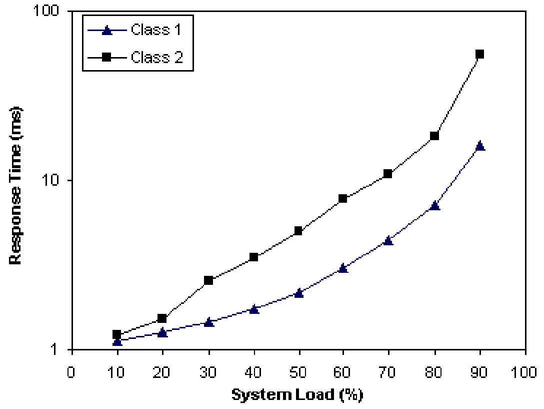


(a) Achieved average response time.

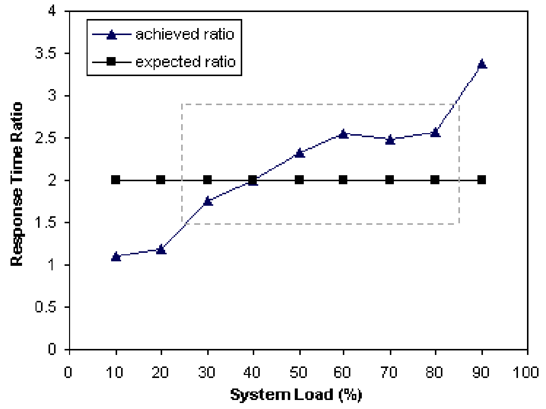


(b) Achieved response time ratio.

Figure 4. Two-class response time DiffServ due to the adaptive process allocation ($\delta_1 : \delta_2 = 1 : 3$).

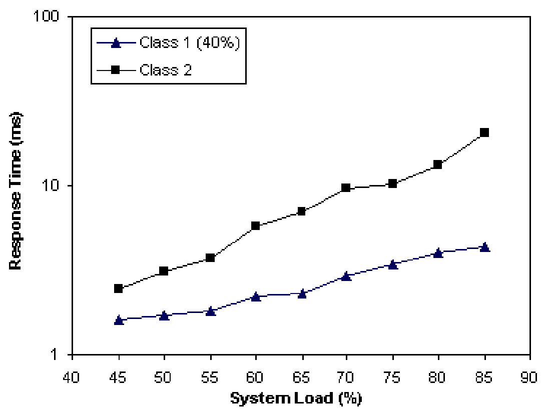


(a) Achieved average response time.

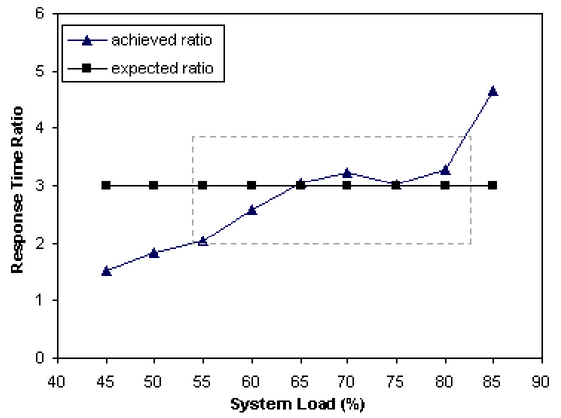


(b) Achieved response time ratio.

Figure 5. Two-class response time DiffServ due to the adaptive process allocation ($\delta_1 : \delta_2 = 1 : 2$).



(a) Achieved average response time.



(b) Achieved response time ratio.

Figure 6. Two-class response time DiffServ due to the adaptive process allocation ($\delta_1 : \delta_2 = 1 : 3$).

6 Conclusions and Future Work

In this paper, we...

Future work is on improving the robustness of the adaptive process allocation strategy under various system load conditions...

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] 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.

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

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

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

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

[9] C. Dovrolis and P. Ramanathan. A case for relative differentiated services and the proportional differentiation model. *IEEE Network*, 13(5):26–34, 1999.

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

[11] 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.

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

[13] L. Essafi, G. Bolch, and A. Andres. An adaptive waiting time priority scheduler for the proportional differentiation model. In *Proc. of the High Performance Computing Symposium*, April 2001.

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

[15] 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.

[16] 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.

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

[18] X. Zhou, J. Wei, and C.-Z. Xu. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proc. IEEE 18th Int'l Parallel and Distributed Processing Symposium (IPDPS)*, April 2004.

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