Design and Implementation of A Feedback Controller for Slowdown Differentiation on Internet Servers[*]

Jianbin Wei and Cheng-Zhong Xu
Department of Electrical and Computer Engineering
Wayne State University, Detroit, MI 48202
{jbwei, czxu}@wayne.edu

ABSTRACT

Proportional slowdown differentiation (PSD) aims to maintain slowdown ratios between different classes of clients according to their pre-specified differentiation parameters. In this paper, we design a feedback controller to allocate processing rate on Internet servers for PSD. In this approach, the processing rate of a class is adjusted by an integral feedback controller according to the difference between the target slowdown ratio and the achieved one. The initial rate class is estimated based on predicted workload using queueing theory. We implement the feedback controller in an Apache Web server. The experimental results under various environments demonstrate the controller's effectiveness and robustness.

Categories and Subject Descriptors: C.4 [Computer Systems Organization]: Performance of Systems

General Terms: Performance.

Keywords: Feedback control, Quality of service, Slowdown

1. DESIGN OF A FEEDBACK CONTROLLER

The past decade has seen a demand for provisioning of different levels of quality of service (QoS) on Internet servers. Performance metric slowdown, defined as a request's queueing delay to its service time, reflects the requirement that a request's queueing delay is kept proportional to its service time. Slowdown or its variant has been taken into account in recently designed QoS-aware systems [2, 3]. These systems mainly focus on how to minimize the average slowdown.

Let $S_i(k)$ denote the average slowdown of class i computed at sampling period k, and δ_i its differentiation parameter. For any two classes i and j, PSD requires

$$\frac{S_i(k)}{S_j(k)} = \frac{\delta_i}{\delta_j}, \quad 1 \le i, j \le N. \tag{1}$$

In this paper, we design and implement a feedback controller for PSD. Its basic structure is shown in Figure 1. The feedback controller is to adjust the rate allocation according to the difference between the target slowdown ratio and the achieved one using integral control. Thus, it is able to eliminate the steady-state error and to avoid over-reactions to measurement noises. We define the error associated with

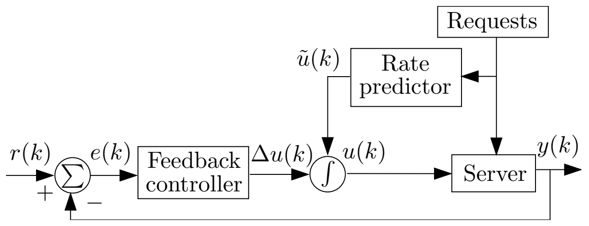


Figure 1: The structure of the feedback controller.

class i as

$$e_i(k) = r_i(k) - y_i(k) = \frac{\delta_i(k)}{\delta_1(k)} - \frac{S_i(k)}{S_1(k)}. \tag{2}$$

The processing rate of class i of sampling period $k+1$ then is

$$c_i(k+1) = \frac{1}{\left(\frac{c_1}{c_i} + g\int e_i(k)\mathrm{d}k\right)\cdot\sum_{i=1}^{N} 1/\left(\frac{c_1}{c_i} + g\int e_i(k)\mathrm{d}k\right)}, \tag{3}$$

where g is the control factor.

The initial processing rate of a class can be calculated by the rate predictor using queueing theory. Let c_i denote class i's allocated processing rate, λ_i its arrival rate, and $E[X]$ its mean service time. In [5], we presented a rate-allocation strategy in which the initial rate of class i is

$$c_i = \lambda_i E[X] + \frac{\lambda_i/\delta_i}{\sum_{i=1}^{N}\lambda_i/\delta_i}(1 - \sum_{i=1}^{N}\lambda_i E[X]). \tag{4}$$

The first part of the initial processing rate is a baseline that prevents the class from being overloaded so as to make the provisioning of PSD feasible. The second part is a portion of surplus processing rate determined by the differentiation parameter of the class and the load conditions (i.e., its normalized arrival rate) controls the quality differences between classes.

2. IMPLEMENTATION AND RESULTS

We implemented the feedback controller on Apache web server 1.3.31 running on Linux 2.6. Apache web server is normally executed with multiple processes (or threads). At the start-up, a parent server process sets up listening sockets and creates a pool of child processes. The child processes then listen on and accept client requests from these sockets. Since every child process in Apache web server is identical, in the implementation, we realize the processing-rate allocation by controlling the number of child processes that a

[*]This research was supported in part by U.S. NSF grant ACI-0203592.

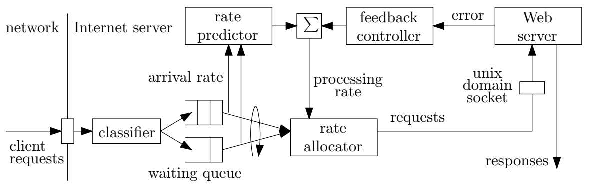


Figure 2: The implementation structure of the feedback controller.

class is allocated. The implementation structure of the feedback controller is presented in Figure 2. It is composed of a classifier, a rate predictor, a feedback controller, and a rate allocator.

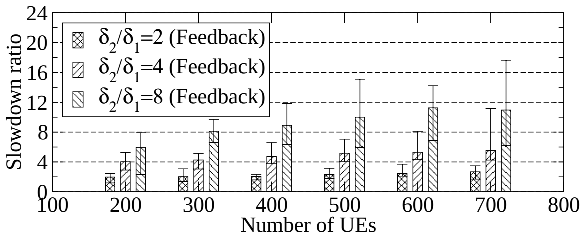
The classifier determines a request's class according to rules defined by service providers. In our implementation, the rules are based on the request's header information (e.g., IP address and port number). After being classified, a request is stored in its corresponding waiting queue. Associated with each waiting queue is a record of traffic information, such as arrival rate and service time. The arrival rate is measured directly from arrived client requests. The service time is returned from the web server and set by the rate allocator.

The rate predictor obtains the arrival rate and service time from waiting queues and estimates the processing rate that a class should be allocated according to (4).

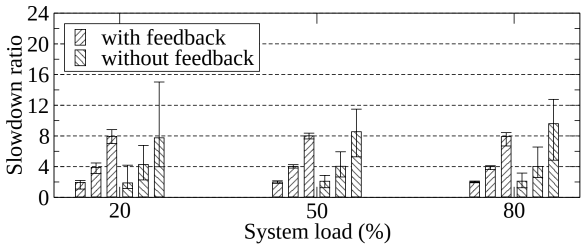
The feedback controller inferences the achieved slowdown ratios between different classes and calculates the processing rate of a class according to (3) by incorporating the rate estimation from the rate predictor. It then sets the number of allocated processes of a class in the rate allocator.

The rate allocator realizes the processing-rate allocation calculated by the feedback controller. In our implementation, the Apache web server is modified to accept requests from a unix domain socket. When a child process calls *accept()* on the unix domain socket, a signal is sent to the rate allocator. Upon receiving such a signal, the rate allocator scans the waiting queues to check if there is a backlogged class whose number of allocated processes is larger than that it is currently occupying. If such class exists, its head-of-line request and the class type are passed to the child process through the unix domain socket; otherwise, a flag is set. Whenever a new request arrives, the flag is checked first and the waiting queues are rescanned. The rate allocator also increases a counter that records the number of occupied processes by the class. After handling a request, the child process sends the request's service time and class type back to the rate allocator. The rate allocator then decreases the corresponding counter and stores the service time in the corresponding waiting queue.

The experimental environment consisted of four PCs running on a 100 Mbps Ethernet. Each PC was a Dell PowerEdge 2450 configured with dual-processor (1 GHz Pentium III) and 512 MB main memory. We installed one Apache web server on one PC while a commonly used web traffic generator SURGE [1] ran on other PCs. The workload of emulated requests was controlled by adjusting the total number of concurrent user equivalents (UEs) in SURGE. Notice that the fixed number of UEs does not affect the representativeness of the generated web traffic [1]. In addition, our experimental configurations are to emulate real-world heavy-load scenarios at a small scale.



(a) Feedback controller with different target ratios.



(b) Performance comparison with and without feedback control.

Figure 3: Effectiveness of the feedback controller.

We carried out experiments to evaluate the performance of the feedback controller under different target slowdown ratios, different workloads, and multiple classes. Figure 3(a) shows the 5th, 50th, and 95th percentiles of achieved slowdown ratios. From the figure we observe that the target ratios (2, 4, and 8) can be achieved under different workload conditions. We also observe that the variance of the achieved slowdown ratios is small in relation to the targets.

The agility of the feedback controller is affected by the control factor g. A controller with large control factor can respond quickly to errors. It, however, may cause large oscillations. We built a simulator to tune g and it was set to 1 in the experiments. The tuning process is presented in [4].

We also carried out simulations to compare the performance with feedback control and without feedback control under different system load conditions. Figure 3(b) depicts the percentiles of the results where the target slowdown ratio was set to 2, 4, or 8. From the figure we observe that variance is significantly decreased with the feedback controller. For example, when the system load is 50% and the target ratio is 4, the difference between the 95th and the 5th percentiles is 3.3 and 0.43 without and with feedback controller, respectively. It further demonstrates the effectiveness of the feedback controller.

3. REFERENCES

[1] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM Sigmetrics '98*.

[2] A. B. Downey. A parallel workload model and its implications for processor allocation. In *Proceedings of HPDC'97*.

[3] M. Harchol-Balter. Task assignment with unknown duration. *Journal of ACM (JACM)*, 49(2), 2002

[4] J. Wei, X. Zhou, and C. Xu. Robust Processing Rate Allocation for Proportional Slowdown Differentiation on Internet Servers. *IEEE Transactions on Computers*, 2005. In Press.

[5] X. Zhou, J. Wei, and C. Xu. Processing rate allocation for proportional slowdown differentiation on Internet servers. In *Proceedings of IPDPS*, 2004.