Feedback Control-based Database Connection Management for Proportional Delay Differentiation-enabled Web Application Servers

Wenping Pan, Dejun Mu, Hangxing Wu, Xinjia Zhang, Lei Yao

College of Automation, Northwestern Polytechnical University, Xi'an, Shaanxi 710072,China
paninxian@gmail.com

Abstract. As an important differentiated service model, proportional delay differentiation (PDD) aims to maintain the queuing delay ratio between different classes of requests or packets according to pre-specified parameters. This paper considers providing PDD service in web application servers through feedback control-based database connection management. To achieve this goal, an approximate linear time-invariant model of the database connection pool (DBCP) is identified experimentally and used to design a proportional-integral (PI) controller. Periodically the controller is invoked to calculate and adjust the probabilities for different classes of dynamic requests to use database connections, according to the error between the measured delay ratio and the reference value. Three kinds of workloads, which follow deterministic, uniform and heavy-tailed distributions respectively, are designed to evaluate the performance of the closed-loop system. Experiment results indicate that, the controller is effective in handling varying workloads, and PDD can be achieved in the DBCP even if the number of concurrent dynamic requests changes abruptly under different kinds of workloads.

1 Introduction

It has become an important issue for Internet servers to provide quality of service (QoS) guarantees to different network applications and clients. Many researchers have highlighted the importance of QoS guarantees in web servers under heavy-load conditions, and there has existed much work focusing on response delay guarantees in web servers [8][12][3][10]. Response delay is a key performance metric for web applications. From a client's perspective, response delay of a request includes three parts, connection delay, processing delay and communication delay. For a dynamic request, processing delay is an important part of users' perceived response time. Most dynamic requests use database connections for data access. Under heavy-load conditions, dynamic requests need to compete for limited number of database connections, which incurs the delay in the DBCP. In order to implement PDD in the DBCP, we divide dynamic requests into two different classes according to their priorities and design a controller to adjust the probabilities for these classes to get idle database connections from the pool. In this approach, requests with different priorities can be served with different delays.

The rest of the paper is organized as follows. Section 2 describes PDD in the DBCP. Section 3 presents the system identification, the controller design, as well as the system implementation and extension. Section 4 describes the experiments and gives the experimental results. Section 5 reviews related work and section 6 concludes the paper.

2 Proportional Delay Differentiation in Database Connection Pools

2.1 Connection Management in DBCP

It is a resource intensive and time consuming operation to open a database connection. As to a specified web application, if each http request opens and then closes a database connection for data access, a significant amount of processing time will be spent on the connection process, which obviously increases users' perceived response time. To solve these kinds of problems, DBCP has been widely adopted in web application servers. DBCP promotes the performance of web applications by reusing active connections rather than opening a new connection for each request. DBCP maintains a pool filled with active connections [11]. Once a new connection request comes in, after checking if there are any idle connections in the pool, DBCP returns one connection if true. If all connections in the pool are busy and the maximum pool size has not been reached, DBCP will create several new connections. When the pool reaches its maximum size, a newly incoming connection request will be queued up waiting for a connection available until the pre-specified waiting time is out, no matter how urgent or important the request is.

2.2 PDD in DBCP

Although DBCP reduces the response time for dynamic requests and enhances the performance of web application servers, it provides service only in a best-effort model and doesn't take the priorities of requests into account. For many business websites, service differentiation becomes necessary because web applications are deployed for online trading and e-commerce. There have existed many mechanisms [4] [17] [22]for service differentiation in web server end-systems, but none of them focuses on the delay differentiation in the DBCP for web application servers. In this paper, we achieve PDD in a DBCP using classical feedback control theory.

PDD, which was first proposed in [23], has gained much attention in recent years [2][20] [21][13][12][18]. The basic principle of PDD is that requests or packets with high priority will receive better performance compared with those with low priority. Suppose that requests or packets in networks can be classified into n classes. Let $\bar{d_i}$ be the average queuing delay of class i and δ_i be the specified delay differentiation parameter for class i. PDD aims to ensure that the delay ratio between class i and j equals the ratio between δ_i and δ_j, as is in Eq.(1). The class with a higher priority usually has a smaller delay differentiation parameter.

$$\frac{\bar{d_i}}{\bar{d_j}} = \frac{\delta_i}{\delta_j}, where\ 1 \leq i \leq n,\ 1 \leq j \leq n. \tag{1}$$

In this paper, dynamic requests in web application servers are classified into two classes, class A with high priority and class B with low priority. When there is no idle database connection in the DBCP, each class of requests will queue up to compete for the next database connection available, as is in Fig.2. According to PDD, the request from class A will get service in a smaller queueing delay than those from class B, and the average delay ratio between class A and class B will be kept as a constant value.

3 Design of A Feedback Controller

3.1 System Identification

As is shown in Fig. 2, let $\bar{d}_A(k)$ and $\bar{d}_B(k)$ denote the average queuing delays for class A and B in the k^{th} sampling period. Let $P_A(k)$ and $P_B(k)$ denote the probabilities for class A and B to get idle database connections in the k^{th} sampling period. Suppose that the DBCP can be approximately modeled as a m^{th} order linear time-invariant system, which can be described as

$$Y(k) = \sum_{i=1}^{m} [a_i Y(k-i) + b_i X(k-i)] \tag{2}$$

where

$$\begin{aligned} Y(k) &= \frac{\bar{d}_A(k)}{\bar{d}_B(k)} \\ X(k) &= \frac{P_B(k)}{P_A(k)} \\ P_B(k) + P_A(k) &= 1. \end{aligned}$$

We need to decide the order m and the parameter vector θ, i.e. $(a_1, \cdots, a_m, b_1, \cdots, b_m)^T$ of the model.

The test-bed is described in section 4. Experimental setup is as follows. The total number of worker threads is configured to be 100, and the pool size is set to be 20. Two client machines, one with high priority and the other with low priority, are started to simulate 50 real clients to send dynamic requests. Requests are classified into class A or B by the classifier according to their source IP, as is shown in Fig. 2. White noise input has been widely used for system identification. In our experiment, we generate a white noise input sequence according to

$$\epsilon(k) = [\epsilon(k-p) + \epsilon(k-q)] mod\, 2 \tag{3}$$

where $p = 8$, $q = 5$ and the sequence period is 255. At the k^{th} sampling instant, $X(k)$ is set to be 1 if $\epsilon(k) = 1$, or else 1.5. The experiment lasts for 40 minutes.

We calculate θ using the recursive least square (RLS) estimation algorithm [14]. According to RLS, θ can be calculated by Eq. (4).Suppose that $\theta_0 = 0$ and $P_0 = 15I$, we can calculate θ under different order $m(1 \leq m \leq 6)$.

$$\theta_{N+1} = \theta_N + \frac{P_N \varphi_{N+1}}{\varphi_{N+1}^T P_N \varphi_{N+1} + 1} Y_{\triangle}(N+m+1) \tag{4}$$

where

$$\begin{aligned} P_{N+1} &= P_N - \frac{P_N \varphi_{N+1} \varphi_{N+1}^T P_N}{\varphi_{N+1}^T P_N \varphi_{N+1} + 1} \\ \varphi_{N+1} &= (Y(N+m-1), \cdots, Y(N), \\ &\quad X(N+m-1), \cdots, X(N))^T \\ Y_{\triangle}(N+m+1) &= Y(N+m+1) - \varphi_{N+1}^T \theta_N \end{aligned}$$

And then we decide the order m using $F - test$ method [14]. We define a loss function $J(m)$, as is shown in Eq. (5), to describe the error between θ and the real parameter vector when the system order is m, and we also construct a statistic $V(n_1, n_2)$, as is shown in Eq. (6), to evaluate the variation of $J(m)$ when the system order is changed from n_1 to n_2. According to $F - test$, $V(n_1, n_2)$ follows the distribution $F(2(n_2 - n_1), L - 2n_2)$ when $n_2 > n_1 \geq m$ and L is large enough, where L is the length of experimental samples.

$$J(m) = \sum_{k=m+1}^{m+N} \left\{ Y(k) - \sum_{i=1}^{m} [a_i Y(k-i) + b_i X(k-i)] \right\}^2 \tag{5}$$

$$V(n_1, n_2) = \frac{J(n_1) - J(n_2)}{J(n_2)} \frac{N - 2n_2}{2(n_2 - n_1)} \tag{6}$$

Through the experiment, we get that, when $n_2 = 2$ and $n_1 = 1$,$V(1, 2) < F_{0.05}(2, 80)$ holds with the confidence of 95%. It means that, when the system order is changed from 1 to 2, there is no significant reduction of the loss function. So $m = 1$, and the DBCP can be modeled as a first order linear time-invariant system with a parameter vector

$$\theta = (-0.0172, 0.7463)^T. \tag{7}$$

3.2 Controller Design

We design a PI controller for the approximate linear model to implement PDD in a DBCP. Integral control is able to eliminate the steady state error and PI controller is easy to be implemented in programme. We can use transfer functions, as are shown in Eq. (8)(9)(10), to describe the PI controller, the linear model given by Eq. (2), and the closed-loop system, which is shown in Fig.1. Performance specifications the closed-loop system should meet are as follows. The steady state error is zero and the settling time is no more than 300 seconds.

$$D(z) = K_P + \frac{K_I T(z+1)}{2(z-1)} \tag{8}$$

$$G(z) = \frac{\sum_{i=1}^{m} b_i z^{m-i}}{z^m - \sum_{i=1}^{m} a_i z^{m-i}} \tag{9}$$

$$G_C(z) = \frac{D(z)G(z)}{1 + D(z)G(z)} \tag{10}$$

$$\begin{aligned} \triangle x(k) &= x(k) - x(k-1) \\ &= K_P[(1 + \frac{TK_I}{K_P})e(k) - e(k-1)] \end{aligned} \tag{11}$$

To construct a system that satisfies the pre-specified performance, we use Root Lotus tool in MATLAB to place the closed-loop poles and get the parameters K_P and K_I for the PI controller. According to the incremental algorithm, we can get the output increment of the controller at the k^{th} sampling time by Eq. (11) and finally we get $\triangle x(k) = 0.42e(k) - 0.1e(k-1)$.

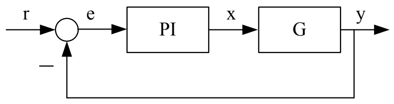

Fig. 1. Feedback control diagram with a PI controller

3.3 The Closed-loop System

We give a description of the closed-loop system components from the perspective of control theory. As is shown in Fig.2, at the k^{th} sampling instant, the monitor is invoked to calculate the average queueing delays $\bar{d}_A(k)$ and $\bar{d}_B(k)$ during the last sampling interval for two classes of requests. Then the controller compares the delay ratio $\frac{\bar{d}_A(k)}{\bar{d}_B(k)}$ with the desired value $\frac{\delta_A}{\delta_B}$, and calculates a new probability ratio $\frac{P_B(k)}{P_A(k)}$ according to the error measured. According to Eq.(11), we can get $\frac{P_B(k)}{P_A(k)} = \triangle x(k) + \frac{P_B(k-1)}{P_A(k-1)}$. Suppose that $\frac{P_B(0)}{P_A(0)} = 1$, we can get $\frac{P_B(k)}{P_A(k)} = 1 + \sum_{i=1}^{k} \triangle x(i)$. The scheduler acts as an actuator. Once a database connection is available, the scheduler generates a random sample r from the uniform distribution $U(0,1)$. If $r \leq P_A(k) = \frac{1}{2+\sum_{i=1}^{k} \triangle x(i)}$, class A gets the connection, or else class B. To reduce the overhead for generating random samples, a sequence of random samples S, can be generated and stored in the scheduler before it works.

3.4 System Implementation and Extension

We firstly introduce the implementation of the closed-loop system based on the Tomcat application server [1]. As a standard JSP/Servlet container, Tomcat supports data access using JSP pages and java classes of Servlet based on DBCP. When a dynamic request, which is mapped to a JSP page or a Servlet class, calls the function $getConnection()$, the classifier puts the request into a virtual queue, Queue A or Queue B, as is in Fig.2, according to its source IP. At the same time the monitor records its arrival time at the queue. When a database connection becomes available, the scheduler decides which queue will get the connection and makes the function $getConnection()$ return from the queue. When the request at the queue head gets the connection available, the monitor will record its departure time and get its queueing delay. When a request finishes data access by calling $closeConnection()$, a connection immediately becomes available for reusing. Periodically, the monitor will recalculate $\frac{\bar{d}_A(k)}{\bar{d}_B(k)}$, and the PI controller, which

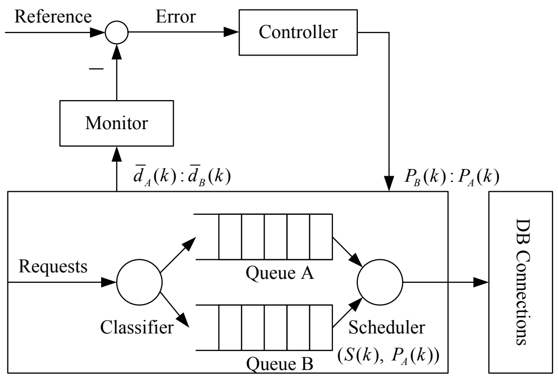

Fig. 2. Control loop for PDD

is implemented as a java thread, will changes $\frac{P_B(k)}{P_A(k)}$ according to Eq.(11). Member functions, such as $SetReferenceValue()$, are also implemented in the DBCP. All of our implementation, which just needs to modify the code of the DBCP, is transparent to web application developers and website managers. It also brings convenience for us to deploy the DBCP in other web application servers.

Secondly, we discuss how our implementation for two classes of requests can be extended to the general case for n $(n \geq 2)$ classes. We can divide n classes into $n-1$ overlapped groups where group $j(1 \leq j \leq (n-1))$ includes class j and $j+1$. Each group has a PI controller and some pre-specified delay differentiation parameters. PDD for each group is the case for two classes, just as described in above. Suppose that at the k^{th} sampling time, the average delay for class $i(1 \leq i \leq n)$ is $\bar{d}_i(k)$, and the probability for class i to get connections is $P_i(k)$ $(\Sigma_{i=1}^{n} P_i(k) = 1)$, the controller output $x_j(k+1)$ for group j can be calculated according to Eq.(11). We can get $P_1(k+1), \ldots, P_n(k+1)$ by solving Eq.(12).In this way, the DBCP can provide PDD service for more than two classes of dynamic requests.

$$\begin{cases} \frac{P_j(k+1)}{P_{j+1}(k+1)} = x_j(k+1), \quad (1 \leq j \leq (n-1)) \\ \Sigma_{i=1}^{n} P_i(k+1) = 1 \end{cases} \tag{12}$$

Thirdly, we discuss how to reduce the average delay for each service class by using the shortest-job first (SJF) scheduling policy. In the closed-loop system in above, dynamic requests of each class are served in a first-come-first-serve (FCFS) manner. For static requests, it has been proven that shortest-remain-processing-time (SRPT) scheduling policy can reduce the mean response time by nearly a factor of ten [25]. However, job sizes for dynamic requests are always unknown in advance and many dynamic requests can not be interrupted. As a result, SRPT scheduling policy can not be used directly in the DBCP. In [24], SJF scheduling policy was implemented for web sites interactions processing. In our implementation of PDD in the DBCP, the sizes of dynamic requests are generated in advance and SJF scheduling policy may bring performance improvement for dynamic requests processing when providing PDD service meanwhile.

4 Experiments

4.1 Test-bed and Workloads

Our test-bed consists of a dispatcher, a back-end server, and three client machines, each with a 2.80GHz Pentium processor and 512 MB RAM. Three client machines run Linux-2.4.18 and generate web traffic using the modified SURGE [7] workload generator, which can simulate a number of real-world clients to send dynamic requests. The back-end server runs Tomcat 5.5.17 and Oracle $9i$ for handling dynamic requests. The feedback control-based DBCP works in Tomcat. On the dispatcher machine, Apache 2.0.53 works as a load balancer and uses Mod_jk [1]communicating with the back-end server. In such a scalable architecture, static requests can be processed by the cache module in Apache, and more back-end servers can be added to share the dynamic workload of the whole system according to pre-specified load-balancing strategies.

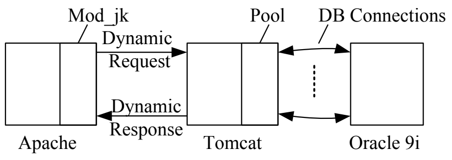

Fig. 3. Architecture of the server system

Previous researchers have illustrated that, the processing time of a static request is approximately linear with the size of the requested file [3], which follows the well-known heavy-tailed distribution. However, it is difficult to find out the distribution of the processing time for dynamic requests because most of them are CPU-intensive or I/O-intensive. For this reason, we designed three kinds of workloads for our experiments. The first workload designed follows a deterministic distribution, where all the dynamic requests have the same processing time, 350ms. The second one fits a uniform distribution, where the processing time of a request ranges from 0 to 700ms. The third one follows a bounded Perato distribution, as is in Eq. (13), which is a typical heavy-tailed distribution with an upper bound. In practice, we generate the third workload using an equivalent bimodal distribution [16], as is in Eq. (14) where $x_a = 50$, $x_b = 6050$, $\alpha = 0.95$, that corresponds to the bounded Pareto distribution.

$$\begin{array}{r} F(x) = \frac{1-(m/x)^{\gamma}}{1-(m/M)^{\gamma}} \\ where\ M \gg m,\ M \geq x \geq m,\ \gamma \in (0,2) \end{array} \tag{13}$$

$$\begin{array}{r} f(x) = \alpha\delta(x-x_a) + (1-\alpha)\delta(x-x_b) \\ where\ \alpha \approx 1,\ \delta(x) = \begin{cases} 1,\ x=0; \\ 0,\ else \end{cases} \end{array} \tag{14}$$

4.2 Experimental Setup and Results

Three kinds of experiments are conducted to evaluate the performance of the closed-loop system. Firstly, we want to compare the impacts of different sampling periods on the closed-loop system and choose the best sampling period T. From the perspective of control theory, the settling time of a closed-loop discrete system is related to its sampling period. In principle, a smaller period leads to a shorter settling time. However, a too small sampling period may make the system enter an oscillatory state and cannot settle down. The experimental setup is as follows. The total number of worker threads is configured to be 100, and the pool size is set to be 15. The reference value is 0.5, i.e. $\frac{\bar{d}_A}{\bar{d}_B} = \frac{1}{2}$. We conduct experiments three times under the uniform workload, and each time with a different sampling period. At the beginning of each experiment, two client machines are started to simulate 50 clients to generate dynamic requests, one with high priority and the other with low priority. The third client machine with low priority is started at 200 seconds to send requests for 10 minutes, which simulates 100 clients to generate bursty traffic. Each experiment lasts for 30 minutes.

Fig.4 shows the results under different sampling periods. When the sampling period T is changed from 10 seconds to 15 seconds, the measured curve of delay ratio becomes much more smooth. But when T is changed from 15 seconds to 20 seconds, there is no significant improvement. To make a tradeoff between stability and response rate, we select 15 seconds as the sampling period for the rest of our experiments.

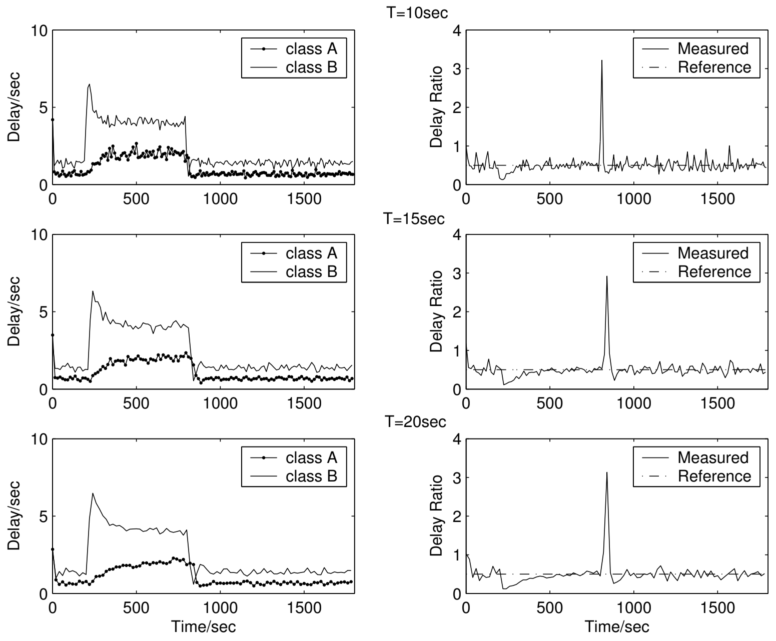

Fig. 4. PDD under different sampling periods

Secondly, we focus on PDD in the DBCP under different kinds of workloads. Parameters of the closed-loop system are configured as in the first kind of experiments. We conduct two experiments, which are under deterministic and heavy-tailed workload respectively. All client machines work as in the first kind of experiments, and there is also a traffic burst during 200 seconds and 800 seconds generated by the third machine. Each experiment lasts for half an hour.

Fig.5 shows the results under different workloads. The figure also includes the result from the first kind of experiment under uniform workload. The DBCP achieves PDD successfully under different workloads, although the average delay for each class is fluctuated all the time. Compared with the other two workloads, heavy-tailed workload makes the average delay for each class vary much more quickly. That is maybe the result of workload distribution. According to Eq. (14), the size of large requests is nearly 121 times of the size of small ones. A large request will significantly increase the service demand in the web application server. However, when the number of concurrent requests changes, the controller reacts quickly to the load variation and ensures that requests with high priority are served with small delays, no matter under what kind of workload.

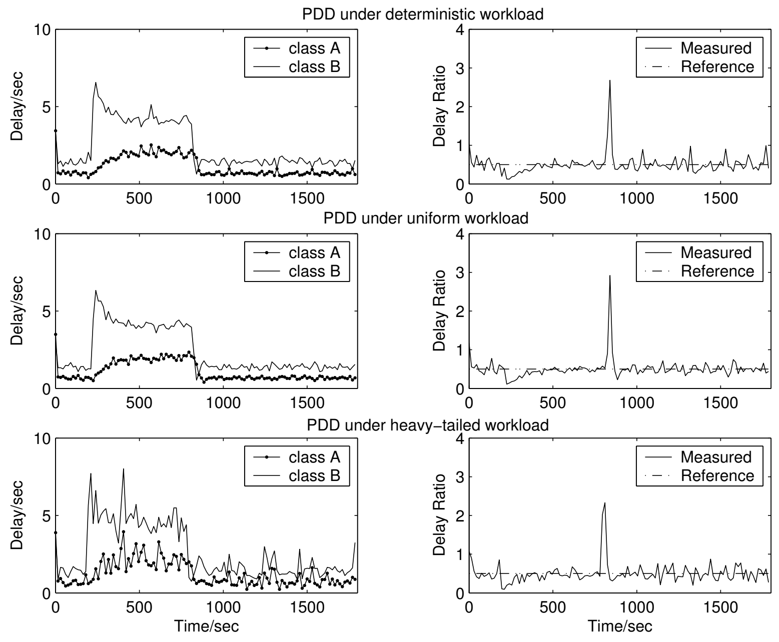

Fig. 5. PDD under different workloads

Thirdly, we want to compare the performance of the closed-loop systems with different scheduling policies. The experiment lasts for 20 minutes. As can be seen from Fig.6, under the uniform workload, the average delay for each service class with

SJF scheduling policy is smaller than that with FCFS scheduling policy. The feedback control-based DBCP is capable of providing PDD service no matter what kind of scheduling policy is used.

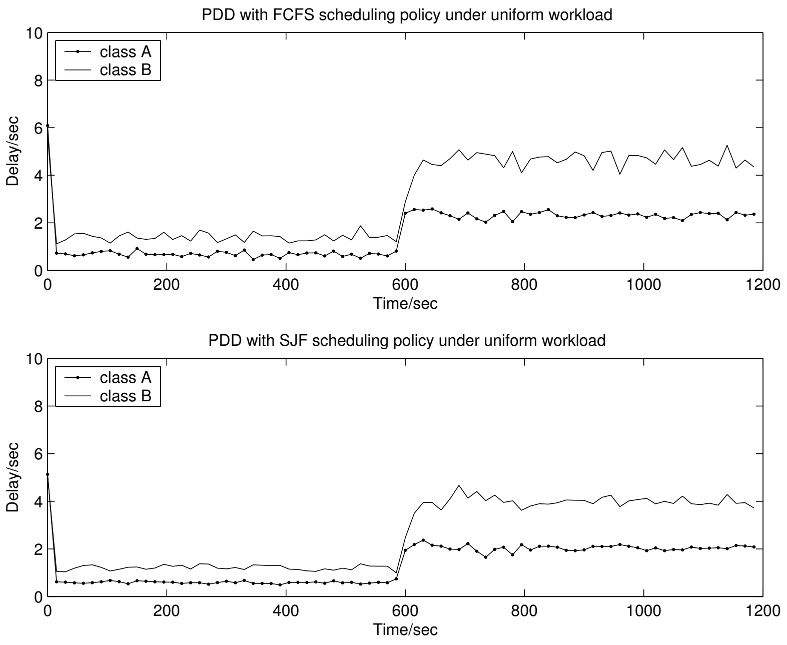

Fig. 6. PDD with different scheduling policies

5 Related Work

Many researchers have highlighted the importance of integrating resource management with quality of service in server systems. In [5], a CPU scheduling algorithm has been proposed to dynamically allocate CPU cycles to Apache processes. In [6], resource containers were proposed as as a kernel mechanism to provide service differentiation by accurate accounting of resource usage. In [8], an observation-based approach was proposed for QoS guarantees at the kernel level by handling bottleneck resources, the CPU cycles and the accept queue. Other kernel-based resource allocation mechanisms for service differentiation can be found in [17]. In [4], a feedback control framework was proposed to guarantee relative/absolute delay in web servers at the connection level and controllers were designed to allocate service threads to clients with different priorities. To reduce latencies and overhead from closing and re-establishing connections, persistent connections are supported as a default by HTTP/1.1. A persistent connection can transmit a sequence of requests, so connection delay just affects the response time of the first request over the connection and request level QoS becomes necessary. Many researchers focused on other QoS metrics e.g. relative hit ratio of web cache [15], relative rejection ratio of requests [9] and system slowdown [19].

Our solution differs from the above works in many respects. Firstly, most of their work only addresses workloads with static requests, whereas this paper fundamentally focuses on dynamic requests and database-driven websites. Secondly, compared with existing work, our work focuses on providing differentiated service at a request level rather than a connection level, and important dynamic requests can get high priority when being handled. Thirdly, in our solution, we implement the controller in the DBCP rather than in the kernel of an operating system, and the DBCP can be deployed in other web application servers besides Tomcat conveniently.

6 Conclusion

It is a great challenge for Internet servers to provide service differentiations in an unpredictable and highly-dynamic environment. Proportional differentiated service is an important service model and response delay is the key performance metric for web application servers. This paper describes the approach for proportional delay differentiations in web application servers through feedback control-based database connection management. We implement a PI controller in a real DBCP for web application servers and design three kinds of workloads for simulation of the closed-loop system. We experimentally demonstrate that, the controller is effective in handling different kinds of workloads and the feedback control-based DBCP is capable of providing service differentiation. Feedback control theory presents its potential for better resource management and QoS guarantees in web application servers.

References

1. The apache tomcat document. http://tomcat.apache.org.
2. M. Abbad and T. Zahratahdi. An algorithm for achieving proportional delay differentiation. Operations Research Letters, 36(2):196-200, March 2008.
3. T. F. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server endsystems: A control-theoretical approach. IEEE Transactions on Parallel and Distributed Systems, 13(1):80-96, 2002.
4. T. F. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu. Feedback performance control in software services. IEEE Control Systems, 23(3), June 2003.
5. J. Almeida, M.Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of service in web content hosting. Proceedings of the SIGMETRICS Workshop on Internet Server Performance, 1998.
6. G. Banga, P.Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. Proceedings of the Symposium on Operating Systems Design and Implementation, 1999.
7. P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. Measurement and Modeling of Computer Systems, pages 151-160, 1998.
8. A. Chandra, P.Pradhan, R. Tewari, S. Sahu, and P. Shenoy. An observation-based approach towards self-managing web servers. Computer Communications, pages 1-15, 2005.

9. C. Huang, C. Cheng, Y. Chuang, and J. R. Jang. Admission control schemes for proportional differentiated services enabled internet servers using machine learning techniques. Expert Systems with Applications, 31:458-471, 2006.
10. V. Kanodia and E. W. Knightly. Ensuring latency targets in multiclass web servers. IEEE Transaction on Parallel and Distributed Systems, 13(10), October 2002.
11. D. Lea. Concurrent Programming in Java: Design Principles and Patterns (Second Edition). Addison Wesley Longman, Inc, 2000.
12. S. C. Lee, J. C. Lui, and D.K. Yau. A proportional-delay diffserv-enabled web server: Admission control and dynamic adaptation. IEEE Transactions on Parallel and Distributed Systems, 15(5):385-400, May 2006.
13. K. M. Lim,J. Paik, J. dong Ryoo, and S.-S. Joo. Prediction error adaptation of input traffic for absolute and proportional delay differentiated services. Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks, pages 38-47, August 2006.
14. L. Ljung. System Identification: Theory for the User. Tsinghua University Press, Beijing China, 2002.
15. Y. Lu, T. F. Abdelzaher, and A. Saxena. Design, implementation, and evaluation of differentiated caching services. IEEE Transactions on Parallel and Distributed Systems, 15(5):440-452, May 2004.
16. K. Psounis, P. Molinero-Fernndex, B. Prabhakar, and F. Papadopoulos. Systems with multiple servers under heavy-tailed workloads. Performance Evaluation, 62(7):456-474, July 2005.
17. T. Voigt, R. Tewari, and D. Freimuth. Kernel mechanisms for service differentiation in overloaded web servers. Proceedings of the Usenix Annual Technical Conference, 2001.
18. K. C. Wang and P. Ramanathan. End-to-end throughput and delay assurances in multihop wireless hotspots. Proceedings of the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspots, pages 93-102, September 2003.
19. J. Wei and C. Xu. Design and implementation of a feedback controller for slowdown differentiation on internet server. WWW2005, pages 10-14, May 2005.
20. J.Wei and C. Z. Xu. Consistent proportional delay differentiation: A fuzzy control approach. Computer Networks, 51(8):2015-2032, June 2007.
21. C. C. Wu, H. M. Wu, and W. Lin. High-performance packet scheduling to provide relative delay differentiation in future high-speed networks. Computer Networks, December 2007.
22. N. Ye, E. S. Gel, X. Li, T. Farley, and Y. Lai. Web server qos models: applying scheduling rules from production planning. Computers and Operations Research, 32(5):1147-1164, 2005.
23. Constantinos Dovrolis, Dimitrios Stiliadis, Parameswaran Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. Proceedings of the ACM SIGCOMM, 10(1):12-26, 1999.
24. Sameh Elnikety, Erich Nahum, John Tracey and Willy Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. WWW2004, pages 276-286, May 2004.
25. Mor Harchol-balter, Bianca Schroeder, Nikhil Bansal and Mukesh Agrawal. Size-Based Scheduling to Improve Web Performance. ACM Transactions on Computer Systems, 21(2): 207-233, 2003.