A Proportional-Delay DiffServ-Enabled Web Server: Admission Control and Dynamic Adaptation

Sam C.M. Lee, John C.S. Lui, *Senior Member*, *IEEE*, and David K.Y. Yau, *Member*, *IEEE*

Abstract—We consider a Web server that can provide differentiated services to clients with different quality of service (QoS) requirements. The Web server can provide $N \geq 1$ classes of proportional-delay differentiated services (PDDS) to heterogeneous clients. An operator can specify *fixed* performance spacings between classes, namely, $r_{i,i+1} > 1$, for $i = 1, \ldots, N-1$. Requests in class $i+1$ are guaranteed to have an average waiting time which is $1/r_{i,i+1}$ of the average waiting time of class i requests. With PDDS, we can provide consistent performance spacings over a wide range of system loading and this simplifies many pricing issues. In addition, each client can specify a maximum average waiting time requirement to be guaranteed by the PDDS-enabled Web server. We show that, in general, the problem of assigning clients to service classes in order to optimize system efficacy is NP-complete. We propose two efficient admission control algorithms so that a Web server can provide the QoS guarantees and, at the same time, classify each client to its "lowest" admissible class, resulting in lowest usage cost for the admitted client. We also consider how to perform end-point dynamic adaptation such that admitted clients can submit requests at a lower class and further reduce their usage costs without violating their QoS requirements. We propose two dynamic adaptation algorithms: one is server-based and the other is client-based. The client-based adaptation is distributed and is based on a noncooperative game technique. We carry out experiments to illustrate the effectiveness of these algorithms under different utility functions and traffic arrival patterns (e.g., Poisson, MMPP, and Pareto). We report extensive experimental results to illustrate the effectiveness of our proposed algorithms.

Index Terms—Proportional differentiated service, admission control, dynamic adaptation, performance evaluation, quality of service.

1 Introduction

The Internet is becoming more commercially oriented and businesses are now using Web servers to disseminate information. Therefore, the effect of access latency at Web servers has become more important. Conventional Web servers use a single class approach in serving client requests. This does not provide adequate performance when different clients may have different QoS requirements and are willing to pay different prices to attain their desired QoS. Hence, there is a need to support *multiple* classes of service at a Web server in order to extend network level service differentiation (e.g., the DiffServ model) to true end-to-end *application level* service differentiation.

There are several ways for a Web server to provide differentiated services. For example, a strict priority policy can be used in which clients submit requests in different priority classes and the Web server always serves the next request from the highest priority class that is backlogged. Some drawbacks of the strict priority policy are 1) the possibility of starvation for requests in the lower priority classes and 2) the performance spacings between different classes are *load dependent*, introducing pricing complication. For example, if a client X is charged at a rate of R_1 and another client Y is charged at a rate of R_2, where $R_2 > R_1$, then Y should expect its performance to be *proportionately* better than that of X (i.e., the performance of Y is R_2/R_1 that of X), regardless of system loading. This type of performance guarantees cannot be easily achieved with a strict priority policy.

We target a Web server which can provide a differentiated service that has the following properties:

- *Consistency:* service differentiation is consistent (i.e., higher classes receive better service) and the performance differentiation is *independent* of variations in class load.
- *Controllability:* the operator of the Web server can specify and control the performance spacings between offered classes of service according to the pricing structure.

In [10], the authors propose an Internet service model called proportional-delay differentiated services, which has the above mentioned consistency and controllability properties. In the service model, the performance spacing between class $i+1$ and class i can be specified as a *fixed* ratio $r_{i,i+1}$. If this ratio can be maintained over a wide range of system loading, then a user of class $i+1$, who is paying at a rate $r_{i,i+1}$ higher than a user of class i, will consistently have a performance that is $r_{i,i+1}$ better than the class i user. To realize proportional-delay differentiated services, the authors in [10] propose to use the *time-dependent priority* (TDP) service discipline. In [15], [16], the authors illustrate the necessary and sufficient conditions under which the

- *S.C.M. Lee and J.C.S. Lui are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: {cmlee, cslui}@cse.cuhk.edu.hk.*
- *D.K.Y. Yau is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907. E-mail: yau@cs.purdue.edu.*

Manuscript received 18 Dec. 2002; revised 28 Aug. 2003; accepted 17 Oct. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 118016.

controllability and consistency properties can be maintained when the requests are of Poisson arrivals.

In this paper, we consider proportional-delay differentiated services at a Web server, say $\mathcal{S}$. $\mathcal{S}$ provides waiting time differentiation for $N > 1$ classes of requests. Let W_i be the expected waiting time of class i requests, for $i = 1, \dots, N$. The operator of the Web server $\mathcal{S}$ specifies a fixed performance spacing $r_{i,i+1} > 1$ such that

$$W_i/W_{i+1} = r_{i,i+1} \qquad \text{for } i = 1, 2, \dots, N-1.$$

For example, if $r_{i,i+1} = 1.5$, then the operator can legitimately charge class $i+1$ clients a usage rate 50 percent higher than that of class i clients. In addition to this performance spacing guarantee, each client specifies a maximum average waiting time for its requests to be guaranteed by $\mathcal{S}$. We consider the following technical issues.

- Efficient admission control so that $\mathcal{S}$ can provide the requested performance differentiation and guarantees.
- Efficient assignment or mapping of client requests into the service classes so that an admitted client's performance requirement can be satisfied.
- Dynamic adaptation such that, depending on the server workload, a client can assign requests to a lower service class (i.e., lower than the class which was initially prescribed at admission control time) and can still receive service consistent with its performance requirement. This way, a client can pay a lower usage cost while still obtaining satisfactory service.

The balance of the paper is organized as follows: In Section 2, we provide the necessary background of proportional-delay differentiated services. We also formulate the problem of admission control, client classification, and dynamic adaptation. We show that, in general, the client classification problem to optimize system efficacy is NP-complete. In Section 3, we present two efficient admission control algorithms and state their important properties. In Section 4, we present two adaptation algorithms: One is server-based (i.e., a centralized algorithm) while the other is client-based (i.e., a distributed algorithm). The client-based algorithm is based on a *noncooperative* game approach and has a low computational complexity. In Section 5, we present experimental results to illustrate the effectiveness of the proposed algorithms. Related work is presented in Section 6. Section 7 concludes.

2 Background and Problem Formulation

Let us present the background of proportional-delay differentiated services (PDDS) [10], [15], [16]. Under PDDS, there are $N > 1$ service classes such that class $i+1$ requests will receive better performance compared with class i requests, for $i = 1, \dots, N-1$. In general, the response time of a request at a Web server consists of the waiting time as well as the corresponding service time. The waiting time of a request is the time the request spends in the server's queue before it receives service. For a popular Web server,

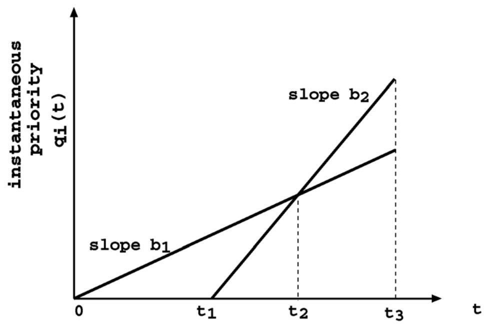


Fig. 1. A two-class TDP where $b_1 < b_2$.

the aggregate traffic loading can be very high and waiting time contributes a significant portion of the request's response time.[1] Controlling the waiting time will, therefore, be highly useful in providing service differentiation. Let W_i be the achieved long-term average waiting time of class i requests. A PDDS Web server tries to guarantee that the ratio of the achieved long-term average waiting time between classes i and $i+1$ is equal to a *fixed* and *prespecified* ratio, $r_{i,i+1}$, where

$$W_i/W_{i+1} = r_{i,i+1} \qquad \text{for } i = 1, \dots, N-1. \tag{1}$$

The objective is to maintain $r_{i,i+1} > 1$ across a wide range of system loading. As mentioned, PDDS can be achieved using the time-dependent priority (TDP) scheduler [10]. In general, TDP is a *nonpreemptive* priority scheduling algorithm with a set of control variables $b_i, 1 \le i \le N$, where $0 \le b_1 \le b_2 \le \dots \le b_N$. The control variable b_i dictates the *instantaneous* priority of a class i request. Specifically, if the kth request of class i arrives at time τ_k, then its priority at time t (for $t \ge \tau_k$), denoted by $q_i^k(t)$, is

$$q_i^k(t) = (t - \tau_k)b_i. \tag{2}$$

To clearly illustrate the concept, we use a two-class TDP as depicted in Fig. 1. Assume that the first request of class 1 arrives at time 0 and the first request of class 2 arrives at time t_1. Both requests remain in the system until time t_3. During the time interval $(t_1, t_2]$, the class 1 request will have a higher priority than the class 2 request. But, since the control parameter b_2 is larger than b_1, after time $t > t_2$, the class 2 request will have a higher priority. Because of this property, requests in higher classes cannot monopolize the system resources and cause the starvation problem.

Let $N_i(t)$ denote the number of class i requests waiting in the queue at time t and $q_i(t)$ the priority of the request at the head of the class i queue. When a Web server $\mathcal{S}$ is ready to service a request at time t, it chooses a request from class i^*, where

$$i^*(t) = arg \max_{i=1..N, N_i(t)>0} \{q_i(t)\}. \tag{3}$$

1. To be precise, response time should also include any queueing and transfer delay along the communication path. However, since the communication overhead cannot be controlled by the Web server, we will not consider it in this paper. Please refer to [16] on how to provide proportional-delay differentiated service for the underlying communication network.

Ties for the highest priority are broken by serving the request that has been waiting the longest in the system. If there is no request in the system, the server is idle and will be activated by any newly arriving request. Note that for the TDP scheduler, a class i request increases in priority at a faster rate than requests of any class j, where $j < i$. In [15], [16], the authors derive the *necessary* and *sufficient* conditions for feasible delay ratios when the requests are of Poisson arrivals. Specifically, for a two-class system, if the system loading ρ satisfies $1 - 1/r_{1,2} < \rho < 1$, then by setting the control parameters $b_1 = 1$ and $b_2 = \rho/(\rho - 1 + \frac{1}{r_{1,2}})$, one can achieve the desired waiting time spacing. For a system with more than two classes of traffic, the authors give the necessary conditions for feasible spacings, and an efficient iterative algorithm for determining the values of the control parameters b_i, $i = 1, \ldots, N$. Please refer to [15], [16] for a detailed derivation of these parameter values.

Consider the utility of a PDDS-enabled Web server offering, say, video-on-demand service. In this case, a class i client who wants to access a video will experience a smaller start-up latency than a client in class $i-1$. In exchange, the class i client will be charged at a higher usage rate than the class $i-1$ client. Our focus is on providing *fundamental understanding* for the design of such a PDDS-enabled Web server.

Assume that there are $M > 0$ potential clients requesting service from a PDDS-enabled Web server $\mathcal{S}$. Each of these clients can be viewed as an aggregation of many individual users (e.g., users from the same company or organization). A client, say j, specifies two parameters for its desired QoS:

- λ_j^{max}: j's maximum offered traffic rate to the server.
- W_j^{max}: the maximum average waiting time for client j's requests before service is obtained.

If a client is admitted to the system and is assigned to class i, the client is charged an admission cost of A_i, where $A_1 \le A_2 \le \cdots \le A_N$. $\mathcal{S}$ also charges a usage cost of ϕ_i for each request in class i, where $\phi_1 \le \phi_2 \le \cdots \le \phi_N$.

The problems we want to address are:

1. *Admission control and class assignment:* Given the workload (λ_j^{max}) and the QoS requirement (W_j^{max}) of client j requesting service, should $\mathcal{S}$ admit this client such that the QoS requirements of all the admitted clients will be satisfied? Also, when a system decides to admit client j, what is the *lowest* possible class assignment for j, such that j will pay the lowest possible usage cost?
2. *Dynamic class adaptation:* For those admitted clients, their request arrival rates may be less than their specified maximum request arrival rates. Therefore, rather than use the assigned class obtained during the admission control process, a client may choose to submit requests at a lower class. This way, the client may enjoy its desired level of service at a reduced usage cost. We consider the problem of how each client can adapt to the traffic loading at a Web server $\mathcal{S}$ and adjust its service class dynamically. The main challenge is to guarantee that we will not violate the maximum average request waiting time required by the client.

Before we proceed, let us define the following notation. Let M' be the number of clients admitted to $\mathcal{S}$. In general, we have $M' \le M$. The admitted class vector, denoted by $\boldsymbol{C}^a = [C_1^a, C_2^a, \ldots, C_{M'}^a]$, represents the class assignment of each admitted client after the admission control and class assignment process.[2] The class assignment for client i is $C_i^a \in \{1, 2, \ldots, N\}$ for $i = 1, \ldots, M'$. After the admission control, an admitted client may dynamically adapt to the loading at $\mathcal{S}$ and lower its assigned class. The class vector at time $t > 0$ for all the admitted clients is denoted by $\boldsymbol{C}(t) = [C_1, C_2, \ldots, C_{M'}]$, where $C_i \in \{1, \ldots, N\}$ is the class chosen by client i. It is easy to observe that $\boldsymbol{C}(0) = \boldsymbol{C}^a$ and $\boldsymbol{C}(t) \le \boldsymbol{C}^a$ for $t > 0$. The total maximum arrival rate of class k, denoted as $\lambda_{c_k}^{max}$, is:

$$\Lambda_{c_k}^{max} = \sum_{j=1}^{M} \lambda_j^{max} \mathbf{1}\{C_j = k\} \quad k = 1, 2, \ldots, N.$$

Let W_{c_k} be the average waiting time of class k requests. Based on the conservation law in queueing system, we have:

$$\sum_{k=1}^{N} \Lambda_{c_k}^{max} W_{c_k} = \sum_{k=1}^{N} \Lambda_{c_k}^{max} W(\lambda), \tag{4}$$

where $W(\lambda)$ represents the average waiting time that would result if the aggregate traffic were serviced by a work-conserving FCFS server of the same capacity as $\mathcal{S}$. Define $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$ for $i = 2, \ldots, N$. Based on (1), we have $r_{i-1,i} = W_{c_{i-1}}/W_{c_i}$. Therefore,

$$\sigma_i = W_{c_i}/W_{c_1} \quad i = 1, \ldots, N.$$

Based on the above equation, we can express W_{c_i} in terms of W_{c_k} as:

$$W_{c_i} = \sigma_i \frac{W_{c_k}}{\sigma_k} \quad i = 1, 2, \ldots, N. \tag{5}$$

Substituting (5) into (4), we have

$$\frac{W_{c_k}}{\sigma_k} \sum_{i=1}^{N} \Lambda_{c_i}^{max} \sigma_i = \sum_{i=1}^{N} \Lambda_{c_i}^{max} W(\lambda).$$

After rearranging terms, we can express W_{c_k} as

$$W_{c_k} = \frac{\sigma_k \Big(\sum_{j=1}^{N} \Lambda_{c_j}^{max} \Big) W(\lambda)}{\sum_{j=1}^{N} \sigma_j \Lambda_{c_j}^{max}} \quad \text{for } k = 1, \ldots, N. \tag{6}$$

Let U_i be a function representing the utility of client i. Each client can have a different utility function. In this paper, the utility function we consider has a form which is illustrated in Fig. 2. Let S_i be the cost of client i. The net utility of client i is then $U_i - S_i$. For the server, the utility is the amount of cost paid by all admitted clients, i.e. $\sum_{i=1}^{M'} S_i$. Let R be the maintenance cost of the server. We define the system efficacy V as the sum of the net utilities of all the admitted clients and the Web server. Our objective is to

2. In this paper, we assume that the system will assign a class value of 0 to those clients that the system cannot admit.

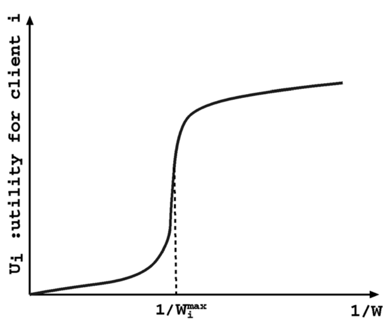


Fig. 2. General form of client's utility function versus inverse of the waiting time.

$$\begin{aligned}\max V &= \sum_{i=1}^{M'}(U_i(W_{c_k}) - S_i) + (\sum_{i=1}^{M'} S_i - R) \\ &= \sum_{i=1}^{M'} U_i(W_{c_k}) - R \\ \text{s.t.} \quad & W_{c_k} \le W_i^{max} \quad i = 1, \ldots, M' \text{ and } k = C_i^a,\end{aligned} \tag{7}$$

i.e., we seek to maximize the system efficacy V under the constraint that the expected waiting time of an admitted client i is less than or equal to its QoS requirement W_i^{max}. In [17], the authors show that, if a request has a utility function with the form similar to Fig. 2, then one needs to apply admission control to maximize the system efficacy V. In the following, we show that the above optimization problem is NP-hard.

Theorem 1. *The constructed optimization problem given in (7) is NP-hard.*

Proof. Consider the decision version of the optimization problem in (7). Assume that 1) there is only one service class in the system, so that a client is in class 1 if it is admitted and in class 0 otherwise, 2) the maximum average waiting time requirements of all clients are the same, and 3) the utility function of each client is a constant function. Because of the first two restrictions, we can find the maximum arrival rate allowed in the system, say Λ^{max}. Now, the question is if there is a class vector C such that the system efficacy is larger than a real number V', while the aggregate arrival rate is less than Λ^{max}.

Given a class vector $\boldsymbol{C}$, it can be checked in linear time if the aggregate arrival rate is larger than Λ^{max}. The average waiting time can be calculated in polynomial time.[3] The system's efficacy can then be found by $\sum_{i=1}^{M'} U_i(W_{c_i})$ in polynomial time. Clearly, the whole process can be done in polynomial time and so the decision problem is in NP. Now, we want to transform the decision problem to a known NP-complete KNAPSACK problem. The arrival rate constraint is transformed to the size constraint of KNAPSACK. The summation of system efficacy is transformed to the summation of values in KNAPSACK. Clearly, the transformation can be done in polynomial time. Therefore, our decision problem is in NP-complete.

Since the above decision problem is only a decision version of the optimization problem in (7) with three restrictions, the optimization problem in (7) must be in NP-hard. □

In general, finding the solution to the optimization problem in (7) can be computationally expensive. A straightforward approach is to perform an exhaustive search. The class value of a client can be $0, 1, \ldots, N$, where a class value of 0 means the client is not admitted into the system. There are a total of M potential clients. The search has a computational complexity of $\Theta((N+1)^M)$ in evaluating the expression of (6) so as to choose the optimal configuration. Since the number of clients M can be very large, the computation cost is prohibitive even for a small number of classes N. In the following, we propose two efficient admission control algorithms such that, at the end of the admission control process, we can determine 1) the clients that the Web server can admit and 2) the lowest possible admitted class vector $\boldsymbol{C}^a = [C_1^a, C_2^a, \ldots, C_{M'}^a]$ for those admitted clients.

3 Admission Control and Resource Provisioning

In this section, we explain how to perform the admission control and the class assignment for a PDDS-enabled Web server.

3.1 Admission Control and Class Assignment

To subscribe service from the server $\mathcal{S}$, each client has to go through the admission control procedure. Each client j will provide the information, λ_j^{max} and W_j^{max}, to the server $\mathcal{S}$. In return, the server $\mathcal{S}$ will indicate whether it can admit client j or not. If the system can admit client j, it will also notify client j of the assigned class index, $C_j^a \in \{1, \ldots, N\}$. As long as client j marks all its requests to $\mathcal{S}$ in class C_j^a, the server $\mathcal{S}$ can *guarantee* that the long term average waiting time for client j is less than or equal to the QoS requirement W_j^{max}. We propose the following two admission control/class assignment algorithms.

3.2 Maximum Profit Algorithm (MPA)

The first algorithm is MPA (see Fig. 3). The objective is to admit a client having a more stringent maximum average waiting time requirement first. The rationale is that if there are two clients i and j with requirements of W_i^{max} and W_j^{max}, respectively, and $W_i^{max} < W_j^{max}$, it is reasonable to assume client i is willing to pay a higher usage cost than client j so as to receive better service. By admitting client i, the service provider may obtain a higher profit. The MPA algorithm is given below.

Let us explain the rationale of the MPA algorithm. Under MPA admission control, we test whether we can admit a tagged client (line 3). For this tagged client i, we first assign it to class one (line 5). By adding this client i, we may change the waiting times of previously admitted clients. We test whether this new additional client will violate the QoS of other clients in Ω' (line 7). If the addition does not violate the QoS of any client, we can admit this tagged client i (line 9). On the other hand, if there is any QoS violation and

3. The value of $W(\lambda)$ can be precomputed offline and values can be obtained via table lookup with polynomial time complexity.

```
MPA Admission Control
1.  Sort the maximum average waiting time requirement of all clients from smallest to largest. After the sorting, we assume
    client 1 (respectively, client $M$) has the smallest (respectively, largest) maximum average waiting time requirement.
2.  Let $\Omega$ be the set of all admitted clients. Initialize $\Omega = \emptyset$ and initialize the admitted class vector $\boldsymbol{C}^a = \mathbf{0}$;
3.  for ($i = 1$ to $M$) { /* test all $M$ clients */
4.    $\Omega' = \Omega \bigcup$ client $i$; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
5.    assign client $i$ in class 1;
6.    while (satisfied_flag == false) {
7.      compute delay of all clients in $\Omega'$ based on Eq.(6);
8.      if (waiting times of all clients in $\Omega'$ are satisfied) {
9.        $\Omega = \Omega'$; satisfied_flag=true; }
10.     else{/* perform class upgrade for unsatisfied clients*/
11.       if (there is any unsatisfied client with class equal $N$)
      /*cannot admit client $i$, restore $\boldsymbol{C}^a$*/
12.         $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
13.       else /* upgrade unsatisfied clients */
14.         increase the class of all unsatisfied clients in $\Omega'$ by 1;
15.     }
16.   } /* termination of while loop */
17. } /* termination of for loop */
18. return ($\Omega$ and $\boldsymbol{C}^a$);
```

Fig. 3. MPA algorithm.

the unsatisfied clients are already in class N, this implies that we cannot admit the tagged client i (line 11-12). If there is QoS violation and none of the unsatisfied clients is in class N, we can upgrade all the unsatisfied clients by one class (line 14) and test whether we can admit the tagged client i again. In the following, we present some important properties of the MPA admission control algorithm, including the computational complexity, and the property that it guarantees an admitted client the minimum class level that can satisfy its QoS requirement.

Lemma 1. *The MPA admission control has a computational complexity of $O(NM^2)$.*

Proof. For MPA, we have to test whether we can admit each of the M clients (line 3). When we test the kth client, the maximum number of clients in Ω' is equal to k (line 4). Each of these clients in Ω' may go through class upgrade (line 14) but never class downgrade. Since there are N classes in the system, we have to test $O(kN)$ configurations in the worst-case. To test for all M clients, we need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$. □

To present the properties of MPA admission control, we first need to define the following notation and then state some preliminary results. Let $\boldsymbol{\Lambda} = [\Lambda^{max}_{c_1}, \cdots, \Lambda^{max}_{c_N}]$ be the arrival rate vector of different classes of requests. We define $\boldsymbol{e}_i$ as a row vector of zero with the ith entry being one. If a client m changes its requests from class i to class j, where $j > i$, then the arrival rate vector is $\boldsymbol{\Lambda}' = \boldsymbol{\Lambda} - \lambda^{max}_m \boldsymbol{e}_i + \lambda^{max}_m \boldsymbol{e}_j$. Let $W_{c_k}(\boldsymbol{\Lambda})$ be the average waiting time of class k requests under loading $\boldsymbol{\Lambda}$.

Lemma 2. *If a client m performs a class upgrade from class i to j ($j > i$), then $W_{c_k}(\boldsymbol{\Lambda}') \geq W_{c_k}(\boldsymbol{\Lambda})$ for all classes $k = 1, 2, \ldots, N$.*

Proof. Equation (6) expresses the average waiting time for each class of traffic under a PDDS system. Since $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$, we have $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_N$, and so $\sigma_i > \sigma_j$. When a client m upgrades from class i to class j, it is easy to observe that the denominator of (6) will not increase while the numerator will remain unchanged. Therefore, $W_{c_k}(\lambda') \geq W_{c_k}(\lambda)$. □

Lemma 3. *If a client m performs a class downgrade from class j to i ($i < j$), then the average waiting times for all classes will not increase.*

Proof. Equation (6) expresses the average waiting time for each class of traffic under a PDDS system. Since $\sigma_1 = 1$ and $\sigma_i = \sigma_{i-1}/r_{i-1,i}$, we have $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_N$, and so $\sigma_j > \sigma_i$. When a client m downgrades from class j to class i, we can easily observe that the denominator of (6) will not decrease while the numerator will remain unchanged. Therefore, $W_{c_k}(\lambda') \leq W_{c_k}(\lambda)$. □

Definition 1. *Let $W_{c_k}(C)$ and $W^{max}_j(C)$ be the average waiting time of class k requests and the maximum average waiting time of client j's requests, respectively, under the class vector C. Also, let $\Lambda(C)$ be the arrival rate under the class vector C.*

Definition 2. *Let $\boldsymbol{C}$ and $\boldsymbol{C}'$ be two class vectors. We say that $\boldsymbol{C} > \boldsymbol{C}'$ iff $C_i \geq C'_i$ and $\exists\ j$, where $C_j > C'_j$.*

Defintion 3. *A class vector $\boldsymbol{C}$ is a* feasible admitted class vector *if the class assignment in $\boldsymbol{C}$ can guarantee the maximum average waiting time requirements for all admitted clients.*

Defintion 4. *A* minimum feasible admitted class vector *$\boldsymbol{C}^*$ is a class vector such that there is no other feasible admitted class vector $\boldsymbol{C}'$ where $\boldsymbol{C}' < \boldsymbol{C}^*$.*

Theorem 2. *The MPA admission control guarantees that, at the end of every stage of testing whether to admit a client, the class vector is always a minimum feasible admitted class vector.*

Proof. Let $\boldsymbol{C}^a(k)$ and $\Omega(k)$ be the admitted class vector and the set of admitted clients after testing whether we can admit the kth client. Initially, we have $\boldsymbol{C}^a(0) = [0, \cdots, 0]$

where 0 indicates rejection and $\Omega(0) = \emptyset$. We proceed to prove the theorem by induction. Consider the first client (or $k = 1$). If MPA rejects this client because the system cannot satisfy its QoS requirement, $\boldsymbol{C}^a(1) = [0, \ldots, 0]$, which is a minimum class vector. If the system admits this client, then because MPA assigns class 1 to the client initially (line 5) and upgrades the class one at a time, the resulting admitted client vector $\boldsymbol{C}^a(1)$ is obviously a minimum feasible class vector.

Assume this property holds for $k = i - 1$. When the system tries to admit the ith client, there are three cases to consider:

1. *The system can admit client i without changing the class assignment in $\Omega(i-1)$:* In this case, since $\boldsymbol{C}^a(i-1)$ is a minimum feasible class vector and we assign the minimum class to client i (line 5), $\boldsymbol{C}^a(i)$ is the minimum admitted class vector for $\Omega(i)$.
2. *The system cannot admit client i because there is at least one client in $\Omega(i-1)$ whose class is in class N (line 11):* In this case, client i will be rejected and we restore the previous admitted class vector (line 12). Therefore, $\boldsymbol{C}^a(i) = \boldsymbol{C}^a(i-1)$, which is the minimum admitted class vector for $\Omega(i)$.

   We divide the analysis of this case into two stages: the present stage and the later stage. In the present stage, there exists a client $l \leq i$ and a class vector $\boldsymbol{C}^T$ such that $C_l^T = N$ and $W_{c_N}(\boldsymbol{C}^T) \geq W_l^{max}(\boldsymbol{C}^T)$.

   Based on Lemma 2, we have:

   $$W_{c_N}(\boldsymbol{C}') \geq W_l^{max}(\boldsymbol{C}^T) \geq W_l^{max}(\boldsymbol{C}^T) \quad \forall \boldsymbol{C}' \geq \boldsymbol{C}^T.$$

   So, there exists no feasible admitted class vector for $\Omega(i)$. In the later stage, we may admit client j where $j > i$. But, admitting client j will not make client i admissible. The reasons are: 1) $\Lambda(\boldsymbol{C}^a(j)) \geq \Lambda(\boldsymbol{C}^a(i-1))x$ implies that $W_{c_k}(\boldsymbol{C}^a(j)) \geq W_{c_k}(\boldsymbol{C}^a(i-1))$ and 2) admitting client j may result in class upgrade for the clients in $\Omega(i-1)$. By Lemma 2, this will increase the waiting time for all the clients in $\Omega(i-1)$. Since admitting client j will not decrease the waiting time of clients in $\Omega(i-1)$, if client i was not admissible, it will not become admissible after the system has admitted client j.
3. *There are $L \geq 1$ unsatisfied clients in $\Omega(i-1)$, but none of them is in class N (line 13):* In this case, the MPA will *simultaneously* upgrade the class of all these L unsatisfied clients by one (line 14). Note that we do not need to upgrade the class of each unsatisfied client sequentially.

   Consider that there are more than one unsatisfied clients in class vector C^T, say i and j. We have $W_{c_k}(\boldsymbol{C}^T) \geq W_i^{max}(\boldsymbol{C}^T)$ and $W_{c_l}(\boldsymbol{C}^T) \geq W_j^{max}(\boldsymbol{C}^T)$ where $k = C_i^T$ and $l = C_j^T$. If only client i does class upgrade and gives the new class vector C', then based on Lemma 2, we have:

   $$W_{c_l}(C') \geq W_{c_l}(C^T) \geq W_j^{max}(C^T).$$

   Client j is still unsatisfied with the new class vector C'. The proof for more than two clients is similar. After the class upgrade, some of these clients may still be unsatisfied, in which case we repeat the process until we reach case 1 or case 2 above. Since we perform class upgrade incrementally, the resulting admitted class vector $\boldsymbol{C}^a(i)$ is a minimum feasible class vector. □

Remark. The implication of Lemma 1 and Theorem 2 is that, not only do we have a computationally efficient admission control algorithm, but the resulting admitted vector $\boldsymbol{C}^a$ is also a minimum feasible class vector. Therefore, we can ensure to provide QoS guarantees to all admitted clients and, at the same time, not overcharge these clients by assigning them to higher classes than needed.

The MPA algorithm assumes that a client with a tighter QoS requirement (i.e., smaller maximum average waiting time) is willing to pay a higher cost for the Web service. On the other hand, a Web server operator may want to maximize the number of admitted clients so as to popularize the Web service. In this case, we propose the following admission control algorithm (see Fig. 4).

3.3 Maximum Admission Algorithm (MAA)

The second admission control algorithm is called MAA. The objective is to admit as many clients as possible into the Web server. The rationale is that by admitting more clients, the Web service will be more popular and the content provider will be able to charge more and generate more profit in the long run. Under MAA, we try to admit those clients with a less stringent QoS requirement (i.e., large maximum average waiting time) first. The MAA algorithm is shown below.

Let us explain the operation of the MAA algorithm. Under MAA admission control, we test whether we can admit a tagged client (line 3). For this tagged client i, we first assign it to class one (line 5). By adding this tagged client i, we may change the waiting times of previously admitted clients. We test whether this new additional client will violate the QoS of other clients in Ω' (line 7). If the addition does not violate the QoS of any client, we can admit this tagged client i (line 9). On the other hand, if there is any QoS violation and the unsatisfied clients are already in class N, this implies that we cannot admit the tagged client i (lines 11-12). If there is QoS violation and none of the unsatisfied clients is in class N, we can upgrade all these unsatisfied clients by one class (line 14) and test whether we can admit the tagged client i again. Once we find the first client that we cannot admit (we call this client i^*), we go to the second phase of the algorithm by testing whether we can admit the remaining clients (clients $i^* + 1$ to M). Because of the initial sorting, the remaining clients will have a maximum average waiting time requirement smaller than or equal to that of client i^*. Therefore, we can do much pruning by skipping those clients whose arrival rates are larger than the arrival rate of client i^* because the server $\mathcal{S}$ cannot admit these clients for sure. In the following, we present some important properties of the MAA admission control: its computational complexity and the property of

```
MAA Admission Control
1.  Sort the maximum average waiting time requirement of all clients from largest to smallest. If there is a tie, sort clients
    based on the maximum arrival rate from smallest to largest. Assume that client 1 (respectively, client $M$) has the largest
    (respectively, smallest) maximum average waiting time requirement.
2.  Let $\Omega$ be the set of all admitted clients. Initialize $\Omega = \emptyset$ and the admitted class vector $\boldsymbol{C}^a = \mathbf{0}$;
3.  for ($i = 1$ to $M$) { /* test all $M$ clients */
4.    $\Omega' = \Omega \bigcup$ client $i$; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
5.    assign client $i$ to class 1;
6.    while (satisfied_flag == false) {
7.      compute delay of all clients in $\Omega'$ based on Eq.(6);
8.      if (waiting times of all clients in $\Omega'$ are satisfied) {
9.        $\Omega = \Omega'$; satisfied_flag=true; }
10.     else{ /* perform class upgrade for unsatisfied clients*/
11.       if (there is any unsatisfied client with class equal $N$) {
            /*can't admit client $i$, restore $\boldsymbol{C}^a$*/
12.         $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
            min_arrival_rate=arrival rate of client $i$;
            $i^* = i; i = M$;}
13.       else /* upgrade unsatisfied clients */
14.         increase the class of all unsatisfied client in $\Omega'$ by 1;
15.     }
16.   } /* termination of while loop */
17. } /* termination of for loop */
      /* test whether we can admit client $i^* + 1$ to $M$ */
18. for ($i = i^*$+1 to $M$) {
19.   if (arrival rate of client $i <$ min_arrival_rate) {
20.     $\Omega' = \Omega \bigcup$ client $i$; $\boldsymbol{C}^a_{temp} = \boldsymbol{C}^a$; satisfied_flag = false;
21.     assign client $i$ to class 1;
22.     while (satisfied_flag == false) {
23.       compute delay of all clients in $\Omega'$ based on Eq.(6);
24.       if (waiting times of all clients in $\Omega'$ are satisfied) {
25.         $\Omega = \Omega'$; satisfied_flag=true; }
26.       else{ /* perform class upgrade for unsatisfied clients*/
27.         if (there is any unsatisfied client with class equal $N$) {
            /*can't admit client $i$, restore $\boldsymbol{C}^a$*/
28.         $\boldsymbol{C}^a = \boldsymbol{C}^a_{temp}$; satisfied_flag = true;
            min_arrival_rate = arrival rate of client $i$;}
29.         else /* upgrade unsatisfied clients */
30.         increase the class of all unsatisfied client in $\Omega'$ by 1;
31.       }
32.     } /* termination of while loop */
33.   } /* termination for if loop */
34. } /* termination of for loop */
35. return ($\Omega$ and $\boldsymbol{C}^a$);
```

Fig. 4. MAA Admission Control algorithm.

guaranteeing an admitted client the minimum service class for its QoS requirement.

Lemma 4. *MAA admission control has a computational complexity of $O(NM^2)$.*

Proof. Under the MAA, we have to test whether we can admit each of the M clients (lines 3). The algorithm is divided into two phases, 1) lines 3-17 and 2) lines 18-34. In the first phase, when we test whether we can admit the kth client, the number of clients in Ω' is equal to k. Each of these clients in Ω' may go through class upgrade (line 14), but never class downgrade. Since there are N classes in the system, we have to test $O(kN)$ configurations in the worst-case. The end of the first phase is signaled by an unsatisfied client in class N (lines 11-12). There are at most M clients to be tested in the first phase. We need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$. When the unsatisfied client (we call this client i^*) is found in the first phase, its arrival rate is marked so that we can perform pruning in the second phase. In the second phase, a user will be tested only if its arrival rate is less than the arrival rate of client i^*. We can perform this pruning because for client $j > i^*$, we have $W_j^{max} < W_{i^*}^{max}$ (due to initial sorting) and if $\lambda_j^{max} > \lambda_{i^*}^{max}$, the server $\mathcal{S}$ cannot admit client j for sure. If client j has a smaller arrival rate than client i^*, the maximum number of clients in Ω' is equal to j, and each of these clients in Ω' may go through class upgrade (line 30), but never class

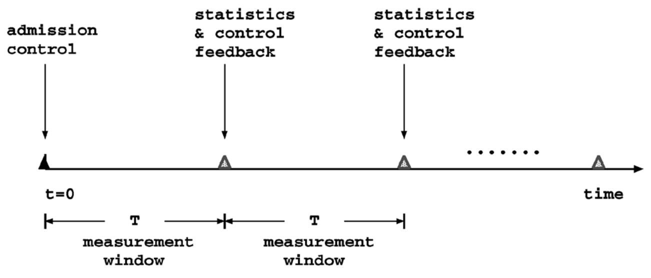


Fig. 5. General framework for server $\mathcal{S}$ to collect statistics and send feedback control back to clients.

downgrade. Since there are N classes in the system, we have to test $O(jN)$ configurations in the worst case. As there are at most M clients to be tested in the second phase, we need to test at most $\sum_{k=1}^{M} kN$ configurations, which is $O(NM^2)$. Therefore, the worst-case computational complexity is $O(NM^2)$ for the MAA algorithm. □

Theorem 3. *MAA admission control guarantees that, at the end of every stage of testing whether to admit a client, the class vector is always a minimum feasible admitted class vector.*

Proof. Please refer to [14]. □

4 Dynamic Class Adaptation

Based on the admission control algorithms proposed in Section 3, the PDDS-enabled Web server $\mathcal{S}$ can provide QoS guarantees to all the admitted clients. In other words, the expected waiting time of each client is guaranteed to be upper bounded by its specified maximum average waiting time. One important point to observe is that the admission control is carried out based on the *maximum* arrival rate specified by each client. It is possible that the average arrival rate of the admitted client is less than or equal to its specified maximum arrival rate. Let λ_j denote the average arrival rate of the admitted client j. If $\sum_{j=1}^{M'} \lambda_j < \sum_{j=1}^{M'} \lambda_j^{max}$, it implies that there is an opportunity for an admitted client, say j, to submit requests to the PDDS-enabled Web server $\mathcal{S}$ with a class value which is less than or equal to C_j^a and still be able to attain its QoS requirement (i.e., the average waiting time is less than W_j^{max}). In this section, we propose two dynamic adaptation algorithms so that the admitted clients can dynamically adapt to the system loading at $\mathcal{S}$.

Before we present these two dynamic adaptation algorithms, let us present the general framework wherein the PDDS-enabled Web server $\mathcal{S}$ can measure the necessary information and send feedback control information back to all admitted clients. Fig. 5 illustrates the general framework. Assume that the server $\mathcal{S}$ has completed the admission control process (either via MPA or MAA) at time $t = 0$. Each admitted client will submit requests to $\mathcal{S}$ based on its class assignment in $\boldsymbol{C}^a$. For every measurement window of length T, the server $\mathcal{S}$ measures the request arrival rates. At the end of each period, the server $\mathcal{S}$ either sends back a new class vector $\boldsymbol{C}$ to all the admitted clients, or sends back the arrival statistics to all the admitted clients, who can then perform their own class adaptation.

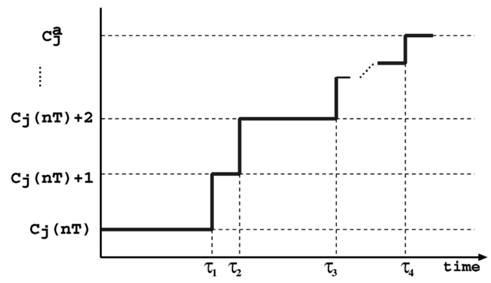


Fig. 6. Class adaptation by client j within a measurement period.

4.1 Centralized Approach: Server-Based Dynamic Adaptation (SBDA)

Under server-based adaptation, the Web server estimates the arrival rate of each client within a measurement window, and then computes a new class vector for each admitted client at the end of the measurement period. The new class assignment will be sent to each admitted client. Each admitted client can then submit requests to the PDDS-enabled Web server in a class range that is between the new class value and the original admitted class value.

Formally, let $\boldsymbol{C}(nT)$ denote the class vector at the end of the nth measurement period. We have $\boldsymbol{C}(0) = \boldsymbol{C}^a$, the initial class vector after the admission control process. Within a measurement window, the server $\mathcal{S}$ estimates the arrival rate of client j. Let $N_j(nT)$ be the number of requests submitted by client j during the nth measurement period. The estimated arrival rate of client j at the end of this measurement period is:

$$\hat{\lambda}_j = \frac{N_j(nT)}{T} \qquad j = 1, 2, \ldots, M'. \tag{8}$$

To generate a new class vector $\boldsymbol{C}(nT)$, the server can use either the MPA or the MAA algorithm described in the previous section. Once the new class vector is computed, the server $\mathcal{S}$ sends the new class value $C_j(nT)$ to client j, $j = 1, 2, \ldots, M'$.

Upon receiving the new class value, the client j can choose to tag the request in class C_j^* where $C_j(nT) \le C_j^* \le C_j^a$. Here, we consider that a client j will initially tag its requests as $C_j(nT)$. During the process of request submission, client j also estimates its waiting time. If it is more than the maximum average waiting time requirement W_j^{max}, then client j will upgrade its requests by one class. The maximum class value that class j can tag its requests is C_j^a. Note that if the estimated average waiting time is less than W_j^{max}, then client j will not perform any class downgrade and will continue to submit requests based on the current class value. This way, client j can reduce its usage cost for $\mathcal{S}$. Fig. 6 illustrates an example in which client j performs a class upgrade at instants τ_1, τ_2, τ_3, and τ_4.

We like to stress two important points here:

- First, $C_j(nT)$ is guaranteed to be less than or equal to C_j^a. The reason is that the original class vector $\boldsymbol{C}(0)$ (or $\boldsymbol{C}^a$) was computed based on the "maximum" arrival rate of each admitted client. Since the arrival rates of all admitted clients within the measurement period are less than or equal to their maximum

```
SBDA Algorithm
1.  for ($i = 1$ to $M'$) { /* test all $M'$ clients */
2.      $C'_i = C^a_i$;
3.  } /* termination of for loop */
4.  finish = FALSE;
    num_series = 0;
    compute delay of all classes using $C'$;
5.  while (finish == FALSE) {
6.      for ($i = 1$ to $M'$) { /* test all $M'$ clients */
7.          num_series++;
8.          if ($C'_i > 1$) {
9.              if ($W_{C'_i - 1} < W^{max}_i$) {
10.                 $C'_i = C'_i - 1$;
                    num_series = 1;
                    compute delays of all classes using $C'$; }
11.         }
12.         if (num_series == $M'$) {
13.             finish = TRUE; }
14.     } /* termination of for loop */
15. } /* termination of while loop */
16. return ($C'$);
```

Fig. 7. SBDA Algorithm.

arrival rates, the resulting class vector $\boldsymbol{C}(nT)$ is guaranteed to be less than or equal to $\boldsymbol{C}^a$.

- Second, if client j tags its requests in class C^a_j, j is assured that its requests will definitely satisfy its QoS requirement. The procedure of SBDA is shown in Fig. 7.

If the future arrival rates of all admitted clients will not change, the SBDA algorithm can find the minimum feasible admitted class vector. SBDA assigns each client i to C^a_i first. Every client then tries to do class downgrade sequentially. If a client finds that the average waiting time of class C'_{i-1}, where $i - 1 \geq 1$, can still satisfy its maximum average waiting time requirement, the client will perform class downgrade. When all admitted clients stop downgrading, the process terminates. The class vector C' computed satisfies the waiting time requirements of all admitted clients and allows them to submit at the lowest possible class. In this paper, the lowest possible class is the class satisfying the waiting time requirement of the admitted client and, at the same time, allowing it to pay the lowest cost. In the following, we present some important properties of the SBDA algorithm.

Theorem 4. *If the future arrival rates of all admitted clients will not change, the class vector computed in SBDA satisfies the waiting time requirements of all admitted clients, and these clients will submit at the lowest possible class.*

Proof. When a client tests if it needs to perform a class downgrade (lines 8-9), there are only two possibilities. Either 1) it moves to one class lower if it finds that its QoS requirement can be satisfied in the lower class, or 2) it remains in the same class if the requirement cannot be satisfied in the lower class. In case 1), when a client performs a class downgrade, the average waiting time of other classes will not increase by Lemma 3. Hence, no user needs to perform class upgrade. In case 2), when a client remains in the same class, this causes no change to the average waiting times of other classes and so no client needs to do class upgrade. Clients always submit at the lowest class. Once two consecutive groups with M' clients not performing any class downgrade are found, the waiting times of all clients will not be changed any more. Therefore, the waiting time requirements of all the admitted clients are guaranteed. □

Lemma 5. *The SBDA algorithm has a computational complexity of $O(NM'^2)$.*

Proof. In the worst-case, for an iteration of M' clients, there is only one client who needs to perform a class downgrade. As there are M' clients and N classes, the maximum number of class downgrade is NM'. Therefore, the worst-case computational complexity is $O(NM'^2)$. □

There are some major drawbacks about the server-based dynamic adaptation approach. For example, it is computationally expensive to estimate the arrival rate of *each* admitted client in (8). Another disadvantage is that the server $\mathcal{S}$ needs to send the new class value $\boldsymbol{C}(nT)$ to each admitted client, which implies that the server needs to perform M' operations to reach all the admitted clients. On the other hand, the advantage of the SBDA approach is that the new class vector $\boldsymbol{C}(nT)$ is very precise. If there is no major change in the future workload, then each admitted client will pay the lowest usage cost and still be able to receive service within its QoS requirement.

4.2 Distributed and Game-Theoretic Approach: Client-Based Dynamic Adaptation (CBDA)

The SBDA algorithm can be computationally expensive, both in tracking the arrival rates of all M' clients and in sending the new class vector to all the clients. We propose an alternative client-based dynamic adaptation algorithm (CBDA), which is a *distributed adaptation* algorithm wherein each client can choose the appropriate class in submitting its requests.

Unlike the SBDA algorithm, the Web server $\mathcal{S}$ does not need to track the arrival rate of each admitted client, but rather estimate the arrival rates of individual classes of requests. Therefore, rather than track M' variables as in SBDA, CBDA only tracks N variables. Since $N << M'$, this results in a major saving for the computation overhead. Let $\tau_{c_k,n}$ be the interarrival time between the $(n-1)\text{th}$ and the $n\text{th}$ requests in class k. We use an exponential weighted time average method to estimate $\hat{\Lambda}_{c_k}(n)$, the arrival rate of class k at the $n\text{th}$ request arrival. The estimate is

$$\hat{\Lambda}_{c_k}(n) = (1-\sigma)\hat{\Lambda}_{c_k}(n-1) + \sigma(\tau_{c_k,n})^{-1} \qquad k = 1, \dots, N, \tag{9}$$

where $0 \leq \sigma \leq 1$. At the end of each measurement period, the Web server $\mathcal{S}$ *multicasts* this class vector $\hat{\boldsymbol{\Lambda}}_c = [\hat{\Lambda}_{c_1}, \hat{\Lambda}_{c_2}, \cdots, \hat{\Lambda}_{c_N}]$ to all the M' admitted clients.

Each client, upon receiving the new class vector $\hat{\boldsymbol{\Lambda}}_c$, can determine the minimum class for its future requests. To illustrate, consider that client j receives $\hat{\boldsymbol{\Lambda}}_c$ from the server $\mathcal{S}$. Let $\hat{\lambda}_{j,c_k}$ be the class k traffic rate submitted by client j in the previous measurement period. Then, the traffic rate vector of client j in the previous measurement period is

```
Adaptation algorithm for client $j$
/*compute rate from previous round*/
1. Let $\tilde{\lambda}_j = \sum_{k=1}^{N} \hat{\lambda}_{j,c_k}$;
2. for ($k = 1$ to $C_j^a$) {
3. /*try new rate vector $\mathbf{\Lambda}^*$ in class $k$ */
4.     $\mathbf{\Lambda}^* = \hat{\mathbf{\Lambda}}_c - \hat{\boldsymbol{\lambda}}_j + \tilde{\lambda}_j \boldsymbol{e}_k$;
5.     Based on Eq.(6) and $\mathbf{\Lambda}^*$, compute delay for client $j$;
6.     if (computed delay $\le W_j^{max}$)
7.       selected_class = $k$; $k = C_j^a$;
8. } /* terminate for loop */
9. return (selected_class);
```

Fig. 8. Adaptation algorithm for client j.

$\hat{\lambda}_j = [\hat{\lambda}_{j,c_1}, \hat{\lambda}_{j,c_2}, \cdots, \hat{\lambda}_{j,c_N}]$. Upon receiving $\hat{\mathbf{\Lambda}}_c$, client j executes the following code (see Fig. 8).

Lemma 6. *The CBDA has a computational complexity of $O(N)$ for each client.*

Proof. Since an admitted client j only needs to test from class 1 to class C_j^a, and C_j^a is upper bounded by N, the worst-case computation complexity for an admitted client is $O(N)$. □

In other words, client j tries to maximize its utility by finding the lowest class such that the average waiting time is less than or equal to the maximum average waiting time requirement, W_j^{max}. In essence, this is a *noncooperative game* problem in which distributed optimization is performed by each client. (For an introduction to the basic concepts of game theory, please refer to [11].) We assume that clients ignore how they influence the class adaptation of other clients when optimizing their own utility. This simplifying assumption corresponds to the standard competitive price taking assumption of economic theory. Also, the above assumption can be justified when:

1. The traffic loading of an individual client is considered to be *small*, as compared to the overall traffic loading at the Web server, so that the class adaptation by a client is considered to be negligible.
2. It is impractical or computationally expensive for a client to determine how to perform class adaptation based on all the other clients' class adaptation decisions.

There are several important properties of the CBDA algorithm as follows:

- **Guaranteed termination:** Each client j searches for the lowest suitable class, from class 1 to C_j^a. In the worst-case, the algorithm will terminate when the class is equal to C_j^a, which is the assigned class during the admission control process. The reason is that the admission control decision was made based on the specified maximum arrival rates for all clients. Therefore, if client j is admitted, by selecting its class equal to C_j^a, we can guarantee that the QoS requirement of client j will be met.
- **Low computational complexity:** Unlike the SBDA approach where the server has to track the arrival rates of *all* M' clients and then *recompute* a new class vector (in essence, reexecute the admission control algorithm), the workload under the CBDA approach is *distributed* among all the clients. The server $\mathcal{S}$ only needs to track the arrival rates for N classes and class adaptation is carried out by the individual clients. If some clients do not want to perform class adaptation, they can simply ignore this optimization step.

TABLE 1
Default Values and Ranges of System Parameters

system parameter	default value or range
N	3
M	1000
λ_r	1.0
$r_{i,i+1}$	1.4
W_j^{max}	(6.0, 20.0)
A	(0.1, 0.3)
B	$D/2$
D	(1.0, 5.0)

One can argue that the adaptation based on the SBDA algorithm is more *precise* than the CBDA algorithm because it uses all available information (i.e., arrival rates of all clients) in making an adaptation decision. We illustrate the performance difference between the two algorithms in the next section.

5 Performance Evaluation

In this section, we compare the performance of the MPA and MAA admission control algorithms. We also present performance results for the SBDA and CBDA adaptation algorithms under various settings—e.g., different arrival rates under Poisson, MMPP, and Pareto arrival process, different waiting time spacing ratios (i.e., $r_{i,i+1}$), and different utility functions. The simulation is done in CSIM. In Section 5.1, we only investigate the class assignment under different system parameters. In Sections 5.2 and 5.3, the simulation is carried out for 50×10^6 time units. The transient period is defined to be the first 5 percent of the total simulation time units. Data collected during the transient period are not used in the data analysis. For experiments with "tagged" clients (i.e., Sections 5.2, 5.3.1, and 5.3.2, and Sections 5.1.2 and 5.1.3), we illustrate the performance of those tagged clients. For the other experiments, the reported results are the averages over 100 independent runs. Table 1 shows the default values and ranges of the system parameters.

5.1 Comparison of MPA and MAA Admission Control

In this experiment, we compare the performance of the MPA and MAA algorithms. In particular, the performance metrics that we are interested in are

1. the number of admitted clients M',
2. the admitted arrival rates of different classes,
3. the achieved waiting times for different classes of requests, and
4. the achieved system efficacy.

TABLE 2
MPA versus MAA for the Number of Admitted Clients M'

# of clients	MPA	MAA
$M = 1,000$	$M' = 497$	$M' = 832$
$M = 2,000$	$M' = 993$	$M' = 1,663$
$M = 5,000$	$M' = 2,479$	$M' = 4,155$

Unless otherwise stated, we assume that the service times of all the requests are exponentially distributed with mean equal to unity. The aggregate request rate from all clients is modeled as a Poisson process with rate λ_r. Note that λ_r is the workload *before* the admission control procedure. The PDDS-enabled Web server supports $N = 3$ classes of requests and their waiting time differentiations are $r_{1,2} = 1.4, r_{2,3} = 1.4$.

5.1.1 Under Different Number of Clients

We vary the number of potential clients that wish to access the server $\mathcal{S}$ to be M = 1,000, 2,000, and 5,000. The maximum average waiting time requirements of these clients are drawn uniformly from $[1.5, 5.5]$ seconds and the aggregate request rate λ_r is set to one. Since this arrival rate can saturate the system ($\rho = 1$), it is necessary for us to perform admission control. Table 2 illustrates the total number of admitted clients M' for the MPA and MAA algorithms under different values of M. We can see that MAA can admit more clients because this algorithm tries to admit clients with less stringent maximum average waiting time requirements first. One can argue that, if the admission cost is fixed on a per class basis, it makes sense to use the MAA algorithm so as to maximize the total admission revenue.

5.1.2 Under Different Arrival Rates

We set the number of potential clients M to 1,000. We vary the aggregate request arrival rate λ_r as 0.5, 0.75, and 1.0. The maximum average waiting time requirements of all clients are drawn uniformly from $[2, 10]$ seconds. Tables 3 and 4 illustrate that, after the admission control and client classification, the arrival rates and the achieved waiting times of the three classes of requests. From Table 3, we observe that at low and moderate workload (e.g., $\lambda_r = 0.5$ or 0.75), both the MPA and MAA can effectively assign clients to the appropriate class so that these admitted clients will pay the lowest possible usage cost. For example, at low workload ($\lambda_r = 0.5$), both algorithms assign all clients to class one (therefore, it becomes single queue scheduling). Under single queue scheduling, the achieved waiting time will be less than the maximum waiting time requirements of all clients. When the system is under high workload ($\lambda_r = 1$), MPA and MAA can filter out those clients whose maximum average waiting time requirements are unrealizable and classify admitted clients to the lowest admissible class.

TABLE 3
MPA versus MAA: Arrival Rates of Different Classes

	class #	MPA	MAA
$\lambda_r = 0.5$	class 3	—	—
	class 2	—	—
	class 1	0.500	0.500
$\lambda_r = 0.75$	class 3	0.058	0.058
	class 2	0.138	0.138
	class 1	0.554	0.554
$\lambda_r = 1.0$	class 3	0.145	0.188
	class 2	0.189	0.233
	class 1	0.432	0.402

For $\lambda_v = 0.5$, all clients are assigned to Class 1.

TABLE 4
MPA versus MAA: Achieved Average Waiting Times of Different Classes

	class #	MPA	MAA
$\lambda_r =$ 0.5	class 3	—	—
	class 2	—	—
	class 1	1.008 (2.000,9.991)	1.008 (2.000,9.991)
	$r_{1,2}$	—	—
	$r_{2,3}$	—	—
$\lambda_r =$ 0.75	class 3	1.701 (2.000,2.378)	1.701 (2.000,2.378)
	class 2	2.381 (2.387,3.331)	2.381 (2.387,3.331)
	class 1	3.334 (3.336,9.991)	3.334 (3.336,9.991)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$
$\lambda_r =$ 1.0	class 3	2.000 (2.000,2.788)	2.950 (2.982,4.119)
	class 2	2.800 (2.802,3.916)	4.130 (4.133,5.776)
	class 1	3.920 (3.926,7.192)	5.782 (5.790,9.991)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$

The numbers in parenthesis indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum waiting time requirements of the admitted clients in that class.

Table 4 depicts the achieved waiting times for different classes under the MPA and MAA algorithms. The numbers in parentheses indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum average waiting time requirements of the admitted clients in that particular class. For example, under $\lambda_r = 0.5$ and MPA, the most stringent (respectively, least stringent) maximum average waiting time requirement is 2.000 (respectively, 9.991) seconds and the achieved waiting time is 1.008 seconds. From Table 4, we observe that both the MPA and MAA algorithms can effectively classify clients to the lowest admissible classes so that their QoS can be satisfied. At the same time, the achieved waiting time ratio is equal to the specified ratio of $r_{i,i+1} = 1.4$.

5.1.3 Under Different Waiting Time Spacings

We compare the effectiveness of the MPA and MAA algorithms under different waiting time spacings. We vary the waiting time spacing $r_{i,i+1}$ to be 1.3, 1.4, and 1.5. The aggregate request arrival rate is $\lambda_r = 1.0$ and the maximum average waiting time requirements of the clients are drawn uniformly from $[2.0, 5.0]$ seconds. From Table 5, we observe that both the MPA and MAA algorithms can effectively classify clients to the lowest admissible classes so that their QoS requirements are satisfied. Also, the achieved waiting time ratio is very close to the specified waiting time ratio $r_{i,i+1}$.

TABLE 5
MPA versus MAA: Achieved Waiting Times of Different Classes under Different Waiting Time Spacings $r_{i,i+1}$

	class #	MPA	MAA
$r_{i,i+1} = 1.3$	class 3	2.000 (2.000,2.597)	2.502 (2.512,3.245)
	class 2	2.600 (2.600,3.377)	3.253 (3.255,4.227)
	class 1	3.380 (3.382,3.772)	4.229 (4.230,4.997)
	W_1/W_2	1.300	1.300
	W_2/W_3	1.300	1.300
$r_{i,i+1} = 1.4$	class 3	1.556 (2.000,2.174)	1.881 (2.527,2.628)
	class 2	2.178 (2.179,3.044)	2.633 (2.634,3.683)
	class 1	3.049 (3.050,3.743)	3.686 (3.687,4.997)
	W_1/W_2	1.400	1.400
	W_2/W_3	1.400	1.400
$r_{i,i+1} = 1.5$	class 3	1.338 (2.000,2.002)	—
	class 2	2.007 (2.007,3.005)	2.484 (2.533,3.722)
	class 1	3.011 (3.011,3.733)	3.725 (3.726,4.997)
	W_1/W_2	1.500	—
	W_2/W_3	1.500	1.500

The numbers in parenthesis indicate the two extremes (i.e., the most stringent and the least stringent) of the maximum waiting time requirements of the admitted clients in that class.

5.1.4 Under Different Utility Functions

This experiment illustrates the effect of the utility function on the system efficacy. The usage cost of each class corresponds to the waiting time ratio; e.g., if $r_{1,2} = 1.4$, the usage costs of classes 1 and 2 are 1.0 and 1.4, respectively. The general form of the utility function is

$$U(A, B, C, D, wt) = D \times \tan^{-1}\left(\frac{C - wt}{A}\right) + B, \qquad (10)$$

where wt is the achieved waiting time of the client, and A, B, C, D are the utility function parameters. (Because of the lack of space, the detailed description of these parameters are presented in [14]). The number of potential clients M is set to 1,000. The aggregate request arrival rate λ_r is 1.2. The maximum average waiting time requirements of clients are drawn uniformly from [6, 20] seconds. Table 6 illustrates the system efficacy of admitted clients for the MPA and MAA algorithms with different utility functions.

TABLE 6
MPA versus MAA: System Efficacy under Different Utility Functions and Waiting Time Spacings $r_{i,i+1}$

	utility function	MPA	MAA
$r_{1,2} = 1.3$ $r_{2,3} = 1.4$	U1	4113.559	4491.842
	U2	3946.108	4319.980
	U3	3732.019	4095.293
	U4	3442.824	3506.946
$r_{1,2} = 1.4$ $r_{2,3} = 1.5$	U1	4124.532	4499.070
	U2	3950.426	4324.481
	U3	3726.556	4092.549
	U4	3453.459	3520.753
$r_{1,2} = 1.5$ $r_{2,3} = 1.6$	U1	4117.647	4518.704
	U2	3944.926	4353.619
	U3	3722.754	4127.143
	U4	3453.258	3540.813

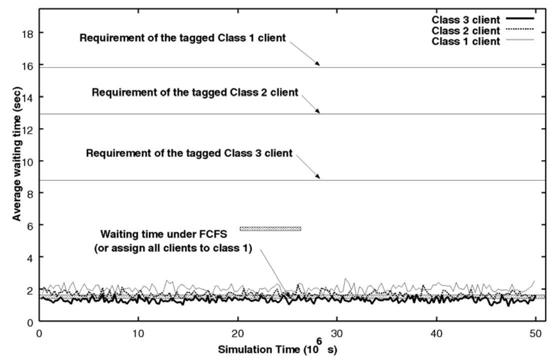


Fig. 9. Waiting times for three different clients wherein the aggregated workload is only half of the maximum specification.

First, we compare the effect of sensitivity to the waiting time on system efficacy. Utility templates U1, U2, and U3 have the same average utility and maximum increase in utility, but U1 is twice as sensitive to the waiting time as U2, while U2 is twice as sensitive as U3. We see that the system efficacy is the largest in U2 both for MPA and MAA. But, MAA always give a larger system efficacy than MPA. For the comparison between utility templates U1 and U4 (both U1 and U4 have the same sensitivity and average utility, but U1 has a larger maximum increase in utility than U4), we see that the decrease of system efficacy in MAA is larger than that in MPA. Hence, the "maximum increase in utility" parameter seems to have a larger effect on MAA than MPA.

5.2 Necessity for Adaptation

In this experiment, we illustrate the necessity to perform class adaption. Consider the scenario wherein after the admission control at $t = 0$, all admitted clients only submit requests to the PDDS-enabled Web server at rates which are only 70 percent of their maximum arrival rates (or λ_j^{max}). Fig. 9 depicts the waiting times of three "tagged" clients from *each* of the three classes as well as their maximum average waiting time requirements. Each point of the plot is the average waiting times of the previous 200 requests of the tagged client. As we can observe, rather than enforce each client to submit requests based on the assigned class, we can simply serve all requests in the system based on a FCFS policy. In other words, if all clients submit requests in class 1, the system can still satisfy all their QoS requirements. This illustrates the necessity of dynamic adaptation. Because the instantaneous workload at the server $\mathcal{S}$ is less than the maximum specified workload, if there is class adaptation, different clients can pay a lower usage cost while still satisfying their QoS.

Another scenario we consider is that, after the admission process, all admitted clients submit requests based on their specified maximum arrival rates. However, at $t^* = 25 \times 10^6$ seconds, 30 percent the of admitted clients stop submitting requests to the Web server. The other *"active"* clients still submit requests based on their specified maximum arrival rates. Fig. 10 illustrates the waiting times of the three tagged clients from their respective classes. One can observe, if all these three clients can mark their requests as class 1 after t^*, they will pay a much lower usage cost and their waiting time requirements can still be satisfied.

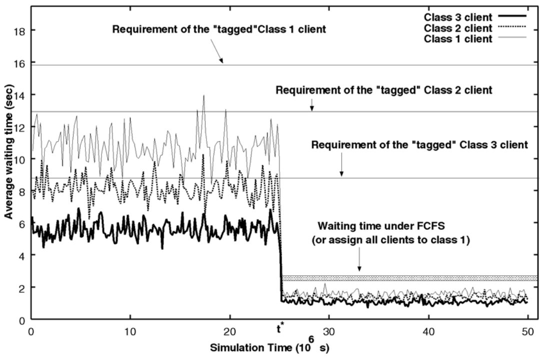


Fig. 10. Waiting times for three differents clients wherein the system workload is reduced by 30 percent at $t = 25 \star 10^6$ seconds.

5.3 Comparison of SBDA and CBDA Adaptation Algorithms

We compare the performance of the SBDA and the CBDA adaptation algorithms. The waiting time requirements of the clients are drawn uniformly from $[6, 20]$ seconds. The aggregate request rate is $\lambda_r = 1.2$. After admission control (by either MPA or MAA), we classify clients into $N = 3$ classes. We simulate the system for 50×10^6 seconds. During the simulation period, admitted clients can change class by using either the SBDA or CBDA algorithms described in the previous section. The arrival rate of each client can change during the simulation. Specifically, within a measurement period of length T, each client can change its arrival rate five times—with probability of 0.8 that the arrival rate is equal to the maximum arrival rate, with probability of 0.1 that the arrival rate is equal to 90 percent of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80 percent of the maximum arrival rate. We consider a *"tagged"* client from *each* of the three classes and we plot their waiting times. Each point of the plot is the average waiting time of the previous 200 requests by the tagged client.

5.3.1 Using MPA and MAA under Poisson Arrivals

Figs. 11 and 12 illustrate the waiting time of the three tagged clients under the SBDA and CBDA adaptation algorithms. The aggregate request rate λ_r (before admission control) is generated by a Poisson process with rate $\lambda_r = 1.2$. For Fig. 11, we use MPA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum waiting time requirements of 12.996, 10.171, and 7.404 seconds, respectively. For Fig. 12, we use MAA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 15.771, 12.988, and 8.775 seconds, respectively. From these figures, we observe that both SBDA and CBDA are very effective in adapting to the workload of the server. All three tagged clients achieve an average waiting time less than their maximum average waiting time requirements. However, note that since CBDA has a much lower computation complexity, it is the preferred algorithm.

5.3.2 Using MAA under Markov-Modulated Poisson Arrivals

We consider the capability of the proposed adaptation algorithms when the input traffic is *non-Poisson*. In this experiment, the aggregate request arrival rate has a mean of $\lambda_r = 1.2$. However, the traffic generation of each client is by a Markov-modulated Poisson Process. Figs. 13 and 14 illustrate the waiting time and probability density function of SBDA and CBDA algorithms under MMPP, respectively. The admission control was carried out using the MAA. The three tagged clients have the maximum average waiting time requirements of 15.771, 12.988, and 8.775 seconds, respectively. From Figs. 13 and 14, we observe that both SBDA and CBDA can adapt to the workload and their waiting time averages are less than their maximum average waiting time requirements. Note that, since CBDA has a much lower computational complexity, we should use CBDA to perform end point adaptation.

5.3.3 Aggregate Utility under Poisson, Pareto, Markov-Modulated Poisson Arrivals

We compare the aggregate utility of all the admitted requests for the SBDA and CBDA adaptation algorithms. The waiting time requirements of the clients are drawn uniformly from $[6, 20]$ seconds. The aggregate request rate is $\lambda_r = 1.0$. The arrival process can be generated by a Poisson, Pareto, or Markov-modulated Poisson process. The desired differentiation ratios are $r_{1,2} = r_{2,3} = 1.4$. After admission

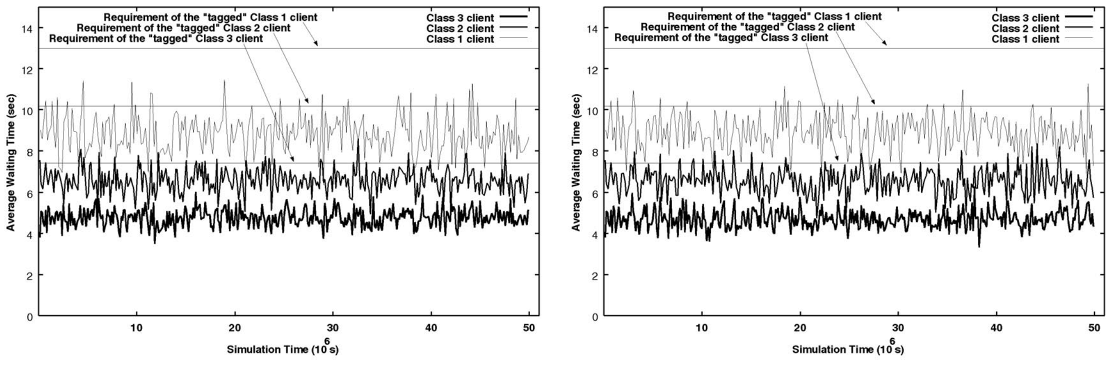


Fig. 11. Waiting times for three different clients under MPA admission control. (a) SBDA: Admission control using MPA. (b) CBDA: Admission control using MPA.

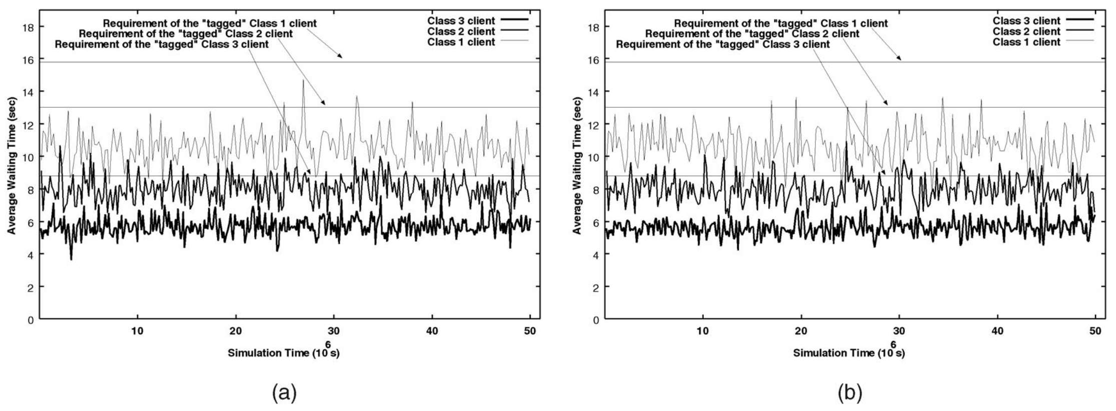


Fig. 12. Waiting times for three different clients under MAA admission control. (a) SBDA: Admission control using MAA. (b) CBDA: Admission control using MAA.

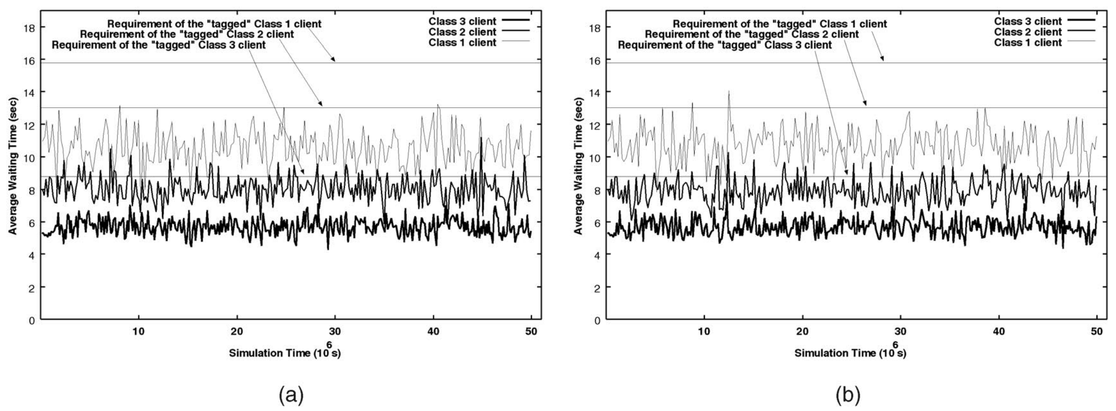


Fig. 13. Waiting time for three different clients under MAA admission control, MMPP arrival process. (a) SBDA: Admission control using MAA. (b) CBDA: Admission control using MAA.

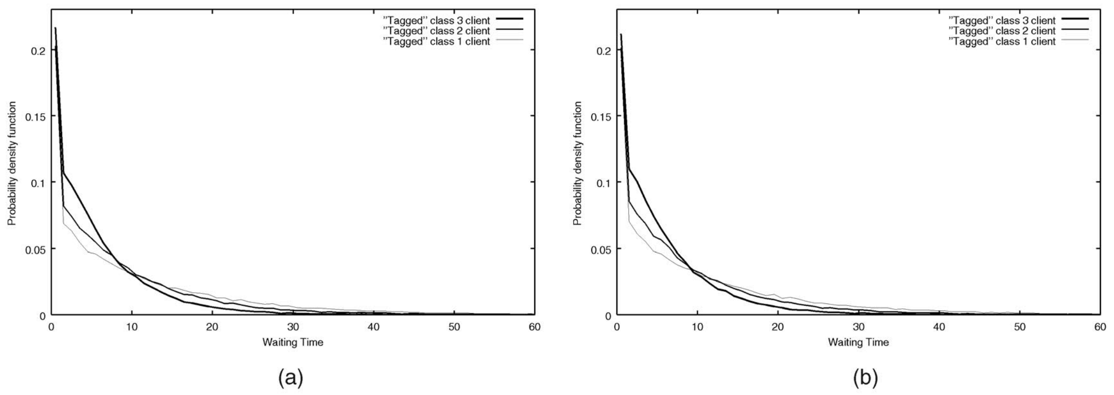


Fig. 14. Probability density function for three different clients under MAA admission control, MMPP arrival process. (a) SBDA: Admission control using MAA. (b) CBDA: Admission control using MAA.

control (by either MPA or MAA), we classify clients into $N = 3$ classes. We simulate the system for 50×10^6 seconds. During the simulation period, admitted clients can change class by using either the SBDA or CBDA algorithms described in the previous section. The arrival rate of each client can change during the simulation. Each client has the same arrival rate within a measurement period of length T. At the end of each measurement period, each client will change the arrival rate with probability of 0.8 that the arrival rate is equal to the maximum arrival rate, with probability of 0.1 that the arrival rate is equal to 90 percent of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80 percent of the maximum arrival rate. Table 7 illustrates the aggregate utility of transmitted packets under the SBDA and CBDA adaptation algorithms with different arrival processes. Irrespective of the adaptation algorithm (MPA or MAA) and the arrival process (Poisson, Pareto, or MMPP), CBDA obtains a larger aggregate utility and the achieved differentiation ratios $r_{i,i+1}$ are also nearer to the desired ratios.

TABLE 7
SBDA versus CBDA: Aggregate Utility of All Transmitted Packets

MPA/ MAA	arrival process	SDBA	CBDA
MPA	Poisson	180625230 (1.37, 1.38)	181859724 (1.39, 1.39)
	Pareto	189769081 (1.35, 1.37)	190240478 (1.38, 1.37)
	mmpp	180899764 (1.38, 1.37)	181713371 (1.39, 1.39)
MAA	Poisson	163760510 (1.36, 1.38)	164177186 (1.39, 1.39)
	Pareto	171645606 (1.34, 1.38)	171931939 (1.38, 1.37)
	mmpp	164110474 (1.38, 1.37)	164739014 (1.39, 1.39)

The numbers in parenthesis indicate the two achieved differentiation ratios.

5.3.4 Aggregate Utility under Real Web Traces

We compare the aggregate utility of all admitted requests for the SBDA and CBDA adaptation algorithms using real Web trace data. There are 200 potential clients. The waiting time requirements of the clients are drawn uniformly from $[6, 20]$ seconds. The arrivals of request are obtained from real traces of Web requests (UC Berkeley home IP Web traces [1]). The target differentiation ratios are $r_{1,2} = r_{2,3} = 1.4$. The service rate is normalized to one. After admission control by MAA, we classify clients into $N = 3$ classes. During the simulation period, admitted clients can change class by using either the SBDA or CBDA algorithms described in the previous section. The arrival rate of each client is drawn from the Web trace data. Each client has the same arrival rate within a measurement period of length T. At the end of each measurement period, each client will change the arrival rate with probability of 0.8 that the arrival rate is equal to the maximum arrival rate, with probability of 0.1 that the arrival rate is equal to 90 percent of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80 percent of the maximum arrival rate. Table 8 illustrates the aggregate utility of all the admitted clients under the SBDA and CBDA algorithms. CBDA usually obtains a larger aggregate utility than SDBA. The achieved differentiation ratios $r_{i,i+1}$ are also closer to the target ratios. Once again, since the CBDA algorithm has a much lower computational complexity as compared to the SBDA algorithm, we should use CBDA for performing the end point adaptation.

6 Related Work

We briefly summarize related research. Recently, various authors have suggested that it is important to consider differentiated services for Web servers [3], [8], [18] in order to complement the Internet differentiated services model. In [3], the authors propose a centralized algorithm to perform server partitioning so as to provide differentiated services. In [8], the authors propose to use the shortest-connection-first algorithm. Differentiation is made for short and long connections. Using their algorithm, short connections have a significant performance gain while long connections pay relatively little penalty. In [18], the authors consider a server that provides prioritized service to different classes of users. In [13], the authors use performance isolation and admission control to design a front-end algorithm to support a new Web service model. In [4] and [7], the authors work on session-based traffic. They propose a dynamic scheduling algorithm to discriminate the scheduling of requests so as to control overload in Web servers. The idea of the algorithm is to admit requests seeming to have a larger contribution in the future. In [6], the authors propose a Service Differentiating Internet Server which uses prioritized services to differentiate services. It consists of two trials of admission control processes: the latter one sends feedback to clients to reduce further retrying from clients so as to reduce congestion. In [5], the authors propose an admission control algorithm, PACERS, to provide delay bounds and different levels of service to incoming requests. It uses estimation of request rates and response time. Inaccurate estimations are dynamically adjusted. In [2], the authors apply control theory to Internet server performance control. They implement a utilization control loop to satisfy a prespecified bound and individual time constraints. They also demonstrate extensions to provide performance isolation, service differentiation, excess-capacity sharing, and QoS guarantees. In [12], the authors consider a Web service which provides bounded latency for different classes of requests. In particular, the authors consider isolation among service classes as well as session control to protect classes from performance degradation due to overload. The latency requirements and service model considered in [12] are not PDDS, but it is interesting to see how one can incorporate the proposed algorithms into our work. Last, the authors in [9] propose an elegant method to select classes under PDDS so that requests can achieve absolute QoS performance. Our work on admission control uses a similar analytical model and similar definitions as their work. The major differences between our work and theirs are: 1) we provide admission control so that we can guarantee the QoS requirements of all admitted clients, and 2) our class selection algorithms (MPA and MAA) have lower (polynomial time) computational complexity.

TABLE 8
SBDA versus CBDA: Aggregate Utility of All Admitted Clients and Achieved Waiting Time Ratios

	aggregated utility	achieved waiting time ratio $r_{1,2}$	achieved waiting time ratio $r_{2,3}$
SBDA	1329882	1.25	1.26
CBDA	1354847	1.31	1.25

7 Conclusion

We consider a PDDS-enabled Web server. The advantage of this type of service is that the operator of the Web server can provide *fixed* and *prespecified* performance spacings between different classes of requests. Based on the performance spacings, the operator can legitimately charge a higher usage cost for clients in a higher service class. Each client has a maximum average waiting time QoS requirement. We prove that the general assignment problem is NP-complete. We present two efficient admission control algorithms that either maximize the potential profit or maximize the number of admitted clients into the system. We show that these admission control algorithms are computationally efficient and, at the same time, the resulting class vector is a minimum feasible admitted class vector. To further reduce the usage cost, we also present two end point adaptation algorithms. One is server-based while the other is distributed. The distributed approach is based on a noncooperative game technique. We show that the distributed approach has lower computational cost and can dynamically adapt to the server's workload. Experiments are carried out to illustrate the effectiveness of these algorithms under different parameters, e.g., different traffic generation processes (Poisson, MMPP, or Pareto), different waiting time ratio specifications and different utility functions. Last, we believe that a similar proportional delay model [10], [9], [16] can be applied to any system for which request waiting time may significantly impact end user performance, such as P2P, database, and networking systems.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous referees for their helpful and insightful comments. The research work of John C.S. Lui was supported in part by an RGC Earmarked Grant. The research work of David K.Y. Yau was supported in part by the US National Science Foundation under grant number CCR-9875742 (CAREER).

REFERENCES

[1] UC Berkeley Home Ip Web Traces: http://ita.ee.lbl.gov/html/contrib/ucb.home-ip-http.html, year?

[2] T.F. Abdelzaher, K.G. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 1, pp. 80-96, Jan. 2002.

[3] V. Cardellini, E. Casalicchico, M. Colajanni, and M. Mambelli, "Web Switch Support for Differentiated Services," *Proc. Workshop Performance and Architecture of Web Servers (PAWS),* June 2001.

[4] H. Chen and P. Mohapatra, "Session-Based Overload Control in QoS-Aware Web Servers," *Proc. IEEE Infocom,* 2002.

[5] X. Chen, H. Chen, and P. Mohapatra, "An Admission Control Scheme for Predictable Server Response Time for Web Accesses," *Proc. 10th World Wide Web Conf.,* pp. 545-554, May 2001.

[6] X. Chen and P. Mohapatra, "Performance Evaluation of Service Differentiating Internet Servers," *IEEE Trans. Computers,* pp. 1368-1375, 2002.

[7] L. Cherkasova and P. Phaal, "Session Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites," *IEEE Trans. Computers,* vol. 51, no. 6, June 2002.

[8] M. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers," *Proc. USENIX Symp. Internet Technologies and Systems,* Oct. 1999.

[9] C. Dovrolis and P. Ramanathan, "Dynamic Class Selection: From Relative Differentiation to Absolute QoS," *Proc. IEEE Int'l Conf. Network Protocols,* Nov. 2001.

[10] C. Dovrolis, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," *Proc. ACM SIGCOMM,* pp. 109-119, Aug. 1999.

[11] R. Gibbons, *Game Theory for Applied Economists.* Princeton Univ. Press, 1992.

[12] V. Kanodia and E.W. Knightly, "Multi-Class Latency-Bounded Web Services," *Proc. IEEE/IFIP Int'l Workshop Quality of Service,* June 2000.

[13] V. Kanodia and E.W. Knightly, "Ensuring Latency Targets in Multi-Class Web Servers," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 1, Jan. 2003.

[14] S.C. Lee, J.C. Lui, and D.K. Yau, "A Proportional-Delay Diffserv-Enabled Web Server: Admission Control and Dynamic Adaptation," Technical Report, CS-TR-2003-02, 2003.

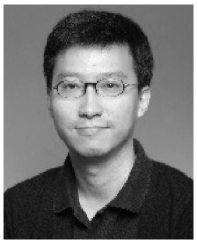
[15] M.K. Leung, J.C. Lui, and D.K. Yau, "Characterization and Performance Evaluation for Proportional Delay Differentiated Services," *Proc. Int'l Conf. Network Protocols,* pp. 295-304, Nov. 2000.

[16] M.K. Leung, J.C. Lui, and D.K. Yau, "Adaptive Proportional Delay Differentiated Services: Characterization and Performance Evaluation," *IEEE/ACM Trans. Networking,* vol. 9, no. 6, Dec. 2001.

[17] S. Shenker, "Fundamental Design Issues for the Future Internet," *IEEE J. Selected Areas in Comm.,* vol. 13, pp. 1141-1149, 1995.

[18] H. Zhu, H. Tang, and T. Yang, "Demand-Driven Service Differentiation in Cluster-Based Network Servers," *Proc. IEEE Infocom,* Apr. 2001.

Sam C.M. Lee received the BSc and MPhil degrees in computer science from the Chinese University of Hong Kong (CUHK). He is presently a PhD student at CUHK. His research interests are in network communication and mathematical optimization theory.



John C.S. Lui received the PhD degree in computer science from the University of California Los Angeles (UCLA). After graduating, he joined the IBM Almaden Research Laboratory/San Jose Laboratory and participated in various research and development projects on file systems and parallel I/O architectures. He later joined the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research encompasses both systems and theory. His current research interests are in theoretical/applied topics in data networks, distributed multimedia systems, network security, OS design, mathematical optimization, and performance evaluation. He received the CUHK Vice-Chancellor's Exemplary Teaching Award in 2001. He is an associate editor of the *Performance Evaluation Journal*, a member of the ACM, a senior member of IEEE, and an elected member in the IFIP WG 7.3.

David K.Y. Yau received the BSc (first class honors) degree from the Chinese University of Hong Kong, and the MS and PhD degrees from the University of Texas at Austin, all in computer sciences. From 1989 to 1990, he was with the Systems and Technology group of Citibank, NA. He was the recipient of an IBM graduate fellowship and is currently an associate professor of computer sciences at Purdue University, West Lafayette, Indiana. He received the US National Science Foundation CAREER award in 1999 for research on network and operating system architectures and algorithms for quality of service provisioning. His other research interests are in network security, value-added services routers, and mobile wireless networking. He is a member of the IEEE.