Admission Control and Dynamic Adaptation for a Proportional-Delay DiffServ-Enabled Web Server

Sam C. M. Lee
Dept of Computer Science & Engineering, The Chinese University of Hong Kong
cmlee@cse.cuhk.edu.hk

John C. S. Lui[*]
Dept of Computer Science & Engineering, The Chinese University of Hong Kong
cslui@cse.cuhk.edu.hk

David K. Y. Yau[†]
Dept of Computer Science
Purdue University
West Lafayette, IN 47907
yau@cs.purdue.edu

ABSTRACT

We consider a web server that can provide differentiated services to clients with different QoS requirements. The web server can provide $N > 1$ classes of service. Rather than using a strict priority policy, which may lead to request starvation, the web server provides a proportional-delay differentiated service (PDDS) to heterogeneous clients. An operator for the web server 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 loadings. In addition, each client can specify a maximum average waiting time requirement to be guaranteed by the web server. 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 client. We also consider how to perform end-point dynamic adaptation such that clients can submit requests at a lower class and further reduce their usage cost, 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 based on a non-cooperative game technique. We report diverse experimental results to illustrate the effectiveness of these algorithms.

1. INTRODUCTION

As the Internet becomes more commercially oriented and different businesses use web servers to disseminate information, the effect of access latency to the web servers becomes more important. Conventional web servers use a *"single class"* approach to service client requests. This will not work well for the scenario wherein different clients have different QoS requirements and are willing to pay different prices to attain their desired QoS levels. Rather, a web server should classify requests into multiple classes which can provide the necessary service differentiation.

There are multiple 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 accepts the next request from the highest priority class that is backlogged. The drawbacks of this approach 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 (e.g., 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 strict priority scheduling.

We target a web server able to 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 [4], the authors propose an Internet service model called proportional differentiated services, which has the above mentioned consistency and controllability properties. In the

[*]Research supported in part by the Mainline and RGC Research Grant.
[†]Research supported in part by the National Science Foundation under grant numbers EIA-9806741 and CCR-9875742 (CAREER).

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 loadings, 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 performance that is $r_{i,i+1}$ better than the class i user. To realize proportional-delay differentiated services, the authors in [4] propose to use the *time-dependent priority* (TDP) service discipline in [7]. In [9, 10], the authors illustrate the necessary and sufficient conditions under which the *controllability* and *consistency* properties can be maintained.

In this paper, we consider a proportional-delay DiffServ-enabled web server, say $\mathcal{S}$. Specifically, $\mathcal{S}$ provides a waiting time differentiated service for $N > 1$ classes of requests. Let W_i be the expected waiting time of class i requests, for $i = 1, \ldots, N$. The web server $\mathcal{S}$ specifies a fixed performance spacing $r_{i,i+1} > 1$ such that

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

For example, if $r_{i,i+1} = 1.5$, then the operator of the web server can legitimately charge class $i+1$ clients a usage rate 50% higher than that of class i clients. In addition, 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 of client requests into different service classes, so that an admitted client's performance requirement can be satisfied.
- Dynamic adaptation so that, depending on the server workload, a client can assign requests to a lower service class (than 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. 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 non-cooperative game approach and has low computational complexity. In Section 5, we present experimental results to illustrate the effectiveness of the proposed algorithms. Section 6 discusses related work, and Section 7 concludes.

2. BACKGROUND & PROBLEM FORMULATION

We review proportional-delay differentiated services (PDDS) [4, 9, 10]. Under PDDS, there are $N > 1$ service classes such that class i requests will receive better performance compared with class $i-1$ requests, for $i = 1, \ldots, N$. We consider performance as the average waiting time of a client's requests. The waiting time of a request is the time the request spends in the server's queue before it receives service. 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, $r_{i,i+1}$. Specifically,

$$W_i/W_{i+1} = r_{i,i+1} \qquad \text{for } i = 1, \ldots, N-1 \qquad (1)$$

The objective is to maintain $r_{i,i+1} > 1$ across a wide range of system loadings. As mentioned, PDDS can be achieved using the time-dependent priority (TDP) scheduler [4]. In general, TDP is a *non-preemptive* priority scheduling algorithm with a set of control variables $b_i, 1 \le i \le N$, where $0 \le b_1 \le b_2 \le \cdots \le b_N$. The control variable b_i dictates the *instantaneous* priority of a class i request. Specifically, if the k-th request of class i arrives at the system at time τ_k, then its priority at time t (for $t \ge \tau_k$), denoted by $q_i^k(t)$, is

$$q_i^k(t) = (t - \tau_k) b_i \ . \qquad (2)$$

Let $N_i(t)$ denotes 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 $\mathcal{S}$ is ready to service a request at time t, it chooses a request from class i^* where

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

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 [9, 10], the authors derive the *necessary* and *sufficient* conditions for feasible delay ratios (Equation (1)). 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, $i = 1, \ldots, N$. For detailed derivation of these control parameters values, please refer to [9, 10].

Consider a PDDS web server offering, say, video-on-demand. 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.

We assume there are $M > 0$ potential clients requesting service from a PDDS-enabled server $\mathcal{S}$. Each of these clients is an aggregation of many individual users, e.g., users from the same company or the same network domain. 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 $A_1 \leq A_2 \leq \cdots \leq A_N$. $\mathcal{S}$ also charges a usage cost of ϕ_i for each request in class i, where $\phi_1 \leq \phi_2 \leq \cdots \leq \phi_N$.

The problems we try 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 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 using 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 condition at $\mathcal{S}$ 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 to the next section, let us define the following notation. Let M' be the number of admitted clients to the PDDS server $\mathcal{S}$. We have $M' \leq M$. 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[1]. The class assignment for client i is $C_i^a \in \{1, 2, \ldots, N\}$, for $i = 1, \ldots, M'$. 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 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 $\boldsymbol{C}(t) \leq \boldsymbol{C}^a$ for $t > 0$. The total maximum arrival rate of

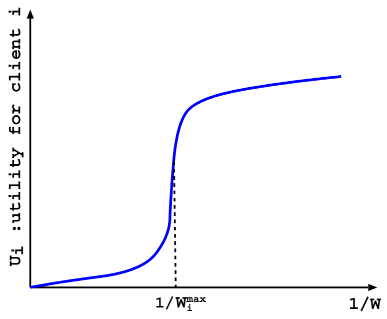


Figure 1: General form of client's utility function vs. inverse of the waiting time

class k, denoted as $\Lambda_{c_k}^{max}$, is:

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

Let W_{c_k} denote the average waiting time of class k requests. Based on the conservation law [7], 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 Equation (1), we have $r_{i-1,i} = W_{c_{i-1}}/W_{c_i}$. Therefore

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

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

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

Substituting Equation (5) into Equation (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

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

Let the function U_i represents 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 Figure 1. We define the system efficacy V as the sum of the utilities of all the admitted clients. Our objective is to

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

i.e., we seek to maximize the system efficacy V under the constraint that the expected waiting time of an admitted

[1]The system will assign a class value of 0 to those clients that the system cannot admit.

client i is less than or equal to its QoS requirement W_i^{max}. In [11], authors show that for a special class of utility functions, then one needs to apply admission control to maximize V. If a request has an utility function with the form similar to Figure 1, then one needs to apply admission control to maximize the system efficacy V.

In general, the optimization problem in Equation (7) can be computationally expensive. A straightforward approach is to perform an exhaustive search. In this case, it has a computational complexity of $\Theta((N+1)^M)$ in evaluating the expression of Equation (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. We propose two efficient admission control algorithms such that at the end of the admission control process, we can determine (1) which are the clients that the system can admit and, (2) the lowest possible admitted class vector $\boldsymbol{C}^a = [C_1^a, \cdots, C_{M'}^a]$ for the admitted clients.

3. ADMISSION CONTROL AND RESOURCE PROVISIONING

In this section, we explain how to perform admission control and class assignment for a PDDS web server.

3.1 Admission Control & 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 requirement W_j^{max}. We propose the following two admission control/class assignment algorithms.

3.2 Maximum Profit Algorithm (MPA)

Under MPA, the objective is to admit a client who has 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 as:

MPA Admission Control

```
1. Sort the maximum average waiting time requirement of
   all clients from smallest to largest. After the sorting, let
   us assume client 1 (respectively, client $M$) has the small-
   est (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 */
18return ($\Omega$ and $\boldsymbol{C}^a$);
```

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 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 the computational complexity and some important properties of the MPA admission control algorithm.

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

Proof: Please refer to [8]. ∎

Before we present the properties of MPA admission control, let us define the following notation and then state some preliminary results. Let $\boldsymbol{\Lambda} = [\Lambda_{c_1}^{max}, \cdots, \Lambda_{c_N}^{max}]$ be the arrival rate vector of different classes of requests. We define $\boldsymbol{e}_i$ as a row vector of zero with the i^{th} 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 $\boldsymbol{\Lambda}' = \boldsymbol{\Lambda} - \lambda_m^{max}\boldsymbol{e}_i + \lambda_m^{max}\boldsymbol{e}_j$. 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 $k = 1, 2, \ldots, N$.*

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 $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_N$. When a client m upgrades from class i to class j, we can easily observe that the denominator of Equation (6) will decrease while the numerator will remain unchanged. Therefore $W_{c_k}(\mathbf{\Lambda}') \geq W_{c_k}(\mathbf{\Lambda})$. ■

Lemma 3. *If a client m performs a class downgrade from class j to i ($i < j$), then the average waiting time for all classes will also decrease.*

Proof: The proof is similar to the previous lemma. ■

Definition 1. *Let $\boldsymbol{C}$ and $\boldsymbol{C}'$ be two class vectors. We say $\boldsymbol{C} > \boldsymbol{C}'$ iff $C_i \geq C_i'$ and $\exists\ j$ where $C_j > C_j'$.*

Definition 2. *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.*

Definition 3. *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 1. *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: Please refer to [8]. ■

Remark: The implication of Lemma 1 and Theorem 1 is that not only do we have an efficient admission control algorithm, but the resulting admitted vector $\boldsymbol{C}^a$ is also a minimum feasible vector. Therefore, we can ensure that we can 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 more 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.

3.3 Maximum Admission Algorithm (MAA)

Under 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 given as:

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 $M$ */
18for ($i = i^*$+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 */
35return ($\Omega$ and $\boldsymbol{C}^a$);
```

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 $\Omega^{'}$ (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 (line 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 greater than or equal to that of client i^*. Therefore, we can do a lot of pruning by skipping those clients whose arrival rates are larger than the arrival rate of client i^* because the server $\mathcal{S}$ cannot admit this client for sure. In the following, we present the computational complexity and properties of the MPA admission control.

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

Proof: Please refer to [8]. ■

Theorem 2. *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 [8] ■

4. DYNAMIC CLASS ADAPTATION

Based on the admission control algorithms proposed in Section 3, the 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 web server $\mathcal{S}$ with a class value less than or equal to C_j^a and still 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 web server $\mathcal{S}$ can measure the necessary information and send feedback control information back to all admitted clients. Assume that the server $\mathcal{S}$ has completed the admission control process (via either 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.

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 web server in a class range that is between the new class value and the original admitted class value.

Formally, let $\boldsymbol{C}(nT)$ denotes the class vector at the end of the n^{th} 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 n^{th} 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 upgrade and will continue to submit requests based on the current class value. This way, client j can reduce its usage cost for $\mathcal{S}$. Figure 2 illustrates an example in which client j performs a class upgrade at instants τ_1, τ_2, τ_3 and τ_4.

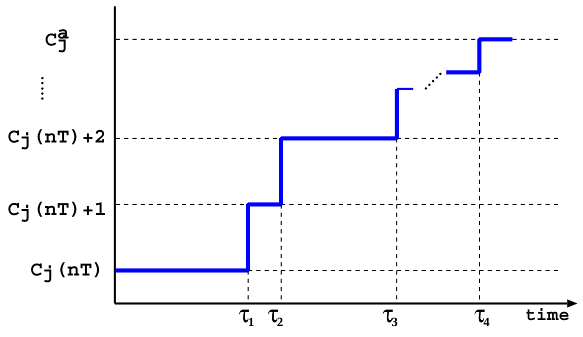


Figure 2: Class adaptation by client j within a measurement period.

There are two important points:

- $C_j(nT)$ 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 arrival rates, the resulting class vector $\boldsymbol{C}(nT)$ is less than or equal to $\boldsymbol{C}^a$.
- If client j tags its requests in class C_j^a, j is assured that its requests will definitely satisfy its QoS requirement.

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 Equation (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 many unicast 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 & Game-Theoretic Approach: Client-Based Dynamic Adaptation (CBDA)

The SBDA algorithm can be 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 tracking 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 $(n-1)^{th}$ and the n^{th} request 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^{th} request arrival. The estimation is

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

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 $\hat{\boldsymbol{\lambda}}_j = [\hat{\lambda}_{j,c_1}, \hat{\lambda}_{j,c_2}, \cdots, \hat{\lambda}_{j,c_N}]$. Upon receiving $\hat{\boldsymbol{\Lambda}}_c$, client j executes the following code:

```
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 $\boldsymbol{\Lambda}^*$ in class $k$ */
4.    $\boldsymbol{\Lambda}^* = \hat{\boldsymbol{\Lambda}}_c - \hat{\boldsymbol{\lambda}}_j + \tilde{\lambda}_j \boldsymbol{e}_k$;
5.    Based on Eq.(6) and $\boldsymbol{\Lambda}^*$, compute delay for client $j$;
6.    if (computed delay $\leq W_j^{max}$)
7.      selected_class = $k$;
8. } /* terminate for loop */
9. return (selected_class);
```

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 *non-cooperative game* problem in which distributed optimization is performed by each client. For an introduction to the basic concepts of game theory, please refer to [5]. 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 the client is considered to be negligible.
2. It is impractical or too 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:

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

Table 1: MPA vs. MAA in number of admitted clients M'.

- **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, re-execute 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.

Of course, 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.

Experiment 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 are (1) the number of admitted clients M', (2) the arrival rates of different classes, and (3) the achieved waiting time for different classes of requests. Unless otherwise state, 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* admission control. The web server supports $N = 3$ classes of requests and their waiting time differentiations are $r_{1,2} = 1.4, r_{2,3} = 1.4$.

In Experiment 1.A, we vary the number of potential clients that want to access the server $\mathcal{S}$ by $M = \{1000, 2000, 5000\}$. The maximum average waiting time requirements of the clients are drawn uniformly between $[1.5, 5.5]$ seconds and the aggregate request rate λ_r is set to one. Since this rate can saturate the system ($\rho = 1$), it is necessary for us to perform admission control. Table 1 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 its tries to admit clients with less stringent maximum average waiting time requirements first. This also indicates 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.

	class #	MPA	MAA
$\lambda_r = 0.5$	class 3	—	—
	class 2	—	—
	class 1	0.500	0.500
$\lambda_r = 0.75$	class 3	0.048	0.048
	class 2	0.129	0.129
	class 1	0.573	0.573
$\lambda_r = 1.0$	class 3	0.137	0.173
	class 2	0.176	0.245
	class 1	0.455	0.406

Table 2: MPA vs. MAA: arrival rates of different classes.

	class #	MPA	MAA
$\lambda_r =$ 0.5	class 3	—	—
	class 2	—	—
	class 1	0.993 (2.010,10.704)	0.993 (2.010,10.704)
	$r_{1,2}$	—	—
	$r_{2,3}$	—	—
$\lambda_r =$ 0.75	class 3	1.649 (2.010,2.307)	1.649 (2.010,2.307)
	class 2	2.309 (2.333,3.229)	2.309 (2.333,3.229)
	class 1	3.232 (3.243,10.704)	3.232 (3.243,10.704)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$
$\lambda_r =$ 1.0	class 3	1.996 (2.010,2.783)	2.946 (3.010,4.118)
	class 2	2.795 (2.797,3.910)	4.125 (4.126,5.739)
	class 1	3.913 (3.915,7.279)	5.775 (5.800,10.704)
	$r_{1,2}$	$r_{1,2} = 1.4$	$r_{1,2} = 1.4$
	$r_{2,3}$	$r_{2,3} = 1.4$	$r_{2,3} = 1.4$

Table 3: MPA vs. MAA: waiting time of different classes. 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 particular class.

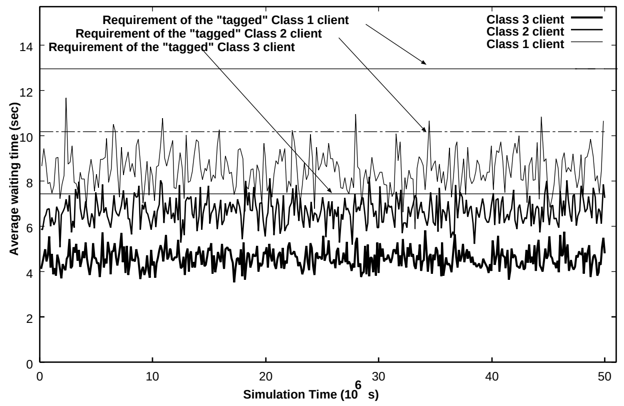
In Experiment 1.B, we set the number of potential clients M to 1000. We vary the aggregate request arrival rate λ_r to be 0.5, 0.75 and 1.0. The maximum average waiting times of all clients are drawn uniformly between $[2, 11]$ seconds. Table 2 and Table 3 illustrate that, after the admission control and client classification, the arrival rates and the achieved waiting times of the 3 classes of requests. From Table 2, 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 1 (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 times are unrealizable and classify clients to the lowest admissible class.

Table 3 depicts the achieved waiting time for different classes under the MPA and MAA algorithms. The numbers in parenthesis 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 is 2.010 (respectively, 10.704) seconds and the achieved waiting time is 0.993 seconds. From Table 3, we observe that both the MPA and the 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 $r_{i,i+1} = 1.4$.

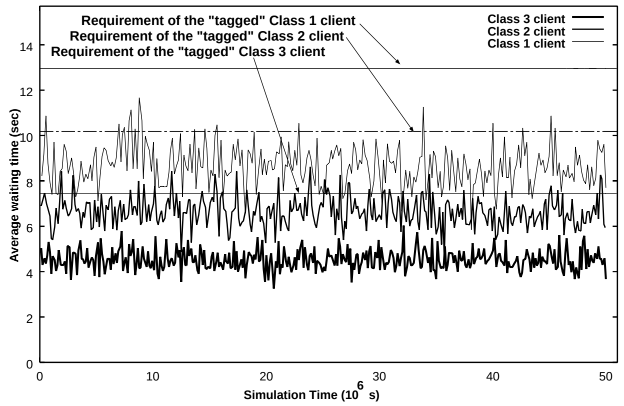
	class #	MPA	MAA
$r_{i,i+1} = 1.3$	class 3	1.485 (1.504,1.927)	2.054 (2.095,2.668)
	class 2	1.931 (1.932,2.508)	2.670 (2.670,3.451)
	class 1	2.510 (2.512,3.262)	3.471 (3.479,5.308)
	W_1/W_2	1.299	1.300
	W_2/W_3	1.300	1.299
$r_{i,i+1} = 1.4$	class 3	1.485 (1.504,2.073)	2.084 (2.095,2.915)
	class 2	2.079 (2.080,2.908)	2.917 (2.919,4.078)
	class 1	2.911 (2.912,3.219)	4.084 (4.087,5.308)
	W_1/W_2	1.400	1.400
	W_2/W_3	1.400	1.399
$r_{i,i+1} = 1.5$	class 3	1.118 (1.504,1.671)	1.616 (2.106,2.373)
	class 2	1.676 (1.679,2.512)	2.424 (2.425,3.621)
	class 1	2.515 (2.515,3.213)	3.636 (3.643,5.308)
	W_1/W_2	1.500	1.500
	W_2/W_3	1.499	1.500

Table 4: MPA vs. MAA: waiting time of different classes under different waiting time spacings $r_{i,i+1}$. 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 particular class.

Lastly, Experiment 1.C illustrates 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 $[1.5, 5.5]$ seconds. From Table 4, we observe that both the MPA and the MAA algorithms can effectively classify clients to the lowest admissible classes so that their QoS requirements are satisfied. At the same time, the achieved waiting time ratio is very close to the specified waiting time ratio $r_{i,i+1}$.



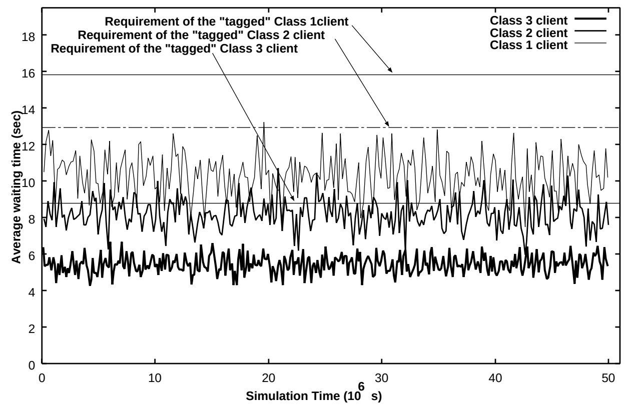
(a) SBDA: Admission control using MPA



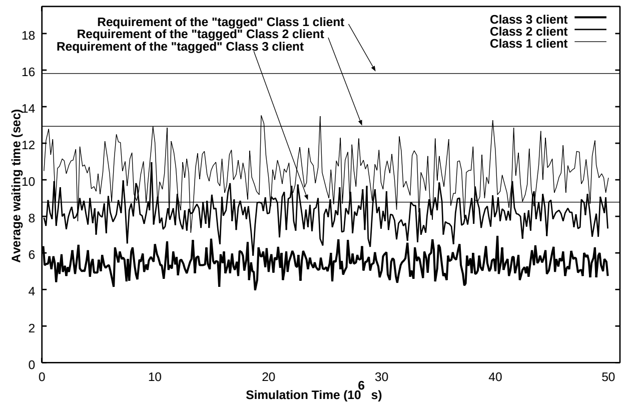
(b) CBDA: Admission control using MPA

Figure 3: Waiting time for three different clients under MPA admission control.

Experiment 2: (Comparison of SBDA and CBDA Adaptation Algorithms) In this experiment, we compare the performance of the SBDA and the CBDA adaptation algorithms. The waiting times of the clients are drawn uniformly from $[6, 22]$ 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



(a) SBDA: Admission control using MAA

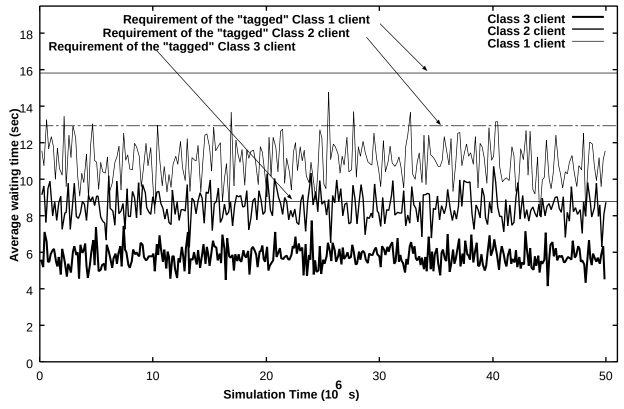


(b) CBDA: Admission control using MAA

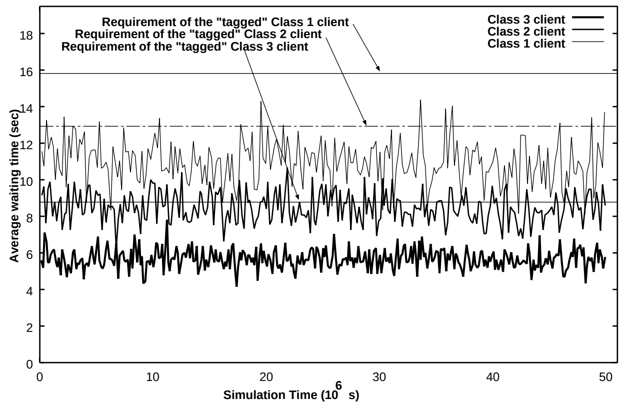
Figure 4: Waiting time for three different clients under MAA admission control.

rate, with probability of 0.1 that the arrival rate is equal to 90% of the maximum arrival rate, and with probability of 0.1 that the arrival rate is equal to 80% 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.

Figure 3 and Figure 4 illustrate the waiting times 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 Figure 3, we use MPA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum waiting time requirements of 12.957, 10.174, 7.44 seconds, respectively. For Figure 4, we use MAA as the admission control algorithm at time $t = 0$. The three tagged clients have maximum average waiting time requirements of 15.824, 12.948, 8.781 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. Note that, since the CBDA algorithm has a much *lower computational complexity* as compared to the SBDA algorithm, we should use CBDA for performing the endpoint adaptation.



(a) SBDA: Admission control using MAA



(b) CBDA: Admission control using MAA

Figure 5: Waiting time for three different clients under MAA admission control, MMPP arrival process.

Lastly, we consider the capability of the proposed adaptation algorithms when the input traffic is *non-Poisson*. In this experiment, the aggregated request arrival rate has a mean of $\lambda_r = 1.2$. However, the traffic generation by each client is modeled as a Markov-modulated Poisson process. Figure 5 illustrates the waiting time of SBDA and CBDA algorithms under the MMPP arrival process. The admission control was carried out using the MAA. The three tagged clients have the maximum average waiting time requirements of 15.824, 12.948, 8.781 seconds respectively. From Figure 5, we observe that both the SBDA and CBDA can adapt to the workload and their waiting time is less than their maximum average waiting time requirement. Once again, since CBDA has a much lower computational complexity, we should use CBDA to perform the end-points 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 [1, 2, 12] in order to complement the Internet differentiated services model. In [1], the authors propose a centralized algorithm to perform server partitioning so as to provide differentiated services. In [2], 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 [12], the authors consider a server that provides prioritized service to different classes of users. In [6], 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 [6] are not PDDS but it is interesting to see how one can incorporate the proposed algorithms into our work. Lastly, the authors in [3] propose a method to select classes under PDDS so that requests can achieve an absolute QoS measure. The major differences between our work and [3] 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.

7. CONCLUSION

In this paper, we consider a web server that can provide proportional delay differentiated services. The advantage of this type of service is that the operator of the web server can provide a *fixed* and *pre-specified* performance spacing between different classes of requests. Based on this performance spacing, 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 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 non-cooperative game technique. We show that the distributed approach has lower computational cost and can dynamically adapt to the server's workload.

Acknowledgments: We wish to thank the anonymous referees for their helpful and insightful comments.

8. REFERENCES

[1] V. Cardellini, E. Casalicchico, M. Colajanni, and M. Mambelli. Web switch support for differentiated services. In *Performance and Architecture of Web Servers (PAWS), Boston*, June 2001.

[2] M. Crovella, R. Frangioso, and M. Harchol-Balter. Connection scheduling in web servers. In *Proceedings of USITS'99, Boulder*, October 1999.

[3] C. Dovrolis and P. Ramanathan. Dynamic class selection: from relative differentiation to absolute qos. *Proceedings of the 2001 IEEE International Conference on Network Protocols*, November 2001.

[4] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional differentiated services: Delay differentiation and packet scheduling. In *ACM SIGCOMM'99*, pages 109–119, August 1999.

[5] R. Gibbons. *Game Theory for Applied Economists.* Princeton University Press, 1992.

[6] V. Kanodia and E. Knightly. Multi-class latency-bounded web services. In *IEEE/IFIP IWQoS 2000, Pittsburgh, PA*, June 2000.

[7] L. Kleinrock. *Queueing Systems: Vol 2.* Wiley-interscience, New York, 1976.

[8] S. C. M. Lee, J. C. S. Lui, and D. K. Y. Yau. Admission control and dynamic adaptation for a proportional delay diffserv-enable web server. In *Technical Report, CUHK, Department of Computer Science & Engineering*, 2001.

[9] M. K. H. Leung, J. C. S. Lui, and D. K. Y. Yau. Characterization and performance evaluation for proportional delay differentiated services. In *International Conference on Network Protocols*, pages 295–304, November 2000.

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

[11] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal of Selected areas in Communication*, 13:1141–1149, 1995.

[12] H. Zhu, H. Tang, and T.Yang. Demand-driven service differentiation in cluster-based network servers. In *Proc. IEEE Infocom 2001, Anchorage, Alaska*, April 2001.