Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Computers & Operations Research 32 (2005) 1147–1164
computers & operations research
www.elsevier.com/locate/dsw

Web server QoS models: applying scheduling rules from production planning

Nong Ye*, Esma S. Gel, Xueping Li, Toni Farley, Ying-Cheng Lai

Department of Industrial Engineering, Arizona State University, P.O. Box 875906, Tempe, AZ 85287, USA

Abstract

Most web servers, in practical use, use a queuing policy based on the Best Effort model, which employs the first-in-first-out (FIFO) scheduling rule to prioritize web requests in a single queue. This model does not provide Quality of Service (QoS). In the Differentiated Services (DiffServ) model, separate queues are introduced to differentiate QoS for separate web requests with different priorities. This paper presents web server QoS models that use a single queue, along with scheduling rules from production planning in the manufacturing domain, to differentiate QoS for classes of web service requests with different priorities. These scheduling rules are Weighted Shortest Processing Time (WSPT), Apparent Tardiness Cost (ATC), and Earliest Due Date. We conduct simulation experiments and compare the QoS performance of these scheduling rules with the FIFO scheme used in the basic Best Effort model with only one queue, and the basic DiffServ model with two separate queues. Simulation results demonstrate better QoS performance using WSPT and ATC, especially when requested services exceed the capacity of a web server.

Keywords: Web server; Quality of Service (QoS); Scheduling rules; Simulation model

1. Introduction

The World Wide Web has become one of the most popular and important applications on the Internet. A web server receives numerous requests for its services, which it cannot handle at the same time, and will typically use a buffer or queue to store incoming requests awaiting service. Requests in the queue are typically stored in order of arrival. The web server will take the request at the front of the queue, and service it first. This is an example of first-in-first-out (FIFO) scheduling. Most existing web servers provide services based on the Best Effort model, using FIFO scheduling.

* Corresponding author. Tel.: +1-480-965-7812; fax: +1-480-965-8692.
E-mail address: nongye@asu.edu (N. Ye).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2003.10.001

Quality of Service (QoS) has three main attributes: timeliness, precision, and accuracy [1,2]. A service request can be considered a process with an input and output. Timeliness measures how fast an output is produced for a given input. Precision measures the quantity, and accuracy measures the quality, of the output. We often see such measures as delay, response time, and jitter used for the timelines attribute; throughput, bandwidth, and loss rate (e.g., packet drop rate) used for the precision attribute; and error rate used for the accuracy attribute. Tradeoffs often need to be made among these attributes.

The Best Effort model of a web server does not provide QoS because the completion time of a request depends on how many requests are already in the queue, and thus cannot be predicted for timeliness assurance. Furthermore, in any set of web requests, some may have a high priority, while others have a low priority. Bhatti and Friedrich [3] discuss classifying web requests into high, medium, and low priorities using such information as IP addresses and requested web sites. The Best Effort model does not support the differentiation of requests based on their priority. Instead, requests are treated "fairly" based on their arrival time.

The lack of QoS in the Internet today makes it a highly vulnerable system in its current role of supporting critical operations in many sectors of our society. QoS has been considered as a "must" for the next-generation Internet [4,5]. Existing work on QoS for the Internet falls into two general frameworks: Integrated Services (IntServ) and Differentiated Services (DiffServ), both of which are concerned mostly with timeliness assurance.

In the IntServ framework, applications are placed in two classes based on timeliness requirements: real-time (hard or strict) and elastic (soft) [6–11]. IntServ aims to provide per flow QoS by reserving bandwidth along a source-destination path to assure timeliness of data delivery. Many scheduling rules and admission control strategies have been proposed in the IntServ framework to assure a bound on end-to-end timeliness [12–15]. Because it requires that every hop on the end-to-end path maintain the state of all bandwidth reservations for each Internet connection, IntServ is not scalable. Furthermore, a bound on end-to-end timeliness usually depends on the number of hops on an end-to-end path, which is difficult to predict, implying that there can exist no absolute bound on end-to-end timeliness.

The DiffServ framework marks data at the edges of a network with two classes of priority: premium and best effort, which, respectively, have a high and low priority for being serviced [16–18]. In a router or host, two separate queues with different capacities are used, one for each class of data. Because the vast majority of data is classified as Best effort, the capacity of the premium queue is usually much smaller. Data in the premium queue is served first, whereas data in the Best Effort queue is served only when the premium queue is empty. Lu et al. [19] use a feedback control method to address the relative delay or jitter in a web server for assuring QoS. Some studies use load balancing to address QoS [20–22]. Still other QoS methods based on DiffServ have been explored [23–25].

Research in providing QoS on a web server takes different approaches. Ferrari [26] investigates the effect of aggregation on performance using Priority Queuing (PQ) and Weighted Fair Queuing (WFQ) scheduling algorithms. Chen and Mohapatra [27] exploit the dependence among session-based requests and propose a dynamic weighted fair sharing (DWFS) scheduling algorithm to control overloads in web servers.

We investigate Internet QoS by providing QoS on web servers. We use only one queue for all web requests, and an advanced scheduling rule to differentiate services and improve overall QoS. Many

advanced scheduling rules have been developed for production planning in the manufacturing domain [28]. If we consider a web server as a production system, those scheduling rules can be adopted. In this study, we investigate the application of three scheduling rules from production planning. These scheduling rules are: Weighted Shortest Processing Time (WSPT), Apparent Tardiness Cost (ATC), and Earliest Due Date (EDD). EDD is also known as Earliest Deadline First (EDF) [29]. We compare the QoS performance of these rules with that of FIFO in the basic Best Effort model with only one queue, and that of a basic DiffServ model for a web server with two separate queues.

The rest of this paper is organized as follows: First, we define the QoS models of a web server using WSPT, ATC, and EDD, along with basic Best Effort and DiffServ models. Next, we describe the simulation of these web server models using a commercial simulation tool for computer networks (OPNET), and the simulation experiments conducted under various data conditions to discover the QoS performance of the models. Finally, we present the simulation results and summarize our findings.

2. Web server QoS models

We introduce high level frameworks for web server QoS models to provide QoS at the application layer. The units queued in these frameworks are considered as complete web requests. Similar to real-time applications where jitter and delay are critical to performance, timeliness is also important to web requests, especially for some commercial web sites where a late response may cause clients to turn to rival web sites. With the increasing development of e-commerce, it is significant to provide timeliness on web sites to attract users. Loss of users means loss of profit. Our study focuses on the server side of a web site. Any incoming request is treated as a new request, regardless of whether it is a resubmitted, previously dropped request, or an entirely new request.

QoS requirements on the Internet primarily come from end users. A web server provides services to end users, who can specify their QoS requirements for these services. Web requests on the Internet today do not come with any indication of their priority or other QoS requirements, such as expected delay. We anticipate that in the future web requests will be accompanied by such data in order to allow for QoS assurance on the Internet. In this study, we assume that each web request comes with the following information:

- Priority
- Requested data (from which we derive its size)
- Due date (required completion time of request)
- Arrival time (determined at time of arrival).

In our study it is assumed that it is not entirely up to the web client to set the priority. Clients cannot set priorities without web server authentication. Otherwise, everyone, including malicious users, will give themselves the highest priority. Clients' access history, user name and password, IP address, and current status (e.g., a customer with a full shopping cart or with a purchasing history gains a higher weight than a customer without a purchasing history or a full cart) can be used to prioritize the requests. Web QoS allows the incoming requests to be categorized as high, medium, or low priority based on IP address, requested URL, and so on [5].

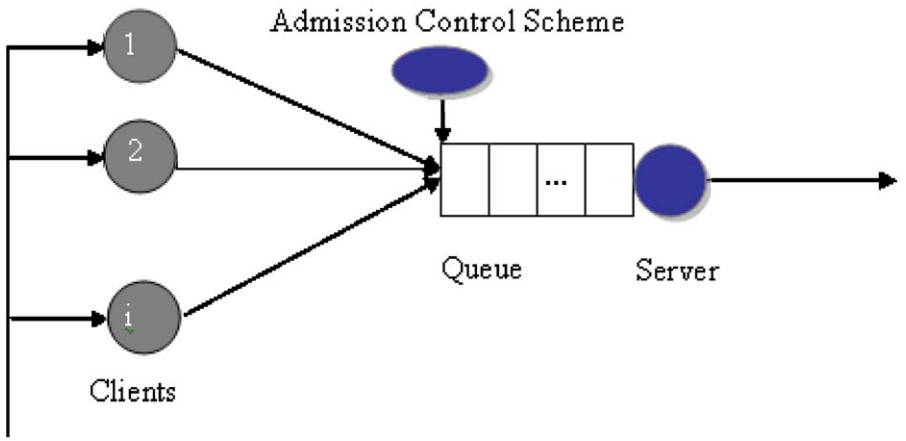

Fig. 1. QoS model of a sequential web server.

The complete round trip time, or delay, of a web request includes time waiting in the queue, finding and retrieving requested data, and sending it to the web client. In this study, for each web request, we ignore the time spent finding and retrieving the requested data, and consider only the waiting time in the queue and the time sending requested data to the requesting web client.

In this paper, we model a web server as a sequential single server, which means that the web server processes incoming requests one at a time as shown in Fig. 1. This concept is different from the traditional multiprocessor or mutlithreaded web server model. We argue that this is reasonable in terms of web server QoS. For example, if bandwidth were the performance bottleneck under scrutiny, then we would argue that high priority requests should not share bandwidth with lower priority requests, thereby requiring a multiprocessor or multithreaded solution.

In our study, we are addressing scheduling issues relating to the web server itself. Such solutions could be scalable to a multiprocessor or multithreaded web server, when applied at each level of redundancy. In this section we present the models used in our experiments.

2.1. The basic Best Effort model

The basic Best Effort model of a web server consists of three elements, as shown in Fig 1: incoming web requests, a queue to sort and keep incoming requests before they are extracted for servicing, and a server to process incoming requests and provide web services. The basic best-effort model uses only the arrival time to sort incoming requests in the queue using the FIFO rule. A request is dropped by the web server if it has been in the queue for longer than some "timeout" threshold (for example, 90 s).

The Internet and most corporate intranets are built using the IP protocol, which is a connectionless protocol that provides no guarantees of service time or the relative ordering of packets. For the Best Effort web servers in such networks, there is no admission control scheme. Hence, if client requests are placed into the web server queue faster than they are removed for processing, congestion occurs, resulting in delayed requests and requests dropped due to the server's timeout threshold. Thus, the Internet can only provide a single level of service; that of Best Effort.

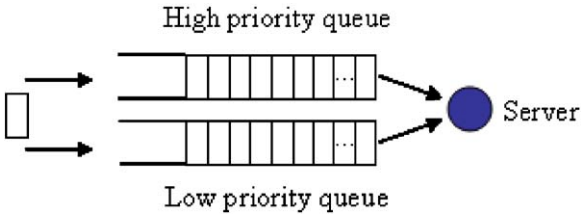

Fig. 2. Basic DiffServ model.

2.2. The basic DiffServ model

Under the DiffServ policy, requests are categorized into priority classes and placed in separate queues. The server always processes the higher priority queue before serving any of the lower priority queues. DiffServ architectures generally define two types of classification: (1) the *behavior aggregate classifier*, which selects packets according to the Diffserv codepoint (DSCP) value stored in the IP header on ingress, and (2) the *multifield classifier*, which uses a more general set of classification conditions like IP header field values and source address.

To implement a DiffServ model, we classify the incoming requests into two categories: high and low (best effort) priority, based on their assigned weights. Fig. 2 shows the two queues used for these two service classes, both of which are serviced in a FIFO manner. In this model, the best effort queue will only be serviced when there are no requests waiting in the high priority queue. Again, as in the Best Effort model, there is no admission control in the basic DiffServ model.

2.3. WSPT

A number of scheduling rules have been developed in the manufacturing domain to schedule a set of jobs on a single machine, given that all of the jobs are available at time=0. The WSPT scheduling rule is one such rule that schedules a set of jobs by decreasing order using a formula which includes the priority weight and processing time of the job. The completion time of a job is defined as the time elapsed between time=0 and the time that the machine finishes processing the job.

It is shown that the WSPT scheduling rule minimizes the weighted completion time for a set of jobs [28]. A similar technique is used in operating systems for scheduling processes on a single CPU. This scheduling algorithm, called shortest job first (SJF), is "provably optimal" [30]. However, the SJF algorithm does not take priorities into account. We investigate the application of WSPT to scheduling in a QoS model of a web server because WSPT incorporates the weight factor, thereby allowing the differentiation of web requests with various priority weights. The WSPT rule also incorporates the processing time of a job, to minimize the sum of the completion times for a set of jobs. Hence, the application of the WSPT rule to scheduling in a QoS model of a web server enables us to minimize the delays of web requests.

Using the WSPT rule, we schedule incoming web requests in a web server queue in decreasing order by their priority values. The priority value of a web request j is determined by w_j/p_j, where

w_j is the given priority weight for the request j, and p_j is its service time, which is calculated from the size of the requested data, d_j, as follows:

$$p_j = \frac{d_j}{s}. \tag{1}$$

In Eq. (1), s is the service rate or data transmission rate (the amount of data that can be transmitted per time unit) of the web server. Note that the higher the weight and the shorter the service time of a web request, the higher its priority value will be.

When a new request arrives at the web server, its priority value is computed, and the request is then inserted into the queue according to this value. Before a new request is inserted into the queue, we first make an admission control decision based on its expected completion and due dates. Deferring an incoming request at the very beginning of the transaction, rather than in the middle, is a desirable scheme for an overloaded web server. First of all, it avoids further frustration on the client side by refusing to accept requests for which it cannot satisfy the respective QoS requirements (e.g., limits on cycle time. lateness and tardiness, etc.). Secondly, it keeps the queue levels relatively stable, resulting in less variable output.

QoS requirements are determined by a web request's QoS factor. We define the QoS factor of an incoming request as:

$$Q_j = D_j - T_j - W_j. \tag{2}$$

In Eq. (2), D_j is the due date of the web request j, T_j is its arrival time, and W_j is its predicted waiting time (i.e., the sum of the processing times of the requests ahead of request j). According to our assumed admission control scheme, if Q_j is less than zero, request j will be rejected.

2.4. ATC

The ATC rule combines WSPT and the Minimum Slack (MS) first rule [28]. MS is a dynamic dispatching rule that orders jobs in increasing order of slack, where the slack of job j at time t is defined as $\max\{d_j - p_j - t, 0\}$, where d_j denotes the due date of job j and p_j denotes its processing time as in Eq. (1).

Under the ATC rule, jobs are scheduled one at a time; that is, every time the machine becomes free, a ranking index is computed for each remaining job. The job with the highest-ranking index is then selected to be processed next. The index is defined as

$$I_j(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max\{d_j - p_j - t, 0\}}{k\bar{p}}\right). \tag{3}$$

In Eq. (3), k is a scaling parameter that can be determined empirically, and $\bar{p}$ represents the average processing time of the remaining jobs in the queue at time t. We can see that if k is big enough, the ATC rule will reduce to WSPT, since the ATC index $I_j(t) \to w_j/p_j$ as $k \to \infty$.

In this paper, we use the following equation in place of Eq. (3) to index incoming requests using the ATC rule:

$$I_j^*(t) = \frac{w_j}{p_j} \exp\left(-\frac{\max\{d_j - t, 0\}}{k\bar{p}}\right). \tag{4}$$

In Eq. (4), the processing time p_j is eliminated because the slack of the request equals its due date minus its waiting time, which we previously defined as the time elapsed between the time a request comes into the web server and the time it gets processed.

2.5. EDD

Sequencing in increasing order by due date, EDD, minimizes the maximum lateness of a set of jobs with given due dates and processing times [28]. While we know that EDD is optimal for the case in which all jobs to be scheduled are available at time zero, it is harder to find an optimal policy for the case in which jobs are released in different points in time, such is the case in a web server. This is due to the fact that when preemption of jobs is not allowed, the optimal schedule is not necessarily a non-idling schedule.

Having noted this point, we adopt a static EDD rule in our web server model by placing the incoming requests into the queue according to their due date. Requests with an earlier due date are placed in the front of the queue, and processed before requests with a later due date.

3. QoS measures

We define four QoS measures in this paper: number of dropped jobs per unit time (*drop rate*), average waiting time in the system, average lateness and throughput.

Under the basic Best Effort and DiffServ policies, request drops only occur when the request's waiting time in the queue reaches the TIMEOUT threshold. Under the WSTP, ATC, and EDD policies, request drops may happen at admission control, as well as when the waiting time exceeds the due date while requests are waiting in the queue. Waiting time is used to measure the responsiveness of the web server. Lateness represents the gap between the waiting and due dates, and can be negative or positive. A negative lateness indicates that a request completed before its due date, and a positive lateness indicates that a request was tardy. Lateness depicts how well the due date requirement of the request is met.

4. Simulation model and experiments

We simulate a web server under the policies discussed above using the OPNET Modeler 8.1.A simulation environment. The simulation experiments were conducted on a Micron PC with a single Pentium4 1.9 GHz CPU and 512 MB of RAM running on the Windows 2000 operating system. Fig. 3 depicts the simulation model based on the web server system presented in Fig. 1. The three generator modules generate web requests with different priority weights. The forwarder module forwards the requests to the queue, where the admission control scheme is implemented. The requests are then placed into the queue based on the scheduling rule currently in use. The sink module destroys the requests after the web server processes them.

We define two fields in the data packet format: weight and due date. We assume that the requested data follows a Pareto distribution with shape parameter 65536 and scale parameter 1.4 [31,32]. Thus the mean of the requested data is $\frac{65536*1.4}{1.4-1}$=229376 bits, about 28 K bytes. We calculate the processing

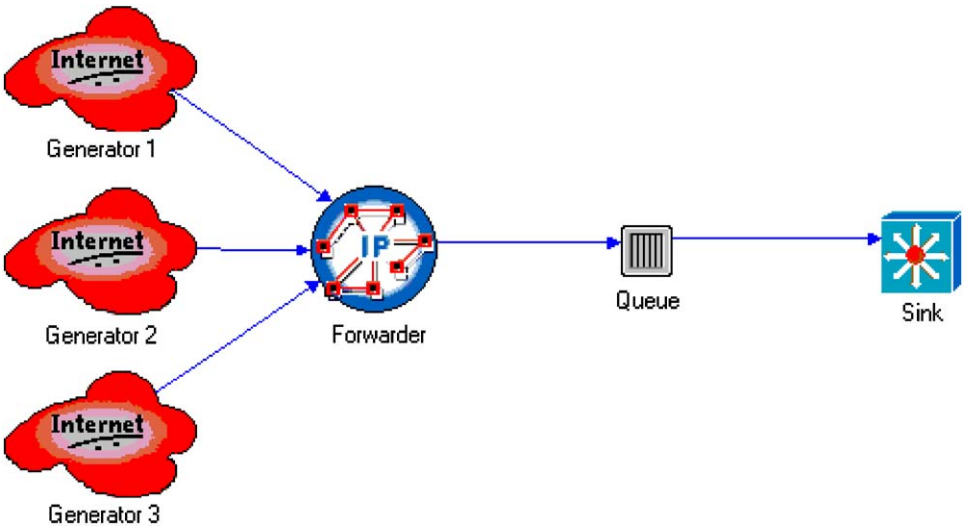

Fig. 3. The topology of the QoS web server simulation.

time by dividing the requested data by a constant, deterministic service rate. For example, if the requested data is 6000 bytes and the service rate of the server is 240,000 bytes per second, the processing time for this request is assumed to be 0.025 s.

The weight field is a 4-bit integer that contains the weight information of the request. The weight value is recorded in the weight field of a request packet when the request is generated. There are multiple choices available to select the proper weight value [3]. For simplicity, we only define three priority classes. In Fig. 3, generators 1, 2, and 3 generate requests with weights 1, 5 and 10, respectively, corresponding to low, medium, and high priority requests.

The due date field in the packets shows the time a request is due. If an application cannot receive a response as required, its QoS requirements cannot be met. Of course, whether or not an application can get its response on time depends not only on how fast the web server processes the request, but also on the Internet transmission delay. How to guarantee a lower bound delay between routers is beyond the scope of this paper. Hence, we focus on the due date, by which the web server is required to have processed a request, and assume that the client will receive its response once the server sends it. We use the same distribution to model the randomness of the due dates for all classes of requests because we assume that they require the same URLs. We also assume that the due dates are normally distributed with a mean of 2 s and a standard deviation of 0.2 s.

In our experiments, we use two traffic conditions: overwhelming and light. We use an exponential distribution to model randomness in the arrival of requests to the web server. For the overwhelming traffic condition, generators 1 and 2 generate requests at a rate of 40 requests per second, while generator 3 generates requests at a rate of 20 requests per second. Hence, the total traffic generated is equal to $(40+40+20)^{*}229376$ bits per second. We set the queue service rate at 12,697,600 bits per second, about T1 speed of 1.55 Mbps, which is around 55.36% of the generated web traffic. The server can process about 55 requests per second.

In the light traffic case, the arrival rate for low and medium priority requests are reduced to 16 requests per second, while the arrival rate for the high priority requests is at a rate of 8 requests

Table 1
Simulation parameters

Traffic case	Class	Weight	Request interarrival time	Due date (s)	Document size (bits)
Overwhelming	LOW	1	Exponential(0.025)	Normal(2,0.2)	Pareto (65536, 1.4)
	MEDIUM	5	Exponential(0.025)	Normal(2,0.2)	Pareto (65536, 1.4)
	HIGH	10	Exponential(0.05)	Normal(2,0.2)	Pareto (65536, 1.4)
Light	LOW	1	Exponential(0.0625)	Normal(2,0.2)	Pareto (65536, 1.4)
	MEDIUM	5	Exponential(0.0625)	Normal(2,0.2)	Pareto (65536, 1.4)
	HIGH	10	Exponential(0.125)	Normal(2,0.2)	Pareto (65536, 1.4)

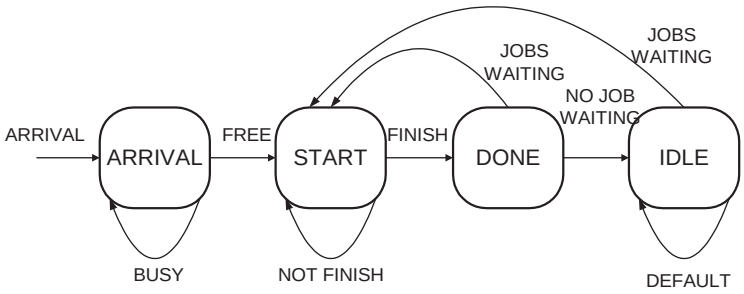

Fig. 4. FSM for sequential web server.

per second. Hence, the total traffic generated reduces to only 72% of the traffic the web server can handle. In both overwhelming and light traffic cases, we use 100 for the scaling parameters in the ATC policy. We set the simulation duration to 4000 s. The parameters for our simulation experiments are given in Table 1.

We use a Finite State Machine (FSM) to simulate processing in a web server. The model actually includes two FSMs. The ARRIVAL state itself is an FSM in which requests come into the queue based on QoS rules. The second FSM includes the three states: START, DONE, and IDLE. The FSMs are shown in Fig. 4.

We describe the processing handled in each state in the FSM shown in Fig. 4 for each of the web server models in our investigation:

ARRIVAL state. Best Effort model: When a request comes in, it enters the ARRIVAL state, and will be placed into the queue using the FIFO rule.

Basic DiffServ model: After a request enters the ARRIVAL state, we get the priority 'weight' value from the packet and place it into the respective queue.

WSPT model: First we execute the admission control algorithm and drop any requests with a QoS factor less than zero. For new requests, we calculate their priority according to equation (1), and insert the request into the queue based on its priority.

ATC model: As an extension of WSPT, we also implement the Admission Control Scheme at the very beginning of the ARRIVAL state. For a new incoming request, we calculate its index using Eq. (4) as its priority for queue placement. We assign $k = 100$; the scaling parameter in our simulation.

EDD model: We perform the Admission Control Scheme when the process enters the ARRIVAL state. We extract the due date information of a new request from the packet's due date field. We use its reciprocal as the priority and insert the request into the queue accordingly.

START state. When the server is not busy, a process enters the START state. In this state, we will remove a request from the head of the queue and schedule it based on its processing time. The FSM will then stay in the START state until the current request is processed.

DONE state. After a request is processed, the FSM enters the DONE state, and the server becomes free again. If there are no jobs waiting for processing, the FSM will go to IDLE, otherwise, the FSM will go back to START to process the waiting requests.

IDLE state. The server is free at the IDLE state. When there are no requests waiting in the queue, the FSM remains in the IDLE state. When a request comes into the ARRIVAL state and is inserted into the queue, the FSM will transition from IDLE to START.

5. Experimental results

In this section, we provide detailed results on our simulation experiments for the heavy traffic and light traffic cases and provide a discussion of our results for each of the QoS performance metrics.

5.1. Overwhelming traffic case

We first present the results of the overwhelming traffic scenario. Due to the "bursty" nature of web traffic, it is not surprising that a web server can become overwhelmed when one minute previous its workload was quite small. A malicious attack that sends a high volume of requests to a web server is another source of overwhelming traffic. We argue that even under the overwhelming traffic condition, our QoS enabled web server can still provide QoS.

The simulation results for this section are given in Table 2. In our tabulated results, STD stands for Standard Deviation which is calculated over time from 600 to 4000 s (a time interval in which the system is in a steady state). BE stands for Best Effort policy and DS stands for basic DiffServ policy. The number in each cell can also be considered a percentage of requests since the columns of each row total the actual number of requests.

5.1.1. Waiting time

The waiting time performance of the Best Effort, basic DiffServ, WSPT, ATC, and EDD policies are shown in Fig. 5. We take all processed requests into account and calculate the waiting time in the steady state window.

We observe that the overall performance is dramatically enhanced under the ATC, EDD, and WSPT policies, as shown in Table 2. These policies employ an admission control scheme. The admission control scheme discards requests whose QoS requirements cannot be met before they are even placed in the queue, which results in a smaller average number of requests waiting in the queue and consequently a shorter waiting time. The average number of requests waiting in the queue is about 25 under WSPT policy, and 67 under EDD policy. Under Best Effort and basic DiffServ, the average number of waiting requests is about 8989 and 7279, respectively.

Table 2
Experimental results of the overwhelming traffic case

Priority class		BE	DS	WSPT	ATC	EDD
Waiting time of requests						
All	Mean	89.96725	56.39645	0.100961	0.104629	0.2742334
	STD	0.004698	7.965471	0.012236	0.013271	0.0301226
High	Mean	89.96703	3.776491	0.021563	0.022253	3.776491
	STD	0.005094	10.2084	0.003994	0.00413	10.2084
Medium	Mean	89.96749	89.95819	0.039224	0.040503	0.273149
	STD	0.00474	0.007214	0.005459	0.005846	0.03198
Low	Mean	89.96762	89.958	0.231369	0.240477	0.274335
	STD	0.004737	0.006525	0.029947	0.032699	0.032978
Requests dropped						
All	Mean	45.34254	46.44422	10.44669	10.63292	46.50868
	STD	11.2525	12.06754	1.174351	1.191997	14.39948
High	Mean	9.001148	0.066765	0.180685	0.199215	9.21321
	STD	2.279903	0.612841	0.067248	0.075172	2.850334
Medium	Mean	18.24159	23.28894	1.001089	1.046383	18.75024
	STD	4.625595	6.022406	0.170591	0.17572	5.782265
Low	Mean	18.10011	23.08864	9.265216	9.387631	18.54573
	STD	4.4637	5.869901	1.061894	1.07052	5.851304
Lateness of requests						
All	Mean	87.964869	54.39429	−1.87864	−1.8751	−1.45155
	STD	0.0124608	7.964984	0.210136	0.2099	0.046376
High	Mean	87.96774	0.4149	−1.95503	−1.95416	−1.4516044
	STD	0.023457	10.20868	0.219513	0.219381	0.051434
Medium	Mean	87.96471	87.95174	−1.93868	−1.93742	−1.45239
	STD	0.01794	0.022036	0.217253	0.217093	0.047151
Low	Mean	87.96591	87.96192	−1.77454	−1.76515	−1.4509
	STD	0.016003	0.028651	0.022221	0.024757	0.049007
Throughput of requests						
All	Mean	12696420	12696350	12697660	12697573	12697636
	STD	2968060	4493832	7463.187	5814.585	3727158
High	Mean	2311544	4241935	3396161	3373713	2290618
	STD	562505.6	1302286	186548.3	178843.3	670560.8
Medium	Mean	5015363	3963317	6083328	6073930	5023274
	STD	1321620	1529590	165117.2	162480	1967933
Low	Mean	5369555	4491137	3218229	3249982	5383762
	STD	2870234	2748046	158414.8	164200.2	3589084

The WSPT and ATC policies also contribute to the stabilized waiting time as shown in Fig. 5. We notice that the variance of the overall Waiting Time under basic DiffServ policy is quite large, which can also be seen from Table 2.

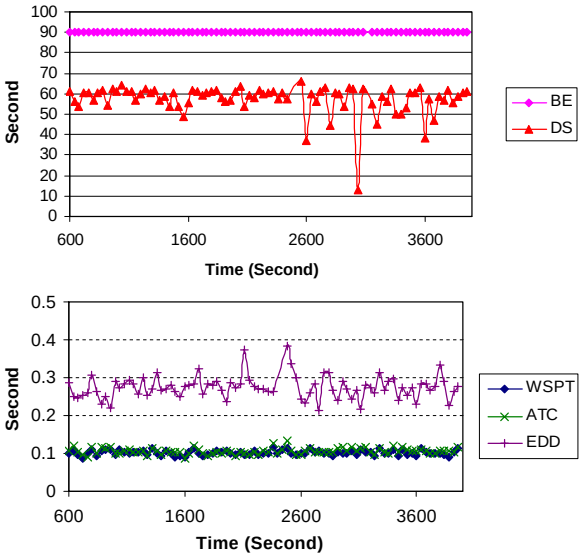

Fig. 5. Overwhelming traffic case: overall waiting time of requests.

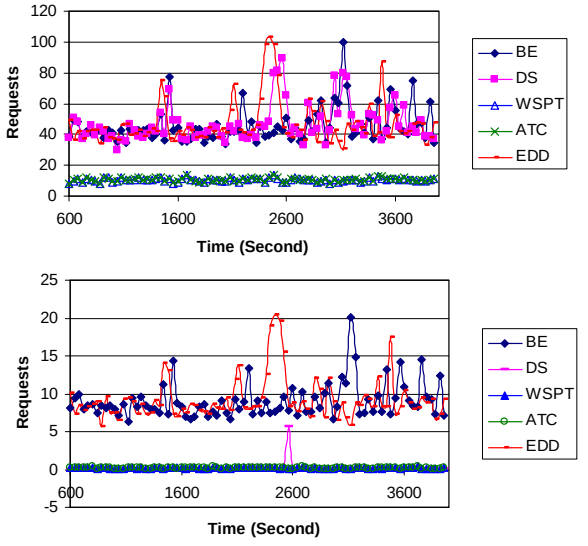

Fig. 6. Overwhelming traffic case: overall drop of all (above) and high priority (below) requests.

5.1.2. *Drop rate*

Fig. 6 shows the overall drop rate of all requests and high priority requests. We observe that the overall dropped requests using Best Effort, basic DiffServ, and EDD is about 45 requests per second as shown in Table 2. However, only about 10 requests are dropped per second under ATC

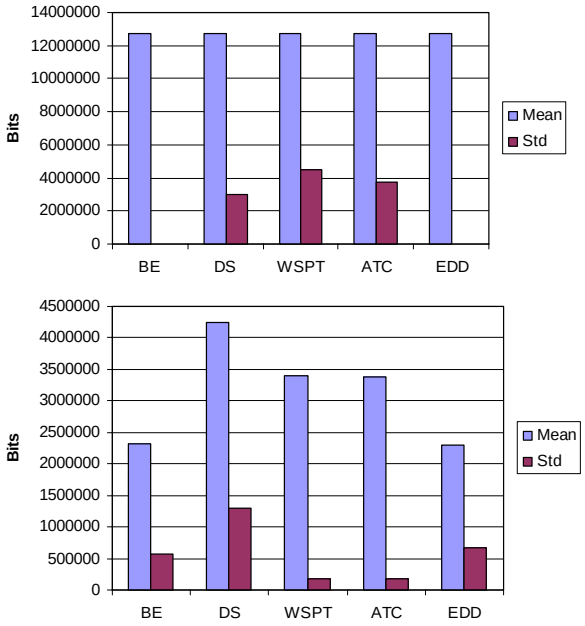

Fig. 7. Overwhelming traffic case: throughput of all (above) and high priority (below) requests.

and WSPT; 22% of Best Effort. This is because the WSPT and ATC policies take processing time into account and discriminate against requests with a long processing time.

In the high priority class of requests, using the basic DiffServ policy has the lowest drop rate because the requests in the low priority queue can only be processed when the high priority queue is empty. We can see from the lower chart in Fig. 6 that some high priority requests may still be dropped. This can happen when the timeout threshold is reached, for example, when a high priority request comes into the system while a request that asked for a large document is being processed.

5.1.3. Lateness

From Table 2, we note that ATC, WSPT and EDD can meet the Lateness QoS requirement. On the other hand, Best Effort and basic DiffServ cannot. In DiffServ, most of the High priority requests can meet the Lateness QoS requirement (negative Lateness), however, the Lateness of some requests can be over 60 s (STD).

5.1.4. Throughput

The mean throughput of all requests is about 12,697,600 bits per second, the service rate of the server, because the web server is overwhelmed (see Table 2). We find that differences in standard deviation of the throughput between the five policies are quite large as shown in Fig. 7.

We observe from the lower chart in Fig. 7 that with High priority class requests, the basic DiffServ policy yields the highest throughput among the five models and EDD has the lowest. The more requests that are dropped, the less the throughput yield, and vice versa.

For Medium priority requests, WSPT has the highest throughput, about 21% more than Best Effort and 53% more than basic DiffServ. For Low priority requests, WSPT and ATC have the lowest throughput and standard deviation (see Table 2). Best Effort and EDD policies have the highest throughput. However, the standard deviations are quite large.

5.1.5. Requests waiting in queue

The average number of requests waiting in the queue under each of the policies is shown in Table 2. There are only 25 requests waiting in the queue under WSPT and ATC policies, while EDD has about 67 requests. Best Effort and basic DiffServ have about 8989 and 7279 requests waiting in the queue, respectively.

5.2. Light traffic case

We now present the results of the light traffic scenario. The simulation results for this section are given in Table 3.

5.2.1. Waiting time

The overall waiting time of the ATC, Best Effort, basic DiffServ, EDD and WSPT policies is shown in Fig. 8. WSPT and ATC have similar performance. The overall waiting time for Best Effort and basic DiffServ are more than 15 times longer than the WSPT and ATC policies.

From Table 3 and Fig. 8, we can see that the deviation of waiting times of the Best Effort and basic DiffServ policies are very large, about 5.87 for Best Effort and 5.49 for basic DiffServ. Congestion can happen under Best Effort and basic DiffServ models at some points, leading to longer waiting times. A request may need to wait more than 40 s before getting service.

5.2.2. Drop rate

Table 3 also includes the drop rate information. There are no requests dropped under the Best Effort and Basic DiffServ policies. There is a tradeoff between waiting time and requests dropped. There are a small amount of requests dropped under ATC, WSPT and EDD policies, due to the admission control scheme, however a shorter waiting time is gained.

5.2.3. Lateness

We find that on the average there is no violation of due date requirements in the Light Traffic scenario under each of the policies because the mean lateness, as shown in Table 3, is negative. But we should keep in mind that, as shown in Fig. 8, the waiting time of some requests can exceed 40 s and thus the lateness can be positive.

5.2.4. Throughput

Since the incoming traffic and service rates are the same under all policies, they produce almost the same level of throughput in the light traffic scenario because the system is stable and able to handle the incoming traffic. The throughput of the Best Effort and Basic DiffServ policies are a little higher than that of the other policies because no requests are dropped, resulting in a higher arrival (and hence, throughput) rate.

Table 3
Experimental results of the light traffic case

Priority class		BE	DS	WSPT	ATC	EDD
Waiting time of requests						
All	Mean	1.530703	1.493246	0.092163	0.090235	0.124289
	STD	5.878742	5.493077	0.04996	0.046535	0.062083
High	Mean	1.529657	0.354413	0.068523	0.069562	0.126305
	STD	5.887196	1.552743	0.03778	0.036143	0.063747
Medium	Mean	1.530479	1.825389	0.078798	0.079292	0.121852
	STD	5.875447	6.939911	0.044733	0.042569	0.056926
Low	Mean	1.531522	1.827767	0.117916	0.112029	0.125791
	STD	5.878587	6.948821	0.065669	0.059874	0.070296
Requests dropped						
All	Mean	0	0	1.352074	1.441486	1.659334
	STD	0	0	4.215406	4.232963	4.426103
High	Mean	0	0	0.217108	0.232718	0.312424
	STD	0	0	0.777847	0.781258	0.828979
Medium	Mean	0	0	0.50021	0.53609	0.676685
	STD	0	0	1.694942	1.698867	1.787868
Low	Mean	0	0	0.635015	0.672957	0.67051
	STD	0	0	1.746721	1.756716	1.810628
Lateness of requests						
All	Mean	−0.47009	−0.50748	−1.91247	−1.91452	−1.87361
	STD	5.881148	5.495734	0.047722	0.04479	0.068402
High	Mean	−0.47449	−1.65009	−1.93752	−1.9364	−1.87521
	STD	5.889278	1.557602	0.046556	0.045589	0.07022
Medium	Mean	−0.46995	−0.17499	−1.92551	−1.92505	−1.87535
	STD	5.877842	6.941919	0.044774	0.042893	0.065856
Low	Mean	−0.46819	−0.17192	−1.88658	−1.89277	−1.87127
	STD	5.881193	6.951396	0.063165	0.058128	0.076792
Throughput of requests						
All	Mean	8995437	9022972	8599323	8592518	8624697
	STD	1564300	1543513	1198582	1187294	1203524
High	Mean	1713773	1719019	1651044	1644867	1636118
	STD	360182.2	333671.4	315147.4	314850.4	317698.7
Medium	Mean	3598025	3609039	3460339	3463858	3459718
	STD	883359.1	913816	747808.3	744651.2	746697.3
Low	Mean	3683673	3694949	3487975	3483828	3528896
	STD	1390708	1345147	1327450	1326310	1347935

5.2.5. *Requests waiting in queue*

There are about 3.6 requests waiting in the queue under the WSPT and ATC policies, 5.3 requests under EDD, 61.5 requests under basic DiffServ and 62.2 requests under Best Effort. The smaller number of waiting requests in the queue contributes to the shorter waiting time.

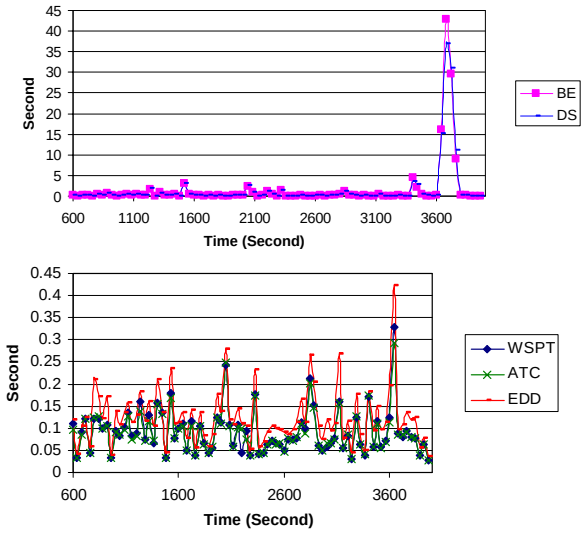

Fig. 8. Light traffic case: overall waiting time.

6. Conclusions and future work

In this paper, we demonstrate how to model a sequential web server as a single machine and apply WSPT, ATC, and EDD queuing disciplines to differentiate the services, and thus provide QoS. We compare these policies with the Best Effort and basic DiffServ policies.

We propose that most web servers can be modeled as a Best Effort Model using the FIFO queuing discipline. We then verify that the Best Effort Model cannot provide quality of service, especially in regard to waiting time. Furthermore, our results show that with the Basic DiffServ Model, the performance of high priority requests cannot be guaranteed because the requests cannot meet the Lateness QoS requirement. Performance using other classes of requests is no better than the Best Effort policy can provide. As we can see from our experimental results, the Basic DiffServ policy cannot provide QoS for a web server.

We introduce an Admission Control Scheme, which contributes to a tremendous improvement in performance. Thanks to the Admission Control Scheme, the overall waiting time of WSPT, ATC, and EDD Models are much shorter than the overall waiting time of Best Effort and Basic DiffServ Models. The waiting time of Best Effort policy is about 900 times longer than that of WSPT in the overwhelming traffic case. In the light traffic case, the waiting time of WSPT is also about 15 times shorter than that of Best Effort. It reveals that the Admission Control Scheme is effective to maintain timeliness for an overwhelmed web server.

We have shown that WSPT and ATC dispatching rules can be used to provide differentiated services. From the simulation results, the performance of ATC is quite similar to that of WSPT when the scaling parameter k is set to 100. We also create other scenarios of ATC with different scaling parameters and find that there is no significant difference if the scaling parameter k is larger than 10 in our simulation.

We can also safely conclude that our QoS models not only provide good performance when the server is overloaded but also work well under a light traffic condition. In a light traffic scenario, the QoS models provide better waiting time for high priority requests at the cost of a very small amount of dropped lower priority requests.

However, as Pinedo [28] stated, real-world scheduling problems are different from the mathematical models in academia. For example, WSPT is a static rule which is not time dependent. It assumes that there are n jobs to be scheduled and the problem is solved after the n jobs are scheduled. For a web server, requests are submitted by clients continuously. WSPT may not be the optimal scheduling rule to gain the minimum total weighted completion time. Another important aspect is that stochastic models usually use special distributions which may not closely represent the behaviors of the real system. Here, we use a requested document size divided by service rate to decide the processing time of a request. Request size follows the Pareto distribution. For a web server, the processing time of a request may also be influenced by the load and configuration of the web server.

In spite of these small differences, scheduling rules in manufacturing can provide valuable insights into scheduling problems in an information infrastructure. From the results of our research, we surmise that some manufacturing scheduling rules may be used to develop a framework for QoS enabled web servers. To implement our web server QoS models, other information about web requests, such as hostname, port, etc. along with the requested links, are also required. We suggest that further work might be done to investigate network costs and delays on implementing models based on these frameworks.

Acknowledgements

This work is sponsored by the Air Force Research Laboratory—Rome (AFRL-Rome) under grant number F30602-01-1-0510, and the Department of Defense (DoD) and the Air Force Office of Scientific Research (AFOSR) under grant number F49620-01-1-0317. The US government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of, AFRL-Rome, DoD, AFOSR, or the US Government.

References

[1] Lawrence TF. The quality of service model and high assurance. In Proceedings of the IEEE High Assurance Systems Engineering Workshop, Washington, DC, 1997.
[2] Ye N. QoS-Centric stateful resource management in information systems. Information Systems Frontiers 2002;4(2):149–60.
[3] Bhatti N, Friedrich R. Web server support for Tiered services. IEEE Network 2000;13(5):64–71.
[4] Strnadl C. At your service: QoS for the Internet. IEEE Multimedia 2002;9(1):93–5.
[5] Bhatti N, Bouch A, Kuchinsky A. Integrating user-perceived quality into web server design. Computer Networks 2000;33(1–6):1–16.
[6] Lzzo P. Gigbit networks: standards and schemes for next-generation networking. New York: Wiley; 2000. p. 177–233 [chapter 6].
[7] Braden R, Clark D, Shenkar S. Integrated services in the internet architecture: an overview, RFC 1633, IETF, 1994.

[8] Braden R, Zhang L, Berson S, Herzog S, Jamin, S. Resource reservation protocol (RSVP) version 1, functional specification, RFC 2205, IETF, 1997.
[9] Wroklawski J. The use of RSVP with IETF integrated services, RFC 2210, 1997.
[10] Wroklawski J. Specification of the controlled-load network element service, RFC 2211, 1997.
[11] Shenker S, Partridge C, Guerin R. Specification of guaranteed quality of service, RFC 2212, 1997.
[12] Almeida J, Dabu M, Manikutty A, Cao P. Providing differentiated levels of services in web content hosting. First Workshop on Internet Server Performance, Madison, WI, 1998.
[13] Cherkasova L, Phaal P. Session-based admission control: a mechanism for peak load management of commercial web sites. IEEE Transactions on Computers 2002;13(6):669–85.
[14] Li K, Jamin S. A measurement-based admission-controlled web server. Proceedings of the IEEE INFCOM 2000, 2000.
[15] Zhang H. Service disciplines for guaranteed performance service in packet-switching networks. Proceedings of the IEEE, 1995;83(10):1374–96.
[16] Chandra S, Ellis CS, Vahdat A. Differentiated multimedia web services using Quality Aware Transcoding, INFOCOM, 2000.
[17] Nichols K, Blake S, Baker F, Black D. Definition of the differentiated services field (DS Field) in the Ipv4 and Ipv6 headers, RFC 2474, 1998.
[18] Nichols K, Jacobson V, Zhang L. A two-bit differentiated services architecture for the Internet, 1999. ftp://ftp.ee.lbl.gov/papers/dsarch.pdf.
[19] Lu C, Abdelzaher TF, Stankovic JA, Son SH. A feedback control approach for guaranteeing relative delays in web servers. IEEE Real-Time Technology and Application Symposium (RTAS' 2001), Taipei, Taiwan, 2001.
[20] Conti M, Gregori E, Panzieri F. Load distribution among replicated web servers: a Qos-based approach. Second Workshop on Internet Server Performance in conjunction with ACM SIGMETRICS 99/FCRC, Atlanta, GA, 1999.
[21] Engelschall RS. Load balancing your web site, Practical Approaches for Distributing HTTP traffic, 1998. www.webtechniques.com/archives/1998/engelschall.
[22] Shan Z, Lin C, Marinescu DC, Yang Y. Modeling and performance analysis of QoS-aware load balancing of web-server clusters. Computer Networks 2002;40(2):235–56.
[23] Rhee, Yoon-Jung, Hyun, Eun-Sil, Kim, Tai-Yun. Connection management for QoS service on the Web. Journal of Network and Computer Applications 2002;25(1):57–68.
[24] Abdelzaher TF, Shin KG, Bhatti N. Performance guarantees for web server end-systems: a control-theoretical approach. IEEE Transactions on Parallel and Distributed Systems 2002;13(1):80–96.
[25] Striegel A, Manimaran G. Packet scheduling with delay and loss differentiation. Computer Communications 2002;25(1):21–31.
[26] Ferrari T. End-to-end performance analysis with traffic aggregation. Computer Networks 2000;34(6):905–14.
[27] Chen H, Mohapatra P. Overload control in QoS-aware web servers. Computer Networks 2003;42(1):119–33.
[28] Pinedo M. Scheduling theory, algorithms, and systems. Englewood Cliffs, NJ: Prentice-Hall; 1995.
[29] Guerin R, Peris V. Quality-of-service in packet networks: basic mechanisms and directions. Computer Networks 1999;31(3):169–89.
[30] Silberschatz A, Galvin PB, Gagne G. Operating systems concepts, 6th ed. New York: Wiley; 2003. p. 158–61 [chapter 6].
[31] David von S. CRC Standard curves and surfaces. Boca Raton, FL: CRC Press; 1993. p. 252–3.
[32] Arlitt MF, Williamson CL. Web server workload characterization: the search for invariants. In ACMSIGMETRICS Performance Evaluation Review 1996:126–37.