Performance Evaluation of Service Differentiating Internet Servers

Xiangping Chen and
Prasant Mohapatra, *Senior Member*, *IEEE*

Abstract—Differentiated service approach has been proposed as a potential solution to provide Quality of Services (QoS) in the next generation Internet. The ultimate goal of end-to-end service differentiation can be achieved by complementing the network-level QoS with the service differentiation at the Internet servers. In this paper, we have presented a detailed study of the performance of service differentiating web servers (SDIS). Various aspects, such as admission control, scheduling, and task assignment schemes for SDIS, have been evaluated through real workload traces. The impact of these aspects has been quantified in the simulation-based study. Under high system utilization, a service differentiating server provides significantly better services to high priority tasks compared to a traditional Internet server. A combination of selective early discard and priority-based task scheduling and assignment is required to provide efficient service differentiation at the servers. The results of these studies could be used as a foundation for further studies on service differentiating Internet servers.

Index Terms—Admission control, performance study, Quality of Service, scheduling, service differentiating Internet servers, task assignment.

1 Introduction

The Internet and the World Wide Web (WWW) provide a global publishing and information exchange infrastructure at low cost. The desire for using the Internet in business transactions is increasing at a fast rate. Thousands of companies have set up Web sites for marketing and interactions with customers [1]. However, many companies find the current generation of Internet servers inadequate for core business transactions due to the problems of reliability, scalability, security, and performance inconsistency.

The existing best-effort service with the *First Come First Serve* (FCFS) scheduling model of the Internet servers leads to misallocation of scarce and expensive network and CPU resources during heavy load periods, causing unpredictable response delay, which is not acceptable to time-sensitive transactions. The problem of unpredictability of response time becomes extremely serious during information retrieval from the Web since more and more applications with time constraints are using the Web as their distributed interfaces. Upgrading hardware to faster CPU, more efficient OS, and broader I/O bandwidth are always solutions for improving the response time and throughput of Internet servers. However, the "bursty" nature and the rapid growth of the Internet traffic volume make it expensive and also inefficient to scale up network bandwidth and server capacity with the increasing peak demand. Previous studies [2], [3] have shown that the peak workload of a web server may exceed its average load by orders of magnitudes. Thus, the server utilization would be very low if we design the system to satisfy requirements of the workload during peak periods.

Another approach of assuring service quality is to preallocate resources for value desirable tasks. Irrespective of the intensity of workload, these cherished tasks enjoy dedicated system resources. The analogous policy for the Internet is the end-to-end resource reservation scheme through the Resource ReSerVation Protocol (RSVP) [4]. One major drawback of RSVP is poor scalability since the number of RSVP control messages processed by each router is proportional to the number of flows passing through the router. Similarly, the resource reservation approach is not appropriate for Internet servers because of poor scalability, low system utilization, and/or long setup delay.

Differentiated service, known as *DiffServ* [5] in the IETF community, has been proposed as an efficient and scalable solution to provide better service for the next generation Internet communication [6]. *DiffServ* creates an "expressway" for high priority traffic by selective reallocation of network resources during peak workload periods. Similarly, prioritized services can be exploited by a Service Differentiating Internet Server (SDIS) to provide better quality services to high priority tasks. Especially, the response delay of mission critical requests can be bounded by allocating CPU and I/O resources with respect to their priorities. Thus, there is a need to design and evaluate Internet servers that can provide fast response to high priority tasks, minimizing the performance penalty of low priority tasks caused by service differentiation without degrading the overall system throughput. To summarize, the primary motivations for SDIS can be itemized as follows:

- Several evolving applications use continuous media (CM) objects, such as audio or video stream data. These data types have time constraints that need to be met for their meaningful delivery. Thus, they may need service priority over the non-real-time tasks.
- The exploding e-commerce market supports several types of transactions and may need to classify services based on revenue generation or request types. Prioritized transactions may thus be necessary to meet the objectives of e-commerce transactions.
- The possible adoption of differentiated services in the next generation Internet will make the best-effort servers unacceptable and may defy the primary purpose of it. Thus, to provide end-to-end QoS assurance, the Internet servers also need to provide differentiated services.

In this paper, we present a model of SDIS and evaluate its performance through simulation. The workload for the study is derived from traces obtained from a live web server log. Priorities are randomly introduced in the trace. During congestion, SDIS provides better services to high-priority requests compared to the best-effort service models. We have also analyzed several features of SDIS, such as admission control, task scheduling, and assignments. The impact of some of these issues on SDIS performance is quantified in our study. The results demonstrate the feasibility and performance advantages of SDIS. A combination of selective early discard and efficient scheduling and task assignment approaches are necessary to derive fair service differentiation from Internet servers.

The remainder of this paper is organized as follows: Section 2 describes policies for providing differentiated services from an Internet Server. The server model, admission control, and scheduling issues are discussed in Section 3. The simulation environment and results are discussed in Sections 4 and 5, respectively. Section 6 describes the related works, followed by the concluding remarks in Section 7.

2 Service Differentiation

A traditional Internet server processes requests in FCFS manner. During a high load period, each task has to wait in a queue for a long time before getting serviced. Overhead from tasks competing

- *X. Chen is with the Network Storage Group, EMC Corp., Hopkinton, MA . E-mail: chen_xiangping@emc.com.*
- *P. Mohapatra is with the Department of Computer Science, University of California, Davis, CA 95616. E-mail: prasant@cs.ucdavis.edu.*

Manuscript received 28 Jan. 2000; revised 5 June 2001; accepted 26 Mar. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 111338.

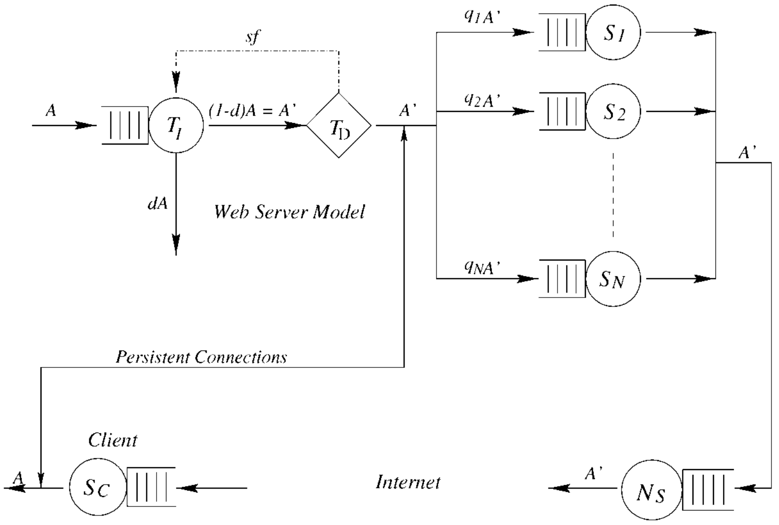

Fig. 1. Queuing network model for a Web server.

for limited resources, such as open connections and network bandwidth, is increased. "Retry" from impatient clients worsens the load situation and causes the "snowball" effect. More elaborate resource allocation schemes, rather than the best-effort service model, need to be adopted to provide predictable services during high load periods. In this section, policies of differentiating services that could be deployed by web servers are discussed.

In the *DiffServ* model for the Internet, different priorities are assigned to packets. High priority packets enjoy timely processing and service assurances. Similarly, at the server, an incoming request can be assigned to a priority group on the basis of the *client*, *network*, *content*, or *owner* of the requested object, as elaborated below.

Client-based service prioritization scheme allows clients to purchase premium services at a certain cost. A client can be authenticated by a combination of cookies, client host ID, session ID provided by the server, or by other profile information of the clients maintained at the server. More detailed user authentication gives the users more flexibility for QoS selection, while increasing the load of the maintenance of status information at the server.

Network-based service differentiation is determined by the underlying network level protocols. Clients specify priority of packets, which is carried by the request. The server extracts the packets priorities and sends reply at the matching service levels.

Content-based service differentiation is determined by the "value" or "importance" of the web object being requested. In an online stock exchange center, priority can be granted to purchase/sell requests over queries. Similarly, in an online shopping web site, the transactions of someone with items in the shopping cart or someone entering payment information should have higher priority than the general browsing transactions.

Owner-based prioritization of service provides selective QoS in a web hosting environment. A web host can grant distinct priorities to requests made to web pages belonging to owner organizations based on the prices they paid.

One or more of the above-mentioned policies can be adopted by the server based on the application environment. Analysis and comparison of these policies is not within the scope of this paper. The levels of priorities that can be set up to provide both flexibility and efficiency of the system are an important metric, but are considered beyond the scope of this work. Similarly, quantification of QoS and pricing issues for different levels of services also need further investigation and analysis and are not emphasized in this study.

3 A Generalized Internet Server

In this section, we present a model of a generalized Internet server and analyze the feasibility of service differentiation. Although other models of Internet servers exist, the goal of the proposed model is to study the issues involved in service differentiation, not the types of Internet servers.

3.1 Service Differentiating Internet Server Model

Fig. 1 shows a queuing model of a generalized Internet server. The system consists of four major logical components, a task initiator T_I, a task dispatcher T_D, a task server pool $S_i(i = 1 \dots N)$, and the communication channel N_S. A represents the request arrival process from the clients. dA represents the dropping probability of incoming requests. A' denotes the requests that get service. The task initiator T_I maintains the open connections of the system. Incoming requests are queued awaiting acceptance by the T_I. Each accepted request is assigned a task and each task is granted an appropriate priority level based on the system settings. The task dispatcher T_D assigns tasks to a task server and each task server schedules and processes tasks according to their priorities. Responses are sent back to clients through the communication channel N_S. To simplify the model, we assume that the server connects to the client through a high speed network. We have ignored blocking and flow control from client side network connection as they are beyond the scope of the proposed study.

The task initiator T_I picks up requests from the incoming request queue. The queue length is limited by the operating system constraints and allowable open connections in the server. Incoming requests are tail-dropped if the queue is full. With the cooperation of intelligent network interface, network layer priority setting of a request can be passed on to the application layer. T_I can be used to collect priority information from low level network protocols and convey it to the task dispatcher T_D. Otherwise, T_D determines the task priority based on the implementation of the service differentiation algorithm. After determining the priority of each task, T_D selects a task server to process the task.

Each task server, $S_i(i = 1...N)$, represents a processing unit that takes care of processing a task. For a static request, the task server translates the URL into the corresponding file path, finds the file, copies the file contents into memory, and sends the reply back to the client through the Internet. For a dynamic request, the server invokes a gateway interface. A web page is generated "on the fly" based on the result returned from the gateway interface and sent back to the client. A task server is an abstract concept in the sense that it can be a child process in a multiprocess server, a thread in a multithreaded server, a processor in a multiprocessor server, or a host in a server cluster. Based on the system environment, each task server shares or has an independent waiting queue, memory, cache space, and other resources. Task servers can be symmetric, i.e., with the same hardware configuration, or can be asymmetric with different hardware configurations.

The capacity of the communication channel N_S is determined by the bandwidth of the server network access point. In the current Internet infrastructure, a server generally has a T1 (192 KB/s) or a T3 (5.4 MB/s) connection. An Intranet web server generally connects to clients through high-speed local area networks such as 10 Mb or 100 Mb Ethernet. Data throughput is determined by the channel bandwidth and network utilization.

3.2 Admission Control

Admission control can be implemented using the following two steps: The first step of admission control is provided by the initiator T_I. Incoming requests are rejected when the arrival rates exceeds the processing capacity of the system. In Fig. 1, *sf* provides feedback of system capacity to the initiator. To avoid bulk rejection, early overload detection can be adopted and two levels of thresholds can be used similar to the Random Early Detection (RED) scheme [8]. Low priority packets are dropped with increasing probability after the system load exceeds the first threshold. All the packets are dropped when the system load exceeds the second level of threshold. Average queue length at the initiator can be used as the threshold indicator. The value of the thresholds can be statically configured or dynamically adjusted, depending on the application environment or on the variety of incoming requests.

The second step of admission control can be provided by the task dispatcher, T_D. The task dispatcher monitors the task server load and makes decisions for dropping requests to assure that the system works within an acceptable load level. Contrary to the first step of uninformed dropping, the task dispatcher sends back a brief explanation to clients stating why it cannot fulfill the request. The informed dropping decreases the "retry" attempts from clients. When one task server is overloaded, tasks are redirected to the other task servers. If all the task servers are overloaded, low priority tasks experience "informed dropping" to relieve the overload situation. In case of communication channel overload, low priority large size tasks are first discarded and the continuous media data requests are discarded when bounded delay QoS cannot be assured. Available bandwidth is used as a metric of channel load.

3.3 Scheduling

In a task server, priority-based scheduling is adapted to provide faster processing of some tasks than others. In this study, we have used strict priority scheduling policy, i.e., low priority tasks do not get service if higher priority tasks are waiting. Tasks in the same priority group are serviced in FCFS order. We consider a nonpreemptive scheduling approach in our study to avoid the complexity of context switching and state maintenance caused by preemption.

Scheduling in the communication channel also follows priority based queuing. Unlike the task server, the data that is being sent out is restricted by the page size of cache memory and network protocols. The scheduling in the communication channel is similar to prioritized process sharing.

3.4 Task Assignment

A distributed server system introduces flexibility of selecting a server for a specific task. Selection can be based on task priority, task type, spatial and temporal locality of web objects, and hardware configuration.

One or more high-performance web servers can be reserved for high priority tasks to make sure that the high priority tasks always encounter short queue length. Another approach of task assignment could be the assignment of task type to different servers. The same type of tasks often consume about the same amount of processing time, while tasks with different types may consume processing time with very high variance. Task assignment schemes which lead toward lower variance of processing time help to improve response time and throughput. CM data require streaming delivery for a sustained time period, for which process sharing performs better than FCFS. In addition, task assignment schemes considering temporal locality have the potential of improving caching efficiency.

Task assignment schemes can be combined with scheduling. If each task server has the facility of prioritized processing, then tasks can be sent to the corresponding server right away to alleviate the queuing load at the task dispatcher. Otherwise, the task dispatcher blocks low priority tasks unless there are no high priority tasks waiting to be served at a server.

4 Simulation

The request arrival patterns at Internet servers are known to exhibit self-similarity and long-range dependencies. Thus, it is difficult to build a good analytical model that can provide us an in-depth view of the performance estimation. In this section, we have used a simulation model and traces from a real web server as workload to examine the performance of SDIS.

4.1 Simulation Model

We have implemented an event-driven simulator for the experimental study of the server model proposed in Section 3. The simulator model is built using the CSIM [9] simulation package according to the model discussed in the previous section. There is one task dispatcher which assigns tasks to four task servers. The service capacity of the task dispatcher is set to be 4,000 requests per second and the service capacity of each task server is 1,000 requests per second. The queue size for each server and the task dispatcher is 1,024. The disk I/O throughput of each server is set to be 10MB per second. Seek overhead of each disk request is set to be 1 ms per second. Cache size of each server is 32 MB and files with size larger than 32 KB are not cached. We have assumed two levels of priority: high priority and low priority. The average system service time of the simulation model is 18ms. The parameters for task processing behavior are derived from [13], [14] and by monitoring the network traffic to and from our departmental web server. Caching hit ratio data is selected based on the study in [15], [16]. Cache size of each server is set to be 32 MB for static objects with size less than 32 KB. Dynamic and big, i.e., sizes equal to or larger than 32 KB, web objects are treated as uncacheable. For a dynamic object request, the service time is dominated by the CPU computation time and, for a large size file request, it is dominated by I/O processing time. The simulation parameters are shown in Table 1.

The queue size for each server process in the simulation is set to be 1,024. This size is quite big compared to the server processing capacity. Thus, the waiting time of low priority tasks exceeds several thousand times of service time before the queue gets full. To save the space, we have not discussed the drop rate here. Later, while examining the EDAC scheme, the queue size is shortened to

TABLE 1
Simulation Parameters

Parameter	value
Number of Task Servers	4
System Capacity for Static Objects	1000 req./sec
Task dispatcher Capacity	4000 req./sec
Disk Bandwidth	10 MBps
Disk Seeking Overhead	1 ms
Network Bandwidth	100 Mbps
Outbound Network BW	80 Mbps
Cache Size	32M bytes/server
Caching Threshold	32 KB
Dynamic Objects Processing Overhead	10 ms
Priority Level	2

100 for each server to improve the response time at the cost of higher drop rate.

4.2 Workload

Previous works [3], [10], [11] have suggested apparent self-similarity and long-term dependency of the WWW traffic pattern. However, accurate synthesis of self-similar traffic remains an open problem [7], [12]. In this study, we propose generating workload from real trace files. We monitor the logs from a departmental web server in the Computer Science department at Michigan State University.[1] The trace files contain 866,587 requests in a one-week period. The access logs provide the request timestamp, client ID, object URL, service status, and reply size of each request. The referrer logs complement burst and session information. Request type distribution from a week access log is listed in Table 2.

The request generators take data extracted from the traces regenerate and send requests to the server. The number of independent request generators is changed to generate different workload intensity. "Burst" of the request flows is well preserved by using multiple independent request regenerators. Timestamps have one-second resolution and requests with the same timestamp are assumed to be distributed exponentially in a one second time period. Data from each day time period, 9 AM to 9 PM, are used as input of the simulator.

4.3 Performance Metrics

The effectiveness of a scheduling scheme is measured in terms of mean response time as well as the 95th percentile response time. Mean response time is defined as the time between the acceptance of a request and the completion of sending back the reply, which is the sum of the waiting time and service time. The 95th percentile response time shows the response time that the majority of tasks experience, which represents the statistical predictability of system responsiveness. Other aspects, including admission control and server side caching algorithms, have been studied as well.

5 Results

In this section, we present and analyze the results obtained from our simulator using the real workload traces. We have examined the impact of priority-based scheduling, task assignment policies, and admission control schemes.

5.1 Effectiveness of Priority-Based Scheduling

In the Internet environment, both access interval and service time distribution are significantly different from the widely used synthetically modeled workload. Therefore, we have used real workload traces for performance evaluation. The high variance of the interarrival time and service rate degrades system performance [17] and forces the web server to operate in a low utilization state. Fig. 2 shows the mean response time of tasks and Fig. 3 shows the 95th percentile response time of tasks. Curve (c) in each figure indicates the performance trends of nonprioritized processing of tasks. As we can see from Fig. 2 and Fig. 3, the response time increases sharply with respect to the system utilization. Curves (a) and (b) in each figure are the response time curves of low priority requests and high priority requests, respectively. High priority tags are assigned to half of the requests randomly and the other half is marked as low priority. Thus, the ratio of high priority to low priority tasks is 1 to 1 and both types of tasks are randomly distributed in the whole arrival sequence. Tasks are assigned to each task server using *Round_Robin* scheduling, irrespective of their priorities. Tasks queued at each server are served based on their priority. Specific scheduling and task assignment approaches are discussed later in this section. Performance degradation of the high priority task group happens at a much higher utilization compared to the nonpriority-based model. On the other hand, the performance curve of low priority task is fairly close to the performance curve without priority differentiation.

We only consider the stable states of the system in the study, i.e., the task response time and slow down before the sharp performance degradation, or the "knee" of the performance curve. From the preceding results, we observe that the delay is bounded in an acceptable range for high priority tasks in a system with "performance knee" at about 50 percent of the system capacity for static objects with 50 percent or less high priority tasks. A steep rise in response time of low priority requests occurs at about 50 percent of the system capacity, which is about the same as the response degradation point without priority differentiation. High priority requests incur average low delay, i.e., less than two seconds, even when the system approaches full utilization. Note that the experiment could not achieve full system capacity due to the overhead incurred by dynamic objects. The 95th percentile response delay of high priority requests is also within a reasonable range, which is less than 10 seconds in the experiment.

The irregularities in the curve at low server utilization are caused by load imbalance among servers while serving continuous media objects. *Round_Robin* scheduling treats each task indifferently, which fails to balance load among servers since continuous media tasks consume much more disk and network bandwidth than other tasks. In the following subsection, we have varied the high priority task proportions from 0.5 to 0.9, corresponding to the portions of low priority task from 0.5 to 0.1. In reality, the proportion of high priority tasks would be lower than that of the low priority tasks.

5.2 High Priority Task Performance

Next, we examine the relationship between the high priority task ratio and the "knee of the curves" in the system. The simulation setup is kept the same as in the last experiment, the only difference is that the high priority task ratio varies from 0.5 to 0.9.

Fig. 4 and Fig. 5 show response delay curves of high priority tasks with the high priority ratio varying from 0.5 to 0.9. As the high priority task ratio increases, the response time curve gets closer to the nonprioritized system response time curve and the benefit margin obtained from differentiating services diminishes. Fig. 4 shows that the "knee"s of the mean response time of high priority tasks move closer to that of the nonprioritized system with the increase in high priority ratio. However, by controlling the proportion of high priority requests, we can obtain a significant performance benefit from the SDIS. Fig. 5 displays the 95th percentile response time of high priority tasks with high priority ratio ranging from 0.5 to 0.9. The monitored time frame is set to be

[1] This work was performed while the authors were at Michigan State University.

TABLE 2
Trace Data Distribution

Item	HTML	Image	Audio	Video	Dynamic	Other	Total
Req. Ratio (%)	19.2	68.8	0.2	0.1	3.9	7.8	100
Traffic Ratio (%)	15.0	49.2	1.6	6.7	5.4	20.2	100
Mean Transfer Size(KB)	5.76	4.98	579.9	2503.9	3.84	19.0	7.39
Transfer Size CoV	1.90	2.46	1.76	1.56	1.33	7.90	14.41

60 seconds, which is the default timeout period used in this study. It can be observed from the figure that the high priority tasks rarely get timed out and retransmitted. Service availability of high priority tasks is much higher than that of low priority tasks.

5.3 Low Priority Task Performance

Fig. 6 and Fig. 7 show the mean response time and 95th percentile response time curves of low priority tasks with the high priority ratio ranging from 0.5 to 0.9, i.e., low priority task ratio from 0.5 to 0.1, versus traffic intensity. It can be observed that the response time of low priority tasks becomes worse with the increase in high priority task proportion, as expected. On the other hand, we find that the spectrum of the occurrence of the "performance knee" in low priority response time curves is relatively narrow, which reflects the minimal influence on low priority tasks with service differentiation.

5.4 Task Assignment Schemes

In a distributed server environment, an appropriate task assignment scheme decreases the waiting time variance and thereby improves the system performance. We studied four types of task assignments, Round_Robin (**rr**), Shortest_Queue_First (**sqf**), Prioritized_Shortest_Queue_First (**psqf**), and Reserved_PSQF (**rpsq**), in the experiment. Tasks are assigned to task servers in rotational order when we use **rr** task assignment scheme. **sqf** task assignment scheme is based on load balancing techniques, which assigns tasks to the server with lowest number of active processes. The effectiveness of the **sqf** assignment scheme depends on the accuracy of system load information the dispatcher uses. In the experiment, we assume that the scheduler always gets the updated process number in each task server and the overhead of system load monitoring is 10 percent of service time. To adapt to the differentiated service environment, a **psqf** task assignment scheme is introduced in which a new task is assigned to the server with the least number of waiting tasks of equal or higher priority than the incoming task. We also tried a resource reservation scheme **rpsq** in assigning a task, i.e., some resources are reserved for high priority tasks. A high priority task waiting time is expected to decrease by reserving one or two servers exclusively serving high priority tasks.

The response times of high priority tasks are shown in Fig. 8 and Fig. 9. Fig. 10 and Fig. 11 illustrate the corresponding response time variations of low priority tasks. Experimental parameters are kept the same in each of the task assignment schemes. The ratio between high priority and low priority tasks is one to one.

The **rr** task assignment scheme consumes the least system overhead, which assigns tasks in rotation order of task servers. There is no need to monitor task server load dynamically to make the decision. However, one big task can cause temporary server overload and degrade the system responsibility and throughput drastically. As we can see from Fig. 8 and Fig. 10, using the **rr** scheme, both high priority and low priority tasks experience poorer response time performance than other task assignment schemes. On the other hand, introducing load balancing techniques in the system improves the system performance on the whole. The experimental results show improved response time performance in both high priority and low priority tasks using the **sqf** scheme compared to the **rr** task assignment scheme.

The performance of high priority tasks under the reservation-based **sqf** and **psqf** schemes does not differ much from nonreservation-based **sqf** and **psqf** schemes. On the contrary, reservation-based **psqf**, denoted as **rpsq**, degrades system response time of low priority tasks. The "curve knee" of low priority task response time moves closer to that of the **rr** scheme. Reservation of resources for high priority tasks is proven to be not as effective as load balancing-based task assignment schemes.

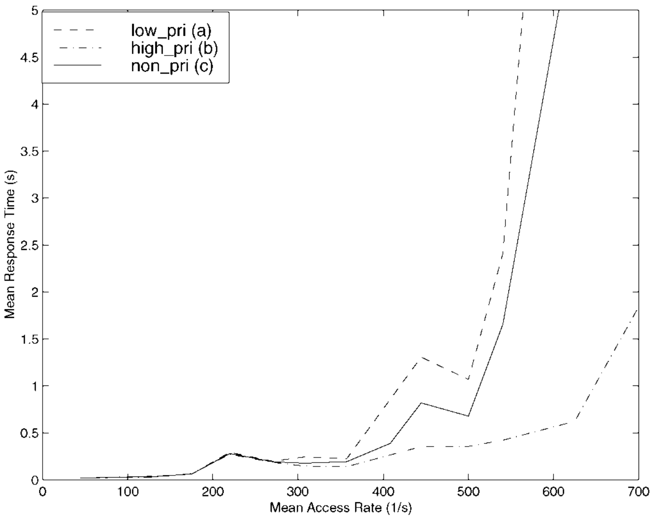

Fig. 2. Mean task response time.

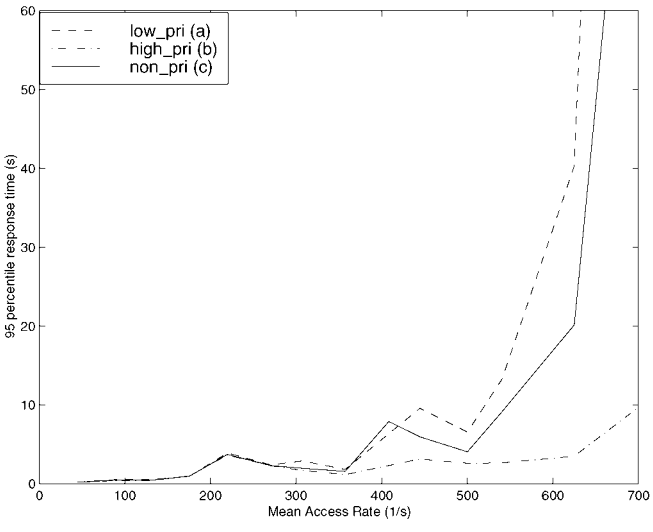

Fig. 3. Ninety-fifth percentile response time.

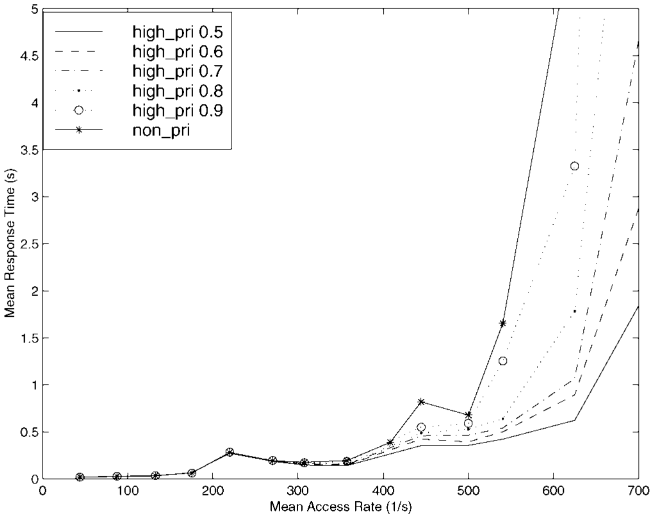

Fig. 4. Mean response time vs. priority ratio.

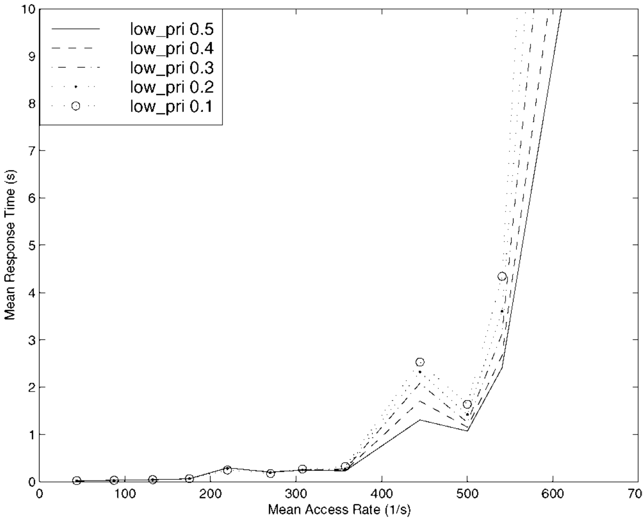

Fig. 6. Mean response time vs. priority ratio.

5.5 Admission Control

The high priority tasks experience the same probability of denial of service in an overloaded system if there is no appropriate admission control scheme. In this study, we explore the effectiveness of reserving buffer space for high priority tasks by an admission control scheme and the early discard admission control (EDAC) scheme discussed in Section 3. We find that EDAC is suited for light to medium workload. During a heavy load period, EDAC is not effective in assuring high priority tasks get system resources. The accepted low priority tasks starve for a long time for service and those waiting tasks occupy buffer space and cause denial of service to high priority tasks, even when the arrival rate of high priority tasks is below the system capacity (see Fig. 12). The early detection threshold of low priority tasks is set to be 0.5 of the buffer space in Fig. 12. We collected the rejection rate data of high and low priority tasks using the **sqf** and **psqf** task assignment schemes. The slope of the rejection rate of high priority tasks is almost the same as the rejection rate of low priority tasks, irrespective of the task assignment schemes. In the following experiment with timeout consideration, we only present rejection rate and abort rate data using **sqf** task assignment, since there are no noticeable differences in performance between **sqf** and **psqf** schemes.

The experimental results shown in Fig. 12 suggest that EDAC is effective in keeping the rejection rate of high priority tasks low. We further consider adding timeout to release system resources from starved tasks and any operations related to tasks that have expired timeout are aborted. The abort rate and rejection rate performance is collected and displayed in Fig. 13. In results shown in Fig. 13, a timeout of two seconds is added to release the system resources from stale low priority tasks. We repeat the experiment with increased high priority task traffic volume till peak arrival rate equals system capacity. The result shows that 99.5 percent of high priority tasks are served within two seconds. No high priority tasks experience denial of services against various task interarrival rate. The abort rate of high priority tasks varies from 0.05 percent to 0.5 percent, depending on the distribution of high priority task access rate. It is worth noticing that the abort rate of high priority requests decreases when the task arrival rate approaches system capacity, which is due to the increase of the rejection rate of low priority tasks. The abort rate of low priority tasks is about the same as high priority tasks under light to medium load and continue to increase with the workload. The results indicate that EDAC and

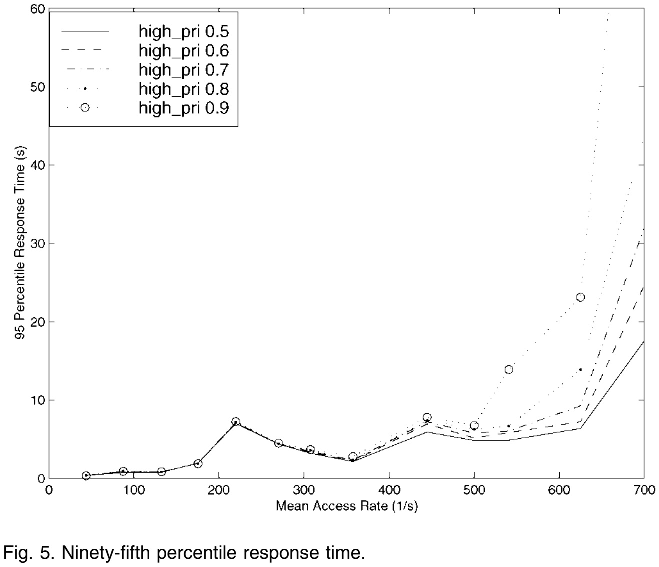

Fig. 5. Ninety-fifth percentile response time.

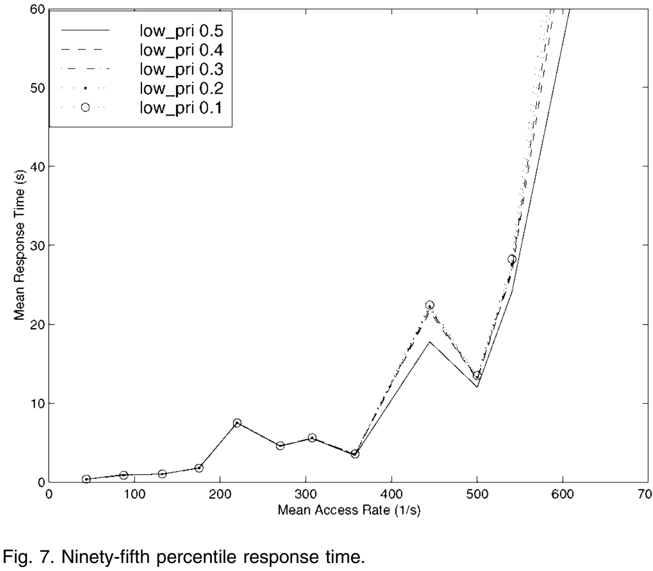

Fig. 7. Ninety-fifth percentile response time.

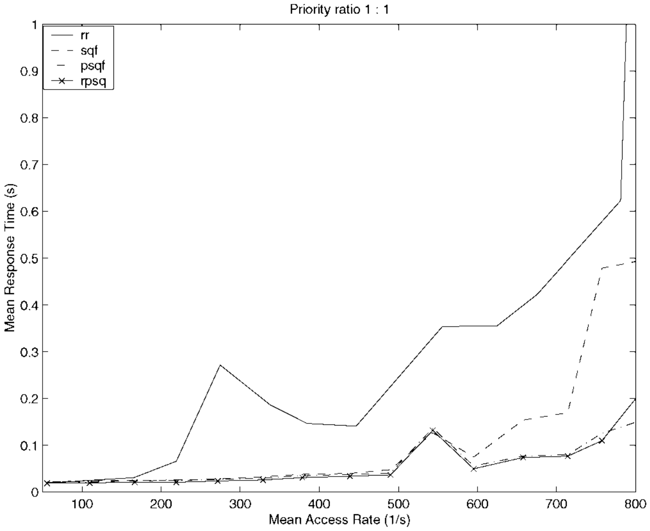

Fig. 8. High priority task mean response time.

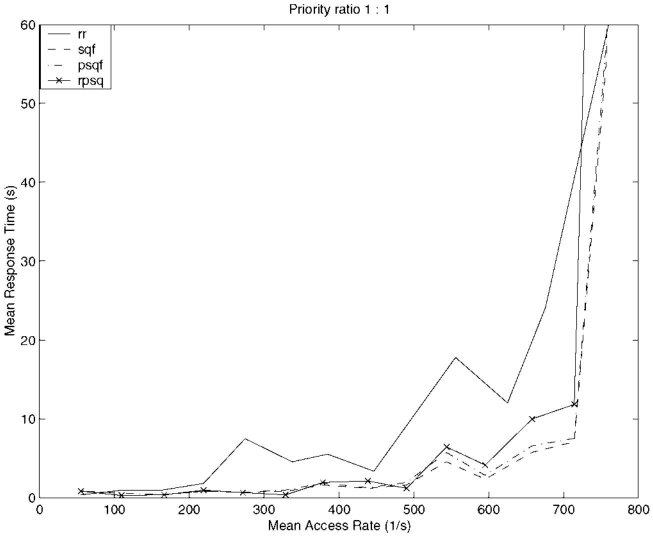

Fig. 11. Ninety-fifth percentile low priority task resp. time.

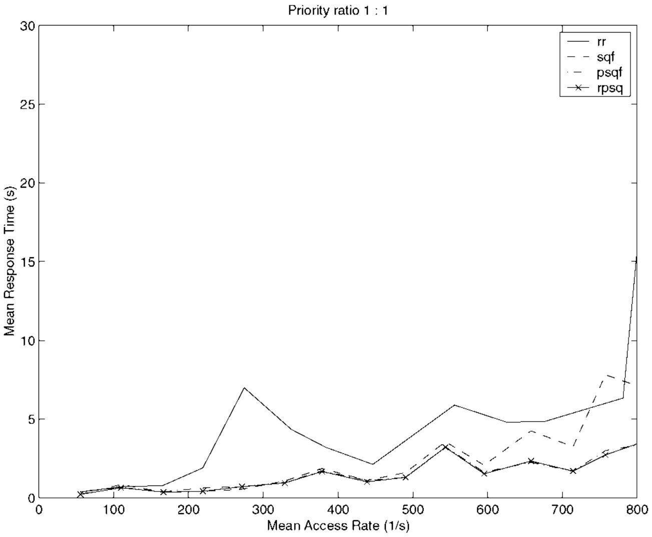

Fig. 9. Ninety-fifth percentile high priority task resp. time.

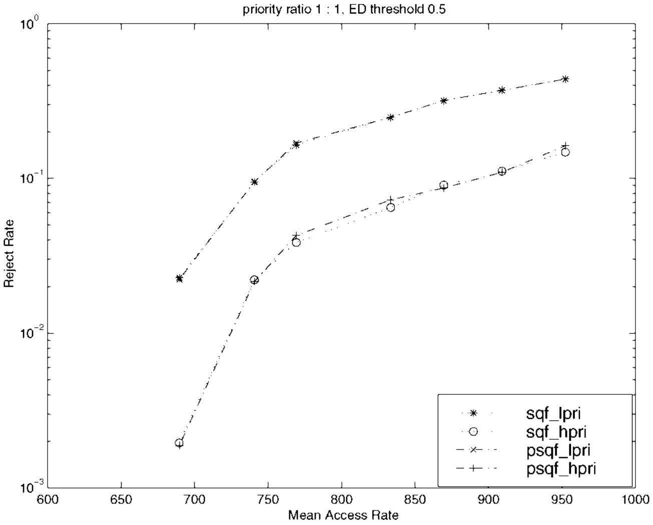

Fig. 12. Rejection ratio vs. task assignment schemes.

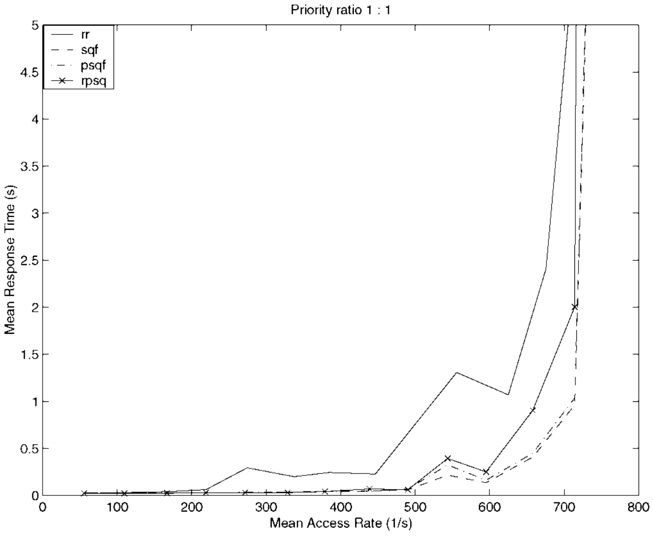

Fig. 10. Low priority task mean response time.

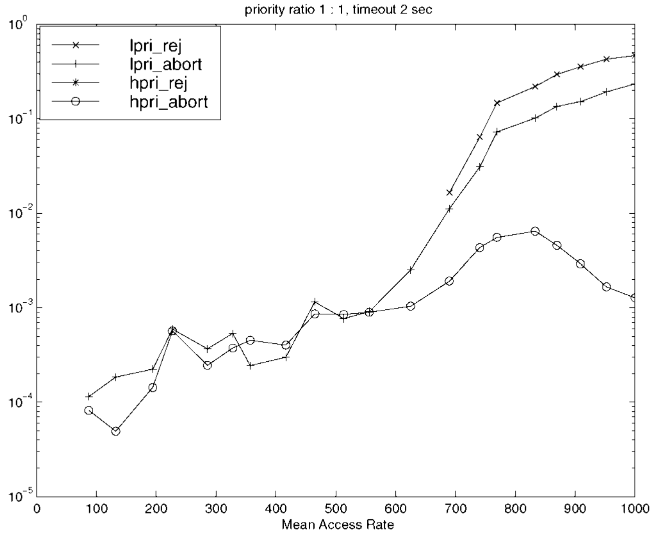

Fig. 13. Rejection and abort ratio of different priority groups.

timeout effectively reallocate system resources to high priority tasks during server overload periods.

6 Related Work

Although a significant amount of research has been done on the service differentiation at the network level, the work on service differentiation in Internet servers has been limited. Almeida et al. [18] have implemented a prototype of web server which can provide prioritized service in a web hosting environment and studied performance issues in both user space and kernel space. They compared performance of high priority tasks and low priority tasks using synthetic benchmarks. Their research results encouraged us to study prioritized service in a general web server environment. Our work is different from theirs in the sense that we use empirical distribution inputs representing the real workload of a web server. In addition, we have also analyzed several other issues of SDIS, including task assignment, admission control, impact of memory affinity, etc. Bhatti and Friedrich at HP Labs [19] have built a prototype of task classifier and scheduler on an Apache web server and studied response time and throughput issues of prioritized services on a web server. They used stress tests which might not be adequate in simulating high variance of web server workload environment. Our work complements the work of these authors. In addition, we have shown the impact of admission control and task assignment schemes and quantified their performance impacts on prioritized tasks. The pricing issues related to the QoS provision on the Internet were analyzed in [20]. They discussed several QoS classification options, which can be also provided by our server model. We extend their work in terms of service differentiation and classification. The authors of [21] studied task assignment policies in a distributed server system model. Each host processes tasks in FCFS order and the task resource demand is known in advance. Their results suggested that a size-based policy performs the best in an environment where the task processing time has high variance. However, they did not consider the QoS assurance issues.

7 Conclusions

The next generation Internet will demand differentiated services from Internet servers, which can be achieved through priority-based service. Service differentiating Internet servers are needed to provide high quality of service to high priority tasks, even under high system utilization. In this study, we show that, under near-saturation of web server utilization, differentiated services provide significantly better services to high priority tasks compared to a traditional web server. In addition, through a simulation-based study, we have shown the impact of admission control and task assignment schemes on the performance of prioritized tasks. The results of our study indicate that the combination of selective early discard, timeout, and priority queuing is necessary and may be sufficient to provide better response from the next generation Internet Servers. We can configure our server system so that the response time of high priority requests can be bounded, irrespective of the volume of low priority requests. This study will instigate further analysis and design of service differentiating Internet servers.

Acknowledgments

This research was supported in part by the US National Science Foundation through grants CCR-0296070 and ANI-0296034.

References

[1] E.J.W. West, "Using the Internet for Business—Web Oriented Routes to Market and Existing It Infrastructures," *Computer Networks and ISDN Systems,* vol. 29, pp. 1769-1776, July 1997.

[2] M.F. Arlitt and C.L. Williamson, "Web Server Workload Characterization: The Search for Invarients," *Proc. ACM SIGMETRICS Conf. Measurement & Modeling of Computer Systems,* pp. 126-137, May 1996.

[3] M.E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes," *IEEE/ACM Trans. Networking,* vol. 5, pp. 835-846, Dec. 1997.

[4] R. Braden et al., "Resource Reservation Protocol (RSVP)—Version 1," RFC 2205, Proposed Standard, Sept. 1997, http://www.isi.edu/div7/rsvp/pub.html.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Dec. 1998.

[6] X. Xiao and L.M. Ni, "Internet QoS: The Big Picture," *IEEE Network,* Mar./Apr. 1999.

[7] P. Barford and M.E. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. Performance '98/ACM SIGMETRICS '98,* pp. 151-160, July 1998.

[8] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networking,* vol. 1, pp. 397-413, Aug. 1993.

[9] H. Schwetman, "Object-Oriented Simulation Modeling with C++/CSIM," *Proc. 1995 Winter Simulation Conf.,* pp. 529-533, 1995.

[10] T.T. Kwan, R.E. McGrath, and D.A. Reed, "User Access Patterns to NCSA's World Wide Web Server," CS Technical Report UIUCDCS-R-95-1934, Univ. of Illinois at Urbana-Champaign, Feb. 1995, ftp://ftp.cs.uiuc.edu/pub/dept/tech_reports/1995/.

[11] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing Reference Locality in the WWW," *Proc. Fourth Int'l Conf. Parallel and Distributed Information Systems (PDIS '96),* Dec. 1996.

[12] V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," *Proc. 1997 Winter Simulation Conf.,* Dec. 1997, http://wwwnrg.ee.lbl.gov/floyd/papers.html.

[13] Y. Hu, A. Nanda, and Q. Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server," *Proc. 18th IEEE Int'l Performance, Computing and Comm. Conf. (IPCCC '99),* Feb. 1999.

[14] J. Hu, S. Mungee, and D. Schmidt, "Techniques for Developing and Measuring High-Performance Web Servers over ATM Networks," *Proc. INFOCOM '98,* 1998.

[15] E.P. Markatos, "Main Memory Caching of Web Documents," *Proc. Fifth Int'l World Wide Web Conf.,* May 1996.

[16] P. Rodriguez, E.W. Biersack, and K.W. Ross, "Improving the Latency in the Web: Caching or Multicast?" *Proc. Third Int'l WWW Caching Workshop,* June 1998.

[17] A. Erramilli, O. Narayan, and W. Willinger, "Experimental Queuing Analysis with Long-Range Dependent Packet Traffic," *IEEE/ACM Trans. Networking,* vol. 4, pp. 209-223, Apr. 1996.

[18] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing Differentiated Quality-of-Service in Web Hosting Services," *Proc. 1998 Workshop Internet Server Performance,* June 1998.

[19] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services," *IEEE Network,* pp. 64-71, Sept./Oct. 1999.

[20] P. Fishburn and A. Odlyzko, "Dynamic Behavior of Differential Pricing and Quality of Service Options for the Internet," *Proc. First Int'l Conf. Information and Computation Economics (ICE '98),* 1998.

[21] M. Harchol-Balter, M.E. Crovella, and C.D. Murta, "On Choosing a Task Assignment Policy for a Distributed Server System," *Proc. Performance Tools '98,* pp. 231-242, 1998.