OPERATING SYSTEMS SUPPORT FOR WWW SERVERS

Sukanya Suranauwarat and Hideo Taniguchi
Graduate School of Information Science and Electrical Engineering,
Kyushu University,
Fukuoka 812-8581, Japan
{sukanya,tani}@csce.kyushu-u.ac.jp

Abstract

The explosive growth of the WWW (World Wide Web) is placing a heavy demand on servers. Thus, improving server performance is important and can be achieved by scheduling resources more efficiently in operating systems. We have proposed a CPU resource scheduling policy for improving response time of a WWW server. This policy gives preferential use of the CPU resource to any process of a server that is predicted to be a server process handling an HTML file request. This allows users to view text data and the general layout of a WWW page in a timely manner during periods of high demand. In order to determine which processes are server processes handling HTML file requests, we introduced scheduling parameters SLP and RW. In this paper, we describe how we predicted and updated parameter RW based on the behavior of a WWW server, and present experimental validation of our method.

1 INTRODUCTION

WWW-based Internet information service has grown enormously over the past five years. It has succeeded because it gives users quick and easy access to a vast amount of worldwide information, the most valuable resource of IT (Information Technology) business and industry. With this success and growth has come a heavy demand on servers. As a result, users experience slow response times on popular or busy sites, which are accessed by many requests simultaneously. Therefore, server performance has become a critical issue for improving the quality of service on the WWW.

A fundamental way to achieve the goal of improving a server's performance is to improve operating systems support for servers, which can be done, for example, by improving stability under overload [Druschel and Banga, 1996][Mogul and Ramakrishnan, 1997], improving server control mechanisms [Banga et al., 1998], and reducing data movement costs [Anderson et al., 1995][Pai et al., 1999]. However, in the area of allocating resources such as the CPU, traditional operating systems still perform poorly when running a WWW server. This is because most traditional operating systems control the allocation of the CPU resource among processes using a fixed scheduling policy, in which the utilization of a computer system (e.g., a real-time or a time-sharing system) is a major concern rather than **contents** or **behavior of processes**. As a consequence, the CPU resource is likely to be used in an inefficient manner, or the processing time of a process might be extended unnecessarily.

For example, in a time-sharing system, each process is assigned a time interval, called a *quantum* or *timeslice*. If the process is still running at the end of its timeslice, the CPU is preempted and given to the next waiting process no matter how much more CPU time the process needs. Thus, a process that needs just a little bit more CPU time to complete its job will not be completed until its next timeslice. Because of this, the processing time and the process switching cost increase unnecessarily. If we had delayed process switching according to the process's behavior allowing the process to continue its execution until it accomplished its job, then the extra costs mentioned above would have been avoided.

Therefore, we have developed and proposed process's behavior-based scheduling policies. One of them is for improving the performance of the applications comprised of multiple processes, like WWW servers. To be more precise, this policy gives preferential use of the CPU resource to any process of a server that is predicted to be a server process handling an HTML file request, by moving it to the head of the *ready queue* where processes waiting for the CPU to become available are placed. **This allows users to view text and the general layout of a WWW page in a timely manner during periods of high demand**.

In order to determine which processes are server processes handling HTML file requests, we introduced scheduling parameters SLP and RW. A description of how to predict and update SLP including experimental validation of our approach have already been reported in [Suranauwarat and Taniguchi, 1999a and 1999b].

In this paper, we describe how we predicted and updated the scheduling parameter RW based on the execution behavior of a WWW server, and present experimental validation of our method.

The remainder of this paper is organized as follows. Section 2 provides background information on the WWW. Section 3 is a brief overview of our scheduling policy and important aspects of its implementation. Section 4 describes how to predict and update RW. Section 5 describes our experiments and explains the results we obtained. Section 6 offers our conclusions and future work.

2 THE WWW

The WWW is based on the client-server model [Comer, 1999][Tanenbaum, 1996]. That is, users access WWW *pages* provided by WWW servers via WWW clients, mainly browsers. WWW pages are written in the HyperText Markup Language (HTML) and stored in a disk as text files called HTML files. An HTML file contains text data that users will view and HTML tags that specify structure for the text data as well as formatting hints. Because an HTML file uses a text representation, non-text data such as images are not included directly in the HTML file. Instead, a tag is placed in the HTML file to specify the place at which an image should be inserted and the source of the file that stores it (Image file). HTML and Image files account for more than 90% of the total requests to a server [Arlitt and Williamson, 1997][Cunha et al., 1995]. Therefore, the WWW page in this paper consists of an HTML file and an Image file.

WWW clients. A user can access the information on the WWW by using a browser, such as Netscape Navigator, Mosaic, or lynx. When the user selects a WWW page to retrieve (usually, by clicking a mouse on a hyperlink), the browser creates a request to be sent to the corresponding WWW server and then waits for a response. When the response arrives, the browser interprets and processes the HTML file sent back by the server, and then displays the text data for the user to view. During the interpretation, if the WWW page also contains other types of data such as an image, then the browser will request the Image file from the WWW server.

WWW servers. The purpose of a WWW sever is to provide WWW pages to WWW clients that request them. The server software we used is *Apache* version 1.2.5, the pre-forking model server [Gaudet, 2000]. In this model, a *master* process accepts new requests and passes them to the pre-forked *worker* processes. We will refer to each pre-forked worker process as a server process in later sections.

3 OVERVIEW

When a WWW server is busy, that is, when it is accessed by a lot of browsers simultaneously, it takes time for the text data stored in HTML file to show up on browsers. As a result, users experience slow response times. This situation could be one in which it is most desirable to improve the response time of a WWW server, since users tend to get frustrated if it takes a long time to view a WWW page [Musciano, 1996]. Hence, in such a situation, **our scheduling goal is to display the text data for the user to view as soon as possible while the images are trickling in and also to allow the user to stop loading if the page is not sufficiently interesting to warrant waiting**. A method to achieve this goal is described below.

Most processes, except the currently executing process (i.e., process that is in the *run* state), are in one of two queues: a ready queue or a sleep queue. Processes that are waiting for the CPU to become available (i.e., in the *ready* state) are placed on a ready queue, whereas processes that are blocked awaiting an event (i.e., in the *wait* state) are located on a sleep queue associated with the event. When a process is blocked awaiting an event to happen, if the resources (e.g., a hard disk) needed for the event are being used by any other process, then that process needs to wait for those resources to become available. Next, that process needs to wait for the operation (e.g., input/output) it initiated to be completed. By reducing the time waiting for the CPU to become available in the ready queue or for the resource needed for an event to become available in the sleep queue, we can achieve an enhanced response time. According to this, we proposed the CPU scheduling policy that when a CPU (a hard disk or a network communication) becomes bottlenecked, any server process handling an HTML file will be moved to the head of the ready queue (sleep queue associated with the event). Note that the bottleneck of the CPU mentioned in this paper is the situation in which the CPU is busy and there is more than one process waiting in the ready queue.

When we discussed the scheduling mechanism that implements the above policy focused on the bottleneck of the CPU, we had **two** problems: **how to detect which processes are server processes handling HTML files**, and **how to operate the ready queue**. To answer these questions we found that we needed to look at the detailed execution behavior of a WWW server. So, we logged the execution behavior of a WWW server and created the predicted execution behavior called *PFS (Program Flow Sequence)* for each server process. PFS is a sequence of entries describing process state and time spent. We analyzed the behavior of server processes based on PFS and found that any server process handling an HTML file has two characteristics: After waiting for a long time in the wait state (characteristic 1), it tends to cycle between run state and wait state a number of times but fewer times than that of a server process handling an Image file (characteristic 2).

To deal with the fist problem, we introduced two parameters into our mechanism in order to determine which processes are server processes handling HTML files: long wait threshold (its value is denoted by SLP) and run state/wait state threshold (its value is denoted by RW). If the time spent by a process in the wait state before moving to the run state is more than SLP, and the number of times the process changes between run state and wait state is less than RW, then we determine that it is a server process handling an HTML file. By these two parameters, we can detect which process appears to be a server process handling an HTML file.

To deal with the second problem, our mechanism puts any process that has characteristic 1 at the head of ready queue and moves that process to the back of the ready queue when that process loses characteristic 2. The reason processes that lose characteristic 2 are moved to the back of the ready queue is that in the case that server processes handling Image files are mistaken as server processes handling HTML files because they exhibit characteristic 1. In other words, a server process handling an Image file will be treated as a server process handling an HTML file when it runs after a long wait, until the number of changes between run state and wait state is more than RW.

4 METHOD OF PREDICTING RW

We analyzed the execution behavior of a WWW server based on its PFS and found that the number of times a server process changes between run state and wait state is proportional to the size of the file it handles (i.e., HTML file or Image file). In other words, the number of changes is proportional to the number of disk accesses required to retrieve requested files. Since HTML files are generally smaller than Image files, the number of times a server process handling an HTML file changes between run state and wait state is smaller than that of a server process handling an Image file. Therefore, the smallest number of times each server process changes between run state and wait state is determined from its PFS for each period, then RW for the next time period is set to the greatest of these values. We describe how to predict RW, in a more general way, in the following steps:

STEP 1: When a server process is created, that process will be registered. For each registered process, a PFS is generated.

STEP 2: The PFS for registered server processes is generated once every period, where the period length is predetermined. The periods are the same for PFS and RW.

STEP 3: At the end each period, the PFS for each registered process will be updated. Based on the PFS for each process, we find the smallest number of times each process changed between run state and wait state. The RW for the next time period is set to the greatest of these values.

STEP 4: When a registered process terminates, it becomes unregistered, in other words, the PFS for that process will no longer be updated.

In STEP 3, we find the smallest number of times each process changes between run state and wait state, this is actually quite an involved process and requires a more detailed description. First, for each process a record is kept of the number of consecutive times it changes between run state and wait state, this is called the *RWbuffer*. Second, we set what the minimum length of time is for a long wait, *SLPmin*. Third, we set a minimum threshold for RW, *RWmin*, in order to avoid considering the number of changes due to anything but the number of disk accesses for requested files, for example, the number of changes due to a *select* system call, which requires one or two changes per occurrence. At the end of each time period, from the updated PFS, the number of changes between run state and wait state (*counter*) are counted until the time in the wait state exceeds SLPmin. When SLPmin is exceeded, the counter is compared to RWmin, if it is greater than RWmin then the value is stored in the RWbuffer for that process, otherwise it is discarded. The counter is then reset and it starts counting from where it left off. The minimum values from each process' RWbuffer are compared and RW for the next time period is assigned the largest value. The reasons for using SLPmin, RWmin and RWbuffer are described below. SLPmin is used so that RW can be determined independently from SLP. RWmin is used so that unrelated changes between run state and wait state will be disregarded. RWbuffer is used so that RW for the next time period is based more on a recent history of the processes rather than just the latest period's behavior. Also by using RWbuffer, if processes run abnormally for a short period of time, then RW will not be effected also it tends to cause RW to reflect the behavior of the server processes more accurately.

5 EXPERIMENTAL EVALUATION

We performed experiments to evaluate the effectiveness of our scheduling mechanism when RW is predicted and updated based on PFS automatically every 500 milliseconds. Our mechanism was implemented as modifications to the BSD/OS 2.1 kernel.

5.1 EXPERIMENTAL SETUP

In all experiments, the server machine was a 233MHz AMD-K6 PC, with 64MB of memory, running our modified version of BSD/OS 2.1. The client machines were 200MHz Pentium Pro PCs, with 64MB of memory, running BSD/OS 2.1. Our server and client software were Apache 1.2.5. and Netscape Navigator 3.04 respectively. All experiments were conducted in single user mode, and the operating system's I/O buffer cache in the server machine as well as each browser's cache were disabled during the experiments, in order to see the effect of our scheduling mechanism clearly. Also, RWbuffer and RWmin are respectively set to 5 and 2 while SLPmin is set to 200 milliseconds.

5.2 EXPERIMENTAL RESULTS

The first experiment is a simple test to verify that our method of predicting RW works well with a real WWW server. This test was conducted using only one of the three of client machines and the server machine in a 10 Mbps Ethernet environment. In order to see the effect of our method clearly, we ran a computation intensive background process that bottlenecked the CPU throughout the experiment. And, we set the browser to access the WWW server in such a way that SLP would be predicted 100% correctly, that is, it accessed the WWW server every 30 seconds while SLP was fixed at 20 seconds.

In this experiment, we varied the size of the HTML file from 1, 3, 5, and 7 times the original size (1,770 bytes) while the size of the Image file was fixed at 43,770 bytes. For each size of the HTML file, we measured the 5 trial times of *time1* and *time2*, when RW was fixed at 1, 3, 5 (*RW=1,3,5*) and when RW was automatically set based on PFS (*RW=auto*). Time1 is the time from requesting a WWW page until text data starts displaying. Time2 is the time from requesting a WWW page until image data displays completely. We will refer to the averages of 5 trail times of time1 and time2 as response time of text data and image data respectively.

Figure 1 shows experimental results plotted with the response time on the y-axis normalized by the response time when using a conventional time-sharing mechanism. To analyze the obtained results, *file read-ahead* operation in operating system needs to be taken into consideration. Since it will have an effect on the number of disk accesses required to retrieve HTML or Image files, which consequently affects the number of times a corresponding process changes between run state and wait state.

In UNIX systems, files are stored in the disk as a number of blocks, each of which has the same size of 4 KB. In order to improve the performance, operating systems usually perform the file read-ahead, when it detects sequential access to a file. In other words, if a file is store as consecutive blocks, then the operating system will asynchronously prefetch the next block of file data with each read request (8 KB of data will be read); otherwise it just performs a normal read request (4 KB of data will be read). Table 1 summarizes the number of disk accesses required to retrieve HTML files when they are stored as consecutive blocks and when they are not.

Table 1 shows that the number of disk accesses is at most 3 when the size of the HTML file is 1, 3 and 5 times the original size. As a result, the response times of text data when RW=3 and when RW=5 are about the same as shown in Figure 1(a). In the same way, the response times when RW=auto are improved. However, when the size of the HTML file is 7 times the original size, the improvement when RW=5 is better than when RW=3. This could be because the HTML file in this case is split up into non-consecutive blocks causing the disk to be accessed 4 times (which is more than the RW value of 3) as shown in Table 1. On the other hand, the results when RW=5 and when RW=auto are about the same.

Table 1 The number of disk access(es) required to retrieve an HTML file.

Size of HTML file	the number of disk accesses	
	consecutive case	non-consecutive case
1 time (1,772 bytes)	1 time	1 time
3 times (5,316 bytes)	1 time	2 times
5 times (8,860 bytes)	2 times	3 times
7 times (12,404 bytes)	2 times	4 times

Figure 1(b) shows that the response times of image data are improved. This could be because our policy of giving any server process handling an Image file that has waited for a long time in the wait state, priority over other processes including the computation intensive background process.

Summary: The response times of text data when RW was automatically set based on PFS are improved in the same way as when RW was fixed, or even better in some cases. Therefore, it can be said that RW is accurately predicted and set by our mechanism.

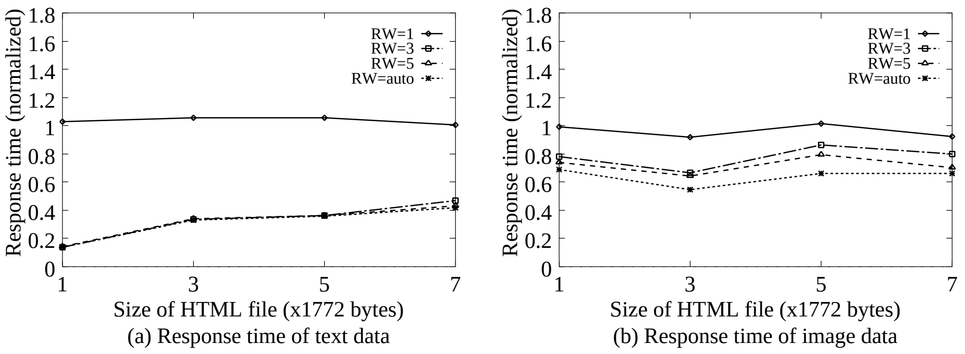

Figure 1 The effect of RW for various sizes of HTML files (1, 3, 5 and 7 times the original size).

However, we did not notice any difference in the degree of improvement between the results when RW=auto and the results when RW=5 in which our mechanism produces the best improvement among fixed values of RW. So, we decided to measure the server performance when the size of the HTML files are increased more, which results in an increase in the number of times a server process handling an HTML file changes between run state and wait state.

The next experiment measured the response times when the size of the HTML files varied from 10, 20, 30, 40 and 50 times the original size while the size of the Image file was the same as in the first experiment. The experimental results are shown in Figure 2.

Figure 2(a) shows that the response times of text data when RW=auto are slightly better than when RW=5, after the sizes of HTML files are more than 40 times the original size at which the number of disk accesses whether the file are stored as consecutive blocks or not is more than the RW value of 5. Moreover, after the sizes of HTML files are more than 30 times the original size, the difference in the degree of improvement between the results when RW=5 as well as when RW=auto and the results when RW=3 are more distinct than those in Figure 1(a).

Response times of image data are also improved in this experiment as shown in Figure 2(b), and the reason is the same as in the first experiment.

Summary: The response times of text data when RW was automatically set based on PFS are improved the most. Therefore, setting RW based on PFS reflects the behavior of server processes.

Generally, HTML files are small, so varying the size in the range as in the first experiment should be enough.

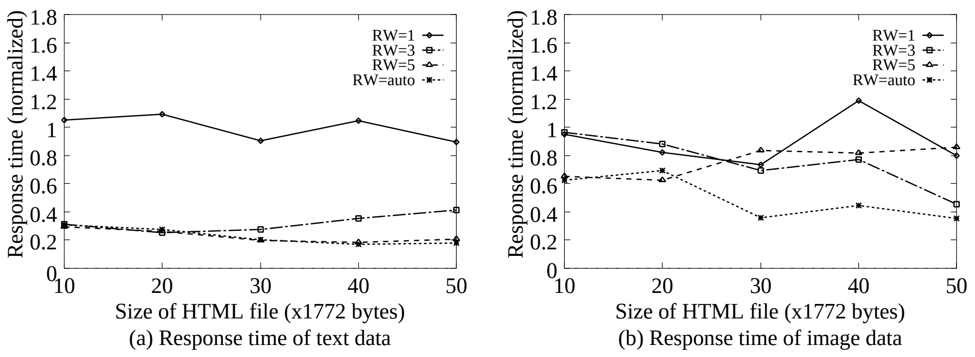

Figure 2 The effect of RW for various sizes of HTML files (10, 20, 30, 40 and 50 times the original size).

The last experiment was designed to examine the performance of the server when it is accessed by many requests simultaneously. This test was conducted using the three client machines and the server machine in a 10 Mbps Ethernet environment. In order to see the effect of our predicting RW method clearly, we ran three browsers from each of the three machines simultaneously; this number can cause bottleneck of the CPU during the short period of simultaneous accesses [Suranauwarat and Taniguchi, 1999b]. And, we set all of the browsers to access the WWW server simultaneous every 30 seconds while SLP was fixed at 20 seconds, so that SLP would be predicted 100% correctly. Note that all browsers accessed 18 unique URLs (Uniform Resource Locator), all of which have the same content. Also, there was no computation intensive background process in order to make the situation more likely to be realistic.

In this experiment, we measured the response times when HTML files and Image file are respectively varied and fixed in the same way as in the first experiment. The experimental results are shown in Figure 3 to Figure 6.

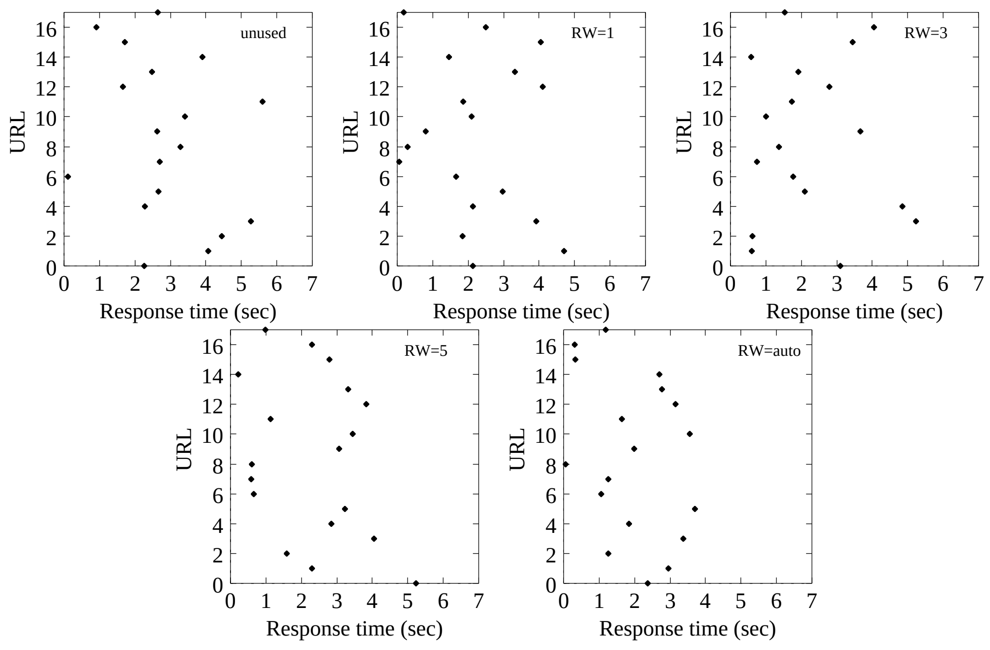

Figure 3 The distribution of response times of text data when the size of HTML file is the original size.

Figure 3 shows the responses time of text data when the size of HTML file is the original size. For comparison, we also measured the performance with a conventional time-sharing mechanism. Figure 3 plots the URLs in numerical sequence on the y-axis against the response time of text data to a request in seconds. This figure shows that the distributions are skewed toward the small values when using our mechanism.

In order to make the experimental results easier to understand and discuss, we put all the data shown in Figure 3 into one graph as shown in Figure 4(a). Figure 4(a) illustrates the minimum, the maximum, the mean, the median values, and the range or distribution of the response times of text data to a request in seconds. Note that the median values can be skewed if we calculated it from the average of 5 trial times, so we calculated them directly from the raw data. Figure 4(a) shows that the mean response time of text data when RW=1, 3, 5 and auto are improved 22.9%, 21.0%, 19.0% and 31.8% respectively, while the median are improved 20.4%, 14.4%, 15.3% and 36.5% respectively. These figures are calculated from by $(a-b)/a \times 100\%$, where a and b are the mean (or the median) values when not using and using our mechanism respectively. Besides, the maximum response times are also pressed down when using our mechanism, especially when RW=auto.

The experimental results pertaining to image data are plotted in the same way as shown in Figure 4(b). This figure shows that the mean and the median response times when using our mechanism are not fast as when not using ours. This is expected and is due to our policy of giving processes handling HTML files priority over all other processes including server processes handling Image files.

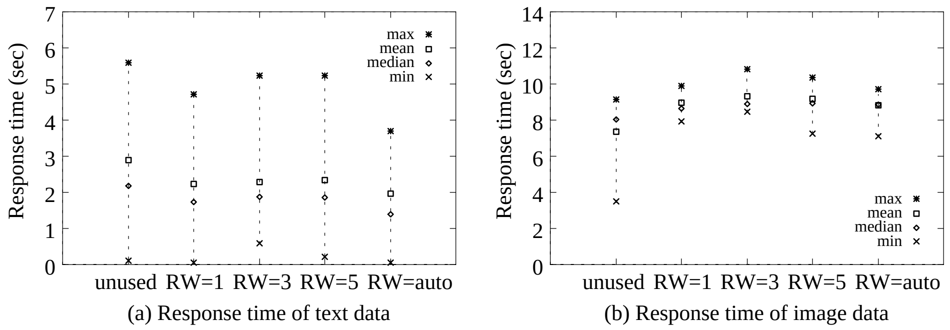

Figure 4 The response times when the size of HTML file is the original size.

In order to compare and discuss each set of results, we plotted the minimum, the maximum, the mean and the median values of response time for each size of the HTML files as shown in Figure 4 on the y-axis, against the size of the HTML files as shown in Figure 5 and Figure 6. Note that each response time on the y-axis is normalized by its response time for a conventional time-sharing mechanism.

Figure 5 shows that the maximum response times are pinned down when using our mechanism even though the minimum values are higher in some cases. That is why the mean and the median values become smaller. As a consequence, the service of the WWW server will be more equally distributed among users and users will perceive a faster response when using our mechanism. Also, our mechanism produces the best improvement for every case when RW=auto.

As expected, in Figure 6, we did not notice any improvement of response time of image data when using our mechanism. However, even though we give processes handling HTML files priority over all other processes including server processes handling Image files, the affect of which on the max, the mean and the median response times are relatively small.

Summary: Our mechanism produces the best improvement when RW is automatically set based on PFS. For example, in the case that the size of the HTML file is the original size, the mean response times of text data when RW was automatically set based on PFS are improved up to 31.8% while those when RW was fixed are improved up to 22.9%.

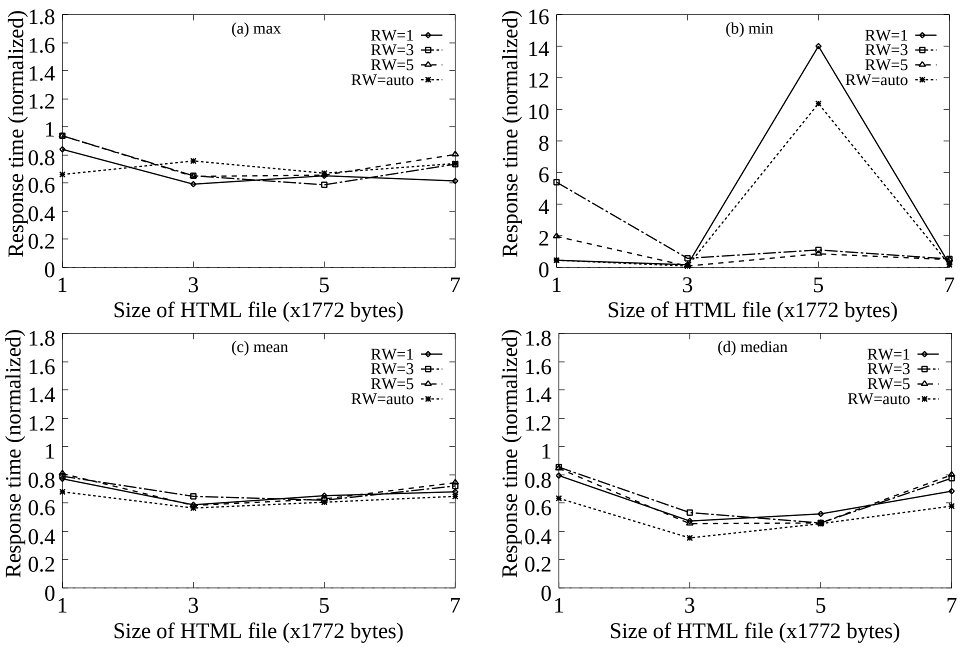

Figure 5 The effect of our mechanism on response times of text data.

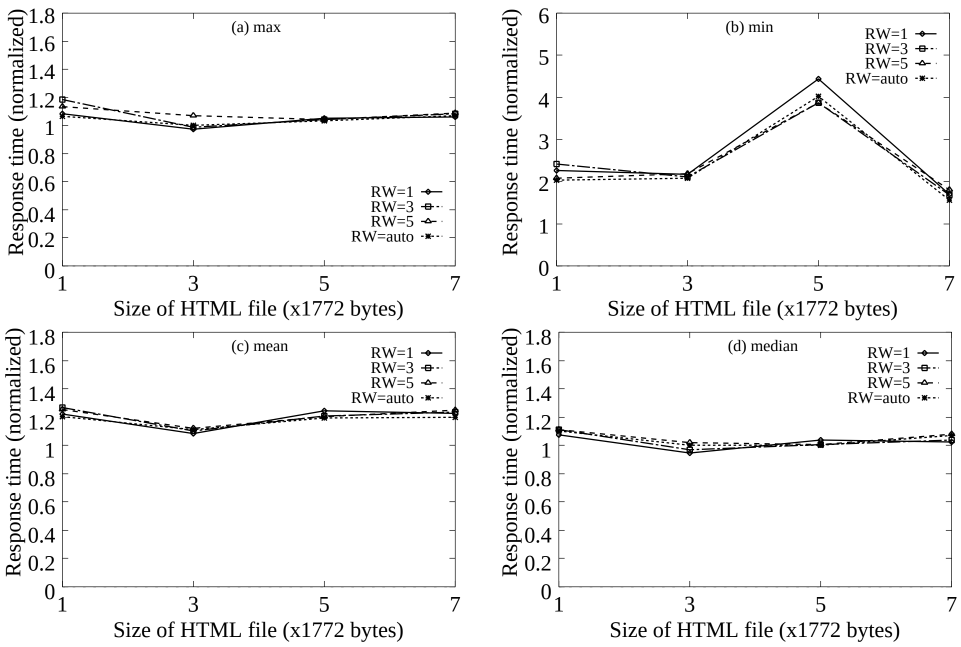

Figure 6 The effect of our mechanism on response times of image data.

6 CONCLUSIONS

The exponential growth in the use of the WWW is placing a heavy demand on servers. As a result, users experience slower response times on popular or busy sites. Thus, improving server performance is important and improving operating systems support (e.g., resource scheduling) for servers is vital to this goal. We have already proposed a CPU resource scheduling policy for improving the response time of a WWW server. This policy gives preferential use of the CPU resource to any process of a server that is predicted to be a server process handling an HTML file request. This allows users to view text data and the general layout of a WWW page in a timely manner during periods of high demand.

In order to determine which processes are server processes handling HTML file requests, we introduced scheduling parameters SLP and RW. In this paper, we described how we predicted and updated parameter RW based on the behavior of a WWW server. And we evaluated the effectiveness of our scheduling policy when RW was automatically set using the proposed method. Our experimental results show that our mechanism produces the best improvement for every case when RW is automatically set using the proposed method. For example, in the case that the size of the HTML file was 1,772 bytes, the mean response times of text data when RW was automatically set are improved up to 31.8% while those when RW was fixed are improved up to 22.9%. They also show that the maximum response times are pinned down in every case even though the minimum values are higher in some cases when RW is set using the proposed method. That is why the mean and the median values become smaller. As a consequence, the service of the WWW server will be more equally distributed among users and users will perceive faster response times. Therefore, our method of predicting and setting RW is effective.

Future work is required to measure the performance of a WWW server when the scheduling parameter SLP and RW are both automatically predicted and updated.

ACKNOWLEDGEMENTS

We would like to thank James Michael Perry for his assistance in proofreading the paper.

REFERENCES

Anderson, E. W. and Pasquale. (1995). The performance of the container shipping I/O system. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*.

Arlitt, M. F. and Williamson, C. L. (1997). Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Trans. Networking*, 5(5):631-645.

Banga, G., Druschel, P., and Mogul, J.C. (1998). Better operating system features for faster network servers. In *Proc of the 1998 Workshop on Internet Server Performance*.

Comer, D.E. (1999). *Computer Networks And Internets, 2nd ed*. Prentice-Hall.

Cunha, C., Bestavros, A., and Crovella, M. (1995). Characteristics of WWW client-based traces, Tech. Rep. BU-CS-95-010, Computer Science Department, Boston University.

Druschel, P. and Banga, G. (1996). Lazy receiver processing (LRP): A network subsystem architecture for server systems. In *Proc. of the 2nd USENIX Symposium on Operating Systems Design and Implementation*.

Gaudet, D. (2000). *Apache Performance Notes*. URL:http://www.apache.org/docs/misc/perf-tuning.html.

Mogul, J. C. and Ramakrishnan, K. K. (1997). Eliminating receive livelock in an interrupt-driven kernel. *ACM Trans. Comput. Systems*, 15(3):217-252.

Musciano, C. (1996). *Tuning Unix for Web Service*. URL:http://www.sunworld.com/swol-01-webmaster.html

Pai, V. S., Druschel, P., and Zwaenepoel, W. (1999). IO-Lite: A unified I/O buffering and caching system. In *Proc. of the 3rd USENIX Symposium on Operating.*

Suranauwarat, S. and Taniguchi, H. (1999a). Process scheduling policy for a WWW server based on its contents. *Trans. IPSJ*, 40(6):2510-2522. (in Japanese)

Suranauwarat, S. and Taniguchi, H. (1999b). Evaluation of process scheduling policy for a WWW server based on its contents. In *Proc. of the 1999 IPSJ Computer System Symposium*.

Tanenbaum, A. (1996). *Computer Networks, 3rd ed*. Prentice-Hall.