Vol. 0 No. 0 Transactions of Information Processing Society of Japan 1959

Evaluation of Process Scheduling Policy for a WWW Server based on Its Contents

SUKANYA SURANAUWARAT † and HIDEO TANIGUCHI†

Traditional process schedulers in operating systems control the sharing of the processor resources among processes using a fixed scheduling policy based on the utilization of a computer system such as a real-time or a timesharing system. Since the control over processor allocation is based on a fixed policy, not based on contents or behavior of processes, this can hinder an effective use of a processor or can extend the processing time of a process unnecessarily. We have already proposed a process scheduling policy, which responds to the behavior of multiple processes of a WWW server, in order to improve the response time of a WWW server. This policy gives any process of a WWW server that is predicted to be a WWW server process handling a text data request from a browser priority over all other processes by moving it to the the head of the ready queue where processes waiting for the processor to become available are placed.

In this paper, we present the experimental evaluation of our proposed process scheduling policy with regard to the number of simultaneous accesses from browsers and the processor load of the server machine, and explain the results we obtained.

1. Introduction

Traditional process schedulers in operating systems control the sharing of the processor resources among processes using a fixed scheduling policy based on the utilization of a computer system such as a real-time or a timesharing system. A real-time system's scheduling policy must be able to analyze or handle data faster than they come in and it must also respond to time events. There are many applications in which computations must be completed before specified deadlines1) and missing these deadlines are catastrophic. Therefore, scheduling such applications has been an important area of research in real-time systems (e.g., 1)∼4)). A timesharing system's scheduling policy is to provide good response to interactive users. Many commonly-used systems such as Unix, Mach, and Windows NT generally use conventional priority-based timesharing schedulers5).

Since the control over processor allocation is based on a fixed policy which is determined by the utilization of a computer system, not based on contents or behavior of processes, where a process is a program in execution, this can hinder an effective use of a processor or can extend the processing time of a process unnecessarily. Therefore, we proposed the idea called POS (Program Oriented Schedule)6). The idea of POS is by increasing operating system ability to alter the execution behavior of a program according to the predicted behavior from the previous execution(s), the operating system could optimize the execution behavior of the program allowing user requirements (e.g., performance enhancement) to be satisfied without making any changes to the existing program. In order to predict the execution behavior of a program, the idea of POS requires that the operating system has the ability to log the execution behavior of the program in terms of process identifier, process state and time. And based on this log, the operating system will create the predicted execution behavior of the program or update it if it already exists.

One function in Windows 98, which users can get "faster program start up" as performance enhancement, uses an idea similar to that of POS. That is the function improves the performance of a user's programs based on the previous usage without making any changes to the programs. In other words, the function creates a log file to determine which programs a user runs most frequently. All such frequently used files are then placed in a single location on the user's hard disk, which further reduces the time needed to start those programs7). However, the function does not alter the execution behavior of programs based on the previous usage which is what our idea does. So, by using the function in Windows 98, the operating system can

† Graduate School of Information Science and Electrical Engineering, Kyushu University

control a user's programs more efficiently until a user's programs start up (i.e., the operating system can locate and open a user's programs faster), but it cannot execute or run a user's programs more efficiently.

We have already applied POS to the process scheduler[6),8)]. In 6), we proposed the process scheduling policy that allows a process to continue its execution even though its quantum has already expired when it is predicted that a process needs a little bit more processor time to complete its job. The objective of this policy is to minimize processing time and/or context switching cost of the process of the target program. However, the target programs of this policy are the programs that consist of only a single process. So, we extended our work to the programs composed of multiple processes such as servers. Server performance is crucial to client/server applications[9)]. We used a WWW server, which is a program that consists of multiple processes, as a sample server. And we proposed a process scheduling policy for improving the response time of a WWW server[8)]. This policy gives any process of a WWW server that is predicted to be a WWW server process handling a text data request from a browser priority over all other processes by moving it to the the head of the ready queue where processes waiting for the processor to become available are placed. We also evaluated this policy experimentally in simple cases for the purpose of verifying that our scheduling mechanism works as expected.

In this paper, we present the experimental evaluation of the proposed process scheduling policy in more complicated and useful cases compared with those in 8). First, we measure the performance of a WWW server in terms of response time and compare the performance of the proposed policy to that of a conventional priority-based timesharing policy when the WWW server is accessed by a lot of browsers simultaneously. For a WWW server, this kind of situation is more likely to be a realistic case compared with those in 8) where the number of browsers accessing the WWW server ranged from 1 to 3 and just two machines (a client machine and a server machine) were used. And this could be the situation in which it is most desirable to improve the response time of a WWW server. Second, we measure the effect of the number of simultaneous accesses on the processing ability of the server machine in terms of response time and processor load. Then, we find the relation between the number of simultaneous accesses and the response time, and the relation between the number of simultaneous accesses and the processor load.

The rest of this paper is organized as follows. Section 2 gives an example of the effect of POS. Section 3 briefly overviews our scheduling mechanism. Section 4 describes our experiments and explains the results we obtained. Section 5 offers our conclusions.

2. An Example of the Effect of POS

We used an easy example shown in **Fig. 1** to identify how performance will be improved when POS is applied to the process scheduler.

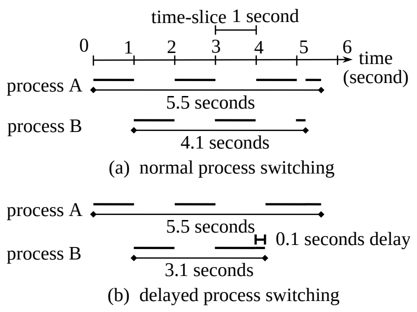

Fig. 1 An Example of the Effect of POS.

In Fig. 1, process A and process B need respectively 3.4 seconds and 2.1 seconds of processor time to accomplish their jobs. Both processes have the same priority and a time-slice of 1 second. Figure 1(a) shows the processing times of process A and process B when using a conventional priority-based timesharing scheduler. The processing times of process A and process B are 5.5 seconds and 4.1 seconds respectively. On the other hand, Figure 1(b) shows the processing times of process A and process B when POS is applied. Based on the predicted behavior that process B needs only 0.1 seconds more processor time to complete its job, the process scheduler delays process switching by 0.1 seconds to allow process B to complete its job. Delaying process switching causes the processing time of process B to be reduced to 3.1 seconds while that of process A is still the same as in Fig. 1(a). Moreover, the context switching cost of process A and B also decrease.

This example also shows how the process scheduling policy, we proposed in 6), controls the time-slice length of the object process, and how the processing time and/or context switching cost of the object process will decrease when using this policy.

3. Overview

Our process scheduling mechanism is composed of two parts: the logging mechanism and the process control mechanism. We log the execution behavior of a WWW server and create/update the predicted execution behavior called PFS (Program Flow Sequence) through the logging mechanism. And we alter the execution behavior of the WWW server by using the process control mechanism.

3.1 Logging Mechanism

When a WWW server is running, a log is collected. A log is a sequence of entries describing process identifier, process state and time. And based on this log a sequence called PFS, which is the predicted behavior of a process, is created or updated for each process of a WWW server. Note that a WWW server is normally composed of multiple processes. PFS is a sequence of entries describing process state and time spent.

3.2 Process Control Mechanism

For this paper, the content of a WWW page is pretty simple, that is, one which contains just text data and image data. Text data and image data are separately saved in a file written in HTML (HTML file) and an image-formatted file (Image file) respectively. After making a request for a WWW page, the browser will interpret and process the HTML file sent back by the WWW server it requested and then display the text data. During the interpretation, if the WWW page also contains image data, then the browser will request the WWW server again for the Image file. When a WWW server is accessed by a lot of browsers simultaneously, it takes time even for the text data which is normally much smaller than image data to show up on browsers. This kind of situation could cause users to give up waiting or to get tired of accessing such WWW servers. So, while a WWW server is accessed by a lot of browsers simultaneously, it is much better if the text data shows up faster. Therefore, we considered improving the time from requesting a WWW page until text data displays (response time).

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 processor 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 first for those resources to become available. Next, that process needs to wait again for the operation (e.g., input/output) it initiated to be completed. By reducing the time waiting for the processor 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 process scheduling policy that when a processor (a hard disk or a network communication) becomes bottlenecked, any WWW server process handling an HTML file will be moved to the head of the ready queue (sleep queue associated with the event)[8]. Note that the bottleneck of the processor mentioned in this paper is the situation in which the processor is busy and there is more than one process waiting in the ready queue.

When we discussed the process control mechanism that implements the above policy focused on the bottleneck of the processor[8], we had two problems: how to detect which processes are WWW 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. We analyzed the behavior of WWW server processes based on PFS and found out that any WWW server process handling an HTML file has 2 characteristics: it runs after waiting for a long time in the wait state (characteristic 1) and it tends to cycle between run state and wait state fewer times than that of WWW server process handling an Image file (characteristic 2).

To deal with the fist problem, we introduced two parameters into our process control mechanism in order to determine which processes are WWW 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 WWW server process handling an HTML file. By these two parameters, we can detect which process appears to be a WWW server process handling an HTML file.

To deal with the second problem, our process control 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 sometimes WWW server processes handling Image files are mistaken as WWW server processes handling HTML files because they exhibit characteristic 1.

How to predict and update SLP: We analyzed the execution behavior of a WWW server based on PFS and found that a WWW server process handling an HTML file is the process that waits for a request from a browser. The time it waits for a request is relatively long. Therefore, the longest time of each WWW server process in the wait state is determined from PFS for each period, then SLP for the next time period is set to the smallest of these values. SLP is updated every time period.

How to predict and update RW: We analyzed the execution behavior of a WWW server based on PFS and found that the number of times a WWW server process handling an HTML file or an Image file changes between run state and wait state is proportional to the size of the HTML file and Image file respectively. In fact, the number of times a WWW server process handling an Image file changes between run state and wait state is greater than that of a WWW server process handling an HTML file, because in general Image files are bigger than HTML files. Therefore, the smallest number of times of each WWW server process changing between run state and wait state is determined from PFS for each period, then RW for the next time period is set to the greatest of these values. RW is updated every time period.

4. Measures of Performance

4.1 Experimental Setup

Our scheduling mechanism is implemented on BSD/OS version 2.1. The software used for the WWW server and the browser in our experiment was Apache version 1.2.5 and Netscape Navigator version 3.04 respectively. The WWW server ran on a personal computer with a 233 MHz AMD-K6 processor and 64 MB of memory, while browsers ran on three personal computers, each with a 200MHz Intel Pentium Pro processor and 64 MB of memory. All machines were running on BSD/OS version 2.1 and were connected by a private 10Mb/s Ethernet. All our experiments were conducted in single user mode, and the operating system's I/O buffer cache in the server machine and each browser's cache were disabled during the experiments in order to see the effect of our scheduling mechanism clearly.

The WWW server was accessed by three browsers from each of the three machines simultaneously. All browsers accessed unique URLs all of which have the same content. In three different experiments in which we varied RW in the range from 1 to 10, we measured the time (t1) from requesting a WWW page until text data starts displaying and the time (t2) from requesting a WWW page until image data displays completely for each access, and then found the mean of the 5 trial times of t1 (response time of text data) and t2 (response time of image data). In experiment 1, all the browsers accessed the WWW server simultaneously every 30 seconds when the WWW server coexisted with a processor-bound process and SLP was fixed at 20 seconds. The purpose of this experiment is to know how the response time of text data would be improved in the situation that is the best for our scheduling mechanism (i.e., the processor of the server machine is bottlenecked [caused by a coexisting processor-bound process] and accesses from browsers are set in such a way that SLP will be predicted 100% correctly). In experiment 2, all the browsers accessed the WWW server randomly at the same time and SLP was fixed at 20 seconds, while in experiment 3, SLP was predicted and updated automatically based on PFS every 500 milliseconds. There was no processor-bound process in experiments 2 and 3. The purpose of experiments 2 and 3 is to know how in a more realistic situation (i.e., no processor-bound process coexists and accesses from browsers are random) the response time of text data would be improved when SLP was fixed and when SLP dynamically varies according to the execution behavior of the WWW server.

4.2 Experimental Result

Figure 2 shows some examples of the results of experiment 1, when not using and using our scheduling mechanism (RW = 3,6,9). We used a conventional priority-based time-sharing mechanism when not using ours. Figure 2 plots the URLs in numerical sequence on the y-axis against the response time of text data to a request in seconds. Figure 2 shows that the smallest and biggest response times when RW = 3,6,9 are better than when not using our scheduling mechanism. It also shows that the range or the variance of response times becomes narrower when using our scheduling mechanism.

Figure 3(a) illustrates the mean and the range of response times of text data from Fig. 2 into one graph. This figure shows that the mean response times of text data when RW = 3,6,9 are improved 33.8%, 21.3% and 26.6% respectively. These figures are calculated by $\frac{a-b}{a} \times 100\%$ where a and b are the mean response times when not using and using our scheduling mechanism respectively. This case produced the most improvement of the response time of text data, because the coexisting processor-bound process always caused the processor to become bottlenecked and the WWW server processes waiting for HTML file requests from browsers were always in the wait state at least 30 seconds which was more than SLP (20 seconds).

The rest of the experimental results will be shown like Fig. 3(a).

Figure 3(b) illustrates the mean and the range of response times of image data to a request in seconds. This figure shows that the smallest response times when RW = 3,6,9 and the mean response times are not as fast as when not using our scheduling mechanism. This is expected and is due to our policy of giving processes handling HTML files priority over all other processes including WWW server processes handling Image files.

Figure 4 and **Figure 5** show respectively the results of experiments 2 and 3 when not using and using our scheduling mechanism (RW = 3,6,9). Figure 4(a) and Figure 5(a) illustrate the mean and the range of response times of text data to a request in seconds. In Fig. 4(a), we did not notice any consistent improvement when using our scheduling mechanism even though the mean response time of text data when RW = 9 is faster than when not using it. On the other hand, in Fig. 5(a), although the smallest response times for RW

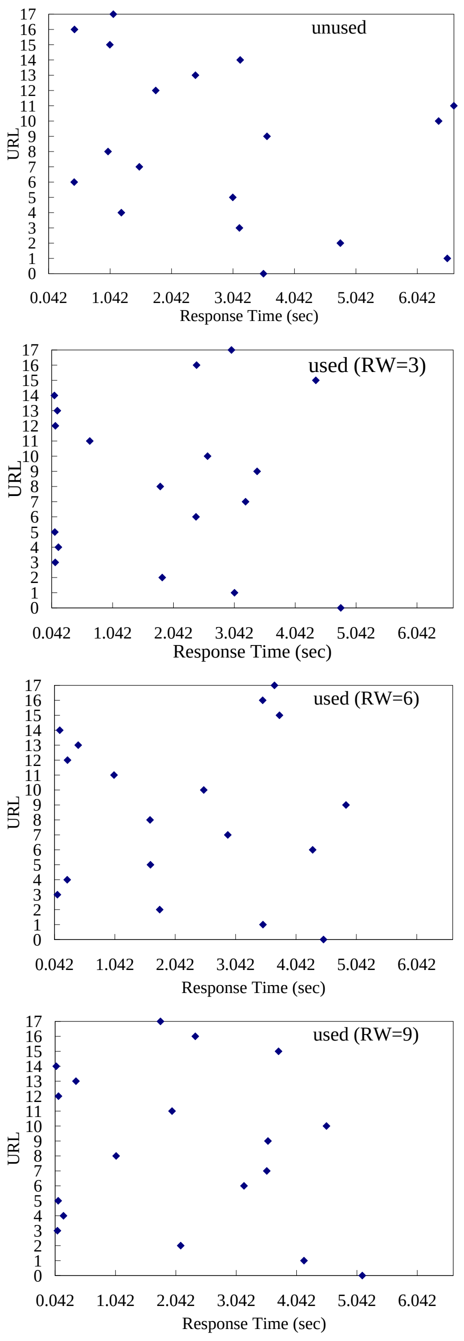

Fig. 2 Response Time of Text Data in Experiment 1.

= 6 and 9 are not better than when not using our scheduling mechanism, the range of response times is narrower and the mean response times of text data when RW = 3,6,9 are improved 22.2%, 17.9% and 6.7% respectively. Even though a processor-bound process causing the bottleneck of the server machine does

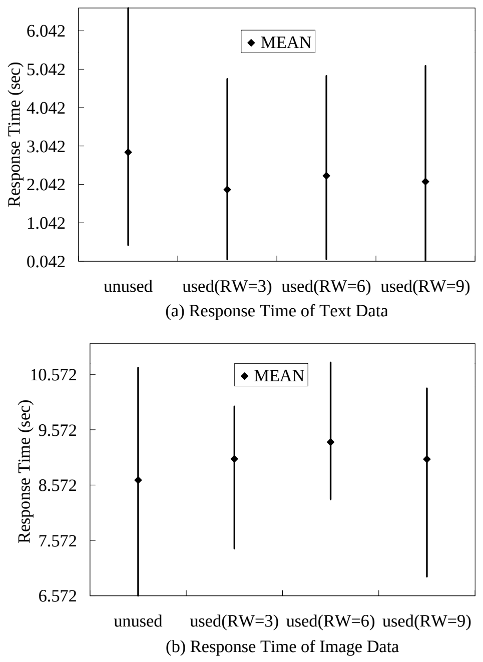

Fig. 3 The Effect of Our Scheduling Mechanism in Experiment 1.

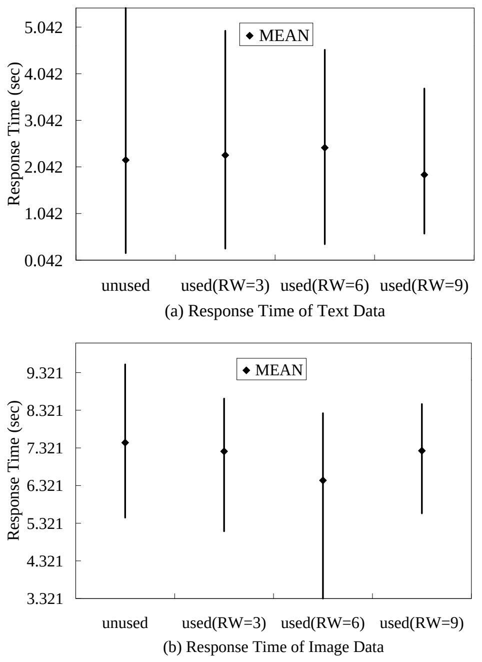

Fig. 4 The Effect of Our Scheduling Mechanism in Experiment 2.

not coexist in Fig. 5(a), our sheduling mechanism still produces a good improvement. This might imply that a lot of simultaneous accesses from browsers could cause the processor of the server machine to become bottlenecked. From now on,our explanations about experiments 2 and 3 will be done on the supposition that a lot of simultaneous accesses from browsers cause the processor of the server machine to become bottlenecked.

The only difference between experiments 2 and 3 is the way SLP was set. In experiment 2, the results of response times of text data shown in Fig. 4(a) indicate that the WWW server processes waiting for HTML file requests from browsers might not be in the wait state more than SLP, because SLP was fixed at 20 seconds while browsers accessed the WWW server randomly. On the other hand, the improvement of response times of text data in experiment 3 shown in Fig. 5(a) means that SLP is accurately predicted and updated by our scheduling mechanism based on the execution behavior of the WWW server, because SLP is predicted and updated automatically based on PFS every 500 milliseconds.

Figure 4(b) and Figure 5(b) illustrate the mean and the range of response times of image data to a request in seconds respectively. In Fig. 4(b), the mean reponse times of image data are improved. This could be because sometimes WWW server processes handling Image files were mistaken as WWW server processes handling HTML files when they were in the wait state more than SLP (20 seconds). In Fig. 5(b), the mean response times of image data when RW = 3,6,9 are not so different from when our scheduling mechanism is not being used as in experiments 1 and 2 (cf. Fig. 3(b), Fig. 4(b)). However, the results when RW = 3 and 9 show that sometimes WWW server processes handling Image files were mistaken as WWW server processes handling HTML files when they were in the wait state more than the predicted SLP.

Summarize experimental results pertaining to response time of text data: The results of experiment 1 show that the mean response times of text data are improved greatly (up to 33.8% when RW=3) when the processor of the server machine is bottlenecked which is the condition that our mechanism favors and SLP is predicted 100% correctly. The results of experiment 2 shows that the mean response times of text data are not improved at all when SLP is fixed, while those of text data in experiment 3 are improved (up to 22.2% when RW=3) when SLP is predicted and updated based on PFS. In experiment 3, even though a processor-bound pro-

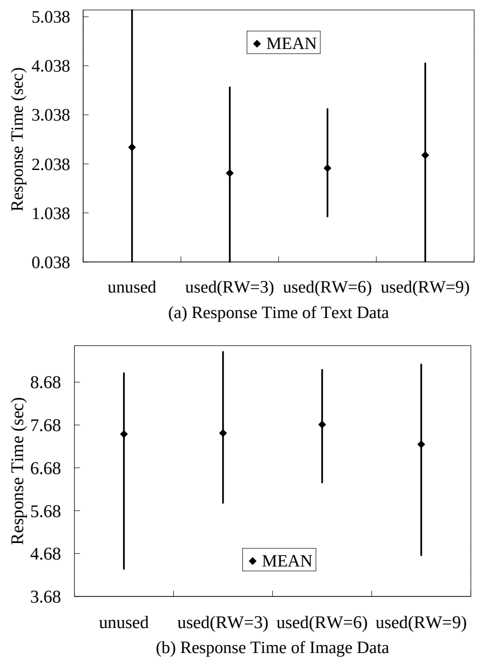

Fig. 5 The Effect of Our Scheduling Mechanism in Experiment 3.

cess causing the bottleneck of the server machine does not coexist, our scheduling mechanism still produces a good improvement. This might imply that a lot of simultaneous accesses from browsers could cause the processor of the server machine to become bottlenecked. If this supposition is true, then the improvement in experiment 3 means that SLP is accurately predicted and updated by our scheduling mechanism based on the execution behavior of the WWW server, because the only difference between experiments 2 and 3 is the way SLP was set.

4.3 Processor Load

In order to prove our supposition about experiment 3, we decided to find the relation between the number of simultaneous accesses and the response time, and the relation between the number of simultaneous accesses and the processor load (PL) or the length of ready queue (LRQ) by setting up experiment 4.

In experiment 4, we varied the number of client machines from 1 to 4 in which each machine has the same specification described in 4.1 and runs three browsers, we measured the response times of text data, PL and LRQ of the server machine under the same conditions as in experiment 3. PL is measured by checking if the processor is busy or idle every 10 milliseconds. If the processor is busy, PL is incremented. The PL ratio is the ratio of PL when the number of client machines is 1,2,3 and 4 to PL when the number of client machines is 1. LRQ is determined by the summation of the total number of processes in the ready queue every 10 milliseconds. The LRQ ratio is the ratio of LRQ when the number of machines is 1,2,3 and 4 to LRQ when the number of machines is 1.

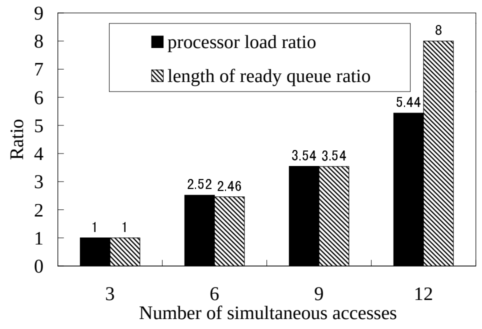

Fig. 6 The Relation between the Number of Simultaneous Accesses and the Processor Load or the Length of Ready Queue.

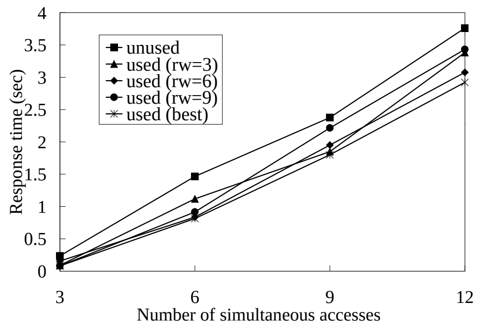

Fig. 7 The Relation between the Number of Simultaneous Accesses and the Response Time.

The results of experiment 4 are shown in **Fig. 6** and **Fig. 7**. Figure 6 plots the PL ratio (1:2.52:3.54:5.44) and the LRQ ratio (1:2.46:3.54:8). These ratios show that PL and LRQ increased at almost the same rate except when the number of simultaneous accesses is 12. Figure 6 shows that the more simultaneous accesses there are, the more PL and LRQ will increase. Therefore, a lot of simultaneous accesses cause the processor to become busy or bottlenecked and also increase LRQ. Figure 7 shows the mean response times of text data while not using our scheduling mechanism, using our scheduling mechanism for RW = 3,6,9 and when our mechanism produced the best result. Figure 7 shows that the more simultaneous accesses there are, the worse response time will be. Due to our policy of moving any server process handling an HTML file to the head of the ready queue when the processor becomes bottlenecked, which in this experiment is caused by the simultaneous accesses as shown in Fig. 6, the mean response times when using our scheduling mechanism shown in Fig. 7 are always better than when not using it. And this is the reason for the improvement in experiment 3.

In addition, in Fig. 7, the best mean response times, when the number of simultaneous accesses is 3,6,9 and 12, are improved 65.7%, 44.4%, 24.2% and 22.3% respectively. However, this improvement declined as the increase in the number of simultaneous accesses caused PL to increase. On the other hand, the effect of PL on the improvement dropped significantly when the number of simultaneous accesses was 12, by that point LRQ was increasing at a higher rate than PL. And this could be the result of our scheduling policy's manipulation on the ready queue.

According to our experimental results, any WWW server which experiences a lot of simultaneous accesses from browsers would benefit from our scheduling mechanism.

5. Conclusions

We evaluated the effectiveness of our proposed process scheduling policy by measuring the response time of a WWW server when it is accessed by a lot of browsers simultaneously, which is likely to be a realistic situation and could be the situation in which it is most desirable to improve the response time of a WWW server. Our experimental results show that the mean response times are improved greatly up to 33.8% in the best case in which the processor of the server machine is bottlenecked caused by a coexisting processor-bound process and accesses from browsers are set in such a way that the scheduling parameter SLP will be predicted 100% correctly. In a more realistic case in which no processor-bound process coexists and accesses from browsers are random, our scheduling mechanism still produces a good improvement of up to 22.2% when the scheduling parameter SLP is predicted and updated automatically by our scheduling mechanism based on the predicted execution behavior of the WWW server, called PFS. PFS is created and updated by our scheduling mechanism from the previous execution(s) of the WWW server. Due to the fact that a lot of simultaneous accesses cause the processor of the server machine to become bottlenecked and increase the length of ready queue, this improvement indicates that the scheduling parameter SLP is accurately predicted and updated by our scheduling mechanism based on the execution behavior of the WWW server. Therefore, any WWW server which experiences a lot of simultaneous accesses from browsers would benefit from our scheduling mechanism.

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

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

References

1) Xu, J. and Parnas, D.L.: On Satisfying Timing Constraints in Hard-Real-Time Systems, *IEEE Trans. Softw. Eng.*, Vol.19, No.1, pp.70–84 (1993).
2) Härbour, M.G., Klein, M.H. and Lehoczky, J.P.: Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 1, pp. 13–28 (1994).
3) Burns, A., Tindell, K. and Wellings, A.: Effective Analysis for Engineering Real-Time Fixed Priority Schedulers, *IEEE Trans. Softw. Eng.*, Vol.21, No.5, pp. 475–479 (1995).
4) Feng, W. and Liu, J.W.S.: Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines, *IEEE Trans. Softw. Eng.*, Vol. 23, No. 2, pp. 93–106 (1997).
5) Ford, B. and Susarla, S.: CPU Inheritance Scheduling, *Proc. 2nd OSDI 1996*, pp. 91–106 (1996).
6) Taniguchi, H.: POS:Program Oriented Schedule, *IPS Japan Proc. of Computer System Symposium'96*, Vol. 96, No. 7, pp. 123-130 (1996). (in Japanese)
7) http://www.microsoft.com/Windows98/usingwindows/maintaining/articles/811Nov/MNT-foundation2a.asp
8) Suranauwarat, S. and Taniguchi, H.: Process Scheduling Policy for a WWW Server Based on its Contents, *Trans. IPS Japan*, Vol. 40, No. 6, pp. 2510–2522 (1999). (in Japanese)
9) Kaashoek, M.F., Engler, D.R., Ganger, G.R., Bricenõ, H.M., Hunt, R., Mazierès, D., Pinckney, T., Grimm, R., Jannotti, J. and Mackenzie, K.: Application Performance and Flexibility on Exokernel Systems, *Proc. 16th SOSP 1997*.